Lecture Notes on Cryptography

Shafi Goldwasser[1] Mihir Bellare[2]

August 2001

[1] MIT Laboratory of Computer Science, 545 Technology Square, Cambridge, MA 02139, USA. E-mail: shafi@theory.lcs.mit.edu ; Web page: http://theory.lcs.mit.edu/ shafi

[2] Department of Computer Science and Engineering, Mail Code 0114, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093, USA. E-mail: mihir@cs.ucsd.edu ; Web page: http://www-cse.ucsd.edu/users/mihir

Foreword

This is a set of lecture notes on cryptography compiled for 6.87s, a one week long course on cryptography taught at MIT by Shafi Goldwasser and Mihir Bellare in the summers of 1996–2001. The notes were formed by merging notes written for Shafi Goldwasser's *Cryptography and Cryptanalysis* course at MIT with notes written for Mihir Bellare's *Cryptography and network security* course at UCSD. In addition, Rosario Gennaro (as Teaching Assistant for the course in 1996) contributed Section 9.6, Section 11.4, Section 11.5, and Appendix D to the notes, and also compiled, from various sources, some of the problems in Appendix E.

Cryptography is of course a vast subject. The thread followed by these notes is to develop and explain the notion of provable security and its usage for the design of secure protocols.

Much of the material in Chapters 2, 3 and 7 is a result of scribe notes, originally taken by MIT graduate students who attended Professor Goldwasser's *Cryptography and Cryptanalysis* course over the years, and later edited by Frank D'Ippolito who was a teaching assistant for the course in 1991. Frank also contributed much of the advanced number theoretic material in the Appendix. Some of the material in Chapter 3 is from the chapter on Cryptography, by R. Rivest, in the Handbook of Theoretical Computer Science.

Chapters 4, 5, 6, 8 and 10, and Sections 9.5 and 7.4.6, were written by Professor Bellare for his *Cryptography and network security* course at UCSD.

Shafi Goldwasser and Mihir Bellare   Cambridge, Massachusetts, August 2001.

Table of Contents

Chapter 1

Introduction to Modern Cryptography

Cryptography is about communication in the presence of an adversary. It encompasses many problems (encryption, authentication, key distribution to name a few). The field of modern cryptography provides a theoretical foundation based on which we may understand what exactly these problems are, how to evaluate protocols that purport to solve them, and how to build protocols in whose security we can have confidence. We introduce the basic issues by discussing the problem of encryption.

1.1 Encryption: Historical Glance

The most ancient and basic problem of cryptography is secure communication over an insecure channel. Party A wants to send to party B a secret message over a communication line which may be tapped by an adversary.

The traditional solution to this problem is called *private key encryption*. In private key encryption A and B hold a meeting before the remote transmission takes place and agree on a pair of encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$, and an additional piece of information S to be kept secret. We shall refer to S as the *common secret key*. The adversary may know the encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$ which are being used, but does not know S.

After the initial meeting when A wants to send B the *cleartext* or *plaintext* message m over the insecure communication line, A *encrypts* m by computing the *ciphertext* $c = \mathcal{E}(S, m)$ and sends c to B. Upon receipt, B *decrypts* c by computing $m = \mathcal{D}(S, c)$. The line-tapper (or adversary), who does not know S, should not be able to compute m from c.

Let us illustrate this general and informal setup with an example familiar to most of us from childhood, the *substitution cipher*. In this method A and B meet and agree on some secret permutation $f\colon \Sigma \to \Sigma$ (where Σ is the alphabet of the messages to be sent). To encrypt message $m = m_1 \dots m_n$ where $m_i \in \Sigma$, A computes $\mathcal{E}(f, m) = f(m_1) \dots f(m_n)$. To decrypt $c = c_1 \dots c_n$ where $c_i \in \Sigma$, B computes $\mathcal{D}(f, c) = f^{-1}(c_1) \dots f^{-1}(c_n) = m_1 \dots m_n = m$. In this example the common secret key is the permutation f. The encryption and decryption algorithms $\mathcal{E}$ and $\mathcal{D}$ are as specified, and are known to the adversary. We note that the substitution cipher is easy to break by an adversary who sees a moderate (as a function of the size of the alphabet Σ) number of ciphertexts.

A rigorous theory of perfect secrecy based on information theory was developed by Shannon [186] in 1943. [1]. In this theory, the adversary is assumed to have unlimited computational resources. Shannon showed

[1] Shannon's famous work on information theory was an outgrowth of his work on security ([187]).

that secure (properly defined) encryption system can exist only if the size of the secret information S that A and B agree on prior to remote transmission is as large as the number of secret bits to be ever exchanged remotely using the encryption system.

An example of a private key encryption method which is secure even in presence of a computationally unbounded adversary is the *one time pad*. A and B agree on a secret bit string $pad = b_1 b_2 \ldots b_n$, where $b_i \in_R \{0,1\}$ (i.e pad is chosen in $\{0,1\}^n$ with uniform probability). This is the common secret key. To encrypt a message $m = m_1 m_2 \ldots m_n$ where $m_i \in \{0,1\}$, A computes $\mathcal{E}(pad, m) = m \oplus pad$ (bitwise exclusive or). To decrypt ciphertext $c \in \{0,1\}^n$, B computes $\mathcal{D}(pad, c) = pad \oplus c = pad \oplus (m \oplus pad) = m$. It is easy to verify that $\forall m, c$ the $\mathbf{P}_{pad}\left[\mathcal{E}(pad, m) = c\right] = \frac{1}{2^n}$. From this, it can be argued that seeing c gives "no information" about what has been sent. (In the sense that the adversary's a posteriori probability of predicting m given c is no better than her a priori probability of predicting m without being given c.)

Now, suppose A wants to send B an additional message m'. If A were to simply send $c = \mathcal{E}(pad, m')$, then the sum of the lengths of messages m and m' will exceed the length of the secret key pad, and thus by Shannon's theory the system cannot be secure. Indeed, the adversary can compute $\mathcal{E}(pad, m) \oplus \mathcal{E}(pad, m') = m \oplus m'$ which gives information about m and m' (e.g. can tell which bits of m and m' are equal and which are different). To fix this, the length of the pad agreed upon a-priori should be the sum total of the length of all messages ever to be exchanged over the insecure communication line.

1.2 Modern Encryption: A Computational Complexity Based Theory

Modern cryptography abandons the assumption that the Adversary has available infinite computing resources, and assumes instead that the adversary's computation is resource bounded in some reasonable way. In particular, in these notes we will assume that the adversary is a probabilistic algorithm who runs in polynomial time. Similarly, the encryption and decryption algorithms designed are probabilistic and run in polynomial time.

The running time of the encryption, decryption, and the adversary algorithms are all measured as a function of a *security parameter* k which is a parameter which is fixed at the time the cryptosystem is setup. Thus, when we say that the adversary algorithm runs in polynomial time, we mean time bounded by some polynomial function in k.

Accordingly, in modern cryptography, we speak of the *infeasibility* of breaking the encryption system and computing information about exchanged messages where as historically one spoke of the *impossibility* of breaking the encryption system and finding information about exchanged messages. We note that the encryption systems which we will describe and claim "secure" with respect to the new adversary are not "secure" with respect to a computationally unbounded adversary in the way that the one-time pad system was secure against an unbounded adversary. But, on the other hand, it is no longer necessarily true that the size of the secret key that A and B meet and agree on before remote transmission must be as long as the total number of secret bits ever to be exchanged securely remotely. In fact, at the time of the initial meeting, A and B do not need to know in advance how many secret bits they intend to send in the future.

We will show how to construct such encryption systems, for which the number of messages to be exchanged securely can be a polynomial in the length of the common secret key. How we construct them brings us to anther fundamental issue, namely that of cryptographic, or complexity, assumptions.

As modern cryptography is based on a gap between efficient algorithms for encryption for the legitimate users versus the computational infeasibility of decryption for the adversary, it requires that one have available *primitives* with certain special kinds of computational hardness properties. Of these, perhaps the most basic is a *one-way function*. Informally, a function is one-way if it is easy to compute but hard to invert. Other primitives include *pseudo-random number generators*, and *pseudorandom function families*, which we will define and discuss later. From such primitives, it is possible to build secure encryption schemes.

Thus, a central issue is where these primitives come from. Although one-way functions are widely believed to exist, and there are several conjectured candidate one-way functions which are widely used, we currently do not know how to mathematically prove that they actually exist. We shall thus design cryptographic schemes assuming we are given a one-way function. We will use the conjectured candidate one-way functions for our working examples, throughout our notes. We will be explicit about what exactly can and cannot be proved and is thus assumed, attempting to keep the latter to a bare minimum.

We shall elaborate on various constructions of private-key encryption algorithms later in the course.

The development of *public key cryptography* in the seventies enables one to drop the requirement that A and B must share a key in order to encrypt. The receiver B can publish authenticated[2] information (called the *public-key*) for anyone including the adversary, the sender A, and any other sender to read at their convenience (e.g in a phone book). We will show encryption algorithms in which whoever can read the public key can send encrypted messages to B without ever having met B in person. The encryption system is no longer intended to be used by a pair of prespecified users, but by many senders wishing to send secret messages to a single recipient. The receiver keeps secret (to himself alone!) information (called the receiver's *private key*) about the public-key, which enables him to decrypt the cyphertexts he receives. We call such an encryption method *public key encryption*.

We will show that secure public key encryption is possible given a *trapdoor function*. Informally, a trapdoor function is a one-way function for which there exists some *trapdoor* information known to the receiver alone, with which the receiver can invert the function. The idea of public-key cryptosystems and trapdoor functions was introduced in the seminal work of Diffie and Hellman in 1976 [67, 68]. Soon after the first implementations of their idea were proposed in [170], [164], [137].

A simple construction of public key encryption from trapdoor functions goes as follows. Recipient B can choose at random a trapdoor function f and its associated trapdoor information t, and set its public key to be a description of f and its private key to be t. If A wants to send message m to B, A computes $\mathcal{E}(f,m) = f(m)$. To decrypt $c = f(m)$, B computes $f^{-1}(c) = f^{-1}(f(m)) = m$. We will show that this construction is not secure enough in general, but construct probabilistic variants of it which are secure.

1.3 A Short List of Candidate One Way Functions

As we said above, the most basic primitive for cryptographic applications is a one-way function which is "easy" to compute but "hard" to invert. (For public key encryption, it must also have a trapdoor.) By "easy", we mean that the function can be computed by a probabilistic polynomial time algorithm, and by "hard" that any probabilistic polynomial time (PPT) algorithm attempting to invert it will succeed with "small" probability (where the probability ranges over the elements in the domain of the function.) Thus, to qualify as a potential candidate for a one-way function, the hardness of inverting the function should not hold only on rare inputs to the function but with high probability over the inputs.

Several candidates which seem to posses the above properties have been proposed.

1. *Factoring.* The function $f : (x,y) \mapsto xy$ is conjectured to be a one way function. The asymptotically proven fastest factoring algorithms to date are variations on Dixon's random squares algorithm [126]. It is a randomized algorithm with running time $L(n)^{\sqrt{2}}$ where $L(n) = e^{\sqrt{\log n \log\log n}}$. The number field sieve by Lenstra, Lenstra, Manasee, and Pollard with modifications by Adlemann and Pomerance is a factoring algorithm proved under a certain set of assumptions to factor integers in expected time
$$e^{((c+o(1))(\log n)^{\frac{1}{3}}(\log\log n)^{\frac{2}{3}})}$$
[128, 3].
2. *The discrete log problem.* Let p be a prime. The multiplicative group $\mathcal{Z}_p^* = (\{x < p | (x,p) = 1\}, \cdot \bmod p)$ is cyclic, so that $\mathcal{Z}_p^* = \{g^i \bmod p | 1 \le i \le p-1\}$ for some generator $g \in \mathcal{Z}_p^*$. The function $f : (p,g,x) \mapsto$

[2] Saying that the information is "authenticated" means that the sender is given a guarantee that the information was published by the legal receiver. How this can be done is discussed in a later chapter.

$(g^x \bmod p, p, g)$ where p is a prime and g is a generator for Z_p^* is conjectured to be a one-way function. Computing $f(p, g, x)$ can be done in polynomial time using repeated squaring. However, The fastest known proved solution for its inverse, called the discrete log problem is the index-calculus algorithm, with expected running time $L(p)^{\sqrt{2}}$ (see [126]). An interesting problem is to find an algorithm which will generate a prime p and a generator g for Z_p^*. It is not known how to find generators in polynomial time. However, in [8], E. Bach shows how to generate random factored integers (in a given range $\frac{N}{2} \ldots N$). Coupled with a fast primality tester (as found in [126], for example), this can be used to efficiently generate random tuples $(p-1, q_1, \ldots, q_k)$ with p prime. Then picking $g \in \mathcal{Z}_p^*$ at random, it can be checked if $(g, p-1) = 1, \forall q_i, g^{\frac{p-1}{q_i}} \bmod p \neq 1$, and $g^{p-1} \bmod p = 1$, in which case $order(g) = p-1$ ($order(g) = |\{g^i \bmod p | 1 \leq i \leq p-1\}|$). It can be shown that the density of $\mathcal{Z}_p^*$ generators is high so that few guesses are required. The problem of efficiently finding a generator for a specific $\mathcal{Z}_p^*$ is an intriguing open research problem.

3. *Subset sum.* Let $a_i \in \{0,1\}^n, \vec{a} = (a_1, \ldots, a_n), s_i \in \{0,1\}, \vec{s} = (s_1, \ldots, s_n)$, and let $f : (\vec{a}, \vec{s}) \mapsto (\vec{a}, \sum_{i=1}^n s_i a_i)$. An inverse of $(\vec{a}, \sum_{i=1}^n s_i a_i)$ under f is any $(\vec{a}, \vec{s}_i')$ so that $\sum_{i=1}^n s_i a_i = \sum_{i=1}^n s_i' a_i$. This function f is a candidate for a one way function. The associated decision problem (given $(\vec{a}, y)$, does there exists $\vec{s}$ so that $\sum_{i=1}^n s_i a_i = y$?) is NP-complete. Of course, the fact that the subset-sum problem is NP-complete cannot serve as evidence to the one-wayness of $f_{\rm ss}$. On the other hand, the fact that the subset-sum problem is easy for special cases (such as "hidden structure" and low density) can not serve as evidence for the weakness of this proposal. The conjecture that f is one-way is based on the failure of known algorithm to handle random high density instances. Yet, one has to admit that the evidence in favor of this candidate is much weaker than the evidence in favor of the two previous ones.

4. *DES with fixed message.* Fix a 64 bit message M and define the function $f(K) = \mathrm{DES}_K(M)$ which takes a 56 bit key K to a 64 bit output $f(K)$. This appears to be a one-way function. Indeed, this construction can even be proven to be one-way assuming DES is a family of pseudorandom functions, as shown by Luby and Rackoff [134].

5. *RSA.* This is a candidate one-way trapdoor function. Let $N = pq$ be a product of two primes. It is believed that such an N is hard to factor. The function is $f(x) = x^e \bmod N$ where e is relatively prime to $(p-1)(q-1)$. The trapdoor is the primes p, q, knowledge of which allows one to invert f efficiently. The function f seems to be one-way. To date the best attack is to try to factor N, which seems computationally infeasible.

In Chapter 2 we discuss formal definitions of one-way functions and are more precise about the above constructions.

1.4 Security Definitions

So far we have used the terms "secure" and "break the system" quite loosely. What do we really mean? It is clear that a minimal requirement of security would be that: any adversary who can see the ciphertext and knows which encryption and decryption algorithms are being used, can not recover the entire cleartext. But, many more properties may be desirable. To name a few:

1. It should be hard to recover the messages from the ciphertext when the messages are drawn from arbitrary probability distributions defined on the set of all strings (i.e arbitrary *message spaces*). A few examples of message spaces are: the English language, the set $\{0,1\}$). We must assume that the message space is known to the adversary.

2. It should be hard to compute partial information about messages from the ciphertext.

3. It should be hard to detect simple but useful facts about traffic of messages, such as when the same message is sent twice.

4. The above properties should hold with high probability.

In short, it would be desirable for the encryption scheme to be the mathematical analogy of opaque envelopes containing a piece of paper on which the message is written. The envelopes should be such that all legal senders can fill it, but only the legal recipient can open it.

We must answer a few questions:

- How can "opaque envelopes" be captured in a precise mathematical definition? Much of Chapters 6 and 7 is dedicated to discussing the precise definition of security in presence of a computationally bounded adversary.
- Are "opaque envelopes" achievable mathematically? The answer is positive . We will describe the the proposals of private (and public) encryption schemes which we prove secure under various assumptions.

We note that the simple example of a public-key encryptions system based on trapdoor function, described in the previous section, does not satisfy the above properties. We will show later, however, probabilistic variants of the simple system which do satisfy the new security requirements under the assumption that trapdoor functions exist. More specifically, we will show probabilistic variants of RSA which satisfy the new security requirement under, the assumption that the original RSA function is a trapdoor function, and are similar in efficiency to the original RSA public-key encryption proposal.

1.5 The Model of Adversary

The entire discussion so far has essentially assumed that the adversary can listen to cyphertexts being exchanged over the insecure channel, read the public-file (in the case of public-key cryptography), generate encryptions of any message on his own (for the case of public-key encryption), and perform probabilistic polynomial time computation. This is called a *passive adversary*.

One may imagine a more powerful adversary who can intercept messages being transmitted from sender to receiver and either stop their delivery all together or alter them in some way. Even worse, suppose the adversary can request a polynomial number of cyphertexts to be decrypted for him. We can still ask whether there exists encryption schemes (public or secret) which are secure against such more powerful adversaries.

Indeed, such adversaries have been considered and encryption schemes which are secure against them designed. The definition of security against such adversaries is more elaborate than for passive adversaries.

In Chapters 6 and 7 we consider a passive adversary who knows the probability distribution over the message space. We will also discuss more powerful adversaries and appropriate definitions of security.

1.6 Road map to Encryption

To summarize the introduction, our challenge is to design both secure private-key and public-key encryption systems which provably meet our definition of security and in which the operations of encryption and decryption are as fast as possible for the sender and receiver.

Chapters 6 and 7 embark on an in depth investigation of the topic of encryption, consisting of the following parts. For both private-key and public-key encryption, we will:

- Discuss formally how to define security in presence of a bounded adversary.
- Discuss current proposals of encryption systems and evaluate them respect to the security definition chosen.
- Describe how to design encryption systems which we can prove secure under explicit assumptions such as the existence of one-way functions, trapdoor functions, or pseudo random functions.

- Discuss efficiency aspects of encryption proposals, pointing out to possible ways to improve efficiency by performing some computations off-line, in batch mode, or in a incremental fashion.

We will also overview some advanced topics connected to encryption such chosen-ciphertext security, non-malleability, key-escrow proposals, and the idea of shared decryption among many users of a network.

Chapter 2

One-way and trapdoor functions

One Way functions, namely functions that are "easy" to compute and "hard" to invert, are an extremely important cryptographic primitive. Probably the best known and simplest use of one-way functions, is for passwords. Namely, in a time-shared computer system, instead of storing a table of login passwords, one can store, for each password w, the value $f(w)$. Passwords can easily be checked for correctness at login, but even the system administrator can not deduce any user's password by examining the stored table.

In Section 1.3 we had provided a short list of some candidate one-way functions. We now develop a theoretical treatment of the subject of one-way and trapdoor functions, and carefully examine the candidate one-way functions proposed in the literature. We will occasionaly refer to facts about number theory discussed in Chapter C.

We begin by explaining why one-way functions are of fundamental importance to cryptography.

2.1 One-Way Functions: Motivation

In this section, we provide motivation to the definition of one-way functions. We argue that the existence of one-way functions is a necessary condition to the existence of most known cryptographic primitives (including secure encryption and digital signatures). As the current state of knowledge in complexity theory does not allow to prove the existence of one-way function, even using more traditional assumptions as $\mathcal{P} \neq \mathcal{NP}$, we will have to assume the existence of one-way functions. We will later try to provide evidence to the plausibility of this assumption.

As stated in the introduction chapter, modern cryptography is based on a gap between efficient algorithms guaranteed for the legitimate user versus the unfeasibility of retrieving protected information for an adversary. To make the following discussion more clear, let us concentrate on the cryptographic task of secure data communication, namely encryption schemes.

In secure encryption schemes, the legitimate user is able to decipher the messages (using some private information available to him), yet for an adversary (not having this private information) the task of decrypting the ciphertext (i.e., "breaking" the encryption) should be infeasible. Clearly, the breaking task can be performed by a non-deterministic polynomial-time machine. Yet, the security requirement states that breaking should not be feasible, namely could not be performed by a probabilistic polynomial-time machine. Hence, the existence of secure encryption schemes implies that there are tasks performed by non-deterministic polynomial-time machines yet cannot be performed by deterministic (or even randomized) polynomial-time machines. In other words, a necessary condition for the existence of secure encryption schemes is that $\mathcal{NP}$ is not contained in $\mathcal{BPP}$ (and hence that $\mathcal{P} \neq \mathcal{NP}$).

However, the above mentioned necessary condition (e.g., $\mathcal{P} \neq \mathcal{NP}$) is not a sufficient one. $\mathcal{P} \neq \mathcal{NP}$ only implies that the encryption scheme is hard to break in the worst case. It does not rule-out the possibility that the encryption scheme is easy to break in almost all cases. In fact, one can easily construct "encryption schemes" for which the breaking problem is NP-complete and yet there exist an efficient breaking algorithm that succeeds on 99% of the cases. Hence, worst-case hardness is a poor measure of security. Security requires hardness on most cases or at least average-case hardness. Hence, a necessary condition for the existence of secure encryption schemes is the existence of languages in $\mathcal{NP}$ which are hard on the average. Furthermore, $\mathcal{P} \neq \mathcal{NP}$ is not known to imply the existence of languages in $\mathcal{NP}$ which are hard on the average.

The mere existence of problems (in NP) which are hard on the average does not suffice. In order to be able to use such problems we must be able to generate such hard instances together with auxiliary information which enable to solve these instances fast. Otherwise, the hard instances will be hard also for the legitimate users and they gain no computational advantage over the adversary. Hence, the existence of secure encryption schemes implies the existence of an efficient way (i.e. probabilistic polynomial-time algorithm) of generating instances with corresponding auxiliary input so that

(1) it is easy to solve these instances given the auxiliary input; and

(2) it is hard on the average to solve these instances (when not given the auxiliary input).

We avoid formulating the above "definition". We only remark that the coin tosses used in order to generate the instance provide sufficient information to allow to efficiently solve the instance (as in item (1) above). Hence, without loss of generality one can replace condition (2) by requiring that these coin tosses are hard to retrieve from the instance. The last simplification of the above conditions essentially leads to the definition of a one-way function.

2.2 One-Way Functions: Definitions

In this section, we present several definitions of one-way functions. The first version, hereafter referred to as strong one-way function (or just one-way function), is the most convenient one. We also present weak one-way functions which may be easier to find and yet can be used to construct strong one way functios, and non-uniform one-way functions.

2.2.1 (Strong) One Way Functions

The most basic primitive for cryptographic applications is a one-way function. Informally, this is a function which is "easy" to compute but "hard" to invert. Namely, any probabilistic polynomial time (PPT) algorithm attempting to invert the one-way function on a element in its range, will succeed with no more than "negligible" probability, where the probability is taken over the elements in the domain of the function and the coin tosses of the PPT attempting the inversion.

This informal definition introduces a couple of measures that are prevalent in complexity theoretic cryptography. An easy computation is one which can be carried out by a PPT algorithm; and a function $\nu \colon \mathsf{N} \to \mathbf{R}$ is negligible if it vanishes faster than the inverse of any polynomial. More formally,

Definition 2.1 ν is negligible if for every constant $c \geq 0$ there exists an integer k_c such that $\nu(k) < k^{-c}$ for all $k \geq k_c$. ▮

Another way to think of it is $\nu(k) = k^{-\omega(1)}$.

A few words, concerning the notion of negligible probability, are in place. The above definition and discussion considers the success probability of an algorithm to be *negligible* if as a function of the input length the success probability is bounded by any polynomial fraction. It follows that repeating the algorithm polynomially (in the input length) many times yields a new algorithm that also has a negligible success probability. In other words, events which occur with negligible (in n) probability remain negligible even if the experiment is repeated for polynomially (in k) many times. Hence, defining negligible success as "occurring with probability smaller than any polynomial fraction" is naturally coupled with defining feasible as "computed within polynomial time". A "strong negation" of the notion of a negligible fraction/probability is the notion of a non-negligible fraction/probability. we say that a function ν is *non-negligible* if there exists a polynomial p such that for all sufficiently large k's it holds that $\nu(k) > \frac{1}{p(k)}$. Note that functions may be neither negligible nor non-negligible.

Definition 2.2 A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is *one-way* if:

(1) there exists a PPT that on input x output $f(x)$;

(2) For every PPT algorithm A there is a negligible function ν_A such that for sufficiently large k,

$$\mathbf{P}\left[\, f(z) = y \;:\; x \stackrel{R}{\leftarrow} \{0,1\}^k \;;\; y \leftarrow f(x) \;;\; z \leftarrow A(1^k, y) \,\right] \;\leq\; \nu_A(k)$$

▮

Remark 2.3 The guarantee is probabilistic. The adversary is not unable to invert the function, but has a low probability of doing so where the probability distribution is taken over the input x to the one-way function where x if of length k, and the possible coin tosses of the adversary. Namely, x is chosen at random and y is set to $f(x)$. ▮

Remark 2.4 The advsersary is not asked to find x; that would be pretty near impossible. It is asked to find some inverse of y. Naturally, if the function is 1-1 then the only inverse is x. ▮

Remark 2.5 Note that the adversary algorithm takes as input $f(x)$ and the security parameter 1^k (expressed in unary notatin) which corresponds to the binary length of x. This represents the fact the adversary can work in time polynomial in $|x|$, even if $f(x)$ happends to be much shorter. This rules out the possibility that a function is considered one-way merely because the inverting algorithm does not have enough time to print the output. Consider for example the function defined as $f(x) = y$ where y is the $\log k$ least significant bits of x where $|x| = k$. Since the $|f(x)| = \log |x|$ no algorithm can invert f in time polynomial in $|f(x)|$, yet there exists an obvious algorithm which finds an inverse of $f(x)$ in time polynomial in $|x|$. Note that in the special case of length preserving functions f (i.e., $|f(x)| = |x|$ for all x's), the auxiliary input is redundant. ▮

Remark 2.6 By this definition it trivially follows that the size of the output of f is bounded by a polynomial in k, since $f(x)$ is a poly-time computable. ▮

Remark 2.7 The definition which is typical to definitions from computational complexity theory, works with asymptotic complexity—what happens as the size of the problem becomes large. Security is only asked to hold for large enough input lengths, namely as k goes to infinity. Per this definition, it may be entirely feasible to invert f on, say, 512 bit inputs. Thus such definitions are less directly relevant to practice, but useful for studying things on a basic level. To apply this definition to practice in cryptography we must typically envisage not a single one-way function but a family of them, parameterized by a *security parameter* k. That is, for each value of the security parameter k there is be a specific function $f : \{0,1\}^k \to \{0,1\}^*$. Or, there may be a family of functions (or cryptosystems) for each value of k. We shall define such familes in subsequent section. ▮

The next two sections discuss variants of the strong one-way function definition. The first time reader is encouraged to directly go to Section 2.2.4.

2.2.2 Weak One-Way Functions

One way functions come in two flavors: strong and weak. The definition we gave above, refers to a strong way function. We could weaken it by replacing the second requirement in the definition of the function by a weaker requirement as follows.

Definition 2.8 A function $f\colon \{0,1\}^* \to \{0,1\}^*$ is *weak one-way* if:

(1) there exists a PPT that on input x output $f(x)$;

(2) There is a polynomial functions Q such that for every PPT algorithm A, and for sufficiently large k,

$$\mathbf{P}\left[\, f(z) \neq y \;:\; x \stackrel{R}{\leftarrow} \{0,1\}^k \;;\; y \leftarrow f(x) \;;\; z \leftarrow A(1^k, y) \,\right] \geq \frac{1}{Q(k)}$$

▮

The difference between the two definitions is that whereas we only require some non-negligible fraction of the inputs on which it is hard to invert a weak one-way function, a strong one-way function must be hard to invert on all but a negligible fraction of the inputs. Clearly, the latter is preferable, but what if only weak one-way functions exist ? Our first theorem is that the existence of a weak one way function implies the existence of a strong one way function. Moreover, we show how to construct a strong one-way function from a weak one. This is important in practice as illustarted by the following example.

Example 2.9 Consider for example the function $f : \mathbf{Z} \times \mathbf{Z} \mapsto \mathbf{Z}$ where $f(x,y) = x \cdot y$. This function can be easily inverted on at least half of its outputs (namely, on the even integers) and thus is not a strong one way function. Still, we said in the first lecture that f is hard to invert when x and y are primes of roughly the same length which is the case for a polynomial fraction of the k-bit composite integers. This motivated the definition of a weak one way function. Since the probability that an k-bit integer x is prime is approximately $1/k$, we get the probability that both x and y such that $|x| = |y| = k$ are prime is approximately $1/k^2$. Thus, for all k, about $1 - \frac{1}{k^2}$ of the inputs to f of length $2k$ are prime pairs of equal length. It is believed that no adversary can invert f when x and y are primes of the same length with non-negligible success probability, and under this belief, f is a weak one way function (as condition 2 in the above definition is satisfied for $Q(k) = O(k^2)$). ▮

Theorem 2.10 Weak one way functions exist if and only if strong one way functions exist. ▮

Proof Sketch: By definition, a strong one way function is a weak one way function. Now assume that f is a weak one way function such that Q is the polynomial in condition 2 in the definition of a weak one way function. Define the function

$$f_1(x_1 \ldots x_N) = f(x_1) \ldots f(x_N)$$

where $N = 2kQ(k)$ and each x_i is of length k.

We claim that f_1 is a strong one way function. Since f_1 is a concatenation of N copies of the function f, to correctly invert f_1, we need to invert $f(x_i)$ correctly for each i. We know that every adversary has a probability of at least $\frac{1}{Q(k)}$ to fail to invert $f(x)$ (where the probability is taken over $x \in \{0,1\}^k$ and the coin tosses of the adversary), and so intuitively, to invert f_1 we need to invert $O(kQ(k))$ instances of f. The probability that the adversary will fail for at least one of these instances is extremely high.

The formal proof (which is omitted here and will be given in appendix) will take the form of a reduction; that is, we will assume for contradiction that f_1 is not a strong one way function and that there exists some adversary A_1 that violates condition 2 in the definition of a strong one way function. We will then show that A_1 can be used as a subroutine by a new adversary A that will be able to invert the original function f with probability better than $1 - \frac{1}{Q(|x|)}$ (where the probability is taken over the inputs $x \in \{0,1\}^k$ and the coin tosses of A). But this will mean that f is not a weak one way function and we have derived a contradiction. ∎

This proof technique is quite typical of proofs presented in this course. Whenever such a proof is presented it is important to examine the cost of the reduction. For example, the construction we have just outlined is not length preserving, but expands the size of the input to the function quadratically.

2.2.3 Non-Uniform One-Way Functions

In the above two definitions of one-way functions the inverting algorithm is probabilistic polynomial-time. Stronger versions of both definitions require that the functions cannot be inverted even by non-uniform families of polynomial size algorithm We stress that the "easy to compute" condition is still stated in terms of uniform algorithms. For example, following is a non-uniform version of the definition of (strong) one-way functions.

Definition 2.11 A function f is called *non-uniformly strong one-way* if the following two conditions hold

(1) *easy to compute:* as before There exists a PPT algorithm to compute for f.

(2) *hard to invert:* For every (even non-uniform) family of polynomial-size algorithms $A = \{M_k\}_{k \in \mathsf{N}}$, there exists a negligble ν_A such that for all sufficiently large k's

$$\mathbf{P}\left[\, f(z) \neq y \;:\; x \stackrel{R}{\leftarrow} \{0,1\}^k \;;\; y \leftarrow f(x) \;;\; z \leftarrow M_k(y) \,\right] \;\leq\; \nu_A(k)$$

∎

Note that it is redundent to give 1^k as an auxiliary input to M_k.

It can be shown that if f is non-uniformly one-way then it is (strongly) one-way (i.e., in the uniform sense). The proof follows by converting any (uniform) probabilistic polynomial-time inverting algorithm into a non-uniform family of polynomial-size algorithm, without decreasing the success probability. Details follow. Let A' be a probabilistic polynomial-time (inverting) algorithm. Let r_k denote a sequence of coin tosses for A' maximizing the success probability of A'. The desired algorithm M_k incorporates the code of algorithm A' and the sequence r_k (which is of length polynomial in k).

It is possible, yet not very plausible, that strongly one-way functions exist and but there are no non-uniformly one-way functions.

2.2.4 Collections Of One Way Functions

Instead of talking about a single function $f : \{0,1\}^* \rightarrow \{0,1\}^*$, it is often convenient to talk about collections of functions, each defined over some finite domain and finite ranges. We remark, however, that the single function format makes it easier to prove properties about one way functions.

Definition 2.12 Let I be a set of indices and for $i \in I$ let D_i and R_i be finite. A collection of strong one way functions is a set $F = \{f_i : D_i \rightarrow R_i\}_{i \in I}$ satisfying the following conditions.

(1) There exists a PPT S_1 which on input 1^k outputs an $i \in \{0,1\}^k \cap I$

(2) There exists a PPT S_2 which on input $i \in I$ outputs $x \in D_i$

(3) There exists a PPT A_1 such that for $i \in I$ and $x \in D_i$, $A_1(i,x) = f_i(x)$.

(4) For every PPT A there exists a negligible ν_A such that $\forall$ k large enough

$$\mathbf{P}\left[f_i(z) = y \ : \ i \stackrel{R}{\leftarrow} I \ ; \ x \stackrel{R}{\leftarrow} D_i \ ; \ y \leftarrow f_i(x) \ ; \ z \leftarrow A(i,y) \right] \ \leq \ \nu_A(k)$$

(here the probability is taken over choices of i and x, and the coin tosses of A).

▮

In general, we can show that the existence of a single one way function is equivalent to the existence of a collection of one way functions. We prove this next.

Theorem 2.13 A collection of one way functions exists if and only if one way functions exist. ▮

Proof: Suppose that f is a one way function.

Set $F = \{f_i : D_i \to R_i\}_{i \in I}$ where $I = \{0,1\}^*$ and for $i \in I$, take $D_i = R_i = \{0,1\}^{|i|}$ and $f_i(x) = f(x)$. Furthermore, S_1 uniformly chooses on input 1^k, $i \in \{0,1\}^k$, S_2 uniformly chooses on input i, $x \in D_i = \{0,1\}^{|i|}$ and $A_1(i,x) = f_i(x) = f(x)$. (Note that f is polynomial time computable.) Condition 4 in the definition of a collection of one way functions clearly follows from the similar condition for f to be a one way function.

Now suppose that $F = \{f_i : D_i \to R_i\}_{i \in I}$ is a collection of one way functions. Define $f_F(1^k, r_1, r_2) = A_1(S_1(1^k, r_1), S_2(S_1(1^k, r_1), r_2))$ where A_1, S_1, and S_2 are the functions associated with F as defined in Definition 2.12. In other words, f_F takes as input a string $1^k \circ r_1 \circ r_2$ where r_1 and r_2 will be the coin tosses of S_1 and S_2, respectively, and then

- Runs S_1 on input 1^k using the coin tosses r_1 to get the index $i = S_1(1^k, r_1)$ of a function $f_i \in F$.
- Runs S_2 on the output i of S_1 using the coin tosses r_2 to find an input $x = S_2(i, r_2)$.
- Runs A_1 on i and x to compute $f_F(1^k, r_1, r_2) = A_1(i, x) = f_i(x)$.

Note that randomization has been restricted to the input of f_F and since A_1 is computable in polynomial time, the conditions of a one way function are clearly met. ▮

A possible example is the following, treated thoroughly in Section 2.3.

Example 2.14 The hardness of computing discrete logarithms yields the following collection of functions. Define $EXP = \{EXP_{p,g}(i) = g^i \bmod p,\ EXPp, g : Z_p \to Z_p^*\}_{<p,g>\in I}$ for $I = \{<p, g> p \text{ prime}, g \text{ generator for } Z_p^*\}$. ▮

2.2.5 Trapdoor Functions and Collections

Infromally, a *trapdoor function* f is a one-way function with an extra property. There also exists a secret inverse function (the*trapdoor*) that allows its possessor to efficiently invert f at any point in the domain of his choosing. It should be easy to compute f on any point, but infeasible to invert f on any point without knowledge of the inverse function . Moreover, it should be easy to generate matched pairs of f's and corresponding trapdoor. Once a matched pair is generated, the publication of f should not reveal anything about how to compute its inverse on any point.

Definition 2.15 A *trapdoor* function is a one-way function $f : \{0,1\}^* \to \{0,1\}^*$ such that there exists a polynomial p and a probabilistic polynomial time algorithm I such that for every k there exists an $t_k \in \{0,1\}^*$ such that $|t_k| \leq p(k)$ and for all $x \in \{0,1\}^*$, $I(f(x), t_k) = y$ such that $f(y) = f(x)$. ▮

An example of a function which may be trapdoor if factoring integers is hard was proposed by Rabin[164]. Let $f(x,n) = x^2 \bmod n$ where $n = pq$ a product of two primes and $x \in Z_n^*$. Rabin[164] has shown that inverting f is easy iff factoring composite numbers product of two primes is easy. The most famous candidate trapdoor function is the RSA[170] function $f(x,n,l) = x^l \bmod n$ where $(l, \phi(n)) = 1$.

Again it will be more convenient to speak of families of trapdoor functions parameterized by security parameter k.

Definition 2.16 Let I be a set of indices and for $i \in I$ let D_i be finite. A collection of strong one way trapdoor functions is a set $F = \{f_i : D_i \to D_i\}_{i \in I}$ satisfying the following conditions.

(1) There exists a polynomial p and a PTM S_1 which on input 1^k outputs pairs (i, t_i) where $i \in I \cap \{0,1\}^k$ and $|t_i| < p(k)$ The information t_i is referred to as the trapdoor of i.

(2) There exists a PTM S_2 which on input $i \in I$ outputs $x \in D_i$

(3) There exists a PTM A_1 such that for $i \in I$, $x \in D_i$ $A_1(i,x) = f_i(x)$.

(4) There exists a PTM A_2 such that $A_2(i, t_i, f_i(x)) = x$ for all $x \in D_i$ and for all $i \in I$ (that is, f_i is easy to invert when t_i is known).

(5) For every PPT A there exists a negligible ν_A such that $\forall$ k large enough

$$\mathbf{P}\left[\, f_i(z) = y \ : \ i \stackrel{R}{\leftarrow} I \ ; \ x \stackrel{R}{\leftarrow} D_i \ ; \ y \leftarrow f_i(x) \ ; \ z \leftarrow A(i,y) \,\right] \ \leq \ \nu_A(k)$$

∎

A possible example is the following treated in in detail in the next sections.

Example 2.17 [The RSA collections of possible trapdoor functions] Let p, q denote primes, $n = pq$, $Z_n^* = \{1 \leq x \leq n, (x,n) = 1\}$ the multiplicative group whose cardinality is $\varphi(n) = (p-1)(q-1)$, and $e \in Z_{p-1}$ relatively prime to $\varphi(n)$. Our set of indices will be $I = \{< n, e > \text{ such that } n = pq \ |p| = |q|\}$ and the trapdoor associated with the particular index $< n, e >$ be d such that $ed = 1 \bmod \phi(n)$. Let $RSA = \{RSA_{<n,e>} : Z_n^* \to Z_n^*\}_{<n,e> \in I}$ where $RSA_{<n,e>}(x) = x^e \bmod n$ ∎

2.3 In Search of Examples

Number theory provides a source of candidates for one way and trapdoor functions. Let us start our search for examples by a digression into number theorey. See also the mini-course on number theory in Appendix C.

Calculating Inverses in Z_p^*

Consider the set $\mathbf{Z}_p^* = \{x : 1 \leq x < p \text{ and } \gcd(x,p) = 1\}$ where p is prime. $\mathbf{Z}_p^*$ is a group under multiplicaton modulo p. Note that to find the inverse of $x \in \mathbf{Z}_p^*$; that is, an element $y \in \mathbf{Z}_p^*$ such that $yx \equiv 1 \bmod p$, we can use the Euclidean algorithm to find integers y and z such that $yx + zp = 1 = \gcd(x,p)$. Then, it follows that $yx \equiv 1 \bmod p$ and so $y \bmod p$ is the desired inverse.

The Euler Totient Function $\varphi(n)$

Euler's Totient Function φ is defined by $\varphi(n) = |\{x : 1 \leq x < p \text{ and } \gcd(x,n) = 1\}$. The following are facts about φ.

(1) For p a prime and $\alpha \geq 1$, $\varphi(p^\alpha) = p^{\alpha-1}(p-1)$.

(2) For integers m, n with $\gcd(m,n) = 1$, $\varphi(mn) = \varphi(m)\varphi(n)$.

Using the rules above, we can find φ for any n because, in general,

$$\begin{aligned}
\varphi(n) &= \varphi(\prod_{i=1}^{k} {p_i}^{\alpha_i}) \\
&= \prod_{i=1}^{k} \varphi({p_i}^{\alpha_i}) \\
&= \prod_{i=1}^{k} {p_i}^{\alpha_i - 1}(p_i - 1)
\end{aligned}$$

Z_p^* Is Cyclic

A group G is cyclic if and only if there is an element $g \in G$ such that for every $a \in G$, there is an integer i such that $g^i = a$. We call g a *generator* of the group G and we denote the index i by $ind_g(a)$.

Theorem 2.18 (Gauss) If p is prime then $\mathbf{Z}_p^*$ is a cyclic group of order $p-1$. That is, there is an element $g \in \mathbf{Z}_p^*$ such that $g^{p-1} \equiv 1 \bmod p$ and $g^i \not\equiv 1 \bmod p$ for $i < p-1$. ∎

¿From Theorem 2.18 the following fact is immediate.

Fact 2.19 Given a prime p, a generator g for $\mathbf{Z}_p^*$, and an element $a \in \mathbf{Z}_p^*$, there is a unique $1 \leq i \leq p-1$ such that $a = g^i$. ∎

The Legendre Symbol

Fact 2.20 If p is a prime and g is a generator of $\mathbf{Z}_p^*$, then

$$g^c = g^a g^b \bmod p \Leftrightarrow c = a + b \bmod p - 1$$

∎

¿From this fact it follows that there is an homomorphism $f : \mathbf{Z}_p^* \rightarrow \mathbf{Z}_{p-1}$ such that $f(ab) = f(a) + f(b)$. As a result we can work with $\mathbf{Z}_{p-1}$ rather than $\mathbf{Z}_p^*$ which sometimes simplifies matters. For example, suppose we wish to determine how many elements in $\mathbf{Z}_p^*$ are perfect squares (these elements will be referred to as *quadratic residues* modulo p). The following lemma tells us that the number of quadratic residues modulo p is $\frac{1}{2}|\mathbf{Z}_p^*|$.

Lemma 2.21 $a \in \mathbf{Z}_p^*$ is a quadratic residue modulo p if and only if $a = g^x \bmod p$ where x satisfies $1 \leq x \leq p-1$ and is even. ∎

Proof: Let g be a generator in $\mathbf{Z}_p^*$.
($\Leftarrow$) Suppose an element $a = g^{2x}$ for some x. Then $a = s^2$ where $s = g^x$.
($\Rightarrow$) Consider the square of an element $b = g^y$. $b^2 = g^{2y} \equiv g^e \bmod p$ where e is even since $2y$ is reduced modulo $p-1$ which is even. Therefore, only those elements which can be expressed as g^e, for e an even integer, are squares. ∎

Consequently, the number of quadratic residues modulo p is the number of elements in $\mathbf{Z}_p^*$ which are an even power of some given generator g. This number is clearly $\frac{1}{2}|\mathbf{Z}_p^*|$.

The *Legendre Symbol* $\mathbf{J}_p(x)$ specifies whether x is a perfect square in $\mathbf{Z}_p^*$ where p is a prime.

$$\mathbf{J}_p(x) = \begin{cases} 1 & \text{if } x \text{ is a square in } \mathbf{Z}_p^* \\ 0 & \text{if } \gcd(x,p) \neq 1 \\ -1 & \text{if } x \text{ is not a square in } \mathbf{Z}_p^* \end{cases}$$

The Legendre Symbol can be calculated in polynomial time due to the following theorem.

Theorem 2.22 [Euler's Criterion] $\mathbf{J}_p(x) \equiv x^{\frac{p-1}{2}} \bmod p$. ▮

Using repeated doubling to compute exponentials, one can calculate $x^{\frac{p-1}{2}}$ in $O(|p|^3)$ steps. Though this $\mathbf{J}_p(x)$ can be calculated when p is a prime, it is not known how to determine for general x and n, whether x is a square in $\mathbf{Z}_n^*$.

2.3.1 The Discrete Logarithm Function

Let EXP be the function defined by $\mathrm{EXP}(p,g,x) = (p,g,g^x \bmod p)$. We are particularly interested in the case when p is a prime and g is a generator of Z_p^*. Deine an index set $I = \{(p,g) : p \text{ is prime and } g \text{ is a generator of } Z_p^*\}$. For $(p,g) \in I$, it follows by Fact 2.19 that $\mathrm{EXP}(p,g,x)$ has a unique inverse and this allows us to define for $y \in \mathbf{Z}_p^*$ the discrete logarithm function DL by $\mathrm{DL}(p,g,y) = (p,g,x)$ where $x \in \mathbf{Z}_{p-1}$ and $g^x \equiv y \bmod p$. Given p and g, $\mathrm{EXP}(p,g,x)$ can easily be computed in polynomial time. However, it is unknown whether or not its inverse DL can be computed in polynomial time unless $p-1$ has very small factors (see [158]). Pohlig and Hellman [158] present effective techniques for this problem when $p-1$ has only small prime factors.

The best fully proved up-to-date algorithm for computing discrete logs is the Index-calculus algorithm. The expected running time of such algorithm is polynomial in $e^{\sqrt{k \log k}}$ where k is the size of the modulos p. There is a recent variant of the number field sieve algorithm for discrete logarithm which seems to work in faster running time of $e^{(k \log k)^{\frac{1}{3}}}$. It interesting to note that working over the finite field $GF(2^k)$ rather than working modulo p seems to make the problem substantially easier (see Coppersmith [57] and Odlyzko [152]). Curiously, computing discrete logarithms and factoring integers seem to have essentially the same difficulty at least as indicated by the current state of the art algorithms.

With all this in mind, we consider EXP a good candidate for a one way function. We make the following explicit assumption in this direction. The assumption basically says that there exists no polynomial time algorithm that can solvethe discrete log problem with prime modulos.

Strong Discrete Logarithm Assumption (DLA):[1] For every polynomial Q and every PPT A, for all sufficiently large k,

$$\Pr[A(p,g,y) = x \text{ such that } y \equiv g^x \bmod p \text{ where } 1 \leq x \leq p-1] < \frac{1}{Q(k)}$$

(where the probability is taken over all primes p such that $|p| \leq k$, the generators g of $\mathbf{Z}_p^*$, $x \in \mathbf{Z}_p^*$ and the coin tosses of A).

An immediate consequence of this assumption we get

Theorem 2.23 Under the strong discrete logarithm assumption there exists a strong one way function; namely, exponentiation modulo a prime p. ▮

[1] We note that a weaker assumption can be made concerning the discrete logarithm problem, and by the standard construction one can still construct a strong one-way function. We will assume for the purpose of the course the first stronger assumption. **Weak Discrete Logarithm Assumption:** There is a polynomial Q such that for every PTM A there exists an integer k_0 such that $\forall k > k_0$ $\Pr[A(p,g,y) = x \text{ such that } y \equiv g^x \bmod p \text{ where } 1 \leq x \leq p-1] < 1 - \frac{1}{Q(k)}$ (where the probability is taken over all primes p such that $|p| \leq k$, the generators g of $\mathbf{Z}_p^*$, $x \in \mathbf{Z}_p^*$ and the coin tosses of A).

Some useful properties of EXP and DL follow.

Remark 2.24 If $\mathrm{DL}(p, g_1, y)$ is easy to calculate for some generator $g_1 \in \mathbf{Z}_p^*$ then it is also easy to calculate $\mathrm{DL}(p, g_2, y)$ for any other generator $g_2 \in \mathbf{Z}_p^*$. (The group $\mathbf{Z}_p^*$ has $\varphi(p-1)$ generators.) To see this suppose that $x_1 = \mathrm{DL}(p, g_1, y)$ and $x_2 = \mathrm{DL}(p, g_2, y)$. If $g_2 \equiv {g_1}^z \bmod p$ where $\gcd(z, p-1)$ then $y \equiv {g_1}^{x_2 z} \bmod p$ and consequently, $x_2 \equiv z^{-1} x_1 \bmod p-1$. ▮

The following result shows that to efficiently calculate $\mathrm{DL}(p, g, y)$ for $(p, g) \in I$ it will suffice to find a polynomial time algorithm which can calculate $\mathrm{DL}(p, g, y)$ on at least a $\frac{1}{Q(|p|)}$ fraction of the possible inputs $y \in \mathbf{Z}_p^*$ for some polynomial Q.

Proposition 2.25 Let ϵ, $\delta \in (0, 1)$ and let S be a subset of the prime integers. Suppose there is a probabilistic algorithm A such that for all primes $p \in S$ and for all generators g of $\mathbf{Z}_p^*$

$$\Pr[A(p, g, y) = x \text{ such that } g^x \equiv y \bmod p] > \epsilon$$

(where the probability is taken over $y \in \mathbf{Z}_p^*$ and the coin tosses of A) and A runs in time polynomial in $|p|$. Then there is a probabilistic algorithm A′ running in time polynomial in $\epsilon^{-1}, \delta^{-1}$, and $|p|$ such that for all primes $p \in S$, generators g of $\mathbf{Z}_p^*$, and $y \in \mathbf{Z}_p^*$

$$\Pr[A'(p, g, y) = x \text{ such that } g^x \equiv y \bmod p] > 1 - \delta$$

(where the probability is taken over the coin tosses of A′). ▮

Proof: Choose the smallest integer N for which $\frac{1}{e^N} < \delta$.

Consider the algorithm A′ running as follows on inputs $p \in S$, g a generator of $\mathbf{Z}_p^*$ and $y \in \mathbf{Z}_p^*$.

Repeat $\epsilon^{-1} N$ times.
  Randomly choose z such that $1 \le z \le p-1$.
  Let $w = A(p, g, g^z y)$
  If A succeeds then $g^w = g^z y = g^{z+x} \bmod p$ where $x = \mathrm{DL}_{p,g}(y)$
    and therefore $\mathrm{DL}_{p,g}(y) = w - z \bmod p-1$.
  Otherwise, continue to next iteration.
End loop

We can estimate the probability that A′ fails:

$$\begin{aligned}\Pr[A'(p, g, y) \text{ fails}] &= \Pr[\text{A single iteration of the loop of A}' \text{ fails}]^{\epsilon^{-1} N} \\ &< (1-\epsilon)^{\epsilon^{-1} N} \\ &< (e^{-N}) \\ &< \delta\end{aligned}$$

Note that since $N = O(\log(\delta^{-1})) = O(\delta^{-1})$, A′ is a probabilistic algorithm which runs in time polynomial in ϵ^{-1}, δ^{-1}, and $|p|$. ▮

The discrete logarithm problem also yields the following collection of functions.

Let $I = \{(p, g) : p \text{ is prime and } g \text{ is a generator of } \mathbf{Z}_p^*\}$ and define

$$\mathrm{EXP} = \{\mathrm{EXP}_{p,g} : \mathbf{Z}_{p-1} \to \mathbf{Z}_p^* \text{ where } \mathrm{EXP}_{p,g}(x) = g^x \bmod p\}_{(p,g) \in I}.$$

Then, under the strong discrete logarithm assumption, EXP is a collection of strong one way functions. This claim will be shown to be true next.

Theorem 2.26 Under the strong discrete logarithm assumption there exists a collection of strong one way functions. ▮

Proof: We shall show that under the DLA EXP is indeed a collection of one way functions. For this we must show that it satisfies each of the conditions in the definition of a collection of one way functions.

For condition 1, define S_1 to run as follows on input 1^k.

(1) Run Bach's algorithm (given in [8]) to get a random integer n such that $|n| = k$ along with its factorization.

(2) Test whether $n+1$ is prime. See primality testing in section C.9.

(3) If so, let $p = n+1$. Given the prime factorization of $p-1$ we look for generators g of $\mathbf{Z}_p^*$ as follows.

  (1) Choose $g \in \mathbf{Z}_p^*$ at random.

  (2) If $p - 1 = \prod_i {q_i}^{\alpha_i}$ is the prime factorization of $p-1$ then for each q_i check that $g^{\frac{p-1}{q_i}} \not\equiv 1 \bmod p$.

  If so, then g is a generator of $\mathbf{Z}_p^*$. Output p and g.

  Otherwise, repeat from step 1.

Claim 2.27 g is a generator of $\mathbf{Z}_p^*$ if for each prime divisor q of $p-1$, $g^{\frac{p-1}{q}} \not\equiv 1 \bmod p$. ▮

Proof: The element g is a generator of $\mathbf{Z}_p^*$ if $g^{p-1} \equiv 1 \bmod p$ and $g^j \not\equiv 1 \bmod p$ for all j such that $1 \leq j < p-1$; that is, g has order $p-1$ in $\mathbf{Z}_p^*$.

Now, suppose that g satisfies the condition of Claim 2.27 and let m be the order of g in $\mathbf{Z}_p^*$. Then $m \mid p-1$. If $m < p-1$ then there exists a prime q such that $m \mid \frac{p-1}{q}$; that is, there is an integer d such that $md = \frac{p-1}{q}$. Therefore $g^{\frac{p-1}{q}} = (g^m)^d \equiv 1 \bmod n$ contradicting the hypothesis. Hence, $m = p-1$ and g is a generator of $\mathbf{Z}_p^*$. ▮

Also, note that the number of generators in $\mathbf{Z}_p^*$ is $\varphi(p-1)$ and in [172] it is shown that

$$\varphi(k) > \frac{k}{6 \log \log k}.$$

Thus we expect to have to choose $O(\log \log p)$ candidates for g before we obtain a generator. Hence, S_1 runs in expected polynomial time.

For condition 2 in the definition of a collection of one way functions, we can define S_2 to simply output $x \in Z_{p-1}$ at random given $i = (p, g)$.

Condition 3 is true since the computation of $g^x \bmod p$ can be performed in polynomial time and condition 4 follows from the strong discrete logarithm assumption. ▮

2.3.2 The RSA function

In 1977 Rivest, Shamir, and Adleman [170] proposed trapdoor function candidate motivated by finding a public-key cryptosystem satisfying the requirements proposed by Diffie and Hellman. The trapdoor function proposed is $RSA(n,e,x) = x^e \bmod n$ where the case of interest is that n is the product of two large primes p and q and $\gcd(e,\phi(n)) = 1$. The corresponding trapdoor information is d such that $d \cdot e \equiv 1 \bmod \phi(n)$.

Viewd as a collection, let RSA $= \{\mathrm{RSA}_{n,e} : \mathbf{Z}_n^* \to \mathbf{Z}_n^*$ where $\mathrm{RSA}_{n,e}(x) = x^e \bmod n\}_{(n,e)\in I}$. for $I = \{< n, e >$ s.t. $n = pq$ $|p| = |q|, (e,\phi(n)) = 1\}$.

RSA is easy to compute. How hard is it to invert? We know that if we can factor n we can invert RSA via the Chinese Remainder Theorem, however we don't know if the converse is true. Thus far, the best way known to invert RSA is to first factor n. There are a variety of algorithms for this task. The best running time for a fully proved algorithm is Dixon's random squares algorithms which runs in time $O(e^{\sqrt{\log n \log\log n}})$. In practice we may consider others. Let $\ell = |p|$ where p is the smallest prime divisor of n. The Elliptic Curve algorithm takes expected time $O(e^{\sqrt{2\ell \log \ell}})$. The Quadratic Sieve algorithm runs in expected $O(e^{\sqrt{\ln n \ln\ln n}})$. Notice the difference in the argument of the superpolynomial component of the running time. This means that when we suspect that one prime factor is substantially smaller than the other, we should use the Elliptic Curve method, otherwise one should use the Quadratic sieve. The new number field sieve algorithm seems to achieve a $O(e^{1.9(\ln n)^{1/3}(\ln\ln n)^{2/3}})$ running time which is a substantial improvement asymptotically although in practice it still does not seem to run faster than the Quadratic Sieve algorithm for the size of integers which people currently attempt to factor. The recommended size for n these days is 1024 bits.

With all this in mind, we make an explicit assumption under which one can prove that RSA provides a collection of trapdoor functions.

Strong RSA Assumption:[2] Let $H_k = \{n = pq : p \neq q$ are primes and $|p| = |q| = k\}$. Then for every polynomial Q and every PTM A, there exists an integer k_0 such that $\forall k > k_0$

$$\Pr[A(n, e, \mathrm{RSA}_{n,e}(x)) = x] < \frac{1}{Q(k)}$$

(where the probability is taken over all $n \in H_k$, e such that $\gcd(e,\varphi(n)) = 1$, $x \in \mathbf{Z}_n^*$, and the coin tosses of A).

We need to prove some auxilary claims.

Claim 2.28 For $(n,e) \in I$, $\mathrm{RSA}_{n,e}$ is a permutation over $\mathbf{Z}_n^*$. ∎

Proof: Since $\gcd(e,\varphi(n)) = 1$ there exists an integer d such that $ed \equiv 1 \bmod \varphi(n)$. Given $x \in \mathbf{Z}_n^*$, consider the element $x^d \in \mathbf{Z}_n^*$. Then $\mathrm{RSA}_{n,e}(x^d) \equiv (x^d)^e \equiv x^{ed} \equiv x \bmod n$. Thus, the function $\mathrm{RSA}_{n,e} : \mathbf{Z}_n^* \longrightarrow \mathbf{Z}_n^*$ is onto and since $|\mathbf{Z}_n^*|$ is finite it follows that $\mathrm{RSA}_{n,e}$ is a permutation over $\mathbf{Z}_n^*$. ∎

Remark 2.29 Note that the above is a constructive proof that RSA has an unique inverse. Since $\gcd(e,\varphi(n)) = 1$ if we run the extended Euclidean algorithm we can find $d \in \mathbf{Z}_n^*$ such that

$$RSA_{n,e}^{-1}(x) = (x^e \bmod n)^d \bmod n = x^{ed} \bmod n = x \bmod n$$

. Note that once we found a d such that $ed \equiv 1 \bmod \varphi(n)$ then we can invert $\mathrm{RSA}_{n,e}$ efficiently because then $\mathrm{RSA}_{n,e}(x)^d \equiv x^{ed} \equiv x \bmod \varphi(n)$. ∎

Theorem 2.30 Under the strong RSA assumption, RSA is a collection of strong one way trapdoor permutations. ∎

[2] A weaker assumption can be made which under standard constructions is equivalent to the stronger one which is made in this class. **Weak RSA Assumption:** Let $H_k = \{n = pq : p \neq q$ are prime and $|p| = |q| = k\}$. There is a polynomial Q such that for every PTM A, there exists an integer k_0 such that $\forall k > k_0$ $\Pr[A(n,e,\mathrm{RSA}_{n,e}(x)) = x] < 1 - \frac{1}{Q(k)}$ (where the probability is taken over all $n \in H_k$, e such that $\gcd(e,\varphi(n)) = 1$, $x \in \mathbf{Z}_n^*$, and the coin tosses of A).

Proof: First note that by Claim 2.28, $\mathrm{RSA}_{n,e}$ is a permutation of $\mathbf{Z}_n^*$. We must also show that RSA satisfies each of the conditions in Definition 2.16. For condition 1, define S_1 to compute, on input 1^k, a pair

$(n,e) \in I \cap \{0,1\}^k$ and corresponding d such that $ed \equiv 1 \bmod \varphi(n)$. The algorithm picks two random primes of equal size by choosing random numbers and testing them for primality and setting n to be their procuct, then $e \in Z_{\phi(n)}$ is chosen at random, and finally d is computed in polynomial time by first computing $\varphi(n) = (p-1)(q-1)$ and then using the extended Euclidean algorithm. For condition 2, define S_2 to randomly generate $x \in \mathbf{Z}_n^*$ on input (n,e). Let $A_1((n,e),x) = \mathrm{RSA}_{n,e}(x)$. Note that exponentiation modulo n is a polynomial time computation and therefore condition 3 holds. Condition 4 follows from the Strong RSA assumption. For condition 5, let $A_2((n,e),d,\mathrm{RSA}_{n,e}(x)) \equiv \mathrm{RSA}_{n,e}(x)^d \equiv x^{ed} \equiv x \bmod n$ and this is a polynomial time computation. ∎

One of the properties of the RSA function is that if we have a polynomial time algorithm that inverts $RSA_{n,e}$ on at least a polynomial proportion of the possible inputs $x \in Z_n^*$ then a subsequent probabilistic expected polynomial time algorithm can be found which inverts $\mathrm{RSA}_{n,e}$ on almost all inputs $x \in Z_n^*$. This can be taken to mean that for a given n,e if the function is hard to invert then it is almost everywhere hard to invert.

Proposition 2.31 Let ϵ, $\delta \in (0,1)$ and let $S \subseteq I$. Suppose there is a probabilistic algorithm A such that for all $(n,e) \in S$

$$\Pr[A(n,e,RSA_{n,e}(x)) = x] > \epsilon$$

(where the probability is taken over $x \in \mathbf{Z}_n^*$ and the coin tosses of A) and A runs in time polynomial in $|n|$. Then there is a probabilistic algorithm A′ running in time polynomial in $\epsilon^{-1}, \delta^{-1}$, and $|n|$ such that for all $(n,e) \in S$, and $x \in \mathbf{Z}_n^*$

$$\Pr[A'(n,e,RSA_{n,e}(x)) = x] > 1-\delta$$

(where the probability is taken over the coin tosses of A′). ∎

Proof: Choose the smallest integer N for which $\frac{1}{e^N} < \delta$.

Consider the algorithm A′ running as follows on inputs $(n,e) \in S$ and $\mathrm{RSA}_{n,e}(x)$.

Repeat $\epsilon^{-1}N$ times.
- Randomly choose $z \in \mathbf{Z}_n^*$.
- Let $y = A(n,e,\mathrm{RSA}_{n,e}(x) \cdot \mathrm{RSA}_{n,e}(z)) = A(n,e,\mathrm{RSA}_{n,e}(xz))$.
- If A succeeds then $y = xz$ and therefore $x = yz^{-1} \bmod n$. Output x.
- Otherwise, continue to the next iteration.

End loop

We can estimate the probability that A′ fails:

$$\begin{aligned}\Pr[A'(n,e,RSA_{n,e}(x)) \neq x] &= \Pr[\text{A single iteration of the loop of A}' \text{ fails}]^{\epsilon^{-1}N} \\ &< (1-\epsilon)^{\epsilon^{-1}N} \\ &< (e^{-N}) \\ &< \delta\end{aligned}$$

Note that since $N = O(\log(\delta^{-1})) = O(\delta^{-1})$, A′ is a probabilistic algorithm which runs in time polynomial in ϵ^{-1}, δ^{-1}, and $|n|$. ∎

Open Problem 2.32 It remains to determine whether a similar result holds if the probability is also taken over the indices $(n,e) \in I$. Specifically, if ϵ, $\delta \in (0,1)$ and A is a PTM such that

$$\Pr[A(n,e,RSA_{n,e}(x)) = x] > \epsilon$$

(where the probability is taken over $(n,e) \in I$, $x \in \mathbf{Z}_n^*$ and the coin tosses of A), does there exist a PTM A' running in time polynomial in ϵ^{-1} and δ^{-1} such that

$$\Pr[A'(n,e,RSA_{n,e}(x)) = x] > 1-\delta$$

(where the probability is taken over $(n,e) \in I$ and the coin tosses of A$'$)?

2.3.3 Connection Between The Factorization Problem And Inverting RSA

Fact 2.33 If some PPT algorithm A can factor n then there exists a PPT A' that can invert $RSA_{\langle n,e\rangle}$. ▮

The proof is obvious as $\phi(n) = (p-1)(q-1)$. The trapdoor information d can be found by using the extended Euclidean algorithm because $d = e^{-1} \bmod \phi(n)$.

Fact 2.34 If there exists a PTM B which on input $\langle n,e\rangle$ finds d such that $ed \equiv 1 \bmod \phi(n)$ then there exists a PTM, B$'$ that can factor n. ▮

Open Problem 2.35 It remains to determine whether inverting RSA and factoring are equivalent. Namely, if there is a PTM C which, on input $\langle n,e\rangle$, can invert $RSA_{\langle n,e\rangle}$, does there exist a PTM C' that can factor n? The answer to this question is unknown. Note that Fact 2.34 does not imply that the answer is yes, as there may be other methods to invert RSA which do not necessarily find d.

2.3.4 The Squaring Trapdoor Function Candidate by Rabin

Rabin in [164] introduced a candidate trapdoor function which we call the squaring function. The squaring function resemble the RSA function except that Rabin was able to actually *prove* that inverting the squaring function is as hard as factoring integers. Thus, inverting the squaring function is a computation which is at least as hard as inverting the RSA function and possibly harder.

Definition 2.36 Let $I = \{n = pq : p \text{ and } q \text{ are distinct odd primes.}\}$. For $n \in$ I, the squaring function $SQUARE_n : \mathbf{Z}_n^* \longrightarrow \mathbf{Z}_n^*$ is defined by $SQUARE_n(x) \equiv x^2 \bmod n$. The trapdoor information of $n = pq \in I$ is $t_n = (p,q)$. We will denote the entire collection of Rabin's functions by RABIN $= \{SQUARE_n : \mathbf{Z}_n^* \longrightarrow \mathbf{Z}_n^*\}_{n\in I}$. ▮

Remark 2.37 Observe that while Rabin's function squares its input, the RSA function uses a varying exponent; namely, e where $\gcd(e,\phi(n)) = 1$. The requirement that $\gcd(e,\phi(n)) = 1$ guarentees that the RSA function is a permutation. On the other hand, Rabin's function is 1 to 4 and thus it does not have a uniquely defined inverse. Specifically, let $n = pq \in I$ and let $a \in \mathbf{Z}_p^*$. As discussed in section C.4, if $a \equiv x^2 \bmod p$ then x and $-x$ are the distinct square roots of a modulo p and if $a \equiv y^2 \bmod q$ then y and $-y$ are the distinct square roots of a modulo q. Then, there are four solutions to the congruence $a \equiv z^2 \bmod n$, constructed as follows. Let $c,d \in \mathbf{Z}_n$ be the Chinese Remainder Theorem coefficients as discussed in Appendix C.4. Then

$$c = \begin{cases} 1 \bmod p \\ 0 \bmod q \end{cases}$$

and

$$d = \begin{cases} 0 \bmod p \\ 1 \bmod q \end{cases}$$

and the four solutions are $cx+dy$, $cx-dy$, $-cx+dy$, and $-cx-dy$. ▮

The main result is that RABIN is a collection of strong one way trapdoor functions and the proof relies on an assumption concerning the difficulty of factoring. We state this assumption now.

Factoring Assumption: Let $H_k = \{pq : p$ and q are prime and $|p| = |q| = k\}$. Then for every polynomial Q and every PTM A, $\exists k_0$ such that $\forall k > k_0$

$$\Pr[A(n) = p : p \mid n \text{ and } p \neq 1, n] < \frac{1}{Q(k)}$$

(where the probability is taken over all $n \in H_k$ and the coin tosses of A).

Our ultimate goal is to prove the following result.

Theorem 2.38 Under the factoring assumption, RABIN is a collection of one way trapdoor functions. ▮

Before proving this, we consider two auxiliary lemmas. Lemma 2.39 constructs a polynomial-time machine A which computes square roots modulo a prime. Lemma 2.42 constructs another polynomial-time machine, SQRT, that inverts Rabin's function using the trapdoor information; specifically, it computes a square root modulo composites given the factorization. SQRT makes calls to A.

Lemma 2.39 Let p be an odd prime and let a be a square modulo p. There exists a probabilistic algorithm A running in expected polynomial time such that $A(p, a) = x$ where $x^2 \equiv a \bmod p$. ▮

Proof: Let p be an odd prime and let a be a quadratic residue in $\mathbf{Z}_p^*$. There are two cases to consider; $p \equiv 1 \bmod 4$ and $p \equiv 3 \bmod 4$.

Case 1 $p \equiv 3 \bmod 4$; that is, $p = 4m + 3$ for some integer m.

$$\text{Since } a \text{ is a square we have } 1 = \mathbf{J}_p(a) \equiv a^{\frac{p-1}{2}} \bmod p \implies a^{2m+1} \equiv 1 \bmod p$$
$$\implies a^{2m+2} \equiv a \bmod p$$

Therefore, a^{m+1} is a square root of a modulo p.

Case 2 $p \equiv 1 \bmod 4$; that is, $p = 4m + 1$ for some integer m.

As in Case 1, we will attempt to find an odd exponent e such that $a^e \equiv 1 \bmod p$.

Again, a is a square and thus $1 = \mathbf{J}_p(a) \equiv a^{\frac{p-1}{2}} \bmod p \implies a^{2m} \equiv 1 \bmod p$.

However, at this point we are not done as in Case 1 because the exponent on a in the above congruence is even. But notice that $a^{2m} \equiv 1 \bmod p \implies a^m \equiv \pm 1 \bmod p$. If $a^m \equiv 1 \bmod p$ with m odd, then we proceed as in Case 1.

This suggests that we write $2m = 2^l r$ where r is an odd integer and compute $a^{2^{l-i}r} \bmod p$ for $i = 1, \ldots, l$ with the intention of reaching the congruence $a^r \equiv 1 \bmod p$ and then proceeding as in Case 1. However, this is not guarenteed as there may exist an integer l' satisfying $0 \leq l' < l$ such that $a^{2^{l'}r} \equiv -1 \bmod p$. If this congruence is encountered, we can recover as follows. Choose a quadratic nonresidue $b \in \mathbf{Z}_p^*$. Then $-1 = \mathbf{J}_p(b) \equiv b^{\frac{p-1}{2}} \bmod p$ and therefore $a^{2^{l'}r} \cdot b^{2^l r} = a^{2^{l'}r} \cdot b^{\frac{p-1}{2}} \equiv 1 \bmod p$. Thus, by multiplying by $b^{2^l r} \equiv -1 \bmod p$, we obtain a new congruence $(a^r b^{2^{l-l'}r})^{2^{l'}} \equiv 1 \bmod p$. We proceed by taking square roots in this congruence. Since $l' < l$, we will, after l steps, arrive at $a^r b^{2s} \equiv 1 \bmod p$ where s is integral. At this point we have $a^{r+1} b^{2s} \equiv a \bmod p \Longrightarrow a^{\frac{r+1}{2}} b^s$ is a square root of $a \bmod p$.

From the above discussion (Cases 1 and 2) we obtain a probabilistic algorithm A for taking square roots. The algorithm A runs as follows on input a, p where $\mathbf{J}_p(a) = 1$.

(1) If $p = 4m + 3$ for some integer m then output a^{m+1} as a square root of $a \bmod p$.

(2) If $p = 4m + 1$ for some integer m then randomly choose $b \in \mathbf{Z}_p^*$ until a value is found satisfying $\mathbf{J}_p(b) = -1$.

  (1) Initialize $i = 2m$ and $j = 0$.

  (2) Repeat until i is odd.

    $i \leftarrow \frac{i}{2}$ and $j \leftarrow \frac{j}{2}$.

    If $a^i b^j = -1$ then $j \leftarrow j + 2m$.

  Output $a^{\frac{i+1}{2}} b^{\frac{j}{2}}$ as a square root of $a \bmod p$.

This algorithm terminates after $O(l)$ iterations because in step 2 (ii) the exponent on a is divided by 2. Note also, that since exactly half of the elements in $\mathbf{Z}_p^*$ are quadratic nonresidues, it is expected that 2 iterations will be required to find an appropriate value for b at the beginning of step 2. Thus, A runs in expected polynomial time and this completes the proof of Lemma 2.39. ∎

Remark 2.40 There is a deterministic algorithm due to René Schoof (see [179]) which computes the square root of a quadratic residue a modulo a prime p in time polynomial in $|p|$ and a (specifically, the algorithm requires $O((a^{\frac{1}{2}+\epsilon} \log p)^9)$ elementary operations for any $\epsilon > 0$). However, it is unknown whether there exists a deterministic algorithm running in time polynomial in $|p|$. ∎

Open Problem 2.41 Does there exist a deterministic algorithm that computes square roots modulo a prime p in time polynomial in $|p|$?

The next result requires knowledge of the Chinese Remainder Theorem. The statement of this theorem as well as a constructive proof is given in Appendix C.4. In addition, a more general form of the Chinese Remainder Theorem is presented there.

Lemma 2.42 Let p and q be primes, $n = pq$ and a a square modulo p. There exists a probabilistic algorithm $SQRT$ running in expected polynomial time such that $SQRT(p, q, n, a) = x$ where $x^2 \equiv a \bmod n$. ∎

Proof: The algorithm SQRT will first make calls to A, the algorithm of Lemma 2.39, to obtain square roots of a modulo each of the primes p and q. It then combines these square roots, using the Chinese Remainder Theorem, to obtain the required square root.

The algorithm SQRT runs as follows.

(1) Let $A(p, a) = x_1$ and $A(q, a) = x_2$.

(2) Use the Chinese Remainder Theorem to find (in polynomial time) $y \in \mathbf{Z}_n$ such that $y \equiv x_1 \bmod p$ and $y \equiv x_2 \bmod q$ and output y.

Algorithm SQRT runs correctly because $y^2 \equiv \begin{cases} x_1^2 \equiv a \bmod p \\ x_2^2 \equiv a \bmod q \end{cases} \Longrightarrow y^2 \equiv a \bmod n$. ∎

On the other hand, if the factors of n are unknown then the computation of square roots modulo n is as hard as factoring n. We prove this result next.

Lemma 2.43 Computing square roots modulo $n \in H_k$ is as hard as factoring n. ▮

Proof: Suppose that I is an algorithm which on input $n \in H_k$ and a a square modulo n outputs y such that $a \equiv y^2 \bmod n$ and consider the following algorithm B which on input n outputs a nontrivial factor of n.

(1) Randomly choose $x \in \mathbf{Z}_n^*$.

(2) Set $y = I(n, x^2 \bmod n)$.

(3) Check if $x \equiv \pm y \bmod n$. If not then $\gcd(x-y, n)$ is a nontrivial divisor of n. Otherwise, repeat from 1.

Algorithm B runs correctly because $x^2 \equiv y^2 \bmod n \Longrightarrow (x+y)(x-y) \equiv 0 \bmod n$ and so $n|[(x+y)(x-y)]$. But $n \not| (x-y)$ because $x \not\equiv y \bmod n$ and $n \not| (x+y)$ because $x \not\equiv -y \bmod n$. Therefore, $\gcd(x-y, n)$ is a nontrivial divisor of n. Note also that the congruence $a \equiv x^2 \bmod n$ has either 0 or 4 solutions (a proof of this result is presented in Appendix C.4). Therefore, if $I(n, x^2) = y$ then $x \equiv \pm y \bmod n$ with probability $\frac{1}{2}$ and hence the above algorithm is expected to terminate in 2 iterations. ▮

We are now in a position to prove the main result, Theorem 2.38.

Proof: For condition 1, define S_1 to find on input 1^k an integer $n = pq$ where p and q are primes of equal length and $|n| = k$. The trapdoor information is the pair of factors (p, q).

For condition 2 in the definition of a collection of one way trapdoor functions, define S_2 to simply output $x \in Z_n^*$ at random given n.

Condition 3 is true since the computation of $x^2 \bmod n$ can be performed in polynomial time and condition 4 follows from the factoring assumption and Lemma 2.43.

Condition 5 follows by applying the algorithm $SQRT$ from Lemma 2.42. ▮

Lemma 2.43 can even be made stronger as we can also prove that if the algorithm I in the proof of Lemma 2.43 works only on a small portion of its inputs then we are still able to factor in polynomial time.

Proposition 2.44 Let ϵ, $\delta \in (0,1)$ and let $S \subseteq H_k$. Suppose there is a probabilistic algorithm I such that for all $n \in S$

$$\Pr[I(n,a) = x \text{ such that } a \equiv x^2 \bmod n] > \epsilon$$

(where the probability is taken over $n \in S$, $a \in {\mathbf{Z}_n^*}^2$, and the coin tosses of I). Then there exists a probabilistic algorithm FACTOR running in time polynomial in ϵ^{-1}, δ^{-1}, and $|n|$ such that for all $n \in S$,

$$\Pr[\text{FACTOR}(n) = d \text{ such that } d \mid n \text{ and } d \neq 1, n] > 1 - \delta$$

(where the probability is taken over n and over the coins tosses of FACTOR). ▮

Proof: Choose the smallest integer N such that $\frac{1}{e^N} < \delta$.

Consider the algorithm FACTOR running as follows on inputs $n \in S$.

Repeat $2\epsilon^{-1}N$ times.
    Randomly choose $x \in \mathbf{Z}_n^*$.
    Set $y = I(n, x^2 \bmod n)$.
    Check if $x \equiv \pm y \bmod n$. If not then $\gcd(x-y, n)$ is a nontrivial divisor of n.
    Otherwise, continue to the next iteration.
End loop

We can estimate the probability that FACTOR fails. Note that even when $I(n, x^2 \bmod n)$ produces a square root of $x^2 \bmod n$, FACTOR(n) will be successful exactly half of the time.

$$\begin{aligned}
\Pr[\text{FACTOR}(n) \text{ fails to factor } n] &= \Pr[\text{A single iteration of the loop of FACTOR fails}]^{\epsilon^{-1}N} \\
&< (1 - \tfrac{1}{2}\epsilon)^{2\epsilon^{-1}N} \\
&< (e^{-N}) \\
&< \delta
\end{aligned}$$

Since $N = O(\log(\delta^{-1})) = O(\delta^{-1})$, FACTOR is a probabilistic algorithm which runs in time polynomial in ϵ^{-1}, δ^{-1}, and $|n|$. ▮

2.3.5 A Squaring Permutation as Hard to Invert as Factoring

We remarked earlier that Rabin's function is not a permutation. If $n = pq$ where p and q are primes and $p \equiv q \equiv 3 \bmod 4$ then we can reduce the Rabin's function $SQUARE_n$ to a permutation g_n by restricting its domain to the quadratic residues in $\mathbf{Z}_n^*$, denoted by Q_n. This will yield a collection of one way permutations as we will see in Theorem 2.3.5. This suggestion is due to Blum and Williams.

Definition 2.45 Let $J = \{pq : p \neq q \text{ are odd primes}, |p| = |q|, \text{ and } p \equiv q \equiv 3 \bmod 4\}$. For $n \in$ J let the function $g_n : Q_n \longrightarrow Q_n$ be defined by $g_n(x) \equiv x^2 \bmod n$ and let BLUM-WILLIAMS $= \{g_n\}_{n \in J}$. ▮

We will first prove the following result.

Lemma 2.46 Each function $g_n \in$ BLUM-WILLIAMS is a permutation. That is, for every element $y \in Q_n$ there is a unique element $x \in Q_n$ such that $x^2 = y \bmod n$. ▮

Proof: Let $n = p_1 p_2 \in$ J. Note that by the Chinese Remainder Theorem, $y \in Q_n$ if and only if $y \in Q_n$ and $y \in Q_{p_1}$ and $y \in Q_{p_2}$. Let a_i and $-a_i$ be the square roots of $y \bmod p_i$ for $i = 1, 2$. Then, as is done in the proof of the Chinese Remainder Theorem, we can construct Chinese Remainder Theorem coefficients c_1, c_2 such that $c_1 = \begin{cases} 1 \bmod p_1 \\ 0 \bmod p_2 \end{cases}$ and $c_2 = \begin{cases} 0 \bmod p_1 \\ 1 \bmod p_2 \end{cases}$ and consequently, the four square roots of $y \bmod n$ are

$$\begin{aligned}
& w_1 = c_1 a_1 + c_2 a_2, \\
& w_2 = c_1 a_1 - c_2 a_2, \\
& w_3 = -c_1 a_1 - c_2 a_2 = -(c_1 a_1 + c_2 a_2) = -w_1, \\
\text{and } & w_4 = -c_1 a_1 + c_2 a_2 = -(c_1 a_1 - c_2 a_2) = -w_2.
\end{aligned}$$

Since $p_1 \equiv p_2 \equiv 3 \bmod 4$, there are integers m_1 and m_2 such that $p_1 = 4m_1 + 3$ and $p_2 = 4m_2 + 3$. Thus, $\mathbf{J}_{p_1}(w_3) = \mathbf{J}_{p_1}(-w_1) = \mathbf{J}_{p_1}(-1)\mathbf{J}_{p_1}(w_1) = (-1)^{\frac{p_1-1}{2}}\mathbf{J}_{p_1}(w_1) = -\mathbf{J}_{p_1}(w_1)$ because $\frac{p_1-1}{2}$ is odd and similarly, $\mathbf{J}_{p_1}(w_4) = -\mathbf{J}_{p_1}(w_2)$, $\mathbf{J}_{p_2}(w_3) = -\mathbf{J}_{p_2}(w_1)$, and $\mathbf{J}_{p_2}(w_4) = -\mathbf{J}_{p_2}(w_2)$. Therefore, without loss of generality, we can assume that $\mathbf{J}_{p_1}(w_1) = \mathbf{J}_{p_1}(w_2) = 1$ (and so $\mathbf{J}_{p_1}(w_3) = \mathbf{J}_{p_1}(w_4) = -1$).

Since only w_1 and w_2 are squares modulo p_1 it remains to show that only one of w_1 and w_2 is a square modulo n or equivalently modulo p_2.

First observe that $\mathbf{J}_{p_2}(w_1) \equiv (w_1)^{\frac{p_2-1}{2}} \equiv (c_1a_1 + c_2a_2)^{2m_2+1} \equiv (a_2)^{2m_2+1} \bmod p_2$ and that $\mathbf{J}_{p_2}(w_2) \equiv (w_2)^{\frac{p_2-1}{2}} \equiv (c_1a_1 - c_2a_2)^{2m_2+1} \equiv (-a_2)^{2m_2+1} \bmod p_2$ (because $c_1 \equiv 0 \bmod p_2$ and $c_2 \equiv 1 \bmod p_2$). Therefore, $\mathbf{J}_{p_2}(w_2) = -\mathbf{J}_{p_2}(w_1)$. Again, without loss of generality, we can assume that $\mathbf{J}_{p_2}(w_1) = 1$ and $\mathbf{J}_{p_2}(w_2) = -1$ and hence, w_1 is the only square root of y that is a square modulo both p_1 and p_2. Therefore, w_1 is the only square root of y in Q_n. ▌

Theorem 2.47 [Williams, Blum] BLUM-Williams is a collection of one-way trapdoor permutations. ▌

Proof: This follows immediately from Lemma 2.46 because each function $g_n \in$ J is a permutation. The trapdoor information of $n = pq$ is $t_n = (p, q)$. ▌

2.4 Hard-core Predicate of a One Way Function

Recall that $f(x)$ does not necessarily hide everything about x even if f is a one-way function. E.g. if f is the RSA function then it preserves the Jacobi symbol of x, and if f is the discrete logarithm function EXP then it is easy to compute the least significant bit of x from $f(x)$ by a simple Legendre symbol calculation. Yet, it seems likely that there is at least one bit about x which is hard to "guess" from $f(x)$, given that x in its entirety is hard to compute. The question is: can we point to specific bits of x which are hard to compute, and how hard to compute are they. The answer is encouraging. A number of results are known which give a particular bit of x which is hard to guess given $f(x)$ for some particular f's such as RSA and the discrete logarithm function. We will survey these results in subsequent sections.

More generally, we call a predicate about x which is impossible to compute from $f(x)$ better than guessing it at random a *hard-core predicate* for f.

We first look at a general result by Goldreich and Levin [94] which gives for any one-way function f a predicate B such that it is as hard to guess $B(x)$ from $f(x)$ as it is to invert f.

Historical Note: The idea of a hard-core predicate for one-way functions was introduced by Blum, Goldwasser and Micali. It first appears in a paper by Blum and Micali [40] on pseduo random number generation. They showed that a if the EXP function ($f_{p,g}(x) = g^x \pmod{p}$) is hard to invert then it is hard to even guess better than guessing at random the most significant bit of x. Under the assumption that quadratic residues are hard to distinguish from quadratic non-residues modulo composite moduli, Goldwasser and Micali in [98] showed that the squaring function has a hard core perdicate as well. Subsequently, Yao [201] showed a general result that given any one way function, there is a predicate $B(x)$ which is as hard to guess from $f(x)$ as to invert f for any function f. Goldreich and Levin's result is a significantly simpler construction than Yao's earlier construction.

2.4.1 Hard Core Predicates for General One-Way Functions

We now introduce the concept of a *hard-core predicate* of a function and show by explicit construction that any strong one way function can be modified to have a hard-core predicate.

Note: Unless otherwise mentioned, the probabilities during this section are calculated uniformly over all coin tosses made by the algorithm in question.

Definition 2.48 A hard-core predicate of a function $f : \{0,1\}^* \to \{0,1\}^*$ is a boolean predicate $B : \{0,1\}^* \to \{0,1\}$, such that

(1) $\exists PPT\, A$, such that $\forall x\, A(x) = B(x)$

(2) $\forall$ PPT G, $\forall$ constants c, $\exists\, k_0$, s.t. $\forall_{k>k_0}$

$$\Pr[G(f(x)) = B(x)] < \frac{1}{2} + \frac{1}{k^c}.$$

The probability is taken over the random coin tosses of G, and random choices of x of length k.

∎

Intuitively, the definition guarantees that given x, $B(x)$ is efficiently computable, but given only $f(x)$, it is hard to even "guess" $B(x)$; that is, to guess $B(x)$ with a probability significantly better than $\frac{1}{2}$.

Yao, in [201], showed that the existence of any trapdoor length-preserving permutation implies the existence of a trapdoor predicate. Goldreich and Levin greatly simplified Yao's construction and show that any one-way function can be modified to have a trapdoor predicate as follows (we state a simple version of their general result).

Theorem 2.49 [94] Let f be a (strong) length preserving one-way function. Define $f'(x \circ r) = f(x) \circ r$, where $|x| = |r| = k$, and $\circ$ is the concatenation function. Then

$$B(x \circ r) = \Sigma_{i=1}^{k} x_i r_i (mod\ 2).$$

is a hard-core predicate for f'. ∎

Note: $v \circ w$ denotes concatenation of strings v and w. Computing B from f' is trivial as $f(x)$ and r are easily recoveravle from $f'(x, r)$. Finaly notice that if f is one-way then so is f'.

For a full proof of the theorem we refer the reader to [94].

It is trivial to extend the definition of a hard-core predicate for a one way function, to a collection of hard core predicates for a collection of one-way functions.

Definition 2.50 A hard-core predicate of a one-way function collection $F = \{f_i : D_i \to R_i\}_{i \in I}$ is a collection of boolean predicates $B = \{B_i : D_i \to R_i\}_{i \in I}$ such that

(1) $\exists PPT\, A$, such that $\forall i, x\, A(i, x) = B_i(x)$

(2) $\forall$ PPT G, $\forall$ constants c, $\exists$, k_0, s.t. $\forall_{k>k_0}$

$$\Pr[G(i, f_i(x)) = B_i(x)] < \frac{1}{2} + \frac{1}{k^c}.$$

The probability is taken over the random coin tosses of G, random choices of $i \in I \cap \{0,1\}^k$ and random $x \in D_i$.

∎

2.4.2 Bit Security Of The Discrete Logarithm Function

Let us examine the bit security of the EXP collection of functions directly rather than through the Goldreich Levin general construction.

We will be interested in the most significant bit of the discrete logarithm x of y modulo p.

For $(p, g) \in I$ and $y \in \mathbf{Z}_p^*$, let $B_{p,g}(y) = \begin{cases} 0 & \text{if } y = g^x \bmod p \\ & \quad \text{where } 0 \le x < \frac{p-1}{2} \\ 1 & \text{if } y = g^x \bmod p \\ & \quad \text{where } \frac{p-1}{2} \le x < p-1 \end{cases}$.

We want to show that if for p a prime and g a generator of $\mathbf{Z}_p^*$, $\mathrm{EXP}_{p,g}(x) \equiv g^x \bmod p$ is hard to invert, then given $y = \mathrm{EXP}_{p,g}(x)$, $B_{p,g}(y)$ is hard to compute in a very strong sense; that is, in attempting to compute $B_{p,g}(y)$ we can do no better than essentially guessing its value randomly. The proof will be by way of a reduction. It will show that if we can compute $B_{p,g}(y)$ in polynomial time with probability greater than $\frac{1}{2} + \epsilon$ for some non-negligible $\epsilon > 0$ then we can invert $\mathrm{EXP}_{p,g}(x)$ in time polynomial in $|p|$, $|g|$, and ϵ^{-1}. The following is a formal statement of this fact.

Theorem 2.51 Let S be a subset of the prime integers. Suppose there is a polynomial Q and a PTM G such that for all primes $p \in S$ and for all generators g of $\mathbf{Z}_p^*$

$$\Pr[G(p,g,y) = B_{p,g}(y)] > \frac{1}{2} + \frac{1}{Q(|p|)}$$

(where the probability is taken over $y \in \mathbf{Z}_p^*$ and the coin tosses of G). Then for every polynomial P, there is a PTM I such that for all primes $p \in S$, generators g of $\mathbf{Z}_p^*$, and $y \in \mathbf{Z}_p^*$

$$\Pr[I(p,g,y) = x \text{ such that } y \equiv g^x \bmod p] > 1 - \frac{1}{P(|p|)}$$

(where the probability is taken over the coin tosses of I). ▮

We point to [40] for a proof of the above theorem.

As a corollary we immediately get the following.

Definition 2.52 Define $MSB_{p,g}(x) = 0$ if $1 \le x < \frac{p-1}{2}$ and 1 otherwise for $x \in Z_{p-1}$, and $MSB = \{MSB_{p,g}(x) : Z_{p-1} \to \{0,1\}\}_{(p,g)\in I}$. for $I = \{(p,g) : p$ is prime and g is a generator of $\mathbf{Z}_p^*\}$. ▮

Corollary 2.53 Under the strong DLA, MSB is a collection of hard-core predicates for EXP. ▮

It can be shown that actually $O(\log\log p)$ of the most significant bits of $x \in \mathbf{Z}_{p-1}$ are hidden by the function $EXP_{p,g}(x)$. We state this result here without proof.

Theorem 2.54 For a PTM A, let

$$\alpha = Pr[A(p,g,g^x, x_{\log\log p}x_{\log\log p-1}\ldots x_0) = 0 \mid x = x_{|p|}\ldots x_0]$$

(where the probability is taken over $x \in \mathbf{Z}_n^*$ and the coin tosses of A) and let

$$\beta = Pr[A(p,g,g^x, r_{\log\log p}r_{\log\log p-1}\ldots r_0) = 0 \mid r_i \in_R \{0,1\}]$$

(where the probability is taken over $x \in \mathbf{Z}_n^*$, the coin tosses of A, and the bits r_i). Then under the Discrete Logarithm Assumption, we have that for every polynomial Q and every PTM A, $\exists k_0$ such that $\forall k > k_0$, $|\alpha - \beta| < \frac{1}{Q(k)}$. ▮

Corollary 2.55 Under the Discrete Logarithm Assumption we have that for every polynomial Q and every PTM A, $\exists k_0$ such that $\forall k > k_0$ and $\forall k_p < \log\log p$

$$\Pr[A(p,g,g^x, x_{k_p}\ldots x_0) = x_{k_p+1}] < \frac{1}{2} + \frac{1}{Q(k)}$$

(where the probability is taken over the primes p such that $|p| = k$, the generators g of $\mathbf{Z}_p^*$, $x \in \mathbf{Z}_p^*$, and the coin tosses of A). ▮

For further information on the simultaneous or individual security of the bits associated with the discrete logarithm see [131, 108].

2.4.3 Bit Security of RSA and SQUARING functions

Let $I = \{< n, e > — n = pq\ |p| = |q|, (e, \phi(n)) = 1\}$, and $RSA = \{RSA_{<n,e>} : Z_n^* \rightarrow Z_n^*\}_{<n,e>\in I}$ be the collection of functions as defined in 2.17.

Alexi, Chor, Goldreich and Schnoor [6] showed that guessing the least significant bit of x from $RSA_{<n,e>}(x)$ better than at random is as hard as inverting RSA.

Theorem 2.56 [6] Let $S \subset I$. Let $c > 0$. If there exists a probabilistic polynomial-time algorithm O such that for $(n, e) \in S$,

$$prob(O(n, e, x^e \bmod n) = \text{ least significant bit of } x \bmod n) \geq \frac{1}{2} + \frac{1}{k^c}$$

(taken over coin tosses of O and random choices of $x \in Z_n^*$) Then there exists a probabilistic expected polynomial time algorithm A such that for all $n, e \in S$, for all $x \in Z_n^*$, $A(n, e, x^e \bmod n) = x \bmod n$. ∎

Now define $LSB = \{LSB_{<n,e>} : Z_n^* \rightarrow Z_n^*\}_{<n,e>\in I}$ where $LSB_{<n,e>}(x)$ =least significant bit of x.

A direct corollary to the above theorem is.

Corollary 2.57 Under the (strong) RSA assumption, LSB is a collection of hard core predicates for RSA. ∎

A similar result can be shown for the most signifant bit of x and in fact for the $\log\log n$ least (and most) significant bits of x simultaneously. Moreover, similar results can be shown for the RABIN and BLUM-WILLIAMS collections. We refer to [6], [199] for the detailed results and proofs. Also see [80] for reductions of improved security.

2.5 One-Way and Trapdoor Predicates

A *one-way predicate*, first introduced in [97, 98] is a notion which preceeds hard core predicates for one-way functions and is strongly related to it. It will be very useful for both design of secure encryption and protocol design.

A *one-way predicate* is a boolean function $B : \{0, 1\}^* \rightarrow \{0, 1\}$ for which

(1) *Sampling is possible*: There exists a PPT algorithm that on input $v \in \{0, 1\}$ and 1^k, selects a random x such that $B(x) = v$ and $|x| \leq k$.

(2) *Guessing is hard*: For all $c > 0$, for all k sufficiently large, no PPT algorithm given $x \in \{0, 1\}^k$ can compute $B(x)$ with probability greater than $\frac{1}{2} + \frac{1}{k^c}$. (The probability is taken over the random choices made by the adversary and x such that $|x| \leq k$.)

A *trapdoor predicate* is a one-way predicate for which there exists, for every k, trapdoor information t_k whose size is bounded by a polynomial in k and whose knowledge enables the polynomial-time computation of $B(x)$, for all x such that $|x| \leq k$.

Restating as a collection of one-way and trapdoor predicates is easy.

Definition 2.58 Let I be a set of indices and for $i \in I$ let D_i be finite. A collection of one-way predicates is a set $B = \{B_i : D_i \rightarrow \{0, 1\}\}_{i\in I}$ satisfying the following conditions. Let $D_i^v = \{x \in D_i, B_i(x) = v$.

(1) There exists a polynomial p and a PTM S_1 which on input 1^k finds $i \in I \cap \{0, 1\}^k$.

(2) There exists a PTM S_2 which on input $i \in I$ and $v \in \{0, 1\}$ finds $x \in D_i$ such that $B_i(x) = v$.

(3) For every PPT A there exists a negligble ν_A such that $\forall$ k large enough

$$\mathbf{P}\left[z = v \ : \ i \stackrel{R}{\leftarrow} I \cap \{0,1\}^k \ ; \ v \stackrel{R}{\leftarrow} \{0,1\} \ ; \ x \stackrel{R}{\leftarrow} D_i^v \ ; \ z \stackrel{R}{\leftarrow} A(i,x) \right] \le \frac{1}{2} + \nu_A(k)$$

∎

Definition 2.59 Let I be a set of indices and for $i \in I$ let D_i be finite. A collection of trapdoor predicates is a set $B = \{B_i : D_i \to \{0,1\}\}_{i \in I}$ satisfying the following conditions. Let $D_i^v = \{x \in D_i, B_i(x) = v$.

(1) There exists a polynomial p and a PTM S_1 which on input 1^k finds pairs (i, t_i) where $i \in I \cap \{0,1\}^k$ and $|t_i| < p(k)$ The information t_i is referred to as the trapdoor of i.

(2) There exists a PTM S_2 which on input $i \in I$ and $v \in \{0,1\}$ finds $x \in D_i$ such that $B_i(x) = v$.

(3) There exists a PTM A_1 such that for $i \in I$ and trapdoor t_i, $x \in D_i$ $A_1(i, t_i, x) = B_i(x)$.

(4) For every PPT A there exists a negligble ν_A such that $\forall$ k large enough

$$\mathbf{P}\left[z = v \ : \ i \stackrel{R}{\leftarrow} I \cap \{0,1\}^k \ ; \ v \stackrel{R}{\leftarrow} \{0,1\} \ ; \ x \stackrel{R}{\leftarrow} D_i^v \ ; \ z \leftarrow A(i,x) \right] \le \frac{1}{2} + \nu_A(k)$$

∎

Note that this definition implies that D_i^0 is roughly the same size as D_i^1.

2.5.1 Examples of Sets of Trapdoor Predicates

A Set of Trapdoor Predicates Based on the Quadratic Residue Assumption

Let Q_n denote the set of all *quadratic residues* (or *squares*) modulo n; that is, $x \in Q_n$ iff there exists a y such that $x \equiv y^2 \bmod n$.

Recall that the Jacobi symbol $(\mathbf{J}_n(x))$ is defined for any $x \in Z_n^*$ and has a value in $\{-1, 1\}$; this value is easily computed by using the law of quadratic reciprocity, even if the factorization of n is unknown. If n is prime then $x \in Q_n \Leftrightarrow (\mathbf{J}_n(x)) = 1$; and if n is composite, $x \in Q_n \Rightarrow (\mathbf{J}_n(x)) = 1$. We let J_n^{+1} denote the set $\{x \mid x \in Z_n^* \wedge (\mathbf{J}_n(x)) = 1\}$, and we let $\tilde{Q}_n$ denote the set of *pseudo-squares* modulo n: those elements of J_n^{+1} which do *not* belong to Q_n. If n is the product of two primes then $|Q_n| = |\tilde{Q}_n|$, and for any pseudo-square y the function $f_y(x) = y \cdot x$ maps Q_n one-to-one onto $\tilde{Q}_n$.

The *quadratic residuousity problem* is: given a composite n and $x \in J_n^{+1}$, to determine whether x is a square or a pseudo-square modulo n. This problem is believed to be computationally difficult, and is the basis for a number of cryptosystems.

The following theorem informally shows for every n, if the quadratic residusosity is hard to compute at all then it is hard to distinguish between squares and non-squares for almost everywhere.

Theorem 2.60 [97, 98]: Let $S \subset \{ns.t.n = pq, p, q, \text{ primes}\}$ If there exists a probabilistic polynomial-time algorithm O such that for $n \in S$,

$$\textit{prob}(O(n,x) \text{ decides correctly whether } x \in J_n^{+1}) > \frac{1}{2} + \epsilon \ , \tag{2.1}$$

where this probability is taken over the choice of $x \in J_n^{+1}$ and O's random choices, then there exists a probabilistic algorithm B with running time polynomial in $\epsilon^{-1}, \delta^{-1}$ and $|n|$ such that for all $n \in S$, for all $x \in J_n^{+1}$,

$$\textit{prob}(B(x,n) \text{ decides correctly whether } x \in Q_n | x \in J_n^{+1}) > 1 - \delta \ , \tag{2.2}$$

where this probability is taken over the random coin tosses of B. ∎

Namely, a probabilistic polynomial-time bounded adversary can not do better (except by a smaller than any polynomial advantage) than guess at random whether $x \in J_n$ is a square mod n, if quadratic residuosity problem is not in polynomial time.

This suggests immediately the following set of predicates: Let

$$QR_{n,z}(x) = \left\{ \begin{array}{l} 0 \text{ if } x \text{ is a square mod } n \\ 1 \text{ if } x \text{ is a non-square mod } n \end{array} \right\}$$

where $QR_{n,z} : J_n^{+1} \rightarrow \{0,1\}$ and $I_k = \{n\#z \mid n = pq,\ |p| = |q| = \frac{k}{2},\ p$ and q primes, $(\mathbf{J}_n(z)) = +1$, z a non-square mod $n\}$. It is clear that $QR = \{QR_{n,z}\}$ is a set of trapdoor predicates where the trapdoor information associated with every index $< n, z >$ is the factorization $< p, q >$. Lets check this explicitly.

(1) To select pairs (i, t_i) at random, first pick two primes, p and q, of size $\left|\frac{k}{2}\right|$ at random, determining n. Next, search until we find a non-square z in Z_n^* with Jacobi symbol $+1$. The pair we have found is then $(< n, z >, < p, q >)$. We already know how to do all of these operations in expected polynomial time .

(2) Follows from the existence of the following algorithm to select elements out of $D_{n,z}^v$:

- To select $x \in D_{n,z}^0$, let $x = y^2 \bmod n$ where y is an element of Z_n^* chosen at random.
- To select $x \in D_{n,z}^1$, let $x = zy^2 \bmod n$ where y is an element of Z_n^* chosen at random.

(3) To compute $QR_{n,z}(x)$ given $< p, q >$, we compute $(\mathbf{J}_p(x))$ and $(\frac{x}{q})$. If both are $+1$ then $QR_{n,z}(x)$ is 0, otherwise it is 1.

(4) This follows from the Quadaratic Residuosity Assumption and the above theorem.

A Set of Trapdoor Predicates Based on the RSA Assumption

Define $B_{n,e}(x) =$ the least significant bit of $x^d \bmod n$ for $x \in Z_n^*$ where $ed = 1 \bmod \phi(n)$. Then, to select uniformly an $x \in Z_n^*$ such that $B_{n,e}(x) = v$ simply select a $y \in Z_n^*$ whose least significant bit is v and set $x = y^e \bmod n$. Given d it is easy to compute $B_{n,e}(x) =$ least significant bit of $x^d \bmod n$.

The security of this construction follows trivially from the definition of collection of hard core predicates for the RSA collection of functions.

Chapter 3

Pseudo-random bit generators

In this chapter, we discuss the notion of pseudo-random generators. Intuitively, a $PSRG$ is a deterministic program used to generate *long* sequence of bits which *looks like random sequence*, given as input a *short random sequence* (the input seed).
The notion of $PSRG$ has applications in various fields:

Cryptography:
: In the case of private key encryption, Shannon showed (see lecture 1) that the length of the clear text should not exceed the length of the secret key, that is, the two parties have to agree on a very long string to be used as the secret key. Using a $PSRG$ G, they need to agree only on a short seed r, and exchange the message $G(r) \bigoplus m$.

Algorithms Design:
: An algorithm that uses a source of random bits, can manage with a shorter string, used as a seed to a $PSRG$.

Complexity Theory:
: Given a probabilistic algorithm, an important question is whether we can make it deterministic. Using the notion of a $PSRG$ we can prove, assuming the existence of one-way function that $BPP \subseteq \cap_\epsilon DTIME(2^{n^\epsilon})$

In this chapter we will define good pseudo random number generators and give constructions of them under the assumption that one way functions exist.

We first ask where do can we actually find **truly** random bit sequences. [1]

3.0.2 Generating Truly Random bit Sequences

Generating a one-time pad (or, for that matter, any cryptographic key) requires the use of a "natural" source of random bits, such as a coin, a radioactive source or a noise diode. Such sources are absolutely essential for providing the initial secret keys for cryptographic systems.

However, many natural sources of random bits may be defective in that they produce *biased* output bits (so that the probability of a one is different than the probability of a zero), or bits which are *correlated* with

[1] some of the preliminary discussion in the following three subsections is taken from Rivest's survey article on cryptography which appears in the handbook of computer science

each other. Fortunately, one can remedy these defects by suitably processing the output sequences produced by the natural sources.

To turn a source which supplies biased but uncorrelated bits into one which supplies unbiased uncorrelated bits, von Neumann proposed grouping the bits into pairs, and then turning 01 pairs into 0's, 10 pairs into 1's, and discarding pairs of the form 00 and 11 [200]. The result is an unbiased uncorrelated source, since the 01 and 10 pairs will have an identical probability of occurring. Elias [75] generalizes this idea to achieve an output rate near the source entropy.

Handling a correlated bit source is more difficult. Blum [38] shows how to produce unbiased uncorrelated bits from a biased correlated source which produces output bits according to a known finite Markov chain.

For a source whose correlation is more complicated, Santha and Vazirani [176] propose modeling it as a *slightly random source*, where each output bit is produced by a coin flip, but where an adversary is allowed to choose *which* coin will be flipped, from among all coins whose probability of yielding "Heads" is between δ and $1-\delta$. (Here δ is a small fixed positive quantity.) This is an extremely pessimistic view of the possible correlation; nonetheless U. Vazirani [197] shows that if one has *two, independent*, slightly-random sources X and Y then one can produce "almost independent" ϵ-biased bits by breaking the outputs of X and Y into blocks $\mathbf{x}, \mathbf{y}$ of length $k = \Omega(1/\delta^2 \log(1/\delta)\log(1/\epsilon))$ bits each, and for each pair of blocks $\mathbf{x}, \mathbf{y}$ producing as output the bit $\mathbf{x} \cdot \mathbf{y}$ (the inner product of $\mathbf{x}$ and $\mathbf{y}$ over $GF(2)$). This is a rather practical and elegant solution. Chor and Goldreich [54] generalize these results, showing how to produce independent ϵ-biased bits from even worse sources, where some output bits can even be completely determined.

These results provide effective means for generating truly random sequences of bits—an essential requirement for cryptography—from somewhat defective natural sources of random bits.

3.0.3 Generating Pseudo-Random Bit or Number Sequences

The one-time pad is generally impractical because of the large amount of key that must be stored. In practice, one prefers to store only a short random key, from which a long pad can be produced with a suitable cryptographic operator. Such an operator, which can take a short *random* sequence x and deterministically "expand" it into a *pseudo-random* sequence y, is called a *pseudo-random sequence generator*. Usually x is called the *seed* for the generator. The sequence y is called "pseudo-random" rather than random since not all sequences y are possible outputs; the number of possible y's is at most the number of possible seeds. Nonetheless, the intent is that for all practical purposes y should be indistinguishable from a truly random sequence of the same length.

It is important to note that the use of pseudo-random sequence generator reduces *but does not eliminate* the need for a natural source of random bits; the pseudo-random sequence generator is a "randomness expander", but it must be given a truly random seed to begin with.

To achieve a satisfactory level of cryptographic security when used in a one-time pad scheme, the output of the pseudo-random sequence generator must have the property that an adversary who has seen a portion of the generator's output y must remain unable to efficiently predict other unseen bits of y. For example, note that an adversary who knows the ciphertext C can guess a portion of y by correctly guessing the corresponding portion of the message M, such as a standardized closing "Sincerely yours,". We would not like him thereby to be able to efficiently read other portions of M, which he could do if he could efficiently predict other bits of y. Most importantly, the adversary should not be able to efficiently infer the seed x from the knowledge of some bits of y.

How can one construct secure pseudo-random sequence generators?

Classical Pseudo-random Generators are Unsuitable

Classical techniques for pseudo-random number generation [120, Chapter 3] which are quite useful and effective for Monte Carlo simulations are typically unsuitable for cryptographic applications. For example, *linear* feedback shift registers [104] are well-known to be cryptographically insecure; one can solve for the feedback pattern given a small number of output bits.

Linear congruential random number generators are also insecure. These generators use the recurrence

$$X_{i+1} = aX_i + b \pmod{m} \tag{3.1}$$

to generate an output sequence $\{X_0, X_1, \ldots\}$ from secret parameters a, b, and m, and starting point X_0. It is possible to infer the secret parameters given just a few of the X_i [157]. Even if only a fraction of the bits of each X_i are revealed, but a, b, and m are known, Frieze, Håstad, Kannan, Lagarias, and Shamir show how to determine the seed X_0 (and thus the entire sequence) using the marvelous *lattice basis reduction* (or "L^3") algorithm of Lenstra, Lenstra, and Lovász [83, 127].

As a final example of the cryptographic unsuitability of classical methods, Kannan, Lenstra, and Lovasz [118] use the L^3 algorithm to show that the binary expansion of any algebraic number y (such as $\sqrt{5} = 10.001111000110111\ldots$) is insecure, since an adversary can identify y exactly from a sufficient number of bits, and then extrapolate y's expansion.

3.0.4 Provably Secure Pseudo-Random Generators: Brief overview

This section provides a brief overview of the history of the modern history of pseudo random bit generators. Subsequent section define these concepts formally and give constructions.

The first pseudo-random sequence generator proposed for which one can prove that it is impossible to predict the next number in the sequence from the previous numbers assuming that it is infeasible to invert the RSA function is due to Shamir [183]. However, this scheme generates a sequence of *numbers* rather than a sequence of *bits*, and the security proof shows that an adversary is unable to predict the next *number*, given the previous numbers output. This is not strong enough to prove that, when used in a one-time pad scheme, each *bit* of the message will be well-protected.

Blum and Micali [40] introduced the first method for designing provably secure pseudo-random *bit* sequence generators, based on the use of one-way predicates. They call a pseudo-random bit generator secure if an adversary cannot guess the next bit in the sequence from the prefix of the sequence, better than guessing at random. Blum and Micali then proposed a particular generator based on the difficulty of computing discrete logarithms. Blum, Blum, and Shub [35] propose another generator, called the *squaring generator*, which is simpler to implement and is provably secure assuming that the quadratic residuosity problem is hard. Alexi, Chor, Goldreich, and Schnorr [6] show that the assumption that the quadratic residuosity problem is hard can be replaced by the weaker assumption that factoring is hard. A related generator is obtained by using the RSA function. Kaliski shows how to extend these methods so that the security of the generator depends on the difficulty of computing elliptic logarithms; his techniques also generalize to other groups [116, 117]. Yao [201] shows that the pseudo-random generators defined above are *perfect* in the sense that no probabilistic polynomial-time algorithm can guess with probability greater by a non-negligible amount than 1/2 whether an input string of length k was randomly selected from $\{0,1\}^k$ or whether it was produced by one of the above generators. One can rephrase this to say that a generator that passes the next-bit test is perfect in the sense that it will *pass all polynomial-time statistical tests*. The Blum-Micali and Blum-Blum-Shub generators, together with the proof of Yao, represent a major achievement in the development of provably secure cryptosystems. Impagliazzo, Luby, Levin and Håstad show that actually the existence of a one-way function is equivalent to the existence of a pseudo random bit generator which passes all polynomial time statistical tests.

3.1 Definitions

Definition 3.1 Let X_n, Y_n be probability distributions on $\{0,1\}^n$ (That is, by $t \in X_n$ we mean that $t \in \{0,1\}^n$ and it is selected according to the distribution X_n). We say that $\{X_n\}$ is **poly-time indistin-**

guishable from $\{Y_n\}$ if $\forall PTM\ A$, $\forall$ polynomial Q, $\exists n_0$, s.t $\forall n > n_0$,

$$|\Pr_{t \in X_n}(A(t) = 1) - \Pr_{t \in Y_n}(A(t) = 1)| < \frac{1}{Q(n)}$$

∎

i.e., for sufficiently long strings, no PTM can tell whether the string was sampled according to X_n or according to Y_n.
Intuitively, Pseudo random distribution would be a indistinguishable from the uniform distribution. We denote the uniform probability distribution on $\{0,1\}^n$ by U_n. That is, for every $\alpha \in \{0,1\}^n$, $\Pr_{x \in U_n}[x = \alpha] = \frac{1}{2^n}$.

Definition 3.2 We say that $\{X_n\}$ is **pseudo random** if it is poly-time indistinguishable from $\{U_n\}$. That is, $\forall PTM\ A$, $\forall$ polynomial Q, $\exists n_0$, such that $\forall n > n_0$,

$$|\Pr_{t \in X_n}[A(t) = 1] - \Pr_{t \in U_n}[A(t) = 1]| < \frac{1}{Q(n)}$$

∎

Comments:

The algorithm A used in the above definition is called a polynomial time *statistical test.* (Knuth, vol. 2, suggests various examples of statistical tests). It is important to note that such a definition cannot make sense for a single string, as it can be drawn from either distribution.

If $\exists A$ and Q such that the condition in definition 2 is violated, we say that X_n fails the test A.

Definition 3.3 A polynomial time deterministic program $G : \{0,1\}^k \to \{0,1\}^{\hat{k}}$ is a pseudo-random generator (PSRG) if the following conditions are satisfied.

1. $\hat{k} > k$
2. $\{G_{\hat{k}}\}_{\hat{k}}$ is pseudo-random where $G_{\hat{k}}$ is the distribution on $\{0,1\}^{\hat{k}}$ obtained as follows: to get $t \in G_{\hat{k}}$:

   pick $x \in U_k$
   set $t = G(x)$

That is, $\forall PTMA, \forall$ polynomial $Q, \forall$ sufficiently large k,

$$|\Pr_{t \in G_{\hat{k}}}(A(t) = 1) - \Pr_{t \in U_{\hat{k}}}(A(t) = 1)| < \frac{1}{Q(\hat{k})} \tag{3.2}$$

∎

3.2 The Existence Of A Pseudo-Random Generator

Next we prove the existence of $PSRG$'s, if length-preserving one way permutations exist. It has been shown that if one-way functions exist (without requiring them to be length-preserving permutations) then one-way functions exist, but we will not show this here.

Theorem 3.4 Let $f : \{0,1\}^* \to \{0,1\}^*$ be a length preserving one-way permutation. Then

1. $\exists\, PSRG \quad G : \{0,1\}^k \to \{0,1\}^{k+1}$ (such G is called an extender).
2. $\forall$ polynomial Q, $\exists\, PSRG \quad G^Q : \{0,1\}^k \to \{0,1\}^{Q(k)}$.

▮

Proof:
Proof of 1*:* Let f be as above. Let B be the *hard core bit* of f. (that is, B is a boolean predicate, $B : \{0,1\}^* \to \{0,1\}$, s.t it is efficient to compute $B(x)$ given x, but given only $f(x)$, it is hard to compute $B(x)$ with probability greater than $\frac{1}{2}+\epsilon$ for non-negligible ϵ.) Recall that we showed last class that every OWF $f(x)$ can be converted into $f_1(x,r)$ for which $B'(x,r) = \sum_1^{|x|} x_i r_i \bmod 2$ is a hard core bit. For notational ease, assume that B is already a hard-core bit for f.

Define
$$G^1(x) = f(x) \circ B(x)$$
($\circ$ denotes the string concatenation operation). We will prove that $G^1(x)$ has the required properties. Clearly, G^1 is computed in poly-time, and for $|x| = k$, $|G^1(x)| = k+1$. It remains to show that the distribution $\{G^1_{k+1}\}$ is pseudo random.

Intuition : Indeed, knowing $f(x)$ should not help us to predict $B(x)$. As f is a permutation, $f(U_k)$ is uniform on $\{0,1\}^k$, and any separation between U_{k+1} and G_{k+1} must caused by the hard core bit. We would like to show that any such separation would enable us to predict $B(x)$ given only $f(x)$ and obtain a contradiction to B being a hard core bit for f.

We proceed by contradiction: Assume that (G is not good) $\exists$ statistical test A, polynomial Q s.t
$$\Pr_{t \in G_{k+1}}(A(t)=1) - \Pr_{t \in U_{k+1}}(A(t)=1) > \frac{1}{Q(k+1)}$$

(Note that we have dropped the absolute value from the inequality 3.2. This can be done wlog. We will later see what would change if the the other direction of the inequality were true).

Intuitively, we may thus interpret that if A answers 1 on a string t it is more likely that t is drawn from distribution G_{k+1}, and if A answers 0 on string t that it is more likely that t is drawn from distribution U_{k+1}.

We note that the probability that $A(f(x) \circ b)$ returns 1, is the sum of the weighted probability that A returns 1 conditioned on the case that $B(x) = b$ and conditioned on the case $B(x) = 1$. By, the assumed separation above, we get that it is more likely that $A(f(x) \circ b)$ will return 1 when $b = B(x)$. This easily translates to an algorithm for predicting the hard core bit of $f(x)$.

Formally, we have

$$\begin{aligned}
\Pr_{x \in U_k, b \in U_1}[A(f(x) \circ b) = 1] &= \Pr[A(f(x) \circ b) = 1 \mid b = B(x)] \cdot \Pr[b = B(x)] \\
&\qquad + \Pr[A(f(x) \circ b) = 1 \mid b = \overline{B(x)}] \cdot \Pr[b = \overline{B(x)}] \\
&= \tfrac{1}{2}(\alpha + \beta)
\end{aligned}$$

where $\alpha = \Pr[A(f(x) \circ b) = 1 \mid b = B(x)]$ and $\beta = \Pr[A(f(x) \circ b) = 1 \mid b = \overline{B(x)}]$.

From the assumption we therefore get

$$\Pr_{x \in U_k}[A(f(x) \circ B(x)) = 1] - \Pr_{x \in U_k}[A(f(x) \circ b) = 1] = \alpha - \tfrac{1}{2}(\alpha + \beta)$$

$$= \tfrac{1}{2}(\alpha - \beta)$$
$$> \tfrac{1}{Q(k)}.$$

We now exhibit a polynomial time algorithm A' that on input $f(x)$ computes $B(x)$ with success probability significantly better than $1/2$.
A' takes as input $f(x)$ and outputs either a 0 or 1.

1. choose $b \in \{0,1\}$
2. run $A(f(x) \circ b)$
3. If $A(f(x) \circ b) = 1$, output b, otherwise output $\overline{b}$.

Notice that, when dropping the absolute value from the inequality 3.2, if we take the second direction we just need to replace b by $\overline{b}$ in the definition of A'.

Claim 3.5 $\Pr[A'(f(x) = B(x))] > \frac{1}{2} + \frac{1}{Q(k)}$. ∎

Proof:

$$\begin{aligned}
\Pr[A'(f(x) = b)] \ & \Pr[A(f(x) \circ b) = 1 \mid b = B(x)] \Pr[b = B(x))\\
& \Pr[A(f(x) \circ b) = 0 | b = \overline{B(x)}) \Pr[b = \overline{B(x)}]\\
& \qquad + \tfrac{1}{2}\alpha + \tfrac{1}{2}(1-\beta)\\
& = \tfrac{1}{2} + \tfrac{1}{2}(\alpha - \beta)\\
& > \tfrac{1}{2} + \tfrac{1}{Q(k)}.
\end{aligned}$$

∎

This contradicts the hardness of computing $B(x)$. It follows that G^1 is indeed a $PSRG$.
Proof of 2: Given a $PSRG$ G that expands random strings of length k to pseudo-random strings of length $k+1$, we need to show that, $\forall$ polynomial Q, $\exists\, PSRG\, G^Q : \{0,1\} \to \{0,1\}^{Q(k)}$. We define G^Q by first using G $Q(k)$ times as follows:

$$\begin{array}{lcl}
x & \to G \to & f(x) \circ B(x)\\
f(x) \circ B(x) & \to G \to & f(f(x)) \circ B(f(x))\\
f^2(x) \circ B(f(x)) & \to G \to & f^3(x) \circ B(f^2(x))\\
 & \bullet & \\
 & \bullet & \\
 & \bullet & \\
f^{Q(k)-2}(x) \circ B(f^{Q(k)-1}(x)) & \to G \to & f^{Q(k)}(x) \circ B(f^{Q(k)-1}(x))
\end{array}$$

The output of $G^Q(x)$ is the concatenation of the last bit from each string *i.e.*,

$$G^Q(x) = B(x) \circ B(f(x)) \circ \cdots \circ B(f^{Q(k)-1}(x) =$$
$$b_1^G(x) \circ b_2^G(x) \circ \cdots \circ b_{Q(|x|)}^G(x)$$

Clearly, G^Q is poly-time and it satisfies the length requirements. We need to prove that the distribution generated by G^Q, $G^Q(U_k)$, is poly-time indistinguishable from $U_{Q(k)}$. We proceed by contradiction, (and

show that it implies that G is not $PSRG$)
If G_{Qk} is not poly-time indistinguishable from $U_{Q(k)}$, $\exists$ statistical test A, and $\exists$ polynomial P, s.t.

$$\Pr_{t \in G_{Q(k)}}(A(t)=1) - \Pr_{t \in U_{Q(k)}}(A(t)=1) > \frac{1}{P(k)}$$

(As before we omit the absolute value). We now define a sequence $D_1, D_2, ..., D_{Q(k)}$ of distributions on $\{0,1\}^{Q(k)}$, s.t. D_1 is uniform (i.e. strings are random), $D_{Q(k)} = G_{Q(k)}$, and the intermediate D_i's are distributions composed of concatenation of random followed by pseudorandom distributions. Specifically,

$t \in D_1$ is obtained by letting	$t = s$	where $s \in U_{Q(k)}$
$t \in D_2$ is ontained by letting	$t = s \circ B(x)$	where $s \in U_{Q(k)-1}, x \in U_k$
$t \in D_3$ is ontained by letting	$t = s \circ B(x) \circ B(f(x))$	where $s \in U_{Q(k)-2}, x \in U_k$
	•	
	•	
	•	
$t \in D_{Q(k)}$ is ontained by letting	$t = B(x) ... \circ B(f^{Q(k)-1}(x))$	where $x \in U_k$

Since the sequence 'moves' from $D_1 = U_{Q(k)}$ to $D_{Q(k)} = G_{Q(k)}$, and we have an algorithms A that distinguishes between them, there must be two successive distributions between which A distinguishes.
i.e. $\exists\ i$, $1 \le i \le Q(k)$, s.t.

$$\Pr_{t \in D_i}(A(t)=1) - \Pr_{t \in D_{i+1}}(A(t)=1) > \frac{1}{P(k)Q(k)}$$

We now present a poly-time algorithm A' that distinguishes between U_{k+1} and G_{k+1}, with success probability significantly better than $\frac{1}{2}$, contradicting the fact that G is a $PSRG$.

A' works as follows on input $\alpha = \alpha_1 \alpha_2 ... \alpha_{k+1} = \alpha' \circ b$

1. Choose $1 \le i \le q(k)$ at random.

2. Let
$$t = \gamma_1 \circ ... \circ \gamma_{Q(k)-i-1} \circ b \circ b_1^G(\alpha') \circ b_2^G(\alpha') \circ ... \circ b_i^G(\alpha')$$
where the γ_j are chosen randomly.

   (Note that $t \in D_i$ if $\alpha' \circ b \in U_{k+1}$, and that $t \in D_{i+1}$ if $\alpha' \circ b \in G_{k+1}$.)

3. We now run $A(t)$. If we get 1, A' returns 0 If we get 0, A' returns 1

   (i.e if A returns 1, it is interpreted as a vote for D_i and therefore for $b \neq B(\alpha')$ and $\alpha \in U_{k+1}$. On the other hand, if A returns 0, it is interpreted as a vote for D_{i+1} and therefore for $b = B(\alpha')$ and $\alpha \in G_{k+1}$.)

It is immediate that:

$$\Pr_{\alpha \in U_{k+1}}(A'(\alpha)=1) - \Pr_{\alpha \in G_{k+1}}(A'(\alpha)=1) > \frac{1}{P(k)Q^2(k)}$$

The extra $\frac{1}{Q(k)}$ factor comes form the random choice of i. This violates the fact that G was a pseudo random generator as we proved in part 1. This is a contradiction

∎

3.3 Next Bit Tests

If a pseudo-random bit sequence generator has the property that it is difficult to predict the next bit from previous ones with accuracy greater than $\frac{1}{2}$ by a non-negligible amount in time polynomial in the size of the seed, then we say that the generator *passes the "next-bit" test.*

Definition 3.6 A *next bit test* is a special kind of statistical test which takes as input a prefix of a sequence and outputs a prediction of the next bit. ∎

Definition 3.7 A (discrete) probability distribution on a set S is a function $D : S \to [0,1] \subset \mathbf{R}$ so that $\sum_{s \in S} D(s) = 1$. For brevity, probability distributions on $\{0,1\}^k$ will be subscripted with a k. The notation $x \in X_n$ means that x is chosen so that $\forall z \in \{0,1\}^n Pr[x = z] = X_n(z)$. In what follows, U_n is the uniform distribution. ∎

Recall the definition of a pseudo-random number generator:

Definition 3.8 A *pseudo-random number generator (PSRG)* is a polynomial time deterministic algorithm so that:

1. if $|x| = k$ then $|G(x)| = \hat{k}$
2. $\hat{k} > k$,
3. $G_{\hat{k}}$ is pseudo-random[2], where $G_{\hat{k}}$ is the probability distribution induced by G on $\{0,1\}^{\hat{k}}$.

∎

Definition 3.9 We say that a pseudo-random generator *passes the next bit test* A if for every polynomial Q there exists ,an integer k_0 such that for all $\hat{k} > k_0$ and $p < \hat{k}$

$$\Pr_{t \in G_{\hat{k}}}[A(t_1 t_2 \ldots t_p) = t_{p+1}] < \frac{1}{2} + \frac{1}{Q(k)}$$

∎

Theorem 3.10 G passes all next bit tests $\Leftrightarrow$ G passes all statistical tests. ∎

Proof:

($\Leftarrow$) Trivial.

($\Rightarrow$) Suppose, for contradiction, that G passes all next bit test but fails some statistical test A. We will use A to construct a next bit test A' which G fails. Define an operator $\odot$ on probability distributions so that $[X_n \odot Y_m](z) = X_n(z_n) \cdot Y_m(z_m)$ where $z = z_n \circ z_m, |z_n| = n, |z_m| = m$ ($\circ$ is concatenation). For $j \leq \hat{k}$ let $G_{j,\hat{k}}$ be the probability distribution induced by $G_{\hat{k}}$ on $\{0,1\}^j$ by taking prefixes. (That is $G_{j,\hat{k}}(x) = \sum_{z \in \{0,1\}^{\hat{k}}, z \text{ extends } x} G_{\hat{k}}(z)$.)

Define a sequence of distributions $H_i = G_{i,\hat{k}} \odot U_{\hat{k}-i}$ on $\{0,1\}^{\hat{k}}$ of "increasing pseudo-randomness." Then $H_0 = U_{\hat{k}}$ and $H_{\hat{k}} = G_{\hat{k}}$. Because G fails A, A can differentiate between $U_{\hat{k}} = H_0$ and $G_{\hat{k}} = H_{\hat{k}}$; that is, $\exists Q \in \mathcal{Q}[x]$ so that $|Pr_{t \in H_0}[A(t) = 1] - Pr_{t \in H_{\hat{k}}}[A(t) = 1]| > \frac{1}{Q(k)}$. We may assume without loss of generality that $A(t) = 1$ more often when t is chosen from $U_{\hat{k}}$ (otherwise we invert the output of A) so

[2] A pseudo-random distribution is one which is polynomial time indistinguishable from $U_{\hat{k}}$

that we may drop the absolute value markers on the left hand side. Then $\exists i, 0 \le i \le \hat{k} - 1$ so that $Pr_{t \in H_i}[A(t) = 1] - Pr_{t \in H_{i+1}}[A(t) = 1] > \frac{1}{\hat{k}Q(k)}$.

The next bit test A' takes $t_1 t_2 \ldots t_i$ and outputs a guess for t_{i+1}. A' first constructs

$$s_0 = t_1 t_2 \ldots t_i 0 r_{i+2} r_{i+3} \ldots r_{\hat{k}}$$
$$s_1 = t_1 t_2 \ldots t_i 1 \hat{r}_{i+2} \hat{r}_{i+3} \ldots \hat{r}_{\hat{k}}$$

where r_j and $\hat{r}_j$ are random bits for $i + 2 \le j \le \hat{k}$. A' then computes $A(s_0)$ and $A(s_1)$.

If $A(s_0) = A(s_1)$, then A' outputs a random bit.

If $0 = A(s_0) = \overline{A(s_1)}$, then A' outputs 0.

If $1 = A(s_0) = \overline{A(s_1)}$, then A' outputs 1.

Claim 3.11 By analysis similar to that done in the previous lecture, $Pr[A'(t_1 t_2 \ldots t_i) = t_{i+1}] > \frac{1}{2} + \frac{1}{\hat{k}Q(k)}$.
∎

Thus we reach a contradiction: A' is a next bit test that G fails, which contradicts our assumption that G passes all next bit tests.

∎

3.4 Examples of Pseudo-Random Generators

Each of the one way functions we have discussed induces a pseudo-random generator. Listed below are these generators (including the Blum/Blum/Shub generator which will be discussed afterwards) and their associated costs. See [40, 35, 170].

Name	One way function	Cost of computing one way function	Cost of computing j^{th}bit of generator
RSA	$x^e \bmod n, n = pq$	k^3	jk^3
Rabin	$x^2 \bmod n, n = pq$	k^2	jk^2
Blum/Micali	$\mathrm{EXP}(p, g, x)$	k^3	jk^3
Blum/Blum/Shub	(see below)	k^2	$\max(k^2 \log j, k^3)$

3.4.1 Blum/Blum/Shub Pseudo-Random Generator

The Blum/Blum/Shub pseudo-random generator uses the (proposed) one way function $g_n(x) = x^2 \bmod n$ where $n = pq$ for primes p and q so that $p \equiv q \equiv 3 \bmod 4$. In this case, the squaring endomorphism $x \mapsto x^2$ on $\mathbf{Z}_n^*$ restricts to an isomorphism on $(\mathbf{Z}_n^*)^2$, so g_n is a permutation on $(\mathbf{Z}_n^*)^2$. (Recall that every square has a unique square root which is itself a square.)

Claim 3.12 The least significant bit of x is a hard bit for the one way function g_n. ∎

The j^{th} bit of the Blum/Blum/Shub generator may be computed in the following way:

$$B(x^{2^j} \bmod n) = B(x^{\alpha} \bmod m)$$

where $\alpha \equiv 2^j \bmod \phi(n)$. If the factors of n are known, then $\phi(n) = (p-1)(q-1)$ may be computed so that α may be computed prior to the exponentiation. $\alpha = 2^j \bmod \phi(n)$ may be computed in $O(k^2 \log j)$ time and x^α may be be computed in k^3 time so that the computation of $B(x^{2^j})$ takes $O(\max(k^3, k^2 \log j))$ time.

An interesting feature of the Blum/Blum/Shub generator is that if the factorization of n is known, the $2^{\sqrt{n}\,th}$ bit can be generated in time polynomial in $|n|$. The following question can be raised: let $G^{BBS}(x,p,q) = B(f^{2^{\sqrt{n}}}(x)) \circ \ldots \circ B(f^{2^{\sqrt{n}}+2k}(x))$ for $n = pq$ and $|x| = k$. Let G^{BBS}_{2k} be the distribution induced by G^{BBS} on $\{0,1\}^{2k}$.

Open Problem 3.13 Is this distribution G^{BBS}_{2k} pseudo-random? Namely, can you prove that

$$\forall Q \in \mathbf{Q}[x], \forall PTM\ A, \exists k_0, \forall k > k_0 |Pr_{t \in G^{BBS}_{2k}}[A(t) = 1] - Pr_{t \in U_{2k}}[A(t) = 1]| < \frac{1}{Q(2k)}$$

The previous proof that G is pseudo-random doesn't work here because in this case the factorization of n is part of the seed so no contradiction will be reached concerning the difficulty of factoring.

More generally,

Open Problem 3.14 Pseudo-random generators, given seed x, implicitly define an infinite string $g^x_1 g^x_2 \ldots$. Find a pseudo-random generator so that the distribution created by restricting to any polynomially selected subset of bits of g^x is pseudo-random. By polynomially selected we mean examined by a polynomial time machine which can see g^x_i upon request for a polynomial number of i's (the machine must write down the i's, restricting $|i|$ to be polynomial in $|x|$).

Chapter 4

Block ciphers and modes of operation

Block ciphers are the central tool in the design of protocols for shared-key cryptography. They are the main available "technology" we have at our disposal. This chapter will take a look at these objects and describe the state of the art in their construction.

It is important to stress that block ciphers are just tools; they don't, by themselves, do something that an end-user would care about. As with any powerful tool, one has to learn to use this one. Even a wonderful block cipher won't give you security if you use don't use it right. But used well, these are powerful tools indeed. Accordingly, an important theme in several upcoming chapters will be on how to use block ciphers well. We won't be emphasizing how to design or analyze block ciphers, as this remains very much an art. The main purpose of this chapter is just to get you acquainted with what typical block ciphers look like. We'll look at two examples, DES, and AES. DES is the "old standby." It is currently (year 2001) the most widely-used block cipher in existence, and it is of sufficient historical significance that every trained cryptographer needs to have seen its description. AES is a more modern block cipher. It is a possible replacement for DES.

4.1 What is a block cipher?

A block cipher is a function $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ that takes two inputs, a k-bit key K and an l-bit "plaintext" M, to return an l-bit "ciphertext" $C = E(K, M)$. The *key-length* k and the *block-length* l are parameters associated to the block cipher and vary from cipher to cipher, as of course does the design of the algorithm itself. For each key $K \in \{0,1\}^k$ we let $E_K\colon \{0,1\}^l \to \{0,1\}^l$ be the function defined by $E_K(M) = E(K, M)$. For any block cipher, and any key K, the function E_K is a *permutation* on $\{0,1\}^n$. This means that it is a bijection, ie. a one-to-one and onto function of $\{0,1\}^l$ to $\{0,1\}^l$. Accordingly it has an inverse, and we can denote it E_K^{-1}. This function also maps $\{0,1\}^l$ to $\{0,1\}^l$, and of course we have $E_K^{-1}(F_K(M)) = M$ and $E_K(E_K^{-1}(C)) = C$ for all $M, C \in \{0,1\}^l$. We let $E^{-1}\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be defined by $E^{-1}(K, C) = E_K^{-1}(C)$; this is the inverse cipher to E.

The block cipher is a public and fully specified algorithm. Both the cipher E and its inverse E^{-1} should be easily computable, meaning given K, M we can compute $E(K, M)$, and given K, C we can compute $E^{-1}(K, C)$.

In typical usage, a random key K is chosen and kept secret between a pair of users. The function E_K is then used by the two parties to process data in some way before they send it to each other. The adversary may see input-output examples of E_K, meaning pairs of the form (M, C) where $C = E_K(M)$, but is not directly shown the key K. Security relies on the secrecy of the key. (This is necessary, but not always sufficient, for security.). So at a first cut at least, you might view the adversary's goal as recovering the key K given some input-output examples of E_K. The block cipher should be designed to make this task computationally difficult. Later we will refine this view, but it is the classical one, so let's go with it for now.

How do block ciphers work? Lets take a look at some of them to get a sense of this.

4.2 Data Encryption Standard

The Data Encryption Standard (DES) is the quintessential block cipher. Even though it is now feeling its age, and on the way out, no discussion of block ciphers can really omit mention of this construction. DES is a remarkably well-engineered algorithm which has had a powerful influence on cryptography. DES is in very widespread use, and probably will be for some years to come. Every time you use an ATM machine, you are using DES.

4.2.1 A brief history

In 1972 the NBS (National Bureau of Standards, now NIST, the National Institute of Science and Technology) initiated a program for data protection and wanted as part of it an encryption algorithm that could be standardized. They put out a request for such an algorithm. In 1974, IBM responded with a design based on their "Lucifer" algorithm. This design would eventually evolve into the DES.

DES has a key-length of $k = 56$ and a block-length of $l = 64$. It consists of 16 rounds of what is called a "Feistel network." We will describe the details shortly.

After NBS, several other bodies adopted DES as a standard, including ANSI (the American National Standards Institute) and the American Bankers Association.

The standard was to be reviewed every five years to see whether or not it should be re-adopted. Although there were claims that it would not be re-certified, the algorithm was re-certified again and again. Only recently did the work for finding a replacement begin in earnest, in the form of the AES (Advanced Encryption Standard) effort.

DES proved remarkably secure. There has, since the beginning, been concerns, especially about exhaustive key-search. But for a fair length of time, the key size of 56 bits was good enough against all but very well-funded organizations. Interesting attacks significantly different from key search emerged only in the nineties, and even then don't break DES in a sense significant in practice. But with today's technology, 56 bits is too small a key size for many security applications.

4.2.2 Construction

The construction is described in FIPS 46 [147]. The following discussion is a quick guide that you can follow if you have the FIPS document at your side.

Begin at page 87 where you see a big picture. The input is 64 bits and in addition there is a 56 bit key K. (They say 64, but actually every eighth bit is ignored. It can be set to the parity of the previous seven.) Notice the algorithm is public. You operate with a hidden key, but nothing about the algorithm is hidden.

The first thing the input is hit with is something called the initial permutation, or IP. This just shuffles bit positions. That is, each bit is moved to some other position. How? In a fixed and specified way: see page 88. Similarly, right at the end, notice they apply the inverse of the same permutation. From now on, ignore these. They do not affect security. (As far as anyone can tell, they are there to make loading the chips easier.)

The essence of DES is in the round structure. There are 16 rounds. Each round i has an associated subkey K_i which is 48 bits long. The subkeys $K_1, \ldots, K_{16}$ are derived from the main key K, in a manner explained on page 95 of the FIPS document.

In each round, the input is viewed as a pair (L_i, R_i) of 32 bit blocks, and these are transformed into the new pair (L_{i+1}, R_{i+1}), via a certain function f that depends on a subkey K_i pertaining to round i. The structure of this transformation is important: it is called the Feistel transformation.

The Feistel transformation, in general, is like this. For some function g known to the party computing the transformation, it takes input (L, R) and returns (L', R') where $L' = R$ and $R' = g(R) \oplus L$. A central property of this transformation is that it is a permutation, and moreover if you can compute g then you can also invert the transform. Indeed, given (L', R') we can recover (L, R) via $R = L'$ and $L = g(R) \oplus R'$. For DES, the role of g in round i is played by $f(K_i, \cdot)$, the round function specified by the subkey K_i. Since $\mathrm{DES}_K(\cdot)$ is a sequence of Feistel transforms, each of which is a permutation, the whole algorithm is a permutation, and knowledge of the key K permits computation of $\mathrm{DES}_K^{-1}(\cdot)$.

Up to now the structure has been quite generic, and indeed many block-ciphers use this high level design: a sequence of Feistel rounds. For a closer look we need to see how the function $f(K_i, \cdot)$ works. See the picture on page 90 of the FIPS document. Here K_i is a 48 bit subkey, derived from the 56 bit key in a way depending on the round number. The 32 bit R_i is first expanded into 48 bits. How? In a precise, fixed way, indicated by the table on the same page, saying E-bit selection table. It has 48 entries. Read it as which inputs bits are output. Namely, output bits 32, 1, 2, 3, 4, 5, then 4, 5 again, and so on. It is NOT random looking! In fact barring that 1 and 32 have been swapped (see top left and bottom right) it looks almost sequential. Why did they do this? Who knows. That's the answer to most things about DES.

Now K_i is XORed with the output of the E-box and this 48 bit input enters the famous S-boxes. There are eight S-boxes. Each takes 8 bits to 6 bits. Thus we get out 32 bits. Finally, there is a P-box, a permutation applied to these 32 bits to get another 32 bits. You can see it on page 91.

What are the S-boxes? Each is a fixed, tabulated function, which the algorithm stores as tables in the code or hardware. You can see them on page 93. How to read them? Take the 6 bit input $b_1, b_2, b_3, b_4, b_5, b_6$. Interpret the first two bits as a row number, between 1 and 4. Interpret the rest as a column number, 1 through 16. Now index into the table.

Well, without going into details, the main objective of the above was to give you some idea of the kind of structure DES has. Of course, the main questions about the design are: why, why and why? What motivated these design choices? We don't know too much about this, although we can guess a little.

4.2.3 Speed

How fast can you compute DES? Let's begin by looking at hardware implementations since DES was in fact designed to be fast in hardware. We will be pretty rough; you can find precise estimates, for different architectures, in various places.

You can get over a Gbit/sec throughput using VLSI. Specifically at least 1.6 Gbits/sec, maybe more. That's pretty fast.

Some software figures I found quoted are: 12 Mbits/sec on a HP 9000/887; 9.8 Mbits/sec on a DEC Alpha 4000/610. The performance of DES in software is not as good as one would like, and is one reason people seek alternatives.

4.3 Advanced Encryption Standard

A new standard was chosen to replace the DES in February 2001. The AES (Advanced Encryption Standard) is a block cipher called Rijndael. It is described in [62]. It has a block length of 128 bits, and the key size is variable: it could be 128, 192 or 256 bits.

4.4 Some Modes of operation

Let a block cipher E be fixed, and assume two parties share a key K for this block cipher. This gives them the ability to compute the functions $E_K(\cdot)$ and $E_K^{-1}(\cdot)$. These functions can be applied to an input of l-bits. An application of E_K is a called *enciphering* and an application of E_K^{-1} is called *deciphering*.

Typically the block size l is 64 or 128. Yet in practice we want to process much larger inputs, say text files to encrypt. To do this one uses a block cipher in some mode of operation. There are several of these. We will illustrate by describing three of them that exhibit different kinds of features. We look at ECB (Electronic Code-Book), CBC (Cipher Block Chaining) and CTR (Counter). In each case there is an encryption process which takes an nl-bit string M, usually called the plaintext, and returns a string C, usually called the ciphertext. (If the length of x is not a multiple of l then some appropriate padding must be done to make it so.) An associated decryption process recovers M from C.

If x is a string whose length is a multiple of l then we view it as divided into a sequence of l-bit blocks, and let $x[i]$ denote the i-th block, for $i = 1, \ldots, |x|/l$. That is, $x = x[1] \ldots x[n]$ where $n = |x|/l$.

4.4.1 Electronic codebook mode

Each plaintext block is individually enciphered into an associated ciphertext block.

Algorithm $\mathcal{E}_K(M[1] \ldots M[n])$ For $i = 1, \ldots, n$ do $C[i] \leftarrow E_K(M[i])$ Return $C[1] \ldots C[n]$	Algorithm $\mathcal{D}_K(C[1] \ldots C[n])$ For $i = 1, \ldots, n$ do $M[i] \leftarrow E_K^{-1}(C[i])$ Return $M[1] \ldots M[n]$

4.4.2 Cipher-block chaining mode

CBC mode processes the data based on some *initial vector* IV which is an l-bit string, as follows.

Algorithm $\mathcal{E}_K(\mathrm{IV}, M[1] \ldots M[n])$ $C[0] \leftarrow \mathrm{IV}$ For $i = 1, \ldots, n$ do $C[i] \leftarrow E_K(C[i-1] \oplus M[i])$ Return $C[0]C[1] \ldots C[n]$	Algorithm $\mathcal{D}_K(C[0]C[1] \ldots C[n])$ For $i = 1, \ldots, n$ do $M[i] \leftarrow E_K^{-1}(C[i]) \oplus C[i-1]$ Return $M[1] \ldots M[n]$

Unlike ECB encryption, this operation is not length preserving: the output is l-bits longer than the input. The initial vector is used for encryption, but is then part of the ciphertext, so that the receiver need not be assumed to know it a priori.

Different specific modes result from different ways of choosing the initial vector. Unless otherwise stated, it is assumed that before applying the above encryption operation, the encryptor chooses the initial vector at random, anew for each message M to be encrypted. Other choices however can also be considered, such as letting IV be a counter that is incremented by one each time the algorithm is applied. The security attributed of these different choices are discussed later.

CBC is the most popular mode, used pervasively in practice.

4.4.3 Counter mode

CTR mode also uses an auxiliary value, an "initial value" IV which is an integer in the range $0, 1, \ldots, 2^l - 1$. In the following, addition is done modulo 2^l, and $\mathsf{NtS}_l(j)$ denotes the binary representation of integer j as an l-bit string.

Algorithm $\mathcal{E}_K(\mathrm{IV}, M[1] \ldots M[n])$	Algorithm $\mathcal{D}_K(\mathsf{NtS}_l(\mathrm{IV})C[1] \ldots C[n])$
For $i = 1, \ldots, n$ do	For $i = 1, \ldots, n$ do
$C[i] \leftarrow E_K(\mathsf{NtS}_l(\mathrm{IV} + i)) \oplus M[i]$	$M[i] \leftarrow E_K(\mathsf{NtS}_l(\mathrm{IV} + i)) \oplus C[i]$
Return $\mathsf{NtS}_l(\mathrm{IV})C[1] \ldots C[n]$	Return $M[1] \ldots M[n]$

Notice that in this case, decryption did not require computation of E_K^{-1}, and in fact did not even require that E_K be a permutation. Also notice the efficiency advantage over CBC: the encryption is parallelizable.

Again, there are several choices regarding the initial vector. It could be a counter maintained by the sender and incremented by $n = |M|/l$ after message M has been encrypted. Or, it could be chosen anew at random each time the algorithm is invoked.

4.5 Key recovery attacks on block ciphers

Practical cryptanalysis of a block cipher $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ focuses on key-recovery. They formulate the problem facing the cryptanalyst like this. A k-bit key K is chosen at random. Let $q \geq 0$ be some integer parameter.

GIVEN: The adversary has a sequence of q input-output examples of E_K, say

$$(M_1, C_1), \ldots, (M_q, C_q)$$

where $C_i = E_K(M_i)$ for $i = 1, \ldots, q$ and $M_1, \ldots, M_q$ are all distinct l-bit strings.

FIND: The adversary must find the key K.

Two kinds of "attack" models are considered within this framework:

KNOWN-MESSAGE ATTACK: $M_1, \ldots, M_q$ are any distinct points; the adversary has no control over them, and must work with whatever it gets.

CHOSEN-MESSAGE ATTACK: $M_1, \ldots, M_q$ are chosen by the adversary, perhaps even adaptively. That is, imagine it has access to an "oracle" for the function E_K. It can feed the oracle M_1 and get back $C_1 = E_K(M_1)$. It can then decide on a value M_2, feed the oracle this, and get back C_2, and so on.

Clearly a chosen-message attack gives the adversary much more power, but is also less realistic in practice.

The most obvious attack is exhaustive key search.

EXHAUSTIVE KEY SEARCH: Go through all possible keys $K' \in \{0,1\}^k$ until you find the right one, namely K. How do you know when you hit K? If $E_{K'}(M_1) = C_1$, you bet that $K' = K$. Of course, you could be wrong. But the "chance" of being wrong is small, and gets much smaller if you do more such tests. For DES, two tests is quite enough. That is, the attack in this case only needs $q = 2$, a very small number of input-output examples.

Let us now describe the attack in more detail. For $i = 1, \ldots, 2^k$ let K_i denote the i-th k-bit string (in lexicographic order). The following algorithm implements the attack.

```
For i = 1, ..., 2^k do
    If E(K_i, M_1) = C_1
        then if E(K_i, M_2) = C_2 then return K_i
```

How long does this take? In the worst case, 2^k computations of the block cipher. For the case of DES, even if you use the above mentioned 1.6 Gbits/sec chip to do these computations, the search takes about 6,000 years. So key search appears to be infeasible.

Yet, this conclusion is actually too hasty. We will return to key search and see why later.

Differential and linear cryptanalysis: The discovery of a less trivial attack waited until 1990. Differential cryptanalysis is capable of finding a DES key using 2^{47} input-output examples (that is, it requires $q = 2^{47}$). However, differential cryptanalysis required a chosen-message attack.

Linear cryptanalysis improved differential in two ways. The number of input-output examples required is reduced to 2^{43}, but also only a known-message attack is required.

These were major breakthroughs in cryptanalysis. Yet, their practical impact is small. Why? It difficult to obtain 2^{43} input-output examples. Furthermore, simply storing all these examples requires about 140 terabytes of data.

Linear and differential cryptanalysis were however more devastating when applied to other ciphers, some of which succumbed completely to the attack.

So what's the best possible attack against DES? The answer is exhaustive key search. What we ignored above is *parallelism.*

Key search machines: A few years back it was argued that one can design a \$1 million machine that does the exhaustive key search for DES in about 3.5 hours. More recently, a DES key search machine was actually built, at a cost of \$250,000. It finds the key in 56 hours, or about 2.5 days. The builders say it will be cheaper to build more machines now that this one is built.

Thus DES is feeling its age. Yet, it would be a mistake to take away from this discussion the impression that DES is weak. Rather, what the above says is that it is an impressively strong algorithm. After all these years, the best practical attack known is still exhaustive key search. That says a lot for its design and its designers.

Later we will see that that we would like security properties from a block cipher that go beyond resistance to key-recovery attacks. It turns out that from that point of view, a limitation of DES is its block size. Birthday attacks "break" DES with about $q = 2^{32}$ input output examples. (The meaning of "break" here is very different from above.) Here 2^{32} is the square root of 2^{64}, meaning to resist these attacks we must have bigger block size. The next generation of ciphers has taken this into account.

4.6 Limitations of key-recovery based security

As discussed above, classically, the security of a block ciphers has been looked at with regard to key recovery. That is, analysis of a block cipher E has focused primarily on the following question: given some number q of input-output examples $(M_1, C_1)), \ldots, (M_q, C_q)$, where K is a random, unknown key and $C_i = E_K(M_i)$, how hard is it for an attacker to find K? A block cipher is viewed as "secure" if the best key-recovery attack is computationally infeasible, meaning requires a value of q that is too large to make the attack practical. In the sequel, we refer to this as *security against key-recovery*

However, as a notion of security, security against key-recovery is quite limited. A good notion should be sufficiently strong to be useful. This means that if a block cipher is secure, then it should be possible to use the block cipher to make worthwhile constructions and be able to have some guarantee of the security of these constructions. But even a cursory glance at common block cipher usages shows that good security in the sense of key recovery is not sufficient for security of the usages of block ciphers.

Take for example the CTR mode of operation discussed in Section 4.4. Suppose that the block cipher had the following weakness: Given $C, F_K(C+1), F_K(C+2)$, it is possible to compute $F_K(C+3)$. Then clearly the encryption scheme is not secure, because if an adversary happens to know the first two message blocks, it can figure out the third message block from the ciphertext. (It is perfectly reasonable to assume the adversary already knows the first two message blocks. These might, for example, be public header information, or the name of some known recipient.) This means that if CTR mode encryption is to be secure, the block cipher must have the property that given $C, F_K(C+1), F_K(C+2)$, it is computationally infeasible to compute

$F_K(C+3)$. Let us call this property SP1, for "security property one".

Of course, anyone who knows the key K can easily compute $F_K(C+3)$ given $C, F_K(C+1), F_K(C+2)$. And it is hard to think how one can do it without knowing the key. But there is no guarantee that someone *cannot* do this without knowing the key. That is, confidence in the security of F against key recovery does *not* imply that SP1 is true.

This phenomenon continues. As we see more usages of ciphers, we build up a longer and longer list of security properties SP1, SP2, SP3, ... that are necessary for the security of some block cipher based application.

Furthermore, even if SP1 is true, CTR mode encryption may still be weak. SP1 is not *sufficient* to guarantee the security of CTR mode encryption. Similarly with other security properties that one might naively come up with.

This long list of necessary but not sufficient properties is no way to treat security. What we need is *one single* "MASTER" property of a block cipher which, if met, *guarantees* security of *lots of* natural usages of the cipher.

A good example to convince oneself that security against key recovery is not enough is to consider the block cipher $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ defined for all keys $K \in \{0,1\}^k$ and plaintexts $x \in \{0,1\}^l$ by $F(K,x) = x$. That is, each instance F_K of the block cipher is the identity function. Is this a "good" block cipher? Surely not. Yet, it is exceedingly secure against key-recovery. Indeed, given any number of input output examples of F_K, an adversary cannot even test whether a given key is the one in use.

This might seem like an artificial example. Many people, on seeing this, respond by saying: "But, clearly, DES and AES are *not* designed like this." True. But that is missing the point. The point is that security against key-recovery *alone* does not make a "good" block cipher. We must seek a better notion of security. Chapter 5 on pseudorandom functions does this.

4.7 Exercises and Problems

Exercise 4.1 Show that for all $K \in \{0,1\}^{56}$ and all $x \in \{0,1\}^{64}$

$$\mathrm{DES}_K(x) = \overline{\mathrm{DES}_{\overline{K}}(\overline{x})}\ .$$

This is called the key-complementation property of DES. ▮

Exercise 4.2 Show how to use the key-complementation property of DES to speed up exhaustive key search by a factor of two. Explain any assumptions that you make. ▮

Exercise 4.3 Find a key K such that $\mathrm{DES}_K(\cdot) = \mathrm{DES}_K^{-1}(\cdot)$. Such a key is sometimes called a "weak" key. How many weak keys can you find? Why do you think they are called "weak?" ▮

CHAPTER 5

Pseudo-random functions

Pseudorandom functions (PRFs) and their cousins, pseudorandom permutations (PRPs), figure as central tools in the design of protocols, especially those for shared-key cryptography. At one level, PRFs and PRPs can be used to model block ciphers, and they thereby enable the security analysis of protocols based on block ciphers. But PRFs and PRPs are also a wonderful conceptual starting points in other contexts. In this chapter we will introduce PRFs and PRPs and try to understand their basic properties.

5.1 Function families

A *function family* is a map $F\colon \mathsf{Keys}(F) \times \mathsf{Dom}(F) \to \mathsf{Range}(F)$. Here $\mathsf{Keys}(F)$ is the set of keys of F; $\mathsf{Dom}(F)$ is the domain of F; and $\mathsf{Range}(F)$ is the range of F. The two-input function F takes a key K and input x to return a point y we denote by $F(K,x)$. For any key $K \in \mathsf{Keys}(F)$ we define the map $F_K\colon \mathsf{Dom}(F) \to \mathsf{Range}(F)$ by $F_K(x) = F(K,x)$. We call the function F_K an *instance* of family F. Thus, F specifies a collection of maps, one for each key. That's why we call F a family of functions.

Most often $\mathsf{Keys}(F) = \{0,1\}^k$ and $\mathsf{Dom}(F) = \{0,1\}^l$ and $\mathsf{Range}(F) = \{0,1\}^L$ for some integer values of $k, l, L \geq 1$. But sometimes the domain or range could be sets of different kinds, containing strings of varying lengths.

There is some probability distribution on the set of keys $\mathsf{Keys}(F)$. Most often, just the uniform distribution. So with $\mathsf{Keys}(F) = \{0,1\}^k$, we are just drawing a random k-bit string as a key. We denote by $K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$ the operation of selecting a random string from $\mathsf{Keys}(F)$ and naming it K. Then $f \stackrel{R}{\leftarrow} F$ denotes the operation: $K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)\,;\, f \leftarrow F_K$. In other words, let f be the function F_K where K is a randomly chosen key. We are interested in the input-output behavior of this randomly chosen instance of the family.

A *permutation* is a map whose domain and range are the same set, and the map is a length-preserving bijection on this set. That is, a map $\pi\colon D \to D$ is a permutation if $|\pi(x)| = |x|$ for all $x \in D$ and also π is one-to-one and onto. We say that F is a family of permutations if $\mathsf{Dom}(F) = \mathsf{Range}(F)$ and each F_K is a permutation on this common set.

Example 5.1 A block cipher is a family of permutations. For example, DES is a family of permutations with $\mathsf{Keys}(\mathrm{DES}) = \{0,1\}^{56}$ and $\mathsf{Dom}(\mathrm{DES}) = \{0,1\}^{64}$ and $\mathsf{Range}(\mathrm{DES}) = \{0,1\}^{64}$. Here $k = 56$ and $l = L = 64$. Similarly AES is a family of permutations with $\mathsf{Keys}(\mathrm{AES}) = \{0,1\}^{128}$ and $\mathsf{Dom}(\mathrm{AES}) = \{0,1\}^{128}$ and $\mathsf{Range}(\mathrm{AES}) = \{0,1\}^{128}$. Here $k = 128$ and $l = L = 128$. ∎

5.2 Random functions and permutations

Let $D, R \subseteq \{0,1\}^*$ be finite sets and let $l, L \geq 1$ be integers. There are two function families that we fix. One is $\mathsf{Rand}^{D\to R}$, the family of all functions of D to R. The other is Perm^D, the family of all permutations on D. For compactness of notation we let $\mathsf{Rand}^{l\to L}$ equal $\mathsf{Rand}^{D\to R}$ with $D = \{0,1\}^l$ and $R = \{0,1\}^L$. We also set Perm^l equal Perm^D with $D = \{0,1\}^l$.

What are these families? The family $\mathsf{Rand}^{D\to R}$ has domain D and range R, while the family Perm^D has domain and range D. The set of instances of $\mathsf{Rand}^{D\to R}$ is the set of all functions mapping D to R, while the set of instances of Perm^D is the set of all permutations on D. The key describing any particular instance function is simply a description of this instance function in some canonical notation. For example, order the domain D lexicographically as $x_1, x_2, \ldots$, and then let the key for a function f be the list of values $(f(x_1), f(x_2), \ldots)$. The key-space of $\mathsf{Rand}^{D\to R}$ is simply the set of all these keys, under the uniform distribution.

Let us illustrate in more detail for the cases in which we are most interested. The key of a function in $\mathsf{Rand}^{l\to L}$ is simply a list of of all the output values of the function as its input ranges over $\{0,1\}^l$. Thus

$$\mathsf{Keys}(\mathsf{Rand}^{l\to L}) \;=\; \{\, (y_1, \ldots, y_{2^l}) \;:\; y_1, \ldots, y_{2^l} \in \{0,1\}^L \,\}$$

is the set of all sequences of length 2^l in which each entry of a sequence is an L-bit string. For any $x \in \{0,1\}^l$ we interpret x as an integer in the range $\{1, \ldots, 2^l\}$ and set

$$\mathsf{Rand}^{l\to L}((y_1, \ldots, y_{2^l}), x) = y_x \;.$$

Notice that the key space is very large; it has size 2^{L2^l}. Naturally, since there is a key for every function of l-bits to L-bits, and this is the number of such functions. The key space is equipped with the uniform distribution, so that $f \stackrel{R}{\leftarrow} \mathsf{Rand}^{l\to L}$ is the operation of picking a random function of l-bits to L-bits.

On the other hand, for Perm^l, the key space is

$$\begin{aligned}\mathsf{Keys}(\mathsf{Perm}^l) \;=\; & \{(y_1, \ldots, y_{2^l}) : \;\; y_1, \ldots, y_{2^l} \in \{0,1\}^l \text{ and} \\ & y_1, \ldots, y_{2^l} \text{ are all distinct}\} \;.\end{aligned}$$

For any $x \in \{0,1\}^l$ we interpret x as an integer in the range $\{1, \ldots, 2^l\}$ and set

$$\mathsf{Perm}^l((y_1, \ldots, y_{2^l}), x) = y_x \;.$$

The key space is again equipped with the uniform distribution, so that $f \stackrel{R}{\leftarrow} \mathsf{Perm}^l$ is the operation of picking a random permutation on $\{0,1\}^l$. In other words, all the possible permutations on $\{0,1\}^l$ are equally likely.

Example 5.2 We exemplify $\mathsf{Rand}^{3\to 2}$, meaning $l = 3$ and $L = 2$. The domain is $\{0,1\}^3$ and the range is $\{0,1\}^2$. An example instance f of the family is illustrated below via its input-output table:

x	000	001	010	011	100	101	110	111
$f(x)$	10	11	01	11	10	00	00	10

The key corresponding to this particular function is

$$(10, 11, 01, 11, 10, 00, 00, 10) \;.$$

The key-space of $\mathsf{Rand}^{3\to 2}$ is the set of all such sequences, meaning the set of all 8-tuples each component of which is a two bit string. There are

$$2^{2\cdot 2^3} \;=\; 2^{16} \;=\; 65{,}536$$

such tuples, so this is the size of the key-space. ∎

We will hardly ever actually think about these families in terms of this formalism. Indeed, it is worth pausing here to see how to think about them more intuitively, because they are important objects.

We will consider settings in which you have black-box access to a function g. This means that there is a box to which you can give any value x of your choice (provided x is in the domain of g), and box gives you back $g(x)$. But you can't "look inside" the box; your only interface to it is the one we have specified. A random function $g\colon \{0,1\}^l \to \{0,1\}^L$ being placed in this box corresponds to the following. Each time you give the box an input, you get back a random L-bit string, with the sole constraint that if you twice give the box the same input x, it will be consistent, returning both times the same output $g(x)$. In other words, a random function of l-bits to L-bits can be thought of as a box which given any input $x \in \{0,1\}^l$ returns a random number, except that if you give it an input you already gave it before, it returns the same thing as last time. It is this "dynamic" view that we suggest the reader have in mind in thinking about random functions.

The dynamic view can be thought of as following program. The program maintains the function in the form of a table T where $T[x]$ holds the value of the function at x. Initially, the table is empty. The program processes an input $x \in \{0,1\}^l$ as follows:

If $T[x]$ is not defined then
    Flip coins to determine a string $y \in \{0,1\}^L$ and let $T[x] \leftarrow y$
Return $T[x]$

The answer on any point is random and independent of the answers on other points.

Another way to think about a random function is as a large, pre-determined random table. The entries are of the form (x, y). For each x someone has flipped coins to determine y and put it into the table.

We are more used to the idea of picking points at random. Here we are picking a *function* at random.

One must remember that the term "random function" is misleading. It might lead one to think that certain functions are "random" and others are not. (For example, maybe the constant function which always returns 0^L is not random, but a function with many different range values is random.) This is not right. The randomness of the function refers to the way it was chosen, not to an attribute of the selected function itself. When you choose a function at random, the constant function is just as likely to appear as any other function. It makes no sense to talk of the randomness of an individual function; the term "random function" just means a function chosen at random.

Example 5.3 Let's do some simple probabilistic computations to understand random functions. Fix $x \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\mathbf{P}\left[\, f(x) = y \;:\; f \stackrel{R}{\leftarrow} \mathsf{Rand}^{l \to L} \,\right] = 2^{-L} \,.$$

Notice it doesn't depend on l. Also it doesn't depend on the values of x, y.

Now fix $x_1, x_2 \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\mathbf{P}\left[\, f(x_1) = f(x_2) = y \;:\; f \stackrel{R}{\leftarrow} \mathsf{Rand}^{l \to L} \,\right] = \begin{cases} 2^{-2L} & \text{if } x_1 \neq x_2 \\ 2^{-L} & \text{if } x_1 = x_2 \end{cases}$$

This illustrates *independence*. Finally fix $x_1, x_2 \in \{0,1\}^l$ and $y \in \{0,1\}^L$. Then:

$$\mathbf{P}\left[\, f(x_1) \oplus f(x_2) = y \;:\; f \stackrel{R}{\leftarrow} \mathsf{Rand}^{l \to L} \,\right] = \begin{cases} 2^{-L} & \text{if } x_1 \neq x_2 \\ 0 & \text{if } x_1 = x_2 \text{ and } y \neq 0^L \\ 1 & \text{if } x_1 = x_2 \text{ and } y = 0^L \end{cases}$$

Similar things hold for the sum of more than two things. ∎

5.3 Pseudorandom functions

A pseudorandom function is a family of functions with the property that the input-output behavior of a random instance of the family is "computationally indistinguishable" from that of a random function. Someone who has only black-box access to a function, meaning can only feed it inputs and get outputs, has a hard time telling whether the function in question is a random instance of the family in question or a random function. The purpose of this section is to arrive at a suitable definition of this notion. Later we will look at motivation and applications.

We fix a family of functions $F\colon \mathsf{Keys}(F) \times D \to R$. (You may want to think $\mathsf{Keys}(F) = \{0,1\}^k$, $D = \{0,1\}^l$ and $R = \{0,1\}^L$ for some integers $k, l, L \geq 1$, since this is the most common case.) Imagine that you are in a room which contains a computer that is connected to another computer outside your room. You can type something into your computer and send it out, and an answer will come back. The allowed questions you can type must be strings from the domain D, and the answers you get back will be strings from the range R. The computer outside your room implements a function $g\colon D \to R$, so that whenever you type a value x you get back $g(x)$. However, your only access to g is via this interface, so the only thing you can see is the input-output behavior of g. We consider two different ways in which g will be chosen, giving rise to two different "worlds."

World 0: The function g is drawn at random from $\mathsf{Rand}^{D\to R}$, namely via $g \stackrel{R}{\leftarrow} \mathsf{Rand}^{D\to R}$. (So g is just a random function of D to R.)

World 1: The function g is drawn at random from F, namely $g \stackrel{R}{\leftarrow} F$. (This means that a key is chosen via $K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$ and then g is set to F_K.)

You are not told which of the two worlds was chosen. The choice of world, and of the corresponding function g, is made before you enter the room, meaning before you start typing questions. Once made, however, these choices are fixed until your "session" is over. Your job is to discover which world you are in. To do this, the only resource available to you is your link enabling you to provide values x and get back $g(x)$. After trying some number of values of your choice, you must make a decision regarding which world you are in. The quality of pseudorandom family F can be thought of as measured by the difficulty of telling, in the above game, whether you are in World 0 or in World 1.

Intuitively, the game just models some way of "using" the function g in an application like an encryption scheme. If it is not possible to distinguish the input-output behavior of a random instance of F from a truly random function, the application should behave in roughly the same way whether it uses a function from F or a random function. Later we will see exactly how this works out; for now let us continue to develop the notion. But we warn that pseudorandom functions can't be substituted for random functions in *all* usages of random functions. To make sure it is OK in a particular application, you have to make sure that it falls within the realm of applications for which the formal definition below can be applied.

The act of trying to tell which world you are in is formalized via the notion of a *distinguisher*. This is an algorithm which is provided oracle access to a function g and tries to decide if g is random or pseudorandom. (Ie. whether it is in world 0 or world 1.) A distinguisher can only interact with the function by giving it inputs and examining the outputs for those inputs; it cannot examine the function directly in any way. We write A^g to mean that distinguisher A is being given oracle access to function g. Intuitively, a family is pseudorandom if the probability that the distinguisher says 1 is roughly the same regardless of which world it is in. We capture this mathematically below. Further explanations follow the definition.

Definition 5.4 Let $F\colon \mathsf{Keys}(F) \times D \to R$ be a family of functions, and let A be an algorithm that takes an oracle for a function $g\colon D \to R$, and returns a bit. We consider two experiments:

Experiment $\mathbf{Exp}^{\mathrm{prf\text{-}1}}_{F,A}$	Experiment $\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{F,A}$
$K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$	$g \stackrel{R}{\leftarrow} \mathsf{Rand}^{D\to R}$
$d \leftarrow A^{F_K}$	$d \leftarrow A^{g}$
Return d	Return d

The *prf-advantage* of A is defined as

$$\mathbf{Adv}^{\mathrm{prf}}_{F,A} \quad = \quad \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf}\text{-}1}_{F,A} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf}\text{-}0}_{F,A} = 1\right] .$$

For any t, q, μ we define the *prf-advantage* of F

$$\mathbf{Adv}^{\mathrm{prf}}_{F}(t,q,\mu) \quad = \quad \max_{A} \{ \mathbf{Adv}^{\mathrm{prf}}_{F,A} \}$$

where the maximum is over all A having time-complexity t and making at most q oracle queries, the sum of the lengths of these queries being at must μ bits. ∎

The algorithm A models the person we were imagining in our room, trying to determine which world he or she was in by typing queries to the function g via a computer. In the formalization, the person is an algorithm, meaning a piece of code. We formalize the ability to query g as giving A an oracle which takes input any string $x \in D$ and returns $g(x)$. Algorithm A can decide which queries to make, perhaps based on answers received to previous queries. Eventually, it outputs a bit d which is its decision as to which world it is in.

It should be noted that the family F is public. The adversary A, and anyone else, knows the description of the family, and have code to implement it, meaning are capable, given values K, x, of computing $F(K, x)$.

The worlds are captured by what we call "experiments." The first experiment picks a random instance F_K of family F and then runs adversary A with oracle $g = F_K$. Adversary A interacts with its oracle, querying it and getting back answers, and eventually outputs a "guess" bit. The experiment returns the same bit. The second experiment picks a random function $g\colon D \to R$ and runs A with this as oracle, again returning A's guess bit. Each experiment has a certain probability of returning 1. The probability is taken over the random choices made in the experiment. Thus, for the first experiment, the probability is over the choice of K and any random choices that A might make, for A is allowed to be a randomized algorithm. In the second experiment, the probability is over the random choice of g and any random choices that A makes. These two probabilities should be evaluated separately; the two experiments are completely different.

To see how well A does at determining which world it is in, we look at the difference in the probabilities that the two experiments return 1. If A is doing a good job at telling which world it is in, it would return 1 more often in the first experiment than in the second. So the difference is a measure of how well A is doing. We call this measure the prf-advantage of A. Think of it as the probability that A "breaks" the scheme F, with "break" interpreted in a specific, technical way based on the definition.

Different distinguishers will have different advantages. There are two reasons why one distinguisher may achieve a greater advantage than another. One is that it is more "clever" in the questions it asks and the way it processes the replies to determine its output. The other is simply that it asks more questions, or spends more time processing the replies. Indeed, we expect that as you see more and more input-output examples of g, your ability to tell which world you are in will go up. The "security" of family F must thus be measured as a function of the resources allowed to the attacker. We want to know, for any given resource limitation, what is the prf-advantage achieved by the most "clever" distinguisher amongst all those who are restricted to the given resource limitations. We associate to the family F a prf-advantage function which on input any values of the resource parameters returns the maximim prf-advantage that an adversary restricted to those resources could obtain. Think of it as the maximum possible achievable probability of "breaking" the scheme F if an attacker is restricted to the given resources.

The choice of resources to consider can vary. In this case we have chosen to measure the time-complexity of A, the number of queries q it makes, and the total length μ of these queries. We associate to the family F an *advantage function* which on input a particular choice of these resource parameters returns the maximum possible advantage that could be obtained by a distinguisher restricted in resource usage by the given parameters. Put another way, it is the advantage of the "cleverest" or "best" distinguisher restricted to the given resources. The advantage function of F captures the security of F as a PRF.

Let us now explain the resources, and some important conventions underlying their measurement, in more detail. The first resource is the time-complexity of A. To make sense of this we first need to fix a model of computation. We fix some RAM model. Think of it as the model used in Algorithms courses, so that you measure the running time of an algorithm just as you did there. However, we adopt the convention that the *time-complexity* of A refers not just to the running time of A, but to the maximum of the running times of the two experiments in the definition, plus the size of the code of A. In measuring the running time of the first experiment, we must count the time to choose the key K at random, and the time to compute the value $F_K(x)$ for any query x made by A to its oracle. In measuring the running time of the second experiment, we count the time to choose the random function g in a dynamic way, meaning we count the cost of maintaining a table of values of the form $(x, g(x))$. Entries are added to the table as g makes queries. A new entry is made by picking the output value at random.

The number of queries made by A captures the number of input-output examples it sees. In general, not all strings in the domain must have the same length, and hence we also measure the sum of the lengths of all queries made.

There is one feature of the above parameterization about which everyone asks. Suppose that F has key-length k. Obviously, the key length is a fundamental determinant of security: larger key length will typically mean more security. Yet, the key length k does not appear explicitly in the advantage function $\mathbf{Adv}_F^{\mathrm{prf}}(t, q, \mu)$. Why is this? The advantage function is in fact a function of k, but without knowing more about F it is difficult to know what kind of function. The truth is that the key length itself does not matter: what matters is just the advantage a distinguisher can obtain. In a well-designed block cipher, $\mathbf{Adv}_F^{\mathrm{prf}}(t, q, \mu)$ should be about $t/2^k$. But that is really an ideal; in practice we should not assume ciphers are this good.

The strength of this definition lies in the fact that it does not specify anything about the kinds of strategies that can be used by a distinguisher; it only limits its resources. A distinguisher can use whatever means desired to distinguish the function as long as it stays within the specified resource bounds.

What do we mean by a "secure" PRF? Definition 5.4 does not have any explicit condition or statement regarding when F should be considered "secure." It only associates to F a prf-advantage function. Intuitively, F is "secure" if the value of the advantage function is "low" for "practical" values of the input parameters. This is, of course, not formal. It is possible to formalize the notion of a secure PRF using a complexity theoretic framework; one would say that the advantage of any adversary whose resources are polynomially-bounded is negligible. This requires an extension of the model to consider a security parameter in terms of which asymptotic estimates can be made. We will discuss this in more depth later, but for now we stick to a framework where the notion of what exactly is a secure PRF remains fuzzy. The reason is that this reflects real life. In real life, security is not some absolute or boolean attribute; security is a function of the resources invested by an attacker. All modern cryptographic systems are breakable in principle; it is just a question of how long it takes.

This is our first example of a cryptographic definition, and it is worth spending time to study and understand it. We will encounter many more as we go along. Towards this end let us summarize the main features of the definitional framework as we will see them arise later. First, there are *experiments*, involving an adversary. Then, there is some *advantage* function associated to an adversary which returns the probability that the adversary in question "breaks' the scheme. Finally, there is an advantage function associated to the cryptographic protocol itself, taking input resource parameters and returning the maximum possible probability of "breaking" the scheme if the attacker is restricted to the given resource parameters. These three components will be present in all definitions. The important component is the experiments; this is where we chose and pin down a model of security.

5.4 Pseudorandom permutations

Recall that a block cipher F is a family of permutations: each instance F_K of the family is a permutation. With the intent of modeling block ciphers we introduce the notion of a pseudorandom permutation. We proceed exactly as above, but replace $\mathsf{Rand}^{D \to R}$ with Perm^D.

In this setting, there are two kinds of attacks that one can consider. One, as before, is that the adversary gets an oracle for the function g being tested. However when g is a permutation one can also consider the case where the adversary gets, in addition, and oracle for g^{-1}. We consider these settings in turn. The first is the setting of chosen-plaintext attacks while the second is the setting of chosen-ciphertext attacks.

5.4.1 PRP under CPA

We fix a family of functions $F\colon \mathsf{Keys}(F) \times D \to D$. (You may want to think $\mathsf{Keys}(F) = \{0,1\}^k$ and $D = \{0,1\}^l$, since this is the most common case. We do not mandate that F be a family of permutations although again this is the most common case.) As before, we consider an adversary A that is placed in a room where it has oracle access to a function g chosen in one of two ways.

World 0: The function g is drawn at random from Perm^D, namely via $g \stackrel{R}{\leftarrow} \mathsf{Perm}^D$. (So g is just a random permutation on D.)

World 1: The function g is drawn at random from F, namely $g \stackrel{R}{\leftarrow} F$. (This means that a key is chosen via $K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$ and then g is set to F_K.)

Notice that World 1 is the same in the PRF setting, but World 0 has changed. As before the task facing the adversary A is to determine in which world it was placed based on the input-output behavior of g.

Definition 5.5 Let $F\colon \mathsf{Keys}(F) \times D \to D$ be a family of functions, and let A be an algorithm that takes an oracle for a function $g\colon D \to D$, and returns a bit. We consider two experiments:

Experiment $\mathbf{Exp}_{F,A}^{\text{prp-cpa-1}}$	Experiment $\mathbf{Exp}_{F,A}^{\text{prp-cpa-0}}$
$K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$	$g \stackrel{R}{\leftarrow} \mathsf{Perm}^D$
$d \leftarrow A^{F_K}$	$d \leftarrow A^{g}$
Return d	Return d

The *prp-cpa-advantage* of A is defined as

$$\mathbf{Adv}_{F,A}^{\text{prp-cpa}} \quad = \quad \mathbf{P}\left[\mathbf{Exp}_{F,A}^{\text{prp-cpa-1}} = 1\right] - \mathbf{P}\left[\mathbf{Exp}_{F,A}^{\text{prp-cpa-0}} = 1\right] .$$

For any t, q, μ we define the *prp-cpa-advantage* of F via

$$\mathbf{Adv}_{F}^{\text{prp-cpa}}(t,q,\mu) \quad = \quad \max_{A} \{ \mathbf{Adv}_{F,A}^{\text{prp-cpa}} \}$$

where the maximum is over all A having time-complexity t and making at most q oracle queries, the sum of the lengths of these queries being at must μ bits. ∎

The intuition is similar to that for Definition 5.4. The difference is that here the "ideal" object with which F is being compared is no longer the family of random functions, but rather the family of random permutations.

Experiment $\mathbf{Exp}_{F,A}^{\text{prp-cpa-1}}$ is actually identical to $\mathbf{Exp}_{F,A}^{\text{prf-1}}$. The probability is over the random choice of key K and also over the coin tosses of A if the latter happens to be randomized. The experiment returns the same bit that A returns. In Experiment $\mathbf{Exp}_{F,A}^{\text{prp-cpa-0}}$, a permutation $g\colon \{0,1\}^l \to \{0,1\}^l$ is chosen at random, and the result bit of A's computation with oracle g is returned. The probability is over the choice of g and the coins of A if any. As before, the measure of how well A did at telling the two worlds apart, which we call the prp-cpa-advantage of A, is the difference between the probabilities that the experiments return 1.

Conventions regarding resource measures also remain the same as before. Informally, a family F is a secure PRP under CPA if $\mathbf{Adv}_{F}^{\text{prp-cpa}}(t,q,\mu)$ is "small" for "practical" values of the resource parameters.

5.4.2 PRP under CCA

We fix a family of permutations $F\colon \mathsf{Keys}(F) \times D \to D$. (You may want to think $\mathsf{Keys}(F) = \{0,1\}^k$ and $D = \{0,1\}^l$, since this is the most common case. This time, we do mandate that F be a family of permutations.) As before, we consider an adversary A that is placed in a room, but now it has oracle access to two functions, g and its inverse g^{-1}. The manner in which g is chosen is the same as in the CPA case, and once g is chosen, g^{-1} is automatically defined, so we do not have to say how it is chosen.

World 0: The function g is drawn at random from Perm^D, namely via $g \stackrel{R}{\leftarrow} \mathsf{Perm}^D$. (So g is just a random permutation on D.)

World 1: The function g is drawn at random from F, namely $g \stackrel{R}{\leftarrow} F$. (This means that a key is chosen via $K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$ and then g is set to F_K.)

In World 1, $g^{-1} = F_K^{-1}$ is the inverse of the chosen instance, while in World 0 it is the inverse of the chosen random permutation. As before the task facing the adversary A is to determine in which world it was placed based on the input-output behavior of its oracles.

Definition 5.6 Let $F\colon \mathsf{Keys}(F) \times D \to D$ be a family of permutations, and let A be an algorithm that takes an oracle for a function $g\colon D \to D$, and also an oracle for the function $g^{-1}\colon D \to D$, and returns a bit. We consider two experiments:

Experiment $\mathbf{Exp}_{F,A}^{\text{prp-cca-1}}$	Experiment $\mathbf{Exp}_{F,A}^{\text{prp-cca-0}}$
$K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$	$g \stackrel{R}{\leftarrow} \mathsf{Perm}^D$
$d \leftarrow A^{F_K, F_K^{-1}}$	$d \leftarrow A^{g, g^{-1}}$
Return d	Return d

The *prp-cca-advantage* of A is defined as

$$\mathbf{Adv}_{F,A}^{\text{prp-cca}} = \mathbf{P}\left[\mathbf{Exp}_{F,A}^{\text{prp-cca-1}} = 1\right] - \mathbf{P}\left[\mathbf{Exp}_{F,A}^{\text{prp-cca-0}} = 1\right].$$

For any $t, q_e, \mu_e, q_d, \mu_d$ we define the *prp-cca-advantage* of F via

$$\mathbf{Adv}_{F}^{\text{prp-cca}}(t, q_e, \mu_e, q_d, \mu_d) = \max_A \{ \mathbf{Adv}_{F,A}^{\text{prp-cca}} \}$$

where the maximum is over all A having time-complexity t, making at most q_e queries to the g oracle, the sum of the lengths of these queries being at must μ_e bits, and also making at most q_d queries to the g^{-1} oracle, the sum of the lengths of these queries being at must μ_d bits, ∎

The intuition is similar to that for Definition 5.4. The difference is that here the adversary has more power: not only can it query g, but it can directly query g^{-1}. Conventions regarding resource measures also remain the same as before. However, we add some resource parameters. Specifically, since there are now two oracles, we count separately the number of queries, and total length of these queries, for each. Informally, a family F is a secure PRP under CCA if $\mathbf{Adv}_{F}^{\text{prp-cca}}(t, q_e, \mu_e, q_d, \mu_d)$ is "small" for "practical" values of the resource parameters.

5.4.3 Relations between the notions

If an adversary above does not query g^{-1}, the latter oracle may as well not be there, and the adversary is effectively mounting a chosen-plaintext attack. Thus we have the following:

Proposition 5.7 Let $F\colon \mathsf{Keys}(F) \times D \to D$ be a family of permutations. Then

$$\mathbf{Adv}_{F}^{\text{prp-cpa}} cpa(t, q, \mu) = \mathbf{Adv}_{F}^{\text{prp-cpa}} cca(t, q, \mu, 0, 0)$$

for any t, q, μ. ∎

5.5 Sequences of families of PRFs and PRPs

Above, the function families we consider have a finite key-space, and typcially also domains and ranges that are finite sets. A *sequence of families of functions* is a sequence $F^1, F^2, F^3, \ldots$, written $\{F^n\}_{n\geq 1}$. Each F^n is a family of functions with input length $l(n)$, output length $L(n)$ and key length $k(n)$, where l, L, k are functions of the security parameter n, called the input, output and key lengths of the sequence, respectively.

In modeling block ciphers, families as we have considered them are the appropriate abstraction. There are several reasons, however, for which we may also want to consider sequences of families. One is that security can be defined asymptotically, which is definitionally more convenient, particulary because in that case we do have a well-defined notion of security, rather than merely having measures of insecurity as above. Also, when we look to designs whose security is based on the presumed hardness of number-theoretic problems, we naturally get sequences of families rather than families. Let us now state the definition of pseudorandomness for a sequence of families. (We omit the permutation case, which is analogous.)

Let $\mathcal{F} = \{F^n\}_{n\geq 1}$ be a sequence of function families with input length $l(\cdot)$ and output length $L(\cdot)$. We say that $\mathcal{F}$ is *polynomial-time computable* if there is an algorithm which given n, K and x outputs $F^n(K, x)$ in time poly(n). To define security, we now consider a sequence $\{D^n\}_{n\geq 1}$ of distinguishers. We say that $\mathcal{D}$ is polynomial time if D^n always halts in poly(n) steps.

Definition 5.8 Let $\mathcal{F} = \{F^n\}_{n\geq 1}$ be a sequence of function families and let $\mathcal{D} = \{D^n\}_{n\geq 1}$ be a sequence of distinguishers. The *prf-advantage* of $\mathcal{D}$ is the function $\mathbf{Adv}^{\rm prf}_{\mathcal{F},\mathcal{D}}(\cdot)$, defined for every n by

$$\mathbf{Adv}^{\rm prf}_{\mathcal{F},\mathcal{D}}(n) \quad = \quad \mathbf{Adv}^{\rm prf}_{F^n,D^n} \; .$$

We say that $\mathcal{F}$ is a PRF if it is polynomial-time computable and also the function $\mathbf{Adv}^{\rm prf}_{\mathcal{F},\mathcal{D}}(\cdot)$ is negligible for every polynomial-time distinguisher sequence $\mathcal{D}$. ∎

Notice that this time the definition insists that the functions themselves can be efficiently computed.

5.6 Usage of PRFs and PRPs

We discuss some motivation for these notions of security.

5.6.1 The shared random function model

In symmetric (ie. shared-key) cryptography, Alice and Bob share a key K which the adversary doesn't know. They want to use this key to achieve various things. In particular, to encrypt and authenticate the data they send to each other. A key is (or ought to be) a short string. Suppose however that we allow the parties a very long shared string. The form it takes is a random function f of l bits to L bits, for some pre-specified l, L. This is called the shared random function model.

The shared random function model cannot really be realized in practice because, as we saw, random functions are just too big to even store. It is a conceptual model. To work in this model, we give the parties oracle access to f. They may write down $x \in \{0,1\}^l$ and in one step be returned $f(x)$.

It turns out that the shared random function model is a very convenient model in which to think about cryptography, formulate schemes, and analyze them. In particular, we will see many examples where we design schemes in the shared random function model and prove them secure. This is true for a variety of problems, but most importantly for encryption and message authentication. The proof of security here is absolute: we do not make any restrictions on the computational power of the adversary, but are able to simply provide an upper bound on the success probability of the adversary.

As an example, consider the CTR mode of operation discussed in Section 4.4.3. Consider the version where the initial vector is a counter. Consider replaceing every invocation of E_K with an invocation of the random function f. (Assume $l = L$). In that case, the mode of operation turns into the one-time pad cryptosystem. The shared random key is just the random function f. As we have discussed, this is well known to meet a strong and well-defined security. So, in the shared random function model, CTR mode is easily seen to be "good".

But now what? We have schemes which are secure but a priori can't be efficiently realized, since they rely on random functions. That's where pseudorandom function or permutation families come in. A PRF family is a family F of functions indexed by small keys (eg. 56 or 128 bits). However, it has the property that if K is shared between Alice and Bob, and we use F_K in place of a random function f in some scheme designed in the shared random function model, the resulting scheme is still secure as long as the adversary is restricted in resource usage.

In other words, instances of PRFs can be used in place of random functions in shared key schemes. The definition of a PRF is crafted to make this possible for as wide a range of applications as possible. An instance of a pseudorandom function is specified by a short key K, and the parties need only store this key. Then, they use this function in place of the random function in the scheme. And things should work out, in the sense that if the scheme was secure when a random function was used, it should still be secure.

This is a very rough idea. Technically, it is not always true: this is the intuition. Pseudorandom functions don't always work. That is, you can't substitute them for random functions in any usage of the latter and expect things to work out. But if used right, it works out in a large number of cases. How do we identify these cases? We have to resort to the formal definition of a pseudorandom function family and prove the security of our construct based on it. We will see how to do this later.

In this context we stress one important point. The security of a PRF relies on the key K being *secret.* The adversary is not given K and cannot directly compute the function. (Of course it might gain some information about values of F_K on various points via the usage of F_K by the legitimate parties, but that will be OK.) In other words, you can substitute *shared, secret* random functions by PRFs, but not public ones.

Pseudorandom functions are an intriguing notion and a powerful tool that enable the following design paradism. When you want to design a scheme for encryption, authentication, or some other purpose, design it in the shared random function model. Then simply substitute the random function with a pseudorandom one, and your scheme should still be secure.

5.6.2 Modeling block ciphers

One of the primary motivations for the notions of pseudorandom functions (PRFs) and pseudorandom permutations (PRPs) is to model block ciphers and thereby enable the security analysis of protocols that use block ciphers.

As discussed in Section 4.6, classically the security of DES or other block ciphers has been looked at only with regard to key recovery. That is, analysis of a block cipher F has focused on the following question: Given some number of input-output examples

$$(x_1, F_K(x_1)), \ldots, (x_q, F_K(x_q))$$

where K is a random, unknown key, how hard is it to find K? The block cipher is taken as "secure" if the resources required to recover the key are prohibitive. Yet, as we saw, even a cursory glance at common block cipher usages shows that hardness of key recovery is not *sufficient* for security. We had discussed wanting a "MASTER" security property of block ciphers under which natural usages of block ciphers could be proven secure. We suggest that this "MASTER" property is that the block cipher be a secure PRP, under either CPA or CCA.

We cannot prove that specific block ciphers have this property. The best we can do is assume they do, and then go on to use them. For quantitative security assessements, we would make specific conjectures about the advantage functions of various block ciphers. For example we might conjecture something like:

$$\mathbf{Adv}^{\text{prp-cpa}}_{\text{DES}}(t, q, 64q) \quad = \quad c_1 \cdot \frac{t/T_{\text{DES}}}{2^{55}} + c_1 \cdot \frac{q}{2^{40}}$$

Here T_{DES} is the time to do one DES computation on our fixed RAM model of computation, and c_1 is some constant. In other words, we are conjecturing that the best attacks are either exhaustive key search or linear cryptanalysis. We might be bolder with regard to AES and conjecture something like

$$\mathbf{Adv}^{\text{prp-cpa}}_{\text{AES}}(t,q,128q) \quad = \quad c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + c_1 \cdot \frac{q}{2^{128}} \,.$$

We could also make similar conjectures regarding the strength of block ciphers as PRPs under CCA rather than CPA.

More interesting is $\mathbf{Adv}^{\text{prf}}_{\text{DES}}(t,q)$. Here we cannot do better than assume that

$$\begin{aligned}
\mathbf{Adv}^{\text{prf}}_{\text{DES}}(t,q,64q) &= c_1 \cdot \frac{t/T_{\text{DES}}}{2^{55}} + \frac{q^2}{2^{64}} \\
\mathbf{Adv}^{\text{prf}}_{\text{AES}}(t,q,128q) &= c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + \frac{q^2}{2^{128}} \,.
\end{aligned}$$

This is due to the birthday attack discussed later. It is possible because a block cipher is a family of permutations, and is a general phenomenon.

We stress that these are all conjectures, and barring the birthday attack, they are based on the assumption that the best possible attacks against the pseudorandomness of the cipher stem from the known key-recovery attacks. Yet, there could exist better attacks that break the cipher as a PRF without recovering the key. So far, we do not know of any such attacks, but the amount of cryptanalytic effort that has focused on this goal is small. Certainly, to assume that a block cipher is a PRF is a much stronger assumption than that it is secure against key recovery. Nonetheless, the motivation and arguments we have outlined in favor of the PRF assumption stay, and our view is that if a block cipher is broken as a PRF then it should be considered insecure, and a replacement should be sought.

5.7 Example Attacks

Let us illustrate the models by providing adversaries that attack different function families in these models.

Example 5.9 We define a family of functions $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ as follows. We let $k = Ll$ and view a k-bit key K as specifying an L row by l column matrix of bits. (To be concrete, assume the first L bits of K specify the first column of the matrix, the next L bits of K specify the second column of the matrix, and so on.) The input string $x = x[1]\ldots x[l]$ is viewed as a sequence of bits, and the value of $F(K,x)$ is the corresponding matrix vector product. That is

$$F_K(x) \;=\; \begin{bmatrix} K[1,1] & K[1,2] & \cdots & K[1,l] \\ K[2,1] & K[2,2] & \cdots & K[2,l] \\ \vdots & & & \vdots \\ K[L,1] & K[L,2] & \cdots & K[L,l] \end{bmatrix} \cdot \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[l] \end{bmatrix} \;=\; \begin{bmatrix} y[1] \\ y[2] \\ \vdots \\ y[L] \end{bmatrix}$$

where

$$\begin{aligned}
y[1] &= K[1,1]\cdot x[1] \oplus K[1,2]\cdot x[2] \oplus \ldots \oplus K[1,l]\cdot x[l] \\
y[2] &= K[2,1]\cdot x[1] \oplus K[2,2]\cdot x[2] \oplus \ldots \oplus K[2,l]\cdot x[l] \\
\vdots &= \vdots \\
y[L] &= K[L,1]\cdot x[1] \oplus K[L,2]\cdot x[2] \oplus \ldots \oplus K[L,l]\cdot x[l] \,.
\end{aligned}$$

Here the bits in the matrix are the bits in the key, and arithmatic is modulo two. The question we ask is whether F is a "secure" PRF. We claim that the answer is no. The reason is that one can design an adversary algorithm A that achieves a high advantage (close to 1) in distinguishing between the two worlds.

We observe that for any key K we have $F_K(0^l) = 0^L$. This is a weakness since a random function of l-bits to L-bits is very unlikely to return 0^L on input 0^l, and thus this fact can be the basis of a distinguishing adversary. Let us now show how the adversary works. Remember that as per our model it is given an oracle $g\colon \{0,1\}^l \to \{0,1\}^L$ and will output a bit. Our adversary D works as follows:

Adversary D^g
  Let $y \leftarrow g(0^l)$
  If $y = 0^L$ then return 1 else return 0

This adversary queries its oracle at the point 0^l, and denotes by y the l-bit string that is returned. If $y = 0^L$ it bets that g was an instance of the family F, and if $y \neq 0^L$ it bets that g was a random function. Let us now see how well this adversary does. We claim that

$$\begin{aligned} \mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,D} = 1\right] &= 1 \\ \mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{F,D} = 1\right] &= 2^{-L} . \end{aligned}$$

Why? Look at Experiment $\mathbf{Exp}^{\text{prf-1}}_{F,D}$ as defined in Definition 5.4. Here $g = F_K$ for some K. In that case it is certainly true that $g(0^l) = 0^L$ so by the code we wrote for D the latter will return 1. On the other hand look at Experiment $\mathbf{Exp}^{\text{prf-0}}_{F,D}$ as defined in Definition 5.4. Here g is a random function. As we saw in Example 5.3, the probability that $g(0^l) = 0^L$ will be 2^{-L}, and hence this is the probability that D will return 1. Now as per Definition 5.4 we subtract to get

$$\begin{aligned} \mathbf{Adv}^{\text{prf}}_{F,D} &= \mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,D} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{F,D} = 1\right] \\ &= 1 - 2^{-L} . \end{aligned}$$

Now let t be the time complexity of D. This is $O(l + L)$ plus the time for one computation of F, coming to $O(l^2L)$. The number of queries made by D is just one, and the total length of all queries is l. Thus we have

$$\begin{aligned} \mathbf{Adv}^{\text{prf}}_{F}(t,1,l) &= \max_A \{ \mathbf{Adv}^{\text{prf}}_{F,A} \} \\ &\geq \mathbf{Adv}^{\text{prf}}_{F,D} \\ &= 1 - 2^{-L} . \end{aligned}$$

The first inequality is true because the adversary D is one member of the set of adversaries A over which the maximum is taken, and hence the maximum advantage is at least that attained by D. Our conclusion is that the advantage function of F as a PRF is very high even for very low values of its resource parameter inputs, meaning F is very insecure as a PRF. ∎

Example 5.10 Suppose we are given a secure PRF $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$. We want to use F to design a PRF $G\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^{2L}$. The input length of G is the same as that of F but the output length of G is twice that of F. We suggest the following candidate construction: for every k-bit key K and every l-bit input x

$$G_K(x) = F_K(x) \| F_K(\overline{x}) .$$

Here "$\|$" denotes concatenation of strings, and $\overline{x}$ denotes the bitwise complement of the string x. We ask whether this is a "good" construction. "Good" means that under the assumption that F is a secure PRF, G should be too. However, this is not true. Regardless of the quality of F, the construct G is insecure. Let us demonstrate this.

We want to specify an adversary attacking G. Since an instance of G maps l bits to $2L$ bits, the adversary D will get an oracle for a function g that maps l bits to $2L$ bits. In World 0, g will be chosen as a random function of l bits to $2L$ bits, while in World 1, g will be set to G_K where K is a random k-bit key. The adversary must tell determine in which world it is placed. Our adversary works as follows:

Adversary D^g
  Let $y_1 \leftarrow g(1^l)$
  Let $y_2 \leftarrow g(0^l)$
  Parse y_1 as $y_1 = y_{1,1} \| y_{1,2}$ with $|y_{1,1}| = |y_{1,2}| = L$
  Parse y_2 as $y_2 = y_{2,1} \| y_{2,2}$ with $|y_{2,1}| = |y_{2,2}| = L$
  If $y_{1,1} = y_{2,2}$ then return 1 else return 0

This adversary queries its oracle at the point 1^l to get back y_1 and then queries its oracle at the point 0^l to get back y_2. Notice that 1^l is the bitwise complement of 0^l. The adversary checks whether the first half of y_1 equals the second half of y_2, and if so bets that it is in World 1. Let us now see how well this adversary does. We claim that

$$\begin{aligned} \mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{G,D} = 1\right] &= 1 \\ \mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{G,D} = 1\right] &= 2^{-L} . \end{aligned}$$

Why? Look at Experiment $\mathbf{Exp}^{\text{prf-1}}_{G,D}$ as defined in Definition 5.4. Here $g = G_K$ for some K. In that case we have

$$\begin{aligned} G_K(1^l) &= F_K(1^l) \| F_K(0^l) \\ G_K(0^l) &= F_K(0^l) \| F_K(1^l) \end{aligned}$$

by definition of the family G. Notice that the first half of $G_K(1^l)$ is the same as the second half of $G_K(0^l)$. So D will return 1. On the other hand look at Experiment $\mathbf{Exp}^{\text{prf-0}}_{G,D}$ as defined in Definition 5.4. Here g is a random function. So the values $g(1^l)$ and $g(0^l)$ are both random and independent $2L$ bit strings. What is the probability that the first half of the first string equals the second half of the second string? It is exactly the probability that two randomly chosen L-bit strings are equal, and this is 2^{-L}. So this is the probability that D will return 1. Now as per Definition 5.4 we subtract to get

$$\begin{aligned} \mathbf{Adv}^{\text{prf}}_{G,D} &= \mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{G,D} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{G,D} = 1\right] \\ &= 1 - 2^{-L} . \end{aligned}$$

Now let t be the time complexity of D. This is $O(l + L)$ plus the time for two computations of G, coming to $O(l + L)$ plus the time for four computations of F. The number of queries made by D is two, and the total length of all queries is $2l$. Thus we have

$$\begin{aligned} \mathbf{Adv}^{\text{prf}}_{G}(t, 2, 2l) &= \max_A \{ \mathbf{Adv}^{\text{prf}}_{G,A} \} \\ &\geq \mathbf{Adv}^{\text{prf}}_{G,D} \\ &= 1 - 2^{-L} . \end{aligned}$$

Our conclusion is that the advantage function of G as a PRF is very high even for very low values of its resource parameter inputs, meaning G is very insecure as a PRF. ▮

Exercise 5.11 Present a secure construction for the problem of the previous example. That is, suppose we are given a secure PRF $F: \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^L$. Design a PRF $G: \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^{2L}$ which is a secure PRF as long as F is secure. For simplicity you may assume $k = l = L$. ▮

5.8 Security against key-recovery

We have mentioned several times that security against key-recovery is not sufficient as a notion of security for a block cipher. However it is certainly necessary: if key-recovery is easy, the block cipher should be declared insecure. We have indicated that we want to adopt as notion of security for a block cipher the notion of a PRF or a PRP. If this is to be viable, it should be the case that any function family that is insecure under key-recovery is also insecure as a PRF or PRP. In this section we verify this fact. Doing so will enable us to exercise the method of reductions.

We begin by formalizing security against key-recovery. We consider an adversary that, based on input-output examples of an instance F_K of family F, tries to find K. Its advantage is defined as the probability that it succeeds in finding K. The probability is over the random choice of K, and any random choices of the adversary itself.

We give the adversary oracle access to F_K so that it can obtain input-output examples of its choice. We do not constrain the adversary with regard to the method it uses. This leads to the following definition.

Definition 5.12 Let $F\colon \mathsf{Keys}(F) \times D \to R$ be a family of functions, and let B be an algorithm that takes an oracle for a function $g\colon D \to R$, and outputs a string. We consider the experiment:

Experiment $\mathbf{Exp}^{\mathrm{kr}}_{F,B}$
  $K \stackrel{R}{\leftarrow} \mathsf{Keys}(F)$
  $K' \leftarrow B^{F_K}$
  If $K = K'$ then return 1 else return 0

The *kr-advantage* of B is defined as

$$\mathbf{Adv}^{\mathrm{kr}}_{F,B} \quad = \quad \mathbf{P}\left[\mathbf{Exp}^{\mathrm{kr}}_{F,B} = 1\right].$$

For any t, q, μ the *kr-advantage* of F is defined via

$$\mathbf{Adv}^{\mathrm{kr}}_{F}(t,q,\mu) \quad = \quad \max_{B}\{\, \mathbf{Adv}^{\mathrm{kr}}_{F,B} \,\}$$

where the maximum is over all B having time-complexity t and making at most q oracle queries, the sum of the lengths of these queries being at must μ bits. ▮

This definition has been made general enough to capture all types of key-recovery attacks. Any of the classical attacks such as exhaustive key search, differential cryptanalysis or linear cryptanalysis correspond to different, specific choices of adversary B. They fall in this framework because all have the goal of finding the key K based on some number of input-output examples of an instance F_K of the cipher. To illustrate let us see what are the implications of the classical key-recovery attacks on DES for the value of the key-recovery advantage function of DES. Assuming the exhaustive search attack is always successful based on testing two examples leads to the fact that

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathrm{DES}}(t, 2, 2\cdot 64) \;=\; 1$$

for t being about 2^{55} times the time T_{DES} for one computation of DES. On the other hand, linear cryptanalysis implies that

$$\mathbf{Adv}^{\mathrm{kr}}_{\mathrm{DES}}(t, 2^{43}, 2^{43}\cdot 64) \;=\; 1$$

for t being about $2^{43}\cdot T_{\mathrm{DES}}$. This gives us a couple of data points on the curve $\mathbf{Adv}^{\mathrm{kr}}_{\mathrm{DES}}(t, q, ql)$. For a more concrete example, let us look at the key recovery advantage of the family of Example 5.9.

Example 5.13 Let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be the family of functions from Example 5.9. We saw that its prf-advantage was very high. Let us now compute is kr-advantage. The following adversary B recovers the key. We let e_j be the l-bit binary string having a 1 in position j and zeros everywhere else. We assume that the manner in which the key K defines the matrix is that the first L bits of K form the first column of the matrix, the next L bits of K form the second column of the matrix, and so on.

Adversary B^{F_K}

Let K' be the empty string
For $j = 1, \ldots, l$ do
  $y_j \leftarrow F_K(e_j)$
  $K' \leftarrow K' \| y_j$
EndFor
Return K'

The adversary B invokes its oracle to compute the output of the function on input e_j. The result, y_j, is exactly the j-th column of the matrix associated to the key K. The matrix entries are concatenated to yield K', which is returned as the key. Since the adversary always finds the key we have

$$\mathbf{Adv}^{\mathrm{kr}}_{F,B} \; = \; 1 \, .$$

The time-complexity of this adversary is $t = O(l^2 L)$ since it makes $q = l$ calls to its oracle and each computation of F_K takes $O(lL)$ time. Thus

$$\mathbf{Adv}^{\mathrm{kr}}_{F}(t, l, l^2) \; = \; 1 \, .$$

The parameters here should still be considered small: l is 64 or 128, which is small for the number of queries. So F is insecure against key-recovery. Note however that F is less secure as a PRF than against key-recovery: its advantage function as a PRF had a value close to 1 for parameter values much smaller than those above. This leads into our next claim, which says that for any given parameter values, the kr-advantage of a family cannot be significantly more than its prf or prp-cpa advantage. ▮

Now we claim that if a block cipher is a secure PRF or PRP then it is also secure against all key-recovery attacks. Put another way, the advantage of F with respect to key recovery cannot be much larger than its advantage as a PRF.

Proposition 5.14 Let $F \colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions. Then for any t, q with $q < 2^l$ we have

$$\mathbf{Adv}^{\mathrm{kr}}_{F}(t, q, ql) \quad \leq \quad \mathbf{Adv}^{\mathrm{prf}}_{F}(t', q+1, (q+1)l) + \frac{1}{2^L} \, , \tag{5.1}$$

and furthermore, if $L = l$, then also

$$\mathbf{Adv}^{\mathrm{kr}}_{F}(t, q, ql) \quad \leq \quad \mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{F}(t', q+1, (q+1)l) + \frac{1}{2^L - q} \, , \tag{5.2}$$

where we set t' to be t plus the time for one computation of F. ▮

The proof introduces the central idea of *reductions*. We will show a transformation $B \mapsto A_B$ of any kr-adversary B into a prf-adversary A_B such that

$$\mathbf{Adv}^{\mathrm{kr}}_{F,B} \; \leq \; \mathbf{Adv}^{\mathrm{prf}}_{F,A_B} + \frac{1}{2^L}$$

and also, if the resources used by B are t, q, ql, then those used by A_B are $t', q+1, (q+1)l$. We claim that barring manipulation, this proves the first equation of the claim. Indeed, by taking maximums on both sides, we will be able to get the equation in question, as we will see later.

The problem that adversary A_B is trying to solve is to determine whether its given oracle g is a random instance of F or a random function of l bits to L-bits. The idea behind a reduction is that A_B will run B as a subroutine and use B's output to solve its own problem.

B is an algorithm that expects to be in a world where it gets an oracle F_K, and it tries to find K via queries to its oracle. For simplicity, first assume that B makes no oracle queries. Now, when A_B runs B, it produces some key K'. A_B can test K' by checking whether $F(K', x)$ agrees with $g(x)$ for some value x. If so, it bets that g was an instance of F, and if not it bets that g was random.

If B does make oracle queries, we must ask how A_B can run B at all. The oracle that B wants is not available. However, B is a piece of code, communicating with its oracle via a prescribed interface. If you start running B, at some point it will output an oracle query, say by writing this to some prescribed memory location, and stop. It awaits an answer, to be provided in another prescribed memory location. When that appears, it continues its execution. When it is done making oracle queries, it will return its output. Now when A_B runs B, it will itself supply the answers to B's oracle queries. When B stops, having made some query, A will fill in the reply in the prescribed memory location, and let B continue its execution. B does not know the difference between this "simulated" oracle and the real oracle except in so far as it can glean this from the values returned.

The value that B expects in reply to query x is $F_K(x)$. That is not what A_B gives it. Instead, it returns $g(x)$, where g is A_B's oracle. When A_B is in World 1, $g(x) = F_K(x)$, and so B is functioning as it would in its usual environment, and will return the key K with a probability equal to its kr-advantage. However when A_B is in World 0, g is a random function, and B is getting back values that bear little relation to the ones it is expecting. That does not matter. B is a piece of code that will run to completion and produce some output. When we are in World 0, we have no idea what properties this output will have. But it is some k-bit string, and A_B will test it as indicated above. It will fail the test with high probability as long as the test point x was not one that B queried, and A_B will make sure the latter is true via its choice of x. Let us now proceed to the actual proof.

Proof of Proposition 5.14: We prove the first equation and then briefly indicate how to alter the proof to prove the second equation.

We will show that given any adversary B whose resources are restricted to t, q, ql we can construct an adversary A_B, using resources $t', q+1, (q+1)l$, such that

$$\mathbf{Adv}^{\mathrm{kr}}_{F,B} \;\leq\; \mathbf{Adv}^{\mathrm{prf}}_{F,A_B} + \frac{1}{2^L} \,. \tag{5.3}$$

If this is true then we can establish Equation (5.3) as follows:

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{kr}}_{F}(t,q,\mu) &= \max_B \{ \mathbf{Adv}^{\mathrm{kr}}_{F,B} \} \\
&\leq \max_B \{ \mathbf{Adv}^{\mathrm{prf}}_{F,A_B} + 2^{-L} \} \\
&\leq \max_A \{ \mathbf{Adv}^{\mathrm{prf}}_{F,A} + 2^{-L} \} \\
&= \mathbf{Adv}^{\mathrm{prf}}_{F}(t, q+1, (q+1)l) + 2^{-L} \,.
\end{aligned}$$

The maximum, in the case of B, is taken over all adversaries whose resources are t, q, ql. In the second line, we apply Equation (5.3). In the third line, we maximize over all A whose resources are $t, q+1, (q+1)l$. The inequality on the third line is true because this set includes all adversaries of the form A_B. The last line is simply by definition. So it remains to show how to design A_B so that Equation (5.3) holds. (This is the core of the argument, namely what is called the "reduction.")

As per Definition 5.4, adversary A_B will be provided an oracle for a function g: $\{0,1\}^l \to \{0,1\}^L$, and will try to determine in which World it is. To do so, it will run adversary B as a subroutine. We provide the description followed by an explanation and analysis.

Adversary A_B^g
  $i \leftarrow 0$
  Run adversary B, replying to its oracle queries as follows
  When B makes an oracle query x do

$i \leftarrow i+1$; $x_i \leftarrow x$
$y_i \leftarrow g(x_i)$
Return y_i to B as the answer
Until B stops and outputs a key K'
Let x be an l bit string not in the set $\{x_1, \ldots, x_q\}$
$y \leftarrow g(x)$
If $F(K', x) = y$ then return 1 else return 0

As indicated in the discussion preceding the proof, A_B is running B and itself providing answers to B's oracle queries via the oracle g. When B has run to completion it returns some k-bit string K', which A_B tests by checking whether $F(K'x)$ agrees with $g(x)$. Here x is a value different from any that B queried, and it is to ensure that such a value can be found that we require $q < 2^l$ in the statement of the Proposition. Now we claim that

$$\begin{aligned} \mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,A_B} = 1\right] &\geq \mathbf{Adv}^{\text{kr}}_{F,B} \\ \mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{F,A_B} = 1\right] &= 2^{-L} . \end{aligned}$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 5.4, we get

$$\begin{aligned} \mathbf{Adv}^{\text{prf}}_{F,A_B} &= \mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,A_B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{F,A_B} = 1\right] \\ &\geq \mathbf{Adv}^{\text{kr}}_{F,B} - 2^{-L} . \end{aligned}$$

Re-arranging terms gives us Equation (5.3). It remains to justify Equations (5.4) and (5.4).

Equation (5.4) is true because in $\mathbf{Exp}^{\text{prf-1}}_{F,A_B}$ the oracle g is F_K for some K, which is the oracle that B expects, and thus B functions as it does in $\mathbf{Exp}^{\text{kr}}_{F,B}$. If B is successful, meaning the key K' it outputs equals K, then certainly A_B returns 1. (It is possible that A_B might return 1 even though B was not successful. This would happen if $K' \neq K$ but $F(K', x) = F(K, x)$. It is for this reason that $\mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,A_B} = 1\right]$ is greater than or equal to $\mathbf{Adv}^{\text{kr}}_{F,B}$ rather than merely equal to it.) Equation (5.4) is true because in $\mathbf{Exp}^{\text{prf-0}}_{F,A_B}$ the function g is random, and since x was never queried by B, the value $g(x)$ is unpredictable to B. Imagine that $g(x)$ is chosen only when x is queried to g. At that point, K', and thus $F(K', x)$, is already defined. So $g(x)$ has a 2^{-L} chance of hitting this fixed point. Note this is true regardless of how hard B tries to make $F(K', x)$ be the same as $g(x)$.

For the proof of Equation (5.2) we seek a reduction $B \mapsto A_B$ with the property that

$$\mathbf{Adv}^{\text{kr}}_{F,B} \leq \mathbf{Adv}^{\text{prp-cpa}}_{F,A_B} + \frac{1}{2^L - q} . \tag{5.4}$$

The reduction is identical to the one given above, meaning the adversary A_B is the same. For the analysis we see that

$$\begin{aligned} \mathbf{P}\left[\mathbf{Exp}^{\text{prp-cpa-1}}_{F,A_B} = 1\right] &= \mathbf{Adv}^{\text{kr}}_{F,B} \\ \mathbf{P}\left[\mathbf{Exp}^{\text{prp-cpa-0}}_{F,A_B} = 1\right] &\leq \frac{1}{2^L - q} . \end{aligned}$$

Subtracting yields

$$\begin{aligned} \mathbf{Adv}^{\text{prp-cpa}}_{F,A_B} &= \mathbf{P}\left[\mathbf{Exp}^{\text{prp-cpa-1}}_{F,A_B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{prp-cpa-0}}_{F,A_B} = 1\right] \\ &\geq \mathbf{Adv}^{\text{kr}}_{F,B} - \frac{1}{2^L - q} \end{aligned}$$

and re-arranging terms gives us Equation (5.4). The first equation above is true for the same reason as before. The second equation is true because in World 0 the map g is now a random permutation of l-bits to l-bits. So $g(x)$ assumes any random value except the values $y_1, \ldots, y_q$, meaning there are $2^L - q$ things it could be. (Remember $L = l$ in this case.) ∎

The following example illustrates that the converse of the above claim is far from true. The kr-advantage of a family can be significantly smaller than its prf or prp-cpa advantage, meaning that a family might be very secure against key recovery yet very insecure as a prf or prp, and thus not useful for protocol design.

Example 5.15 Define the block cipher $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ by $E_K(x) = x$ for all k-bit keys K and all l-bit inputs x. We claim that it is very secure against key-recovery but very insecure as a PRP under CPA. More precisely, we claim that for all values of t, q, however high,

$$\mathbf{Adv}_E^{\mathrm{kr}}(t, q, ql) \;=\; 2^{-k} \,,$$

and on the other hand

$$\mathbf{Adv}_E^{\mathrm{prp\text{-}cpa}}(t, 1, l) \;\geq\; 1 - 2^{-l}$$

for $t = O(l)$. In other words, given an oracle for E_K, you may make as many queries as you want, and spend as much time as you like, before outputting your guess as to the value of K, yet your chance of getting it right is only 2^{-k}. On the other hand, using only a single query to a given oracle $g\colon \{0,1\}^l \to \{0,1\}^l$, and very little time, you can tell almost with certainity whether g is an instance of E or is a random function of l bits to l bits. Why are these claims true? Since E_K does not depend on K, an adversary with oracle E_K gets no information about K by querying it, and hence its guess can as to the value of K can be correct only with probability 2^{-k}. On the other hand, an adversary can test whether $g(0^l) = 0^l$, and by returning 1 if and only if this is true, attain a prp-advantage of $1 - 2^{-l}$. ∎

5.9 The birthday attack

Suppose $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ is a family of permutations, meaning a block cipher. If we are given an oracle $g\colon \{0,1\}^l \to \{0,1\}^l$ which is either an instance of E or a random function, there is a simple test to determine which of these it is. Query the oracle at distince points $x_1, x_2, \ldots, x_q$, and get back values $y_1, y_2, \ldots, y_q$. You know that if g were a permutation, the values $y_1, y_2, \ldots, y_q$ must be distinct. If g was a random function, they may or may not be distinct. So, if they are distinct, bet on a permutation.

Surprisingly, this is pretty good distinguisher, as we will argue below. Roughly, it takes $q = \sqrt{2^l}$ queries to get an advantage that is quite close to 1. The reason is the birthday paradox. If you are not familiar with this, you may want to look at Section A.1, and then come back to the following.

This tells us that an instance of a block cipher can be distinguished from a random function based on seeing a number of input-output examples which is approximately $2^{l/2}$. This has important consequences for the security of block cipher based protocols.

Proposition 5.16 Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a family of permutations. Suppose q satisfies $2 \leq q \leq 2^{(l+1)/2}$. Then

$$\mathbf{Adv}_E^{\mathrm{prf}}(t, q, ql) \quad\geq\quad 0.3 \cdot \frac{q(q-1)}{2^l} \,,$$

where and t is the time for q computations of E, plus $O(ql)$. ∎

Proof of Proposition 5.16: The birthday attack is implemented by an adversary D who, given an oracle $g\colon \{0,1\}^l \to \{0,1\}^l$, works like this:

Adversary D^g
For $i = 1, \ldots, q$ do
  Let x_i be the i-th l-bit string in lexicographic order
  $y_i \leftarrow g(x_i)$
End For
If $y_1, \ldots, y_q$ are all distinct then return 1, else return 0

We claim that

$$\mathbf{Adv}^{\mathrm{prf}}_{E,D} \quad \geq \quad 0.3 \cdot \frac{q(q-1)}{2^l} ,$$

from which the Proposition follows. Let us now justify this lower bound. Letting $N = 2^l$, we claim that

$$\mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf}\text{-}1}_{E,D} = 1\right] \quad = \quad 1 \tag{5.5}$$

$$\mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf}\text{-}0}_{E,D} = 1\right] \quad = \quad 1 - C(N,q) \; . \tag{5.6}$$

Here $C(N,q)$, as defined in Section A.1, is the probability that some bin gets two or more balls in the experiment of randomly throwing q balls into N bins. We will justify these claims shortly, but first let us use them to conclude. Subtracting, we get

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prf}}_{E,D} &= \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf}\text{-}1}_{E,D} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf}\text{-}0}_{E,D} = 1\right] \\
&= 1 - [1 - C(N,q)] \\
&= C(N,q) \\
&\geq 0.3 \cdot \frac{q(q-1)}{2^l} \; .
\end{aligned}$$

The last line is by Proposition A.1. It remains to justify Equations (5.5) and (5.6).

Equation (5.5) is clear because in World 1, $g = E_K$, and since E is a family of permutations, g is a permutation, and thus $y_1, \ldots, y_q$ are all distinct. Now, suppose D is in World 0, so that g is a random function of l bits to l bits. What is the probability that $y_1, \ldots, y_q$ are all distinct? Since g is a random function and $x_1, \ldots, x_q$ are distinct, $y_1, \ldots, y_q$ are random, independently distributed values in $\{0,1\}^l$. Thus we are looking at the birthday problem. We are throwing q balls into $N = 2^l$ bins and asking what is the probability of there being no collisions, meaning no bin contains two or more balls. This is $1 - C(N,q)$, justifying Equation (5.6). ▮

5.10 PRFs versus PRPs

When we come to analyses of block cipher based constructions, we will find a curious dichotomy. Analyses are considerably simpler and more natural assuming the block cipher is a PRF. Yet, PRPs are what most naturally model block ciphers. To bridge the gap, we relate the prf and prp-cpa advantage functions of a given block cipher. The following says, roughly, that the birthday attack is the best possible one. A particular family of permutations E may have prf-advantage that is greater than its prp-advantage, but only by an amount of $q(q-1)/2^{l+1}$, the collision probability term in the birthday attack.

Proposition 5.17 Suppose E: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ is a family of permutations. Then

$$\mathbf{Adv}^{\mathrm{prf}}_{E}(t,q,ql) \quad \leq \quad \frac{q(q-1)}{2^{l+1}} + \mathbf{Adv}^{\mathrm{prp}\text{-}\mathrm{cpa}}_{E}(t,q,ql)$$

for any t, q. ▮

The proof is again by reduction, but a very simple one. A given prf-adversary A is mapped to prp-adversary A, meaning the adversary is unchanged. Accordingly, the following does not explicitly talk of reductions.

Proof: Let A be an adversary that takes an oracle for a function g: $\{0,1\}^l \to \{0,1\}^l$. Then we claim that

$$\mathbf{Adv}^{\mathrm{prf}}_{E,A} \quad \leq \quad \mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{E,A} + \frac{q(q-1)}{2^{l+1}}\,, \tag{5.7}$$

where q is the number of oracle queries made by A. The Proposition follows by taking maximums, so it remains to prove Equation (5.7).

Let B denote the adversary that first runs A to obtain an output bit b and then returns $\bar{b}$, the complement of b. Then

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{prf}}_{E,A} &= \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}1}}_{E,A} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,A} = 1\right] \\
&= \left(1 - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}1}}_{E,B} = 1\right]\right) - \left(1 - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B} = 1\right]\right) \\
&= \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}1}}_{E,B} = 1\right] \\
&= \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}1}}_{E,B} = 1\right] \\
&= \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}0}}_{E,B} = 1\right] \\
&\quad + \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}0}}_{E,B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}1}}_{E,B} = 1\right] \\
&= \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}0}}_{E,B} = 1\right] + \mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{E,A}\,.
\end{aligned}$$

So it suffices to show that

$$\mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}0}}_{E,B} = 1\right] \quad \leq \quad \frac{q(q-1)}{2^{l+1}}\,. \tag{5.8}$$

Let $\mathbf{P}[\cdot]$ denote the probability in Experiment $\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B}$, and let g denote the oracle in that experiment. Assume without loss of generality that all oracle queries of A —they are the same as those of B— are distinct. Let D denote the event that all the answers are distinct, and let $\overline{D}$ denote the complement of event D. Then

$$\begin{aligned}
\mathbf{P}\left[\mathbf{Exp}^{\mathrm{prf\text{-}0}}_{E,B} = 1\right] &= \mathbf{P}\left[B^g = 1\right] \\
&= \mathbf{P}\left[B^g = 1 \mid D\right] \cdot \mathbf{P}\left[D\right] + \mathbf{P}\left[B^g = 1 \mid \overline{D}\right] \cdot \mathbf{P}\left[\overline{D}\right] \\
&\leq \mathbf{P}\left[B^g = 1 \mid D\right] + \mathbf{P}\left[\overline{D}\right] \\
&= \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}0}}_{E,B} = 1\right] + \mathbf{P}\left[\overline{D}\right] \\
&\leq \mathbf{P}\left[\mathbf{Exp}^{\mathrm{prp\text{-}cpa\text{-}0}}_{E,B} = 1\right] + \frac{q(q-1)}{2^{l+1}}\,.
\end{aligned}$$

In the last step we used Proposition A.1. Re-arranging terms gives us Equation (5.8) and concludes the proof. ▮

5.11 Constructions of PRF families

Where can we find PRFs? There are a variety of ways. We can build them out of pseudorandom bit generators or one-way functions, a conservative but to date inefficient approach. There are more efficient constructions whose security is based on the presumed hardness of specific number-theoretic problems. Finally in practice we might be willing to assume that block ciphers like the AES have the property, as discussed above.

The notion of a pseudorandom bit generator (PRBG) was discussed in Chapter 3. Recall it is a polynomial time computable function G which takes a k bit seed and produces a $p(k) > k$ bit sequence of bits that look random to any efficient test.

The first construction of PRF families was from PRBGs which are length doubling: the output length is twice the input length.

Theorem 5.18 [92] Given a length-doubling pseudorandom bit generator we can construct a sequence of families $\mathcal{F}$ which is a PRF. ▮

The construction, called the binary tree construction, is like this. The function G induces a tree of functions G_z in the following way:

- Define $G_0(x) \circ G_1(x) = G(x)$ where $k = |G_0(x)| = |G_1(x)|$.
- Define $G_{z\circ 0}(x) \circ G_{z\circ 1}(x) = G_z(x)$ where $k = |G_{z\circ 0}| = |G_{z\circ 1}|$.

Then $f_i(x)$ is defined in terms of the binary tree induced by G as follows: $\forall x f_i(x) = G_x(i)$. We now let $\mathcal{F} = \{F^k\}_{k\geq 1}$ where F^k is $\{f_i : \{0,1\}^k \to \{0,1\}^k | |i| = k\}$. It is shown in [92] that this is secure.

Another construction based on a primitive called synthesizers was given by Naor and Reingold [143]. This yields a PRBG based construction which is more parallelizable than the binary tree based one.

We saw before that we can construct PRBGs from one-way functions [111, 107]. It follows from the above that we can build (infinite) PRF families from one-way functions. Furthermore, one can see that given any pseudorandom function family one can construct a one-way function [110]. Thus we have the following.

Theorem 5.19 There exists a sequence of families which is a PRF if and only if there exist one-way functions. ▮

This is quite a strong statement. One-way functions are a seemingly weak primitive, a priori quite unrelated to PRFs. Yet the one can be transformed into the other. Unfortunately the construction is not efficient enough to be practical.

Naor and Reingold have suggested a construction of a sequence of families $\mathcal{F} = \{F^n\}_{n\geq 1}$ which they prove is a PRF assuming that the DDH (Decisional Diffie-Hellman) problem is hard [144]. In this construct, evaluation of particular function from F^n on an $l(n)$-bit input requires $l(n)$ modular multiplications and one modular exponentiation, over an underlying group.

5.11.1 Extending the domain size

Suppose we start with a finite PRF family with input length l. Often we need to extend the functions to a larger domain in such a way that they are still PRFs on the larger domain. (One of the important reasons for this is that pseudorandom functions make good message authentication codes. See Theorem 8.6.) There are various ways to do this. One is to use the CBC (cipher block chaining) construction.

Here we are given a finite PRF family with input length l and output length also l. An integer m is fixed and we want to construct a family of functions mapping $\{0,1\}^{lm}$ to $\{0,1\}^l$. The construction is like this. The keys for the new family are the same as for the old one. Let $f = F_K$ be the l bit to l bit function indicated by key K. The new function, given $M_1 \ldots M_m$, does the following:

- Set $Y_0 = 0^l$
- Set $Y_1 = f(M_1 \oplus Y_0)$

- Set $Y_2 = f(M_2 \oplus Y_1)$
- $\vdots$
- Set $Y_m = f(M_m \oplus Y_{m-1})$
- Output Y_m.

Let $F^{(m)}$ denote the family of functions in which the function indexed by K maps $\{0,1\}^{ml}$ to $\{0,1\}^l$ and is given by the CBC using $f = F_K$.

Theorem 5.20 [12] Let $l, m \geq 1$ and $q, t \geq 0$ be integers. Let $F\colon \mathsf{Keys}F \times \{0,1\}^l \to \{0,1\}^l$ be a family of functions. Then

$$\mathbf{Adv}^{\mathrm{prf}}_{F^{(m)}}(q,t) \leq \mathbf{Adv}^{\mathrm{prf}}_{F}(q',t') + 1.5 \cdot \frac{q^2m^2}{2^l} \tag{5.9}$$

$$\leq \mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{F}(q',t') + \frac{q^2m^2}{2^{l-1}} \tag{5.10}$$

where $q' = mq$ and $t' = t + O(mql)$. ▮

We stress that the input must be of *exactly* nl bits, not at most nl bits. Else the construction is not secure.

There are also other constructions. For example, the cascade construction of [19]. Also, based on the ideas of XOR MACs [11], a construction called the Protected Counter Sum was given in [30]. Different constructions have different properties in terms of security and efficiency.

Similarly (or simultaneously) we may want to extend the output length. It turns out this is easier, so we won't discuss it in detail.

5.12 Some applications of PRFs

5.12.1 Cryptographically Strong Hashing

Let P_1, P_2 be polynomials so that $\forall x, P_1(x) > P_2(x)$. Define $F^{P_1,P_2} = \{f : \{0,1\}^{P_1(k)} \to \{0,1\}^{P_1(k)}\}$. Then we wish to hash names into address where $|\mathrm{Name}| = P_1(k)$ and $|\mathrm{Address}| = P_2(k)$. We may use pseudo-random functions to hash these names so that $\mathrm{Address} = f_i(\mathrm{Name})$.

Claim 5.21 If there exist one way functions, then for all polynomials P, and for all integers k sufficiently large, the previous hashing algorithm admits no more than $O(\frac{1}{2^{\sqrt{\mathrm{Address}}}}) + \frac{1}{P(k)}$ collisions even if, after fixing the scheme, the names are chosen by an adversary with access to previous $(\mathrm{Name}, \mathrm{Address})$ pairs. ▮

5.12.2 Prediction

A *prediction test* $T(1^k)$

1. queries an oracle for $f \in F_k$, discovering $(x_1, f(x_1)), \ldots, (x_l, f(x_l))$,
2. outputs an "exam", x, and
3. is given y so that with probability $\frac{1}{2}$, $y = f(x)$ (otherwise, y is chosen randomly in $\{0,1\}^{|f(x)|} - \{f(x)\}$).
4. outputs 1 if it guesses that $y = f(x)$, 0 otherwise.

F is said to *pass the prediction test* T if $\forall Q \in \mathbf{Q}[x], \exists k_0, \forall k > k_0$,

$$Pr[T(1^k) \text{ guesses correctly given } y \text{ in step } 3] < \frac{1}{2} + \frac{1}{Q(k)}$$

The above pseudo-random functions then pass all prediction tests (assuming there exist one way functions).

5.12.3 Learning

Define a *concept space* S and a *concept* $C \subseteq S$. A learner is exposed to a number of pairs $(e_i, \pm_i)$ where $e_i \in S$ and $\pm_i = + \Leftrightarrow e_i \in C$. The learner is then requested to determine if a given $e \in S$ is an element of C.

The above pseudo-random function show that if there exist one way functions, then there exist concepts not learnable in polynomial time. (The concept in this case would be $\{x, f(x)\} \subseteq \{x, y\}$.)

5.12.4 Identify Friend or Foe

Consider the situation of two forces of war planes fighting an air battle. Each plane wishes to identify potential targets as friendly or enemy. This can be done using pseudo-random functions in the following way:

1. All the planes on a certain force know i.
2. To identify a target, a plane sends the target a random number r and expects to receive back $f_i(r)$ if the target is a friend.

Then, even though the enemy planes see many pairs of the form $(x, f(x))$, they cannot compute $f(y)$ for y they have not yet seen.

5.12.5 Private-Key Encryption

Let A and B privately agree on i. Then to encrypt message m, A produces a random string r and sends $(r, f_i(r) \oplus m)$. B can compute $f_i(r)$ and so compute $f_i(r) \oplus m \oplus f_i(r) = m$. Assuming that there exist one way functions, such a system is secure again chosen ciphertext attack, that is, secure even if the adversary can compute $(r, f_i(r))$ for a collection of r's. See Chapter 6 for more on this.

5.13 Historical Notes

The basic definition of a pseudorandom functions is due to Goldreich, Goldwasser and Micali [92]. In particular these authors introduced the important notion of distinguishers. The notion of a pseudorandom permutation is due to Luby and Rackoff [133]. These works are in the complexity-theoretic or "asymptotic" setting, where one considers sequences of families rather than just one family, and defines security as per Definition 5.8. The approach used for the bulk of the current chapter, motivated by the desire to model block ciphers, is called "concrete security," and originates with [13]. Definitions 5.4 and 5.5 are from [13], as are Propositions 5.16 and 5.17.

5.14 Exercises and Problems

Problem 5.22 Let $a[i]$ denote the i-th bit of a binary string i, where $1 \le i \le |a|$. The *inner product* of n-bit binary strings a, b is

$$\langle\, a, b\,\rangle \;=\; a[1]b[1] \oplus a[2]b[2] \oplus \cdots \oplus a[n]b[n] \;.$$

A family of functions $F: \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^L$ is said to be *inner-product preserving* if for every $K \in \{0,1\}^k$ and every distinct $x_1, x_2 \in \{0,1\}^l - \{0^l\}$ we have

$$\langle\, F(K,x_1), F(K,x_2) \,\rangle \;=\; \langle\, x_1, x_2 \,\rangle\ .$$

Prove that if F is inner-product preserving then

$$\mathbf{Adv}^{\mathrm{prf}}_{F}(t,2,2l) \geq \frac{1}{2}\cdot\left(1+\frac{1}{2^L}\right)$$

for $t = q \cdot T_F + O(\mu)$, where T_F denotes the time to perform one computation of F. Explain in a sentence why this shows that if F is inner-product preserving then F is not a secure PRF. ▮

Problem 5.23 Let $E: \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^l$ be a block cipher. The *two-fold cascade* of E is the block cipher $E^{(2)}: \{0,1\}^{2k} \times \{0,1\}^l \rightarrow \{0,1\}^l$ defined by

$$E^{(2)}(K_1 \| K_2, x) = E(K_1, E(K_2, x))$$

for all $K_1, K_2 \in \{0,1\}^k$ and all $x \in \{0,1\}^l$. (Here "$\|$" stands for concatenation of strings.) Prove that

$$\mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{E^{(2)}}(t,q,lq) \;\leq\; \mathbf{Adv}^{\mathrm{prp\text{-}cpa}}_{E}(t,q,lq)$$

for all t, q. Explain in a sentence why this shows that if E is a secure PRP then so is $E^{(2)}$. ▮

Chapter 6

Private-key encryption

The *private-key setting*, also called the *symmetric setting*, considers two parties who share a key and will use this key to imbue communicated data with various security attributes. (How they came into possession of a shared key is not part of the problem considered here, and will be addressed later.) The main security goals are privacy and authenticity of the communicated data. A symmetric encryption scheme (also called a private-key encryption scheme) enables parties in possession of a shared secret key to achieve the goal of data privacy. This is the canonical goal of cryptography.

6.1 Symmetric encryption schemes

A symmetric encryption scheme specifies an encryption algorithm, which tells the sender how to process her data as a function of the key to produce the object that is actually transmitted. It also specifies a decapsulation algorithm which tells the receiver how to retrieve the original data from the transmission while possibly also performing some verification. Finally, there is a key generation algorithm, which produces a key that the parties need to share. The formal description follows.

Definition 6.1 A *symmetric encryption scheme* $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms, as follows:

- The *key generation* algorithm $\mathcal{K}$ is a randomized algorithm that returns a string K. We let $\mathsf{Keys}(\mathcal{SE})$ denote the set of all strings that have non-zero probability of being output by $\mathcal{K}$. The members of this set are called *keys*. We write $K \stackrel{R}{\leftarrow} \mathcal{K}$ for the operation of executing $\mathcal{K}$ and letting K denote the key returned.
- The *encryption* algorithm $\mathcal{E}$ takes the key $K \in \mathsf{Keys}(\mathcal{SE})$ and a *plaintext* $M \in \{0,1\}^*$ to return a *ciphertext* $C \in \{0,1\}^* \cup \{\perp\}$. This algorithm might be randomized or stateful. We write $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$.
- The deterministic *decryption* algorithm $\mathcal{D}$ takes a key $K \in \mathsf{Keys}(\mathcal{SE})$ and a ciphertext $C \in \{0,1\}^*$ to return some $M \in \{0,1\}^* \cup \{\perp\}$. We write $M \leftarrow \mathcal{D}_K(C)$.

We require that for any key $K \in \mathsf{Keys}(\mathcal{SE})$ and any message $M \in \{0,1\}^*$, if $\mathcal{E}_K(M)$ returns a ciphertext $C \neq \perp$ then $\mathcal{D}_K(C) = M$. ∎

The key generation algorithm, as the definition indicates, is randomized. It takes no inputs. When it is run, it flips coins internally and uses these to select a key K. Typically, the key is just a random string of some length, in which case this length is called the *key length* of the scheme. When two parties want to use the scheme, it is assumed they are in possession of K generated via $\mathcal{K}$. How they came into joint possession of this key K in such a way that the adversary did not get to know K is not our concern here; it is an assumption we make.

Once in possession of a shared key, the parties can encrypt data for transmission. To encrypt plaintext M, the sender (or encrypter) runs the encryption algorithm with key K and input M to get back a string we call the ciphertext.

The encryption algorithm may be either randomized or stateful. If randomized, it flips coins and uses those to compute its output on a given input K, M. Each time the algorithm is invoked, it flips coins anew, and in particular invoking it twice on the same inpupts may not yield the same response both times. If the encryption algorithm is stateful, its operation depends on a global variable such as a counter, which is updated upon each invocation of the encryption algorithm. Thus, the encrypter maintains a counter that is initialized in some pre-specified way. When the encryption algorithm is invoked on inputs K, M, it computes a ciphertext based on K, M and the current counter value. It then updates the counter, and the new counter value is stored. (The receiver does not need to maintain a counter, and in particular decryption does not require access to any global variable or call for any synchronization between parties.)

When there is no such counter or global variable, the scheme is *stateless*. In stateful schemes the encryption algorithm typically does not flip coins internally. (It is still OK to call it a randomized algorithm. It just happens to not make use of its given random number generator.) In stateless schemes, randomization is essential to security, as we will see.

Once a ciphertext C is computed, it is transmitted to the receiver. The latter can recover the message by running the decryption algorithm with the same key used to create the ciphertext, namely via $M \leftarrow \mathcal{D}_K(C)$. The decryption algorithm is neither randomized nor stateful.

Many encryption schemes restrict the set of strings that they are willing to encrypt. (For example, perhaps the algorithm can only encrypt plaintexts of length a positive multiple of some block length l, and can only encrypt plaintext of length up to so maximum length.) These kinds of restrictions are captured by having the encryption algorithm return the special symbol $\perp$ when fed a message not meeting the required restriction. In a stateless scheme, there is typically a set of strings, called the *plaintext space*, such that $\mathcal{E}_K(M) \neq \perp$ for all K and all M in the plaintext space. In a stateful scheme, whether or not $\mathcal{E}_K(M)$ returns $\perp$ depends not only on M but also possibly on the value of the state variable. For example, when a counter is being used, it is typical that there is a limit to the number of encryptions performed, and when the counter reaches a certain value the encryption algorithm returns $\perp$ no matter what message it is fed.

6.2 Some encryption schemes

Let us begin with a few examples.

Scheme 6.2 The one-time-pad encryption scheme (also called the Vernam cipher) $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is stateful and deterministic. The key generation algorithm simply returns a random k-bit string K, where the key-length k is a parameter of the scheme, so that the key space is $\mathsf{Keys}(\mathcal{SE}) = \{0,1\}^k$. The encryptor maintains a counter *ctr* which is initially zero. The encryption and decryption algorithms operate as follows:

Algorithm $\mathcal{E}_K(M)$	Algorithm $\mathcal{D}_K((C, ctr))$
Let $n = \|M\|$	Let $n \leftarrow \|M\|$
If $ctr + n > k$ then return $\perp$	For $i = 1$ to n do
For $i = 1$ to n do	$M[i] \leftarrow K[ctr + i] \oplus C[i]$
$C[i] \leftarrow K[ctr + i] \oplus M[i]$	EndFor
EndFor	$M \leftarrow M[1] \ldots M[n]$
$ctr \leftarrow ctr + n$	Return M
$C \leftarrow C[1] \ldots C[n]$	
Return (ctr, C)	

Here $X[i]$ dennotes the i-th bit of a binary string X. The encryption algorithm XORs the message bits with key bits, starting with the key bit indicated by the current counter value. The counter is then incremented by the length of the message. Key bits are not reused, and thus if not enough key bits are available to encrypt a message, the encryption algorithm returns $\perp$. Note that the ciphertext returned includes the value of the counter. This is in order to enable decryption. (Recall that the decryption algorithm, as per Definition 6.1, must be stateless and deterministic, so we do not want it to have to maintain a counter as well.) ▮

The following schemes rely either on a family of permutations (ie. a block cipher) or a family of functions. It is convenient if the length of the message to be encrypted is a positive multiple of a block length associated to the family. Accordingly, the encryption algorithm returns $\perp$ if this is not the case. In practice, however, one would first pad the message appropriately so that the padded message always had length a positive multiple of the block length, and apply the encryption algorithm to the padded message. The padding function should be injective and easily invertible.

Scheme 6.3 Let $E: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher. Operating it in ECB (Electronic Code Book) mode yields a stateless symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm simply returns a random key for the block cipher, meaning it picks a random k-bit string key and returns it, so that the key space is $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathcal{E}_K(M)$	Algorithm $\mathcal{D}_K(C)$
If $\lvert M\rvert < l$ then return $\perp$	If $\lvert C\rvert < l$ then return $\perp$
If $\lvert M\rvert \bmod l \neq 0$ then return $\perp$	If $\lvert C\rvert \bmod l \neq 0$ then return $\perp$
Parse M as $M[1] \ldots M[n]$	Parse C as $C = C[1] \ldots C[n]$
For $i = 1, \ldots, n$ do	For $i = 1, \ldots, n$ do
$C[i] \leftarrow E_K(M[i])$	$M[i] \leftarrow E_K^{-1}(M[i])$
EndFor	EndFor
$C \leftarrow C[1] \ldots C[n]$	$M \leftarrow M[1] \ldots M[n]$
Return C	Return M

Parsing M means that we divide it into l-bit blocks and let $M[i]$ denote the i-th such block, and similarly for C. Notice that here the encryption algorithm did not make any random choices. That does not mean we are not allowed to call it a randomized algorithm; it is simply a randomized algorithm that happened to choose to not make random choices. ▮

Cipher-block chaining (CBC) is the most popular mode, used pervasively in practice.

Scheme 6.4 Let $E: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher. Operating it in CBC mode with random IV yields a stateless symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm simply returns a random key for the block cipher, meaning it picks a random k-bit string key and returns it, so that the key space is $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathcal{E}_K(M)$	Algorithm $\mathcal{D}_K(C)$
If $\lvert M\rvert < l$ then return $\perp$	If $\lvert C\rvert < 2l$ then return $\perp$
If $\lvert M\rvert \bmod l \neq 0$ then return $\perp$	If $\lvert C\rvert \bmod l \neq 0$ then return $\perp$
Parse M as $M[1] \ldots M[n]$	Parse C as $C[0]C[1] \ldots C[n]$
$C[0] \stackrel{R}{\leftarrow} \{0,1\}^l$	For $i = 1, \ldots, n$ do
For $i = 1, \ldots, n$ do	$M[i] \leftarrow E_K^{-1}(C[i]) \oplus C[i-1]$
$C[i] \leftarrow F_K(C[i-1] \oplus M[i])$	EndFor
EndFor	$M \leftarrow M[1] \ldots M[n]$
$C \leftarrow C[0]C[1] \ldots C[n]$	Return M
Return C	

Parsing M means that we divide it into l-bit blocks and let $M[i]$ denote the i-th such block. In parsing C we also divide it into l-bit blocks, but this time the blocks are numbered starting at 0. The IV is $C[0]$, which is chosen at random by the encryption algorithm. This choice is made independently each time the algorithm is invoked. ▮

For the following schemes it is useful to introduce some notation. If $l \geq 1$ and i are integers with $0 \leq i \leq 2^l - 1$ then we let $\mathsf{NtS}_l(i)$ (read "number to string") denote the l-bit string which is the binary representation of integer i. If s is a string we let $\mathsf{StN}(s)$ (read "string to number") denote the non-negative integer whose binary representation is s.

The CTR (counter) modes that follow are not much used, to the best of our knowledge, but perhaps wrongly so. We will see later that they have good security properties. In contrast to CBC, the encryption and decryption procedures are parallelizable, which can be exploited to speed up these processes in the presence of hardware support. There are two variants of the mode, one random and the other stateful, and, as we will see later, their security properties are different.

Scheme 6.5 Let $F: \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^L$ be a family of functions. (Not necessarily a family of permutations.) Operating it in CTR mode with starting point chosen at random anew for each message yields a stateless symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, which we call R-CTR mode or the R-CTR symmetric encryption scheme. The key generation algorithm simply returns a random key for F, meaning it picks a random k-bit string key and returns it, so that the key space is $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

Algorithm $\mathcal{E}_K(M)$	Algorithm $\mathcal{D}_K(C)$
If $\lvert M\rvert < L$ then return $\perp$	If $\lvert C\rvert < l + L$ then return $\perp$
If $\lvert M\rvert \bmod L \neq 0$ then return $\perp$	If $(\lvert C\rvert - l) \bmod L \neq 0$ then return $\perp$
Parse M as $M[1] \ldots M[n]$	Let $C[0]$ be the first l bits of C
$R \stackrel{R}{\leftarrow} \{0, 1, \ldots, 2^l - 1\}$	Parse the rest of C as $C[1] \ldots C[n]$
For $i = 1, \ldots, n$ do	$R \leftarrow \mathsf{StN}(C[0])$
$\quad C[i] \leftarrow F_K(\mathsf{NtS}_l(R + i)) \oplus M[i]$	For $i = 1, \ldots, n$ do
EndFor	$\quad M[i] \leftarrow F_K(\mathsf{NtS}_l(R + i)) \oplus C[i]$
$C[0] \leftarrow \mathsf{NtS}_l(R)$	EndFor
$C \leftarrow C[0]C[1] \ldots C[n]$	$M \leftarrow M[1] \ldots M[n]$
Return C	Return M

Parsing M means that we divide it into L-bit (not l-bit!) blocks and let $M[i]$ denote the i-th such block. For C the decryption algorithm first chops off the first l bits, and then divides the rest of the string into L-bit blocks. The random value chosen by the encryption algorithm is an integer in the range $0, \ldots, 2^l - 1$. It is used to define a sequence of values on which F_K is applied to produce a "pseudo one-time pad" to which the data is XORed. The random value is included in the ciphertext in order to enable decryption. ▮

Scheme 6.6 Let $F: \{0,1\}^k \times \{0,1\}^l \rightarrow \{0,1\}^L$ be a family of functions. (Not necessarily a family of permutations.) Operating it in CTR mode with counter yields a stateful symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, which we call C-CTR mode or C-CTR symmetric encryption scheme. The key generation algorithm simply returns a random key for F, meaning it picks a random k-bit string key and returns it, so that the key space is $\{0,1\}^k$. The encryptor maintains a counter *ctr* which is initially zero. The encryption and decryption algorithms are as follows:

Algorithm $\mathcal{E}_K(M)$
If $\lvert M\rvert < L$ then return $\perp$
If $\lvert M\rvert \bmod L \neq 0$ then return $\perp$
Parse M as $M[1]\ldots M[n]$
If $ctr + n \geq 2^l$ then return $\perp$
For $i = 1,\ldots,n$ do
$C[i] \leftarrow F_K(\mathsf{NtS}_l(ctr+i))\oplus M[i]$
EndFor
$C[0] \leftarrow \mathsf{NtS}_l(ctr)$
$C \leftarrow C[0]C[1]\ldots C[n]$
$ctr \leftarrow ctr + n$
Return C
	Algorithm $\mathcal{D}_K(C)$
	If $\lvert C\rvert < l + L$ then return $\perp$
	If $(\lvert C\rvert - l) \bmod L \neq 0$ then return $\perp$
	Let $C[0]$ be the first l bits of C
	Parse the rest of C as $C[1]\ldots C[n]$
	$ctr \leftarrow \mathsf{StN}(C[0])$
	For $i = 1,\ldots,n$ do
	$M[i] \leftarrow F_K(\mathsf{NtS}_l(ctr+i))\oplus C[i]$
	EndFor
	$M \leftarrow M[1]\ldots M[n]$
	Return M

Parsing M means that we divide it into L-bit (not l-bit!) blocks and let $M[i]$ denote the i-th such block. For C the decryption algorithm first chops off the first l bits, and then divides the rest of the string into L-bit blocks. The counter is not allowed to wrap around: the encryption algorithm returns $\perp$ if this would happen. The counter is included in the ciphertext in order to enable decryption. The encryption algorithm updates the counter upon each invocation, and begins with this updated value the next time it is invoked. ▮

We will return to the security of these schemes after we have developed the appropriate notions.

6.3 Issues in security

Let us fix a particular symmetric encryption scheme $\mathcal{SE} = (\mathcal{K},\mathcal{E},\mathcal{D})$. Two parties share a key K for this scheme, this key having being generated as $K \stackrel{R}{\leftarrow} \mathcal{K}$. The adversary does not a priori know K. We now want to explore the issue of what security (in this case, privacy) of the scheme might mean.

The adversary is assumed able to capture any ciphertext that flows on the channel between the two parties. It can thus collect ciphertexts, and try to glean something from them. Our question is: what exactly does "glean" mean? What tasks, were the adversary to accomplish them, would make us declare the scheme insecure? And, correspondingly, what tasks, were the adversary unable to accomplish them, would make us declare the scheme secure?

It is much easier to think about *insecurity* than security, because we can certainly identify adversary actions that indubitably imply the scheme is insecure. For example, if the adversary can, from a few ciphertexts, derive the underlying key K, it can later decrypt anything it sees, so if the scheme allowed easy key recovery from a few ciphertexts it is definitely insecure. Yet, an absence of easy key recovery is certainly not enough for the scheme to be secure; maybe the adversary can do something else.

One might want to say something like: given C, the adversary has no idea what M is. This however cannot be true, because of what is called a priori information. Often, something about the message is known. For example, it might be a packet with known headers. Or, it might be an English word. So the adversary, and everyone else, has some information about the message even before it is encrypted.

One might also try to say that what we want is: given ciphertext C, the adversary can't easily recover the plaintext M. But actually, this isn't good enough. The reason is that the adversary might be able to figure out *partial information* about M. For example, even though she might not be able to recover M, the adversary might, given C, be able to recover the first bit of M, or the sum of all the bits of M. This is not good, because these bits might carry valuable information.

For a concrete example, say I am communicating to my broker a message which is a sequence of "buy" or "sell" decisions for a pre-specified sequence of stocks. That is, we have certain stocks, numbered 1 through l, and bit i of the message is 1 if I want to buy stock i and 0 otherwise. The message is sent encrypted. But if the first bit leaks, the adversary knows whether I want to buy or sell stock 1, which may be something

I definitely don't want to reveal. If the sum of the bits leaks, the adversary knows how many stocks I am buying.

Granted, this might not be a problem at all if the data was in a different format. However, making assumptions, or requirements, on how users format data, or how they use it, is a bad and dangerous approach to secure protocol design. It is an important principle of our approach that the encryption scheme should yield security no matter what is the format of the data. That is, we don't want people to have to worry about how they format their data: it should be secure regardless.

In other words, as designers of security protocols, we cannot make assumptions about data content or formats. Our protocols must protect any data, no matter how formatted. We view it as the job of the protocol designer to ensure this is true. And we want schemes that are secure in the strongest possible natural sense.

So what is the best we could hope for? It is useful to make a thought experiment. What would an "ideal" encryption be like? Well, it would be as though some angel took the message M from the sender and delivered it to the receiver, in some magic way. The adversary would see nothing at all. Intuitively, our goal is to "approximate" this as best as possible. We would like encryption to have the properties of ideal encryption. In particular, no partial information would leak.

As an example, consider the ECB encryption scheme of Example 6.3. Given the ciphertext, can an eavesdropping adversary figure out the message? Hard to see how, since it does not know K, and if F is a "good" block cipher, then it ought to have a hard time inverting F_K without knowledge of the underlying key. Nonetheless this is not a good scheme. Consider just the case $n = 1$ of a single block message. Suppose I have just two messages, 0^l for "buy" and 1^l for "sell." I keep sending data, but always one of these two. What happens? The adversary sees which are the same. That is, it might see that the first two are the same and equal to the third, etc.

In a secure encryption scheme, it should not be possible to co-relate ciphertexts of different messages in such a way that information is leaked.

This has a somewhat dramatic implication. Namely, *encryption must be probabilistic* or *depend on state information*. If not, you can always tell if the same message was sent twice. Each encryption must use fresh coin tosses, or, say, a counter, and an encryption of a particular message may be different each time. In terms of our setup it means $\mathcal{E}$ is a *probabilistic* or *stateful* algorithm. That's why we defined symmetric encryption schemes, above, to allow these types of algorithms.

The reason this is dramatic is that it goes in many ways against the historical or popular notion of encryption. Encryption is thought of as a code, a fixed mapping of plaintexts to ciphertexts. This is no longer true. A single plaintext will have many possible ciphertexts. (Depending on the random choices or state of the encryption algorithm.) Yet, it should be possible to decrypt. How is this possible? We have seen several examples above.

Let us now start looking at privacy more formally. We will begin with the information-theoretic notion of perfect privacy introduced by Shannon, and analyze the one-time pad scheme in this light. Perfect security, however, requires a key as long as the total amount of data encrypted, and is not practical. So we then look at a notion of "computational security." The security will only hold with respect to adversaries of limited computing power. If the adversary works harder, she can figure out more, but a "feasible" amount of effort yields no noticeable information. This is the important notion for us and will be used to analyze the security of schemes such as those presented above.

6.4 Information-theoretic security

We discuss the information-theoretic notion of security called perfect security which we will show is possessed by the one-time-pad scheme.

We fix a particular symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Two parties share a key K for this scheme and the adversary does not a priori know K. The adversary is assumed able to capture any ciphertext that flows on the channel between the two parties. Having captured a ciphertext, it attempts to glean information about the corresponding plaintext message.

Take for example the one-time-pad scheme, and assume a single k-bit message is encrypted and transmitted, where k is the length of the key. Due to the random choice of the key (pad), this certainly seems very "secure." We would like to say that the adversray, given the ciphertext, has "no idea" what the message was. But it is not clear how to say this, or if it is even really true. The adversary could always guess the message. Or, it could have a pretty good idea what the message was from some context surrounding the encryption. For example, it may know that the first few bytes of the message is a packet header containing the sender's (known) ip address.

So we can't really say the adversary has no idea what the message is given the ciphertext. Instead, we adopt a comparitive measure of security. We are interested in how much more the adversary knows about the message given the ciphertext as opposed to what it knew before it saw the ciphertext. Perfect security holds if "the adversary's best guess as to the message after having seen the ciphertext is the same as before it saw the ciphertext." In other words, the ciphertext was no help in figuring out anything new about the message.

This is captured this as follows. We assume a single message will be encrypted, and are interested only in the security of this encryption. There is some plaintext space $\mathsf{Plaintexts} \subseteq \{0,1\}^*$ of messages that the encryptor is willing to encrypt. (For example, with the one-time pad scheme, if the key length is k bits then $\mathsf{Plaintexts} = \{0,1\}^k$.) Notice that this effectively makes the scheme stateless.

We model the a priori information (the information the adversary already possesses about the message) as a probability distribution on the set of possible messages. Formally, a *message distribution* on $\mathsf{Plaintexts}$ is a function $D\colon \mathsf{Plaintexts} \to [0,1]$ such that

$$\sum_{M \in \mathsf{Plaintexts}} D(M) = 1\ ,$$

and also $D(M) > 0$ for all $M \in \mathsf{Plaintexts}$. For example, there might be four messages, $00, 01, 10, 11$, with

$$D(00) = 1/6, \quad D(01) = 1/3, \quad D(10) = 1/4, \quad \text{and} \quad D(11) = 1/4\ .$$

We imagine that the sender chooses a message at random according to D, meaning that a specific message $M \in \mathsf{Plaintexts}$ has probability $D(M)$ of being chosen. In our example, the sender would choose 00 with probability $1/6$, and so on.

The message distribution, and the fact that the sender chooses according to it, are known to the adversary. Before any ciphertext is transmitted, the adversary's state of knowledge about the message chosen by the sender is given by D. That is, it knows that the message was 00 with probability $1/6$, and so on.

We say that the encryption scheme is perfectly secure if the possession of the ciphertext does not impart any *additional* information about the message than was known a priori via the fact that it was chosen according to D. The setup is like this. After the sender has chosen the message according to D, a key K is also chosen, according to the key generation algorithm, meaning $K \leftarrow \mathcal{K}$, and the message is encrypted to get a ciphertext, via $C \leftarrow \mathcal{E}_K(M)$. The adversary is given C. We ask the adversary: given that you know C is the ciphertext produced, for each possible value of the message, what is the probability that that particular value was actually the message chosen? If the adversary can do no better than say that the probability that M was chosen was $D(M)$, it means that the possession of the ciphertext is not adding any new information to what is already known. This is perfect security.

To state this more formally, we first let

$$S \;=\; \mathsf{Keys}(\mathcal{SE}) \times \mathsf{Plaintexts} \times \{0,1\}^r$$

denote the sample space underlying our experiment. Here r is the number of coins the encryption algorithm tosses. (This is zero if the encryption algorithm is deterministic, as is the case for the one-time pad.) We let introduce the following random variables:

$$\begin{array}{lll}
\mathbf{K}\colon S \to \mathsf{Keys}(\mathcal{SE}) & \text{defined by} & (K,M,R) \mapsto K \\
\mathbf{M}\colon S \to \mathsf{Plaintexts} & \text{defined by} & (K,M,R) \mapsto M \\
\mathbf{C}\colon S \to \{0,1\}^* & \text{defined by} & (K,M,R) \mapsto \mathcal{E}_K(M;R)
\end{array}$$

Thus $\mathbf{K}$ simply returns the value of the chosen key while $\mathbf{M}$ returns the value of the chosen message. The last random variable returns the encryption of the message using key K and coins R. The probability distribution underlying this sample space is denoted $\mathbf{P}_{D,\mathcal{SE}}[\cdot]$ and is given by a choice of K as per $\mathcal{K}$, a choice of M as per D, and a random choice of R, all these being made independently.

Definition 6.7 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme with associated message space $\mathsf{Plaintexts}$. Let $D\colon \mathsf{Plaintexts} \to [0,1]$ be a message distribution on $\mathsf{Plaintexts}$. We say that $\mathcal{SE}$ is *perfectly secure with respect to* D if for every $M \in \mathsf{Plaintexts}$ and every possible ciphertext C it is the case that

$$\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M \mid \mathbf{C} = C] \quad = \quad D(M)\ . \tag{6.1}$$

We say that $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is *perfectly secure* if it is perfectly secure with respect to *every* message distribution on $\mathsf{Plaintexts}$. ▮

Here "$\mathbf{M} = M$" is the event that the message chosen by the sender was M, and "$\mathbf{C} = C$" is the event that the ciphertext computed by the sender and received by the adversary was C. The definition considers the conditional probability that the message was M given that the ciphertext was C. It says that this probability is exactly the a priori probability of the message M, namely $D(M)$.

In considering the one-time pad encryption scheme (cf. Scheme 6.2) we omit the counter as part of the ciphertext since only a single message is being encrypted. Thus, the ciphertext is a k-bit string where k is the length of the key and also of the message. Also note that in this scheme $r = 0$ since the encryption algorithm is not randomized.

Example 6.8 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the one-time-pad encryption scheme with the key length (and thus also message length and ciphertext length) set to $k = 2$ bits and the message space set to $\mathsf{Plaintexts} = \{0,1\}^k$. Let D be the message distribution on $\mathsf{Plaintexts}$ defined by $D(00) = 1/6$, $D(01) = 1/3$, $D(10) = 1/4$ and $D(11) = 1/4$. For each possible ciphertext $C \in \{0,1\}^k$, the first table of Figure 6.1 shows the value of $\mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = C \mid \mathbf{M} = M]$, the probability of obtaining this particular ciphertext if you encrypt M with the one-time pad scheme. As the table indicates, this probability is always 0.25. Why? Having fixed M, the possible ciphertexts are $M \oplus K$ as K ranges over $\{0,1\}^k$. So, regardless of the value of M, all different k bit strings are equally likely as ciphertexts. The corresponding general statement is stated and proved in Lemma 6.9 below. The second table shows the value of $\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M \mid \mathbf{C} = C]$, the probability that the message was M given that an adversary sees ciphertext C. Notice that this always equals the a priori probability $D(M)$. ▮

The following lemma captures the basic security property of the one-time-pad scheme: no matter what is the message, each possible k-bit ciphertext is produced with probability 2^{-k}, due to the random choice of the key. .

Lemma 6.9 Let $k \geq 1$ be an integer and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the one-time-pad encryption scheme of Scheme 6.2 with the key length set to k bits and the message space set to $\mathsf{Plaintexts} = \{0,1\}^k$. Let D be a message distribution on $\mathsf{Plaintexts}$. Then

$$\mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = Y \mid \mathbf{M} = X] \quad = \quad 2^{-k}\ .$$

for any $X \in \mathsf{Plaintexts}$ and any $Y \in \{0,1\}^k$. ▮

Proof of Lemma 6.9: If X is fixed and known, what's the probability that we see Y? Since $Y = K \oplus X$ for the one-time-pad scheme, it only happens if $K = Y \oplus X$. The probability that K is this particular string is exactly 2^{-k} since K is a randomly chosen k-bit string. ▮

This enables us to show that the one-time-pad scheme meets the notion of perfect security we considered above.

$D(M)$	M \ C	00	01	10	11
1/6	00	0.25	0.25	0.25	0.25
1/3	01	0.25	0.25	0.25	0.25
1/4	10	0.25	0.25	0.25	0.25
1/4	01	0.25	0.25	0.25	0.25

$D(M)$	M \ C	00	01	10	11
1/6	00	1/6	1/6	1/6	1/6
1/3	01	1/3	1/3	1/3	1/3
1/4	10	1/4	1/4	1/4	1/4
1/4	01	1/4	1/4	1/4	1/4

Figure 6.1: In the first table, the entry corresponding to row M and column C shows the value of $\mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = C \mid \mathbf{M} = M]$, for the one-time-pad scheme of Example 6.8. Here the key and message length are both $k = 2$. In the second table, the entry corresponding to row M and column C shows the value of $\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M \mid \mathbf{C} = C]$, for the same scheme.

Theorem 6.10 Let $k \geq 1$ be an integer and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the one-time-pad encryption scheme of Scheme 6.2 with the key length set to k bits and the message space set to $\mathsf{Plaintexts} = \{0,1\}^k$. Let D be a message distribution on $\mathsf{Plaintexts}$. Then $\mathcal{SE}$ is perfectly secure with respect to D. ▮

Proof of Theorem 6.10: Let $M \in \mathsf{Plaintexts}$ be a message and let $C \in \{0,1\}^k$ be a possible ciphertext. We need to show that Equation (6.1) is true. We have

$$\begin{aligned}
\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M \mid \mathbf{C} = C] &= \mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = C \mid \mathbf{M} = M] \cdot \frac{\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M]}{\mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = C]} \\
&= 2^{-k} \cdot \frac{\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M]}{\mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = C]} .
\end{aligned}$$

The first equality was by Bayes' rule. The second equality was obtained by applying Lemma 6.9 with $X = M$ and $Y = C$. By definition

$$\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M] = D(M)$$

is the a priori probability of M. Now for the last term:

$$\begin{aligned}
\mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = C] &= \sum_X \mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = X] \cdot \mathbf{P}_{D,\mathcal{SE}}[\mathbf{C} = C \mid \mathbf{M} = X] \\
&= \sum_X D(X) \cdot 2^{-k} \\
&= 2^{-k} \cdot \sum_X D(X) \\
&= 2^{-k} \cdot 1 .
\end{aligned}$$

The sum here was over all possible messages $X \in \mathsf{Plaintexts}$, and we used Lemma 6.9. Plugging all this into the above we get

$$\mathbf{P}_{D,\mathcal{SE}}[\mathbf{M} = M \mid \mathbf{C} = C] = 2^{-k} \cdot \frac{D(M)}{2^{-k}} = D(M)$$

as desired. ∎

The one-time-pad scheme is not the only scheme possessing perfect security, but it seems to be the simplest and most natural one.

6.5 Indistinguishability under chosen-plaintext attack

Perfect security can only be achieved by schemes with keys as long as the message encrypted, which is not practical. (We will not prove this.) We wish to find a notion of security that, even though not perfect in the above sense, is as good in practice. Roughly, the main difference is to take into account the fact that adversaries are computationally restricted. There may be "useful" information in a ciphertext, but if you can't compute it, the ciphertext hasn't really given you anything.

In some ways, this is where modern cryptography really begins. For us, what is most relevant is not the above, but what follows.

We have already discussed the issues in Section 6.3 above and will now distill a formal definition of security.

6.5.1 Definition

We discussed above the case where the sender is encrypting one of two known messages. It turns out that this is the "worst" case for security. We consider an adversary (not in possession of the secret key) who knows (in fact, is allowed to choose) two messages of the same length. Then, one is encrypted and the ciphertext given to the adversary. The scheme is secure if the adversary has a hard time telling which message was encrypted.

We actually want to consider encryption not of one message, but a whole sequence of them, so this idea must be extended. There is a sequence of pairs of messages, $(M_{1,0}, M_{1,1}), \ldots, (M_{q,0}, M_{q,1})$, where, in each pair, the two messages have the same length. This sequence is known to the adversary. Now, a "challenge" bit b is chosen at random, and a sequence of ciphertexts $C_1, \ldots, C_q$ is produced, where $C_i \leftarrow \mathcal{E}_K(M_{i,b})$. Note that in these encryptions, the encryption algorithm uses fresh coins, or an updated state, each time. The adversary gets the sequence of ciphertexts and must guess the bit b to win. In other words, the adversary is trying to determine whether the sender sent $M_{1,0}, \ldots, M_{q,0}$ or $M_{1,1}, \ldots, M_{q,1}$.

To further empower the adversary, it is allowed to choose the sequence of message pairs via a *chosen plaintext attack*. This means that it chooses the first pair, then receives C_1, then chooses the second pair, and so on.

Let us now formalize this. We fix a specific encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. (It could be either stateless or stateful). We consider an adversary A. It is a program which has access to an oracle to which it can provide as input any pair (M_0, M_1) of equal-length messages. The oracle will return a ciphertext. We will consider two possible ways in which this ciphertext is computed by the oracle, corresponding to two possible "worlds" in which the adversary "lives". To do this, first define the *left-or-right encryption oracle* $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))$, as follows:

Oracle $\mathcal{E}_K(\mathrm{LR}(M_0, M_1, b))$ // $b \in \{0,1\}$ and $M_0, M_1 \in \{0,1\}^*$
  $C \leftarrow \mathcal{E}_K(M_b)$
  Return C

The oracle encrypts one of the messages, the choice of which being made according to the bit b. Now the two worlds are as follows:

World 0: The oracle provided to the adversary is $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,0))$. So, whenever the adversary makes a query (M_0, M_1) to its oracle, the latter computes $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M_0)$, and returns C as the answer.

World 1: The oracle provided to the adversary is $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,1))$. So, whenever the adversary makes a query (M_0, M_1) to its oracle, the latter computes $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M_1)$, and returns C as the answer.

We call the first world, or oracle, the "left" world or oracle, and the second the "right" world or oracle. The problem for the adversary is, after talking to its oracle for some time, to tell which of the two oracles it was given. Before we pin this down, let us further clarify exactly how the oracles operate.

Think of an oracle as a subroutine to which A has access. A can make an oracle query (M_0, M_1) by writing (M_0, M_1) in some special, specified location in memory, and, in one step, the answer is returned. A has no control on how the answer is computed, nor can A even see the working of the oracle, which will typically depend on secret information that A is not given. A just has an interface to this subroutine; the ability to call it as a black-box, and get back an answer.

First assume the given symmetric encryption scheme $\mathcal{SE}$ is stateless. The oracle, in either world, is probabilistic, because it calls the encryption algorithm. Recall that this algorithm is probabilistic. Above, when we say $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M_b)$, it is implicit that $\mathcal{E}$ picks its own random coins implicitly and uses them to compute C.

The random choices of the encryption function are somewhat "under the rug" here, but should not be forgotten; they are central to the meaningfulness of the notion, as also the security of the schemes.

If the given symmetric encryption scheme $\mathcal{SE}$ was stateful, the oracles, in either world, become stateful too. (Think of a subroutine that maintains a global variable across calls to the subroutine.) In world b the oracle begins with a state value initialized according to the specification of the scheme. For example, for CTR mode with a counter, it is a counter ctr set to 0. Now, each time the oracle is invoked, it computes $\mathcal{E}_K(M_b)$ according to the specification of algorithm $\mathcal{E}$. This algorithm will, as a side-effect, update the counter, and upon the next invocation of the oracle, the new counter value will be used.

We clarify that the choice of which world we are in is made once, a priori, and then the adversary executes. In world 0, *all* message pairs sent to the oracle are answered by the oracle encrypting the left message in the pair, while in world 1, all message pairs are answered by the oracle encrypting the right message in the pair. The choice of which does not flip-flop from oracle query to oracle query; it is made once and then remains the same for all messages.

We consider an encryption scheme to be "secure against chosen-plaintext attack" if a "reasonable" adversary cannot obtain "significant" advantage in distinguishing the cases $b = 0$ and $b = 1$ given access to the oracle, where reasonable reflects its resource usage. The technical notion is called indistinguishability under chosen-plaintext attack, denoted IND-CPA.

Before presenting it we need to discuss a subtle point. There are certain queries that an adversary can make to its lr-encryption oracle which will definately enable it to learn the value of the hidden bit b (meaning figure out in which world it is) but which we consider illegitimate. One is to query the oracle with messages M_0, M_1 of different lengths. We do not ask that encryption hide the length of the plaintext, and indeed common schemes reveal this because the length of the ciphertext depends on the length of the plaintext, so an adversary making such a query could easily win. Another, less obvious attack is for the adversary to make a query M_0, M_1 of equal-length messages such that $\mathcal{E}_K(M_0) \neq \perp$ and $\mathcal{E}_K(M_1) = \perp$. (If the scheme is stateless, this means M_0 is in the plaintext space and M_1 is not.) For some schemes, it is easy for the adversary to find such messages. However, the response of the lr-encryption oracle then gives away the bit b. We have chosen to deal with these issues by simply disallowing the adversary from making such queries. That is, let us say that an adversary is *illegitimate* if it either makes a lr-encryption query consisting of two messages of different lengths, or makes an lr-encryption query M_0, M_1 for which $\mathcal{E}_K(M_c) = \perp$ with positive probability for some value of c. The adversary is legitimate if it is not illegitimate.

The issue of legitimacy can, once discussed, be forgotten, since in all our reductions and results we will have only legitimate adversaries, but we do have to deal with it in the definition.

Definition 6.11 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme, let $b \in \{0, 1\}$, and let A be an algorithm that has access to an oracle that takes input a pair of strings and returns a string. We consider the following experiment:

Experiment $\mathbf{Exp}^{\text{ind-cpa-}b}_{\mathcal{SE},A}$
$K \stackrel{R}{\leftarrow} \mathcal{K}$
$d \leftarrow A^{\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))}$
Return d

The *ind-cpa-advantage* of A is defined as

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} \quad = \quad \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-1}}_{\mathcal{SE},A} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-0}}_{\mathcal{SE},A} = 1\right]$$

if A is legitimate, and 0 otherwise. For any t, q, μ we define the *ind-cpa-advantage* of $\mathcal{SE}$ via

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t,q,\mu) \quad = \quad \max_{A} \left\{ \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} \right\}$$

where the maximum is over all legitimate A having time-complexity t, making to the oracle at most q queries the sum of whose lengths is at most μ bits. ∎

We discuss some important conventions. The *time-complexity* mentioned above is the worst case total execution time of the experiment, plus the size of the code of the adversary, in some fixed RAM model of computation. We stress that the the total execution time of the experiment is more than the running time of the adversary. It includes the time of *all* operations in the experiment, including the time for key generation and the computation of answers to oracle queries. This convention for measuring time complexity is the same as used in other parts of these notes.

Another convention is that the length of a query M_0, M_1 to a left-or-right encryption oracle is defined as $|M_0|$. (We can assume this equals $|M_1|$ since the adversary is assumed to be legitimate.) This convention is used in measuring the parameter μ.

If $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A}$ is small, it means that A is outputting 1 just about as often in world 0 as in world 1, meaning it is not doing a good job of telling which world it is in. If this quantity is large (meaning close to one) then the adversary A is doing well, meaning our scheme $\mathcal{SE}$ is not secure.

For symmetric encryption scheme $\mathcal{SE}$ to be secure against chosen plaintext attack, the ind-cpa-advantage of an adversary must be small, no matter what strategy the adversary tries. However, we expect that the advantage grows as the adversary invests more effort in the process. To capture this we have defined the advantage function $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(\cdot,\cdot,\cdot)$ as above. This is a function associated to any symmetric encryption scheme $\mathcal{SE}$, which becomes fixed once we fix the scheme. The resources of the adversary we have chosen to use in the parameterization are three. First, its time-complexity, measured according to the convention above. Second, the number of oracle queries, or the number of message pairs the adversary asks of its oracle. These messages may have different lengths, and our third parameter is the sum of all these lengths, denoted μ, again measured according to the convention above. The ind-cpa-advantage function of the scheme measures the maximum probability that the security of the scheme $\mathcal{SE}$ can be compromised by an adversary using the indicated resources.

6.5.2 Alternative interpretation of advantage

Why is the $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A}$ called the "advantage" of the adversary? We can view the task of the adversary as trying to guess which world it is in. A trivial guess is for the adversary to return a random bit. In that case, it has probability 1/2 of being right. Clearly, it has not done anything damaging in this case. The advantage of the adversary measures how much better than this it does at guessing which world it is in, namely the excess over 1/2 of the adversary's probability of guessing correctly. In this subsection we will see how the above definition corresponds to this alternative view, a view that lends some extra intuition to the definition and is also useful in later usages of the definition.

As usual we fix a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. We now consider the following game, or experiment.

Experiment $\mathbf{Exp}^{\text{ind-cpa-cg}}_{\mathcal{SE},A}$
  Pick a bit b at random
  Let $K \stackrel{R}{\leftarrow} \mathcal{K}$
  $g \leftarrow A^{\mathcal{E}_K(\text{LR}(\cdot,\cdot,b))}$
  If $b = g$ return 1 else return 0

Here, A is run with an oracle for world b, where the bit b is chosen at random. A eventually outputs a bit g, its guess as to the value of b. The experiment returns 1 if A's guess is correct. Thus

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-cg}}_{\mathcal{SE},A} = 1\right]$$

is the probability that A correctly guesses which world it is in. (The probability is over the initial choice of world as given by the bit b, the choice of K, the random choices of $\mathcal{E}_K(\cdot)$ if any, and the coins of A if any.) The following proposition says that one-half of the advantage is exactly the excess above one-half of the chance that A correctly guesses which world it is in.

Proposition 6.12 Let $\mathcal{SE}$ be a symmetric encryption scheme. Then

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-cg}}_{\mathcal{SE},A} = 1\right] \quad = \quad \frac{1}{2} + \frac{\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A}}{2} \; .$$

for any ind-cpa-adversary A. ▮

Proof of Proposition 6.12: We let $\mathbf{P}[\cdot]$ be the probability of event "$\cdot$" in the experiment $\mathbf{Exp}^{\text{ind-cpa-cg}}_{\mathcal{SE},A}$, and refer below to quantities in this experiment. The claim of the Proposition follows by a straightforward calculation:

$$\begin{aligned}
&\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-cg}}_{\mathcal{SE},A} = 1\right] \\
&\quad = \quad \mathbf{P}[b = g] \\
&\quad = \quad \mathbf{P}[b = g \mid b = 1] \cdot \mathbf{P}[b = 1] + \mathbf{P}[b = g \mid b = 0] \cdot \mathbf{P}[b = 0] \\
&\quad = \quad \mathbf{P}[b = g \mid b = 1] \cdot \frac{1}{2} + \mathbf{P}[b = g \mid b = 0] \cdot \frac{1}{2} \\
&\quad = \quad \mathbf{P}[g = 1 \mid b = 1] \cdot \frac{1}{2} + \mathbf{P}[g = 0 \mid b = 0] \cdot \frac{1}{2} \\
&\quad = \quad \mathbf{P}[g = 1 \mid b = 1] \cdot \frac{1}{2} + (1 - \mathbf{P}[g = 1 \mid b = 0]) \cdot \frac{1}{2} \\
&\quad = \quad \frac{1}{2} + \frac{1}{2} \cdot (\mathbf{P}[g = 1 \mid b = 1] - \mathbf{P}[g = 1 \mid b = 0]) \\
&\quad = \quad \frac{1}{2} + \frac{1}{2} \cdot \left(\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-1}}_{\mathcal{SE},A} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-0}}_{\mathcal{SE},A} = 1\right]\right) \\
&\quad = \quad \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} \; .
\end{aligned}$$

We began by expanding the quantity of interest via standard conditioning. The term of $1/2$ in the third line emerged because the choice of b is made at random. In the fourth line we noted that if we are asking whether $b = g$ given that we know $b = 1$, it is the same as asking whether $g = 1$ given $b = 1$, and analogously for $b = 0$. In the fifth line and sixth lines we just manipulated the probabilities and simplified. The next line is important; here we observed that the conditional probabilities in question are exactly the success probabilities in the real and random games respectively. That meant we had recovered the advantage, as desired. ▮

6.6 Example chosen-plaintext attacks

We illustrate the use of the model in finding attacks by providing an attack on ECB mode, and also a general attack on deterministic, stateless schemes.

6.6.1 Attack on ECB

Let us fix a block cipher $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$. The ECB symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is that of Scheme 6.3. Suppose an adversary sees a ciphertext $C = \mathcal{E}_K(M)$ corresponding to some unknown plaintext text M, encrypted under the key K also unknown to the adversary. Can the adversary recover M? Not easily, if E is a "good" block cipher. For example if E is AES, it seems quite infeasible. Yet, we have already discussed how infeasability of recovering plaintext from ciphertext is not an indication of security. ECB has other weaknesses. Notice that if two plaintexts M and M' agree in the first block, then so do the corresponding ciphertexts. So an adversary, given the ciphertexts, can tell whether or not the first blocks of the corresponding plaintexts are the same. This is loss of partial information about the plaintexts, and is not permissible in a secure encryption scheme.

It is a test of our definition to see that the definition captures these weaknesses and also finds the scheme insecure. It does. To show this, we want to show that there is an adversary that has a high ind-cpa-advantage while using a small amount of resources. This is what the following proposition says.

Proposition 6.13 Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher, and $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ the corresponding ECB symmetric encryption scheme as described in Scheme 6.3. Then

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, 1, 2l) \quad = \quad 1$$

for $t = O(l)$ plus the time for two applications of the block cipher. ▮

The advantage of this adversary is 1 even though it uses hardly any resources: just one query, and not a long one at that. That is clearly an indication that the scheme is insecure.

Proof of Proposition 6.13: We will present an adversary algorithm A, having time-complexity t, making 1 query to its oracle, this query being of length $2l$, and having

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} \quad = \quad 1\,.$$

The Proposition follows.

Remember the adversary A is given a lr-encryption oracle $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))$ which takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b. The goal of A is to determine the value of b. Our adversary works like this:

Adversary $A^{\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))}$
  $M_1 \leftarrow 0^{2l}$; $M_0 \leftarrow 0^l \| 1^l$
  $C[1]C[2] \leftarrow \mathcal{E}_K(\mathrm{LR}(M_0, M_1, b))$
  If $C[1] = C[2]$ then return 1 else return 0

The adversary's single oracle query is the pair of messages M_0, M_1. Since each of them is two blocks long, so is the ciphertext computed according to the ECB scheme. Now, we claim that

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-1}}_{\mathcal{SE},A} = 1\right] \quad = \quad 1$$

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-0}}_{\mathcal{SE},A} = 1\right] \quad = \quad 0\,.$$

Hence $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} = 1 - 0 = 1$. And A achieved this advantage by making just one oracle query, whose length, which as per our conventions is just the length of M_0, is $2l$ bits. So $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, 1, 2l) = 1$.

Why are the two equations claimed above true? You have to return to the definitions of the quantities in question, and trace through the experiments defined there. In World 1, meaning when $b = 1$, the oracle returns $C[1]C[2] = E_K(0^l)\|E_K(0^l)$, so $C[1] = C[2]$ and A returns 1. In World 0, meaning when $b = 0$, the oracle returns $C[1]C[2] = E_K(0^l)E_K(1^l)$. Since E_K is a permutation, $C[1] \neq C[2]$. So A returns 0 in this case. ∎

As an exercise, try to analyze the same adversary as an adversary against CBC or CTR modes, and convince yourself that the adversary will not get a high advantage.

There is an important feature of this attack that must be emphasized. Namely, ECB is an insecure encryption scheme *even if the underlying block cipher E is highly secure.* The weakness is not in the tool being used, but in the manner we are using it. It is the ECB mechanism that is at fault. Even a good tool is useless if you don't use it well.

This is the kind of design flaw that we want to be able to spot and eradicate. Our goal is to find symmetric encryption schemes that are secure as long as the underlying block cipher is secure. In other words, the scheme has no inherent flaw. As long as you use good ingredients, the recipe produces a good meal. If you don't use good ingredients? Well, that is your problem.

6.6.2 Deterministic, stateless schemes are insecure

ECB mode is deterministic and stateless, so that if the same message is encrypted twice, the same ciphertext is returned. It turns out that this property, in general, results in an insecure scheme, and provides perhaps a better understanding of why ECB fails. Let us state the general fact more precisely.

Proposition 6.14 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a deterministic, stateless symmetric encryption scheme. Assume there is an integer m such that the plaintext space of the scheme contains two distinct strings of length m. Then

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, 2, 2m) \quad = \quad 1$$

for $t = O(l)$ plus the time for two encryptions. ∎

The requirement being made on the message space is minimal; typical schemes have messages spaces containing all strings of lengths between some minimum and maximum length. Note that this Proposition applies to ECB and is enough to show the latter is insecure, but Proposition 6.13 shows something a little stronger because there there is only one query rather than two.

Proof of Proposition 6.14: We will present an adversary algorithm A, having time-complexity t, making 2 queries to its oracle, each query being of length m, and having

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} \quad = \quad 1\,.$$

The Proposition follows.

Remember the adversary A is given a lr-encryption oracle $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))$ which takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b. The goal of A is to determine the value of b. Our adversary works like this:

Adversary $A^{\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))}$
  Let X, Y be distinct, m-bit strings in the plaintext space
  $C_1 \leftarrow \mathcal{E}_K(\mathrm{LR}(X, Y, b))$
  $C_2 \leftarrow \mathcal{E}_K(\mathrm{LR}(Y, Y, b))$
  If $C_1 = C_2$ then return 1 else return 0

Now, we claim that

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-1}}_{\mathcal{SE},A} = 1\right] = 1$$
$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-0}}_{\mathcal{SE},A} = 1\right] = 0\,.$$

Hence $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} = 1 - 0 = 1$. And A achieved this advantage by making two oracle query, each of whose length, which as per our conventions is just the length of the first message, is m bits. So $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, 2, 2m) = 1$.

Why are the two equations claimed above true? In World 1, meaning when $b = 1$, the oracle returns $C_1 = \mathcal{E}_K(Y)$ and $C_2 = \mathcal{E}_K(Y)$, and since the encryption function is deterministic and stateless, $C_1 = C_2$, so A returns 1. In World 0, meaning when $b = 0$, the oracle returns $C_1 = \mathcal{E}_K(X)$ and $C_2 = \mathcal{E}_K(Y)$, and since it is required that decryption be able to recover the message, it must be that $C_1 \neq C_2$. So A returns 0. ∎

6.7 Security against plaintext recovery

In Section 6.3 we noted a number of security properties that are necessary but not sufficient for security. For example, it should be computationally infeasible for an adversary to recover the key from a few plaintext-ciphertext pairs, or to recover a plaintext from a ciphertext. A test of our definition is that it implies these properties, in the sense that a scheme that is secure in the sense of our definition is also secure against key-recovery or plaintext-recovery.

The situation is analogous to what we saw in the case of PRFs. There we showed that a secure PRF is secure against key-recovery. In order to have some variation, this time we choose a different property, namely plaintext recovery. We formalize this, and then show if there was an adversary B capable of recovering the plaintext from a given ciphertext, then this would enable us to construct an adversary A that broke the scheme in the IND-CPA sense, meaning figured out which of the two Worlds it is in. But if the scheme is secure in the IND-CPA sense, that latter adversary could not exist. Hence, neither could the former.

The idea of this argument illustrates how we convince ourselves that the above definition is good, and captures all the properties we might want for security against chosen plaintext attack. Take some other property that you feel a secure scheme should have: infeasability of key recovery from a few plaintext-ciphertext pairs; infeasabilty of predicting the XOR of the plaintext bits; etc. Imagine there was an adversary B that was successful at this task. We claim this would enable us to construct an adversary A that broke the scheme in the left-or-right sense, and hence B does not exist if the scheme is secure in the left-or-right sense. More precisely, we would use the insecurity function of the scheme to bound the probability that adversary B succeeds. Assuming the insecurity function is small at the specified parameter values, so is the chance that adversary B succeeds.

Let us now go through the plaintext recovery example in detail. The task facing the adversary will be to decrypt a ciphertext which was formed by encrypting a randomly chosen challenge message of some length m. In the process we want to give the adversary the ability to see plaintext-ciphertext pairs, and capture this by giving it access to an encryption oracle. This encryption oracle is not the lr-encryption oracle we saw above: instead, it simply takes input a single message M and returns a ciphertext $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$ computed by encrypting M. To capture providing the adversary with a challenge ciphertext, we introduce another oracle that takes no inputs, and upon being invoked picks a random m-bit string (plaintext) M, computes $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$, and returns C. The adversary is allowed only a single query to this oracle, and wins if it can output the plaintext M corresponding to the ciphertext C that this challenge oracle returns. We denote the challenge oracle by $\mathcal{E}_K(\$^m)$. Here $\$^m$ is meant to indicate the choosing of a random m-bit string as the input to $\mathcal{E}_K$.

For simplicity we assume the encryption scheme is stateless, and that $\{0,1\}^m$ is a subset of the plaintext space associated to the scheme. As usual, when either the encryption or the challenge oracle invoke the encryption function, it is implicit that they respect the randomized nature of the encryption function, meaning the latter tosses coins anew upon each invocation of the oracle.

Definition 6.15 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a stateless symmetric encryption scheme whose plaintext space includes $\{0,1\}^m$, and let B be an algorithm that has access to two oracles. We consider the following experiment:

Experiment $\mathbf{Exp}^{\mathrm{pr}}_{\mathcal{SE},B}$
 $K \stackrel{R}{\leftarrow} \mathcal{K}$
 $M \leftarrow B^{\mathcal{E}_K(\cdot), \mathcal{E}_K(\$^m)}$
 If $\mathcal{D}_K(C) = M$, where C was the response to B's query to $\mathcal{E}_K(\$^m)$
  then return 1
  else return 0

The *pr-advantage* of B is defined as

$$\mathbf{Adv}^{\mathrm{pr}}_{\mathcal{SE},B} \quad = \quad \mathbf{P}\left[\mathbf{Exp}^{\mathrm{pr}}_{\mathcal{SE},B} = 1\right] .$$

For any t, q, μ we define the *pr-advantage* of $\mathcal{SE}$ via

$$\mathbf{Adv}^{\mathrm{pr}}_{\mathcal{SE}}(t, q, \mu) \quad = \quad \max_B \left\{ \mathbf{Adv}^{\mathrm{pr}}_{\mathcal{SE},B} \right\}$$

where the maximum is over all B having time-complexity t, making to the encryption oracle at most q queries the sum of whose lengths is at most μ bits. ▮

In the experiment above, B is executed with its two oracles. Recall that it is allowed exactly one query to its challenge oracle. We denote the ciphertext returned as response to this query by C. (Recall the challenge oracle takes no inputs.) The adversary B wins if it can correctly decrypt C, and in that case the experiment returns 1. In the process, the adversary can make encryption oracle queries as it pleases.

The following Proposition says that the probability that an adversary successfully recovers a plaintext from a challenge ciphertext cannot exceed the ind-cpa-advantage of the scheme (with resource parameters those of the plaintext recovery adversary) plus the chance of simply guessing the plaintext. In other words, security in the IND-CPA sense implies security against plaintext recovery.

Proposition 6.16 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a stateless symmetric encryption scheme whose plaintext space includes $\{0,1\}^m$. Then

$$\mathbf{Adv}^{\mathrm{pr}}_{\mathcal{SE}}(t, q, \mu) \quad \leq \quad \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q+1, \mu+m) + \frac{1}{2^m}$$

for any t, q, μ. ▮

The reason this is true is quite simple. If an adversary B were capable of decrpting a challenge ciphertext, we could easily build an adversary A_B that, using B as a subroutine, would be able to tell whether it is in World 0 or World 1. In other words, it is a reduction.

Proof of Proposition 6.16: We will show that given any adversary B whose resources are restricted to t, q, μ we can construct an adversary A_B, using resources $t, q+1, \mu+m$, such that

$$\mathbf{Adv}^{\mathrm{pr}}_{\mathcal{SE},B} \quad \leq \quad \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A_B} + \frac{1}{2^m} . \tag{6.2}$$

The Proposition follows by the usual maximization process.

As per Definition 6.1, adversary A_B will be provided a lr-encryption oracle, and will try to determine in which World it is. To do so, it will run adversary B as a subroutine. We provide the description followed by an explanation and analysis.

Adversary $A_B^{\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))}$
  Run adversary B, replying to its oracle queries as follows
  When B makes an encryption oracle query X do
    $Y \leftarrow \mathcal{E}_K(\mathrm{LR}(X,X,b))$
    Return Y to B as the answer
  When B makes its challenge oracle query do
    $M_0 \stackrel{R}{\leftarrow} \{0,1\}^m$; $M_1 \stackrel{R}{\leftarrow} \{0,1\}^m$
    $C \leftarrow \mathcal{E}_K(\mathrm{LR}(M_0,M_1,b))$
    Return C to B as the answer
  Until B stops and outputs a plaintext M
  If $M = M_1$ then return 1 else return 0

Here A_B is running B and itself providing answers to B's oracle queries. When B makes an encryption oracle query X, adversary A_B needs to return $\mathcal{E}_K(X)$. It does this by invoking its lr-encryption oracle with both messages in the pair set to X, so that regardless of the value of the bit b, the ciphertext returned is an encryption of X, just as B wants. When B makes its (one and only) query to the challenge oracle, A_B picks two random messages, each of length m, and invokes its lr-encryption oracle on them to get back a ciphertext C. Now B returns a message M which is supposed to be the decryption of C. Adversary A_B tests whether $M = M_1$ and if so bets that it is in World 1. Else it bets that it is in World 0. Now we claim that

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-}=}_{\mathcal{SE},A_B,1}1\right] \;\geq\; \mathbf{Adv}^{\text{pr}}_{\mathcal{SE},B}$$

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-}=}_{\mathcal{SE},A_B,0}1\right] \;\leq\; 2^{-m}\,.$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 6.1, we get

$$\begin{aligned}\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A_B} &= \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-}=}_{\mathcal{SE},A_B,1}1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-}=}_{\mathcal{SE},A_B,0}1\right] \\ &\geq \mathbf{Adv}^{\text{pr}}_{\mathcal{SE},B} - 2^{-m}\,.\end{aligned}$$

Re-arranging terms gives us Equation (6.2). It remains to justify Equations (6.3) and (6.3).

Adversary B will return the $M = \mathcal{D}_K(C)$ with probability at least $\mathbf{Adv}^{\text{pr}}_{\mathcal{SE},B}$. In World 1, ciphertext C is an encryption of M_1, so this means that $M = M_1$ with probability at least $\mathbf{Adv}^{\text{pr}}_{\mathcal{SE},B}$, and thus Equation (6.3) is true. Now assume A_B is in World 0. In that case, A_B will return 1 only if B returns $M = M_1$. But B is given no information about M_1, since C is an encryption of M_0 and M_1 is chosen randomly and independently of M_0. It is simply impossible for B to output M_1 with probability greater than 2^{-m}. Thus Equation (6.3) is true. ∎

Similar arguments can be made to show that other desired security properties of a symmetric encryption scheme follow from this definition. For example, is it possible that some adversary B, given some plaintext-ciphertext pairs and then a challenge ciphertext C, can compute the XOR of the bits of $M = \mathcal{D}_K(C)$? Or the sum of these bits? Or the last bit of M? Its probability of doing any of these cannot be more than marginally above 1/2 because were it so, we could design an adversary A that won the left-or-right game using resources comparable to those used by B. We leave as an exercise the formulation and working out of other such examples along the lines of Proposition 6.16.

Of course one cannot exhaustively enumerate all desirable security properties. But you should be moving towards being convinced that our notion of left-or-right security covers all the natural desirable properties of security under chosen plaintext attack. Indeed, we err, if anything, on the conservative side. There are some attacks that might in real life be viewed as hardly damaging, yet our definition declares the scheme insecure if it succumbs to one of these. That is all right; there is no harm in being a little conservative. What is more important is that if there is any attack that in real life would be viewed as damaging, then the scheme will fail the left-or-right test, so that our formal notion too declares it insecure.

6.8 Security of CTR against chosen-plaintext attack

Let $F: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions. The CTR symmetric encryption scheme comes in two variants: the randomized (stateless) one of Scheme 6.5 and the counter-based (stateful) one of Scheme 6.6. Both are secure against chosen-plaintext attack, but, interestingly, the counter version is more secure than the randomized version. We will first state the main theorems about the schemes, discuss them, and then prove them. For the counter version we have:

Theorem 6.17 Let $F: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding C-CTR symmetric encryption scheme as described in Scheme 6.6. Then for any t, q, μ with $\mu < L2^l$ we have

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q, \mu) \quad \le \quad 2 \cdot \mathbf{Adv}^{\text{prf}}_{F}(t, q', lq') ,$$

where $q' = \mu/L$. ∎

And for the randomized version:

Theorem 6.18 Let $F: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding R-CTR symmetric encryption scheme as described in Scheme 6.5. Then for any t, q, μ with $\mu < L2^l$ we have

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q, \mu) \quad \le \quad 2 \cdot \mathbf{Adv}^{\text{prf}}_{F}(t, q', lq') \; + \; \frac{\mu(q-1)}{L2^l} ,$$

where $q' = \mu/L$. ∎

This kind of result is what this whole approach is about. Namely, we are able to provide provable guarantees of security of some higher level cryptographic construct (in this case, a symmetric encryption scheme) based on the assumption that some building block (in this case an underlying block cipher treated as a PRF) is secure. They are the first example of the "punch-line" we have been building towards. So it is worth pausing at this point and trying to make sure we really understand what these theorems are saying and what are their implications.

If we want to entrust our data to some encryption mechanism, we want to know that this encryption mechanism really provides privacy. If it is ill-designed, it may not. We saw this happen with ECB. Even if we used a secure block cipher, the design flaws of ECB mode made it an insecure encryption scheme.

Flaws are not apparent in CTR at first glance. But maybe they exist. It is very hard to see how one can be convinced they do *not* exist, when one cannot possible exhaust the space of all possible attacks that could be tried. Yet this is exactly the difficulty that the above theorems circumvent. They are saying that CTR mode *does not have design flaws.* They are saying that as long as you use a good block cipher, you are *assured* that nobody will break your encryption scheme. One cannot ask for more, since if one does not use a good block cipher, there is no reason to expect security anyway. We are thus getting a conviction that *all attacks fail* even though we do not even know exactly how these attacks operate. That is the power of the approach.

Now, one might appreciate that the ability to make such a powerful statement takes work. It is for this that we have put so much work and time into developing the definitions: the formal notions of security that make such results meaningful. For readers who have less experience with definitions, it is worth knowing, at least, that the effort is worth it. It takes time and work to understand the notions, but the payoffs are big: you actually have the ability to get guarantees of security.

How, exactly, are the theorems saying this? The above discussion has pushed under the rug the quantitative aspect that is an important part of the results. It may help to look at a concrete example.

Example 6.19 Let us suppose that F is AES. So the key size is $k = 128$ and the block size is $l = L = 128$. Suppose I want to encrypt $q = 2^{40}$ messages, each $128 * 2^3$ bits long, so that I am encrypting a total of $\mu = 2^{50}$ bits of data. Can I do this securely using counter-mode CTR? What is the chance that an adversary figures out something about my data? Well, if the adversary has $t = 2^{60}$ computing cycles, then by definition its chance is not more than $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q, \mu)$. That has nothing to do with the theorem: it is just our definitions, which say that this is the maximum probability of being able to break the encryption scheme in these given resources. So the question of whether the scheme is secure for my chosen parameters boils down to asking what is the value of $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q, \mu)$. A priori, we have no idea. But now, we appeal to Theorem 6.17, which says that this chance is at most $2 \cdot \mathbf{Adv}^{\text{prf}}_{F}(t, q', 128q')$, where q' is as given in the theorem. Namely $q' = \mu/L = 2^{50}/128 = 2^{43}$. So the question is, what is the value of $\mathbf{Adv}^{\text{prf}}_{F}(t, q', 128q')$ with these values of t', q'?

Thus, what the theorem has done is reduce the question of estimating the probability of loss of privacy from the encryption scheme to the question of estimating the pseudorandomness of AES. As per Section 5.6.2, one might conjecture that

$$\mathbf{Adv}^{\text{prf}}_{\text{AES}}(t, q', 128q') \;=\; c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + \frac{(q')^2}{2^{128}} ,$$

where T_{AES} is the time to do one AES computation on our fixed RAM model of computation. Now plug in $t = 2^{60}$ and $q' = 2^{43}$ and take into account what we computed above. We get

$$\begin{aligned}
\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q, \mu) &\le 2 \cdot \mathbf{Adv}^{\text{prf}}_{\text{AES}}(t, q', 128q') \\
&\le 2c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + \frac{2(q')^2}{2^{128}} \\
&= \frac{2^{61}}{2^{128}} \cdot \frac{c_1}{T_{\text{AES}}} + \frac{2^{43*2+1}}{2^{128}} \\
&= \frac{1}{2^{67}} \cdot \frac{c_1}{T_{\text{AES}}} + \frac{1}{2^{41}} \\
&\le \frac{1}{2^{41}} .
\end{aligned}$$

In the last step, we made the (very reasonable) assumption that c_1/T_{AES} is at most 2^{26}. Thus, the chance the adversary gets any information about our encrypted data is about 2^{-41}, even though we allow this adversary computing time up to 2^{60}, and are encrypting 2^{50} bits of data. This is a very small chance, and we can certainly live with it. It is in this sense that we say the scheme is secure. ∎

Example 6.20 You are encouraged to work out another example along the following lines. Don't assume F is AES, but rather assume it is an even better PRF. It still has $k = l = L = 128$, but assume it is not a permutation, so that there are no birthday attacks; specifically, assume

$$\mathbf{Adv}^{\text{prf}}_{F}(t, q', 128q') \;=\; c_1 \cdot \frac{t/T_{\text{AES}}}{2^{128}} + c_1 \cdot \frac{q}{2^{128}} .$$

Now, consider both the counter-based CTR scheme and the randomized one. In the theorems, the difference is the $\mu(q-1)/L2^l$ term. Try to see what kind of difference this makes. For each scheme, consider how high you can push q, μ, t and still have some security left. For which scheme can you push them higher? Which scheme is thus "more secure"? ∎

These examples illustrate how to use the theorems to figure out how much security you will get from the CTR encryption scheme in some application.

6.8.1 Proof of Theorem 6.17

The paradigm used is quite general in many of its aspects, and we will use it again, not only for encryption schemes, but for other kinds of schemes that are based on pseudorandom functions.

Algorithm $\mathcal{E}_g(M)$	Algorithm $\mathcal{D}_f(C)$
If $\lvert M\rvert < L$ then return $\perp$	If $\lvert C\rvert < l + L$ then return $\perp$
If $\lvert M\rvert \bmod L \neq 0$ then return $\perp$	If $(\lvert C\rvert - l) \bmod L \neq 0$ then return $\perp$
Parse M as $M[1]\ldots M[n]$	Let $C[0]$ be the first l bits of C
If $ctr + n \geq 2^l$ then return $\perp$	Parse the rest of C as $C[1]\ldots C[n]$
For $i = 1, \ldots, n$ do	$ctr \leftarrow \mathsf{StN}(C[0])$
$\quad C[i] \leftarrow g(\mathsf{NtS}_l(ctr + i)) \oplus M[i]$	For $i = 1, \ldots, n$ do
EndFor	$\quad M[i] \leftarrow g(\mathsf{NtS}_l(ctr + i)) \oplus C[i]$
$C[0] \leftarrow \mathsf{NtS}_l(ctr)$	EndFor
$C \leftarrow C[0]C[1]\ldots C[n]$	$M \leftarrow M[1]\ldots M[n]$
$ctr \leftarrow ctr + n$	Return M
Return C	

Figure 6.2: Version $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of the C-CTR scheme parameterized by a family of functions G.

An important observation regarding the CTR scheme is that the encryption and decryption operations do not need direct access to the key K, but only access to a subroutine, or oracle, that implements the function F_K. This is important because one can consider what happens when F_K is replaced by some other function. To consider such replacements, we reformulate the scheme. We introduce a scheme that takes as a parameter any given family of functions G having domain $\{0,1\}^l$ and range $\{0,1\}^L$. As we will see later the cases of interest are $G = F$ and $G = \mathsf{Rand}^{l \to L}$. Let us first however describe this parameterized scheme. In the rest of this proof, $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ denotes the symmetric encryption scheme defined as follows. The key generation algorithm simply returns a random instance of G, meaning picks a function $g \stackrel{R}{\leftarrow} G$ from family G at random, and views g as the key. The encryptor maintains a counter ctr which is initially zero. The encryption and decryption algorithms are shown in Figure 6.2. In the figure, parsing M means that we divide it into L-bit (not l-bit!) blocks and let $M[i]$ denote the i-th such block. For C the decryption algorithm first chops off the first l bits, and then divides the rest of the string into L-bit blocks. The encryption algorithm updates the counter upon each invocation, and begins with this updated value the next time it is invoked. As the description indicates, the scheme is exactly C-CTR, except that function g is used in place of F_K. This seemingly cosmetic change of viewpoint is quite useful, as we will see.

We observe that the scheme in which we are interested, and which the theorem is about, is simply $\mathcal{SE}[F]$ where F is our given family of functions as per the theorem. Now, the proof breaks into two parts. The first step removes F from the picture, and looks instead at an "idealized" version of the scheme. Namely we consider the scheme $\mathcal{SE}[\mathsf{Rand}^{l \to L}]$. Here, a random function g of l-bits to L-bits is being used where the original scheme would use F_K. We then assess an adversary's chance of breaking this idealized scheme. We argue that this chance is actually zero. This is the main lemma in the analysis.

This step is definitely a thought experiment. No real implementation can use a random function in place of F_K because even storing such a function takes an exorbitant amount of memory. But this analysis of the idealized scheme enables us to focus on any possible weaknesses of the CTR mode itself, as opposed to weaknesses arising from properties of the underlying block cipher. We can show that this idealized scheme is secure, and that means that the mode itself is good.

It then remains to see how this "lifts" to a real world, in which we have no ideal random functions, but rather want to assess the security of the scheme $\mathcal{SE}[F]$ that uses the given family F. Here we exploit the notion of pseudorandomness to say that the chance of an adversary breaking the $\mathcal{SE}[F]$ can differ from its chance of breaking the ideal-world scheme $\mathcal{SE}[\mathsf{Rand}^{l \to L}]$ by an amount not exceeding the probability of breaking the pseudorandomness of F.

Lemma 6.21 Let A be any ind-cpa-adversary attacking $\mathcal{SE}[\mathsf{Rand}^{l \to L}]$. Then

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[\mathsf{Rand}^{l \to L}],A} = 0$$

as long as the total length of A's oracle queries is at most $L2^l$. ∎

The lemma considers an aribtrary adversary. Let us say this adversary has time-complexity t, makes q queries to its lr-encryption oracle, these totalling μ bits. The lemma does not care about the values of t and q, but insists that μ be at most $L2^l$. Under this restriction, it makes a strong statement. It says that adversary has zero advantage, meaning no chance at all of breaking the scheme. The strength of this claim is enhanced when one considers that the only restriction, that μ be at most $L2^l$, is hardly a restriction in practice, where l is typically at least 64. The fact that no restriction is made on t indicates that the result is information-theoretic: it holds regardless of how much computing time the adversary invests.

Of course, this lemma refers to the idealized scheme, namely the one where the function g being used by the encryption algorithm is random. But remember that ECB was insecure even in this setting. (The attacks we provided for ECB work even if the underlying cipher E is Perm^l, the family of all permutations.) So the statement is not content-free; it is saying something quite meaningful and important about the CTR mode. It is not true of all modes.

We postpone the proof of the lemma. Instead we will first see how to use it to conclude the proof of the theorem. The argument here is quite simple and generic.

The lemma tells us that the C-CTR encryption scheme is (very!) secure when g is a random function. But we are interested in the case where g is is an instance of our given family F. So our worry is that the actual scheme $\mathcal{SE}[F]$ is insecure even though the idealized scheme $\mathcal{SE}[\mathsf{Rand}^{l\to L}]$ is secure. In other words, we worry that there might be an adversary having large ind-cpa-advantage in attacking $\mathcal{SE}[F]$, even though we know that its advantage in attacking $\mathcal{SE}[\mathsf{Rand}^{l\to L}]$ is zero. But we claim that this is not possible if F is a secure PRF. Intuitively, the existence of such an adversary indicates that F is not approximating $\mathsf{Rand}^{l\to L}$, since there is some detectable event, namely the success probability of some adversary in a certain experiment, that happens with high probability when F is used and with low probability when $\mathsf{Rand}^{l\to L}$ is used. To concretize this intuition, let A be a ind-cpa-adversary attacking $\mathcal{SE}[F]$. We associate to A a distinguisher D_A that is given oracle access to a function $g\colon \{0,1\}^l \to \{0,1\}^L$ and is trying to determine which world it is in, where in World 0 g is a random instance of $\mathsf{Rand}^{l\to L}$ and in World 1 g is a random instance of F. We suggest the following strategy to the distinguisher. It runs A, and replies to A's oracle queries in such a way that A is attacking $\mathcal{SE}[\mathsf{Rand}^{l\to L}]$ in D_A's World 0, and A is attacking $\mathcal{SE}[\mathsf{Rand}^{l\to L}]$ in D_A's World 1. The reason it is possible for D_A to do this is that it can execute the encryption algorithm $\mathcal{E}_g(\cdot)$ of Figure 6.2, which simply requires access to the function g. If the adversary A wins, meaning breaks the encryption scheme, D_A bets that g is an instance of F, and otherwise D_A bets that g is an instance of $\mathsf{Rand}^{l\to L}$.

We stress the key point that makes this argument work. It is that the encryption function of the C-CTR scheme invokes the function F_K purely as an oracle. If it had, instead, made direct some direct use of the key K, the paradigm above would not work. The full proof follows.

Proof of Theorem 6.17: Let A be any ind-cpa-adversary attacking $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Assume A makes q oracle queries totalling μ bits, and has time-complexity t. We will design a distinguisher D_A such that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE},A} \quad \leq \quad 2 \cdot \mathbf{Adv}^{\text{prf}}_{F,D_A} \, . \tag{6.3}$$

Furthermore, D_A will make μ/L oracle queries and have time-complexity t. Now, the statement of Theorem 6.17 follows as usual, by taking maximums. So the main thing is to provide the distinguisher for which Equation (6.3) is true. This distinguisher uses A as a subroutine.

Remember that D_A takes an oracle $g\colon \{0,1\}^l \to \{0,1\}^L$. This oracle is either drawn at random from F or from $\mathsf{Rand}^{l\to L}$ and D_A does not know a priori which. To find out, D_A will use A. But remember that A too gets an oracle, namely a lr-encryption oracle. From A's point of view, this oracle is simply a subroutine: A can write, at some location, a pair of messages, and is returned a response by some entity it calls its oracle. When D_A runs A as a subroutine, it is D_A that will "simulate" the lr-encryption oracle for A, meaning D_A will provide the responses to any oracle queries that A makes. Here is the description of D_A:

Distinguisher D_A^g

$b \stackrel{R}{\leftarrow} \{0,1\}$
Run adversary A, replying to its oracle queries as follows
When A makes an oracle query (M_0, M_1) do
  $C \stackrel{R}{\leftarrow} \mathcal{E}_g(M_b)$
  Return C to A as the answer
Until A stops and outputs a bit d
If $d = b$ then return 1 else return 0

Here $\mathcal{E}_g(\cdot)$ denotes the encryption function of the generalized C-CTR scheme that we defined in Figure 6.2. The crucial fact we are exploiting here is that this function can be implemented given an oracle for g. Distinguisher D_A itself picks the challenge bit b representing the choice of worlds for A, and then sees whether or not A succeeds in guessing the value of this bit. If it does, it bets that g is an instance of F, and otherwise it bets that g is an instance of $\mathsf{Rand}^{l\to L}$. For the analysis, we claim that

$$\mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,D_A} = 1\right] = \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[F],A} \tag{6.4}$$

$$\mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{F,D_A} = 1\right] = \frac{1}{2} + \frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[\mathsf{Rand}^{l\to L}],A} . \tag{6.5}$$

We will justify these claims shortly, but first let us use them to conclude. Subtracting, as per Definition 5.4, we get

$$\begin{aligned}
\mathbf{Adv}^{\text{prf}}_{F,D_A} &= \mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,D_A} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{F,D_A} = 1\right] \\
&= \frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[F],A} - \frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[\mathsf{Rand}^{l\to L}],A} \\
&= \frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[F],A} .
\end{aligned} \tag{6.6}$$

The last inequality was obtained by applying Lemma 6.21, which told us that the term $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[\mathsf{Rand}^{l\to L}],A}$ was simply zero. Re-arranging terms gives us Equation (6.3). Now let us check the resource usage. Each computation $\mathcal{E}_g(M_b)$ requires $|M_b|/L$ applications of g, and hence the total number of queries made by D_A to its oracle g is μ/L. The time-complexity of D_A equals that of A once one takes into account the convention that time-complexity refers to the time of the entire underlying experiment. It remains to justify Equations (6.4) and (6.5).

Distinguisher D_A returns 1 when $b = d$, meaning that ind-cpa-adversary A correctly identified the world b in which it was placed, or, in the language of Section 6.5.2, made the "correct guess." The role played by D_A's world is simply to alter the encryption scheme for which this is true. When D_A is in World 1, the encryption scheme, from the point of view of A, is $\mathcal{SE}[F]$, and when D_A is in World 0, the encryption scheme, from the point of view of A, is $\mathcal{SE}[\mathsf{Rand}^{l\to L}]$. Thus, using the notation from Section 6.5.2, we have

$$\mathbf{P}\left[\mathbf{Exp}^{\text{prf-1}}_{F,D_A} = 1\right] = \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-cg}}_{\mathcal{SE}[F],A} = 1\right]$$

$$\mathbf{P}\left[\mathbf{Exp}^{\text{prf-0}}_{F,D_A} = 1\right] = \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-cg}}_{\mathcal{SE}[\mathsf{Rand}^{l\to L}],A} = 1\right] .$$

To obtain Equations (6.4) and (6.5) we can now apply Proposition 6.12. ∎

For someone unused to PRF based proofs of security the above may seem complex, but the underlying idea is actually very simple, and will be seen over and over again. It is simply that one can view the experiment of the ind-cpa-adversary attacking the encryption scheme as information about the underlying function g being used, and if the adversary has more success in the case that g is an instance of F than that g is an instance of $\mathsf{Rand}^{l\to L}$, then we have a distinguishing test between F and $\mathsf{Rand}^{l\to L}$. Let us now prove the lemma about the security of the idealized C-CTR scheme.

Proof of Lemma 6.21: The intuition is simple. When g is a random function, its value on successive counter values yields a one-time pad, a truly random and unpredictable sequence of bits. As long as the number of data bits encrypted does not exceed $L2^l$, we invoke g only on distinct values in the entire encryption process. The outputs of g are thus random. Since the data is XORed to this sequence, the adversary gets no information whatsoever about it.

Now, we must make sure that this intuition carries through in our setting. Our lemma statement makes reference to our notions of security, so we must use the setup in Section 6.5.1. The adversary A has access to a lr-encryption oracle. Since the scheme we are considering is $\mathcal{SE}[\mathsf{Rand}^{l\to L}]$, the oracle is $\mathcal{E}_g(\mathrm{LR}(\cdot,\cdot,b))$, where the function $\mathcal{E}_g$ was defined in Figure 6.2, and g is a random instance of $\mathsf{Rand}^{l\to L}$, meaning a random function.

The adversary makes some number q of oracle queries. Let $(M_{i,0}, M_{i,1})$ be the i-th query, and let n_i be the number of blocks in $M_{i,0}$. (This is the same as the number of blocks in $M_{i,1}$.) Let $M_{i,c}[j]$ be the value of the j-th L-bit block of $M_{i,c}$ for $c \in \{0,1\}$. Let C_i be the response returned by the oracle to query $(M_{i,0}, M_{i,1})$. It consists of $n_i + 1$ blocks, the first block being the l-bit binary representation of the counter at the start of the encryption $\mathcal{E}_g(M_{i,b})$, where b is the challenge bit underlying the $\mathcal{E}_g(\mathrm{LR}(\cdot,\cdot,b))$ oracle, and the other blocks are denoted $C_i[1]\ldots C_i[n_i]$. Pictorially:

$$\begin{array}{rcl}
M_{1,b} & = & M_{1,b}[1]M_{1,b}[1]\ldots M_{1,b}[n_1] \\
C_1 & = & \mathsf{NtS}_l(0)C_1[1]\ldots C_1[n_1] \\
\\
M_{2,b} & = & M_{2,b}[1]M_{2,b}[2]\ldots M_{2,b}[n_2] \\
C_2 & = & \mathsf{NtS}_l(n_1)C_2[1]\ldots C_2[n_2] \\
\\
\vdots & & \vdots \\
\\
M_{q,b} & = & M_{q,b}[1]M_{q,b}[2]\ldots M_{q,b}[n_q] \\
C_q & = & \mathsf{NtS}_l(n_1+\cdots+n_{q-1})C_q[1]\ldots C_q[n_q]
\end{array}$$

What kind of distribution do the outputs received by A have? We claim that the $n_1 + \cdots + n_q$ values $C_i[j]$ ($i = 1,\ldots,q$ and $j = 1,\ldots,n_i$) are randomly and independently distributed, not only of each other, but of the queried messages and the bit b, and moreover this is true in both worlds. Why? Here is where we use a crucial property of the CTR mode, namely that it XORs data with the value of g on a counter. We observe that according to the scheme

$$C_i[j] \quad = \quad g(\mathsf{NtS}_l(n_1+\cdots+n_{i-1}+j)) \oplus \begin{cases} M_{i,1}[j] & \text{if we are in world 1} \\ M_{i,0}[j] & \text{if we are in world 0.} \end{cases}$$

Now, we can finally see that the idea we started with is really the heart of it. The values on which g is being applied above are all distinct. So the outputs of g are all random and independent. It matters not, then, what we XOR these outputs with; what comes back is just random.

This tells us that any given output sequence from the oracle is equally likely in both worlds. Since the adversary determines its output bit based on this output sequence, its probability of its returning 1 must be the same in both worlds,

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-1}}_{\mathcal{SE}[\mathsf{Rand}^{l\to L}],A} = 1\right] \quad = \quad \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cpa-0}}_{\mathcal{SE}[\mathsf{Rand}^{l\to L}],A} = 1\right].$$

Hence A's ind-cpa-advantage is zero. ∎

Algorithm $\mathcal{E}_g(M)$	Algorithm $\mathcal{D}_f(C)$
If $\lvert M\rvert < L$ then return $\perp$	If $\lvert C\rvert < l + L$ then return $\perp$
If $\lvert M\rvert \bmod L \neq 0$ then return $\perp$	If $(\lvert C\rvert - l) \bmod L \neq 0$ then return $\perp$
Parse M as $M[1] \ldots M[n]$	Let $C[0]$ be the first l bits of C
$R \stackrel{R}{\leftarrow} \{0, 1, \ldots, 2^l - 1\}$	Parse the rest of C as $C[1] \ldots C[n]$
For $i = 1, \ldots, n$ do	$R \leftarrow \mathsf{StN}(C[0])$
$\quad C[i] \leftarrow g(\mathsf{NtS}_l(R + i)) \oplus M[i]$	For $i = 1, \ldots, n$ do
EndFor	$\quad M[i] \leftarrow g(\mathsf{NtS}_l(R + i)) \oplus C[i]$
$C[0] \leftarrow \mathsf{NtS}_l(R)$	EndFor
$C \leftarrow C[0]C[1] \ldots C[n]$	$M \leftarrow M[1] \ldots M[n]$
Return C	Return M

Figure 6.3: Version $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ of the R-CTR scheme parameterized by a family of functions G.

6.8.2 Proof of Theorem 6.18

The proof of Theorem 6.18 re-uses a lot of what we did for the proof of Theorem 6.17 above. We first look at the scheme when g is a random function, and then use the pseudorandomness of the given family F to deduce the theorem. As before we associate to a family of functions G having domain $\{0,1\}^l$ and range $\{0,1\}^L$ a parameterized version of the R-CTR scheme, $\mathcal{SE}[G] = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm simply returns a random instance of G, meaning picks a function $g \stackrel{R}{\leftarrow} G$ from family G at random, and views g as the key, and the encryption and decryption algorithms are shown in Figure 6.3. Here is the main lemma.

Lemma 6.22 Let A be any ind-cpa-adversary attacking $\mathcal{SE}[\mathsf{Rand}^{l \to L}]$. Then

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[\mathsf{Rand}^{l \to L}], A} \quad \leq \quad \frac{\mu(q-1)}{L 2^l} ,$$

where q is the number of oracle queries made by A and $\mu < L2^l$ is the total length of these queries. ▮

The proof of Theorem 6.18 given this lemma is easy at this point because it is almost identical to the above proof of Theorem 6.17. So let us finish that first, and then go on to prove Lemma 6.22.

Proof of Theorem 6.18: Let A be any ind-cpa-adversary attacking $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. Assume A makes q oracle queries totalling μ bits, and has time-complexity t. We will design a distinguisher D_A such that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}, A} \quad \leq \quad 2 \cdot \mathbf{Adv}^{\text{prf}}_{F, D_A} + \frac{\mu(q-1)}{L 2^l} .$$

Furthermore, D_A will make μ/L oracle queries and have time-complexity t. Now, the statement of Theorem 6.18 follows as usual, by taking maximums.

The code for D_A is the same as in the proof of Theorem 6.17. However note that the underlying algorithm $\mathcal{E}_g(\cdot)$ has changed, now being the one of Figure 6.3 rather than that of Figure 6.2. For the analysis, the only change is that the term

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[\mathsf{Rand}^{l \to L}], A}$$

in Equation (6.6), rather than being zero, is upper bounded as per Lemma 6.22, and thus

$$\mathbf{Adv}^{\text{prf}}_{F, D_A} \quad \geq \quad \frac{1}{2} \cdot \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[F], A} - \frac{1}{2} \cdot \frac{\mu(q-1)}{L 2^l} . \qquad (6.7)$$

The rest is as before. ▮

The above illustrates how general and generic was the "simulation" argument of the proof of Theorem 6.17. Indeed, it adapts easily not only to the randomized version of the scheme, but to the use of pseudorandom functions in many other schemes, even for different tasks like message authentication. The key point that makes it work is that the scheme itself invokes g as an oracle.

Before we prove Lemma 6.22, we will analyze a certain probabilistic game. The problem we isolate here is purely probabilistic; it has nothing to do with encryption or even cryptography.

Lemma 6.23 Let n, q, l be positive integers, and let $n_1, \ldots, n_q < 2^l$ also be positive integers. Suppose we pick q integers $r_1, \ldots, r_q$ from the range $\{0, 1, \ldots, 2^l - 1\}$, uniformly and independently at random. We consider the following $n_1 + \cdots + n_q$ numbers:

$$\begin{array}{cccc} r_1+1, & r_1+2, & \cdots, & r_1+n_1 \\ r_2+1, & r_2+2, & \cdots, & r_2+n_2 \\ \vdots & & & \vdots \\ r_q+1, & r_q+2, & \cdots, & r_q+n_q\,, \end{array}$$

where the addition is performed modulo 2^l. We say that a *collision* occurs if some two (or more) numbers in the above table are equal. Then

$$\mathbf{P}[\mathsf{Col}] \quad \leq \quad \frac{(q-1)(n_1 + \cdots + n_q)}{2^l}\,,$$

where Col denotes the event that a collision occurs. ▮

Proof of Lemma 6.23: As with many of the probabilistic settings that arise in this area, this is a question about some kind of "balls thrown in bills" setting, related to the birthday problem studied in Section A.1, and indeed a reader may find it helpful to have studied that appendix first.

Think of having 2^l bins, numbered $0, 1, \ldots, 2^l - 1$. We have q balls, numbered $1, \ldots, q$. For each ball we choose a random bin which we call r_i. We choose the bins one by one, so that we first choose r_1, then r_2, and so on. When we have thrown in the first ball, we have defined the first row of the above table, namely the values $r_1 + 1, \ldots, r_1 + n_1$. Then we pick the assignment r_2 of the bin for the second ball. This defines the second row of the table, namely the values $r_2 + 1, \ldots, r_2 + n_2$. A collision occurs if any value in the second row equals some value in the first row. We continue, up to the q-th ball, each time defining a row of the table, and are finally interested in the probability that a collision occurred somewhere in the process. To upper bound this, we want to write this probability in such a way that we can do the analysis step by step, meaning view it in terms of having thrown, and fixed, some number of balls, and seeing whether there is a collision when we throw in one more ball. To this end let Col_i denote the event that there is a collision somewhere in the first i rows of the table, for $i = 1, \ldots, q$. Let NoCol_i denote the event that there is no collision in the first i rows of the table, for $i = 1, \ldots, q$. Then by conditioning we have

$$\begin{aligned} \mathbf{P}[\mathsf{Col}] &= \mathbf{P}[\mathsf{Col}_q] \\ &= \mathbf{P}[\mathsf{Col}_{q-1}] + \mathbf{P}[\mathsf{Col}_q \mid \mathsf{NoCol}_{q-1}] \cdot \mathbf{P}[\mathsf{NoCol}_{q-1}] \\ &\leq \mathbf{P}[\mathsf{Col}_{q-1}] + \mathbf{P}[\mathsf{Col}_q \mid \mathsf{NoCol}_{q-1}] \\ &\leq \vdots \\ &\leq \mathbf{P}[\mathsf{Col}_1] + \sum_{i=2}^{q} \mathbf{P}[\mathsf{Col}_i \mid \mathsf{NoCol}_{i-1}] \\ &= \sum_{i=2}^{q} \mathbf{P}[\mathsf{Col}_i \mid \mathsf{NoCol}_{i-1}]\,. \end{aligned}$$

Thus we need to upper bound the chance of a collision upon throwing the i-th ball, given that there was no collision created by the first $i-1$ balls. Then we can sum up the quantities obtained and obtain our bound.

We claim that for any $i = 2, \ldots, q$ we have

$$\mathbf{P}\left[\mathsf{Col}_i \mid \mathsf{NoCol}_{i-1}\right] \quad \leq \quad \frac{(i-1)n_i + n_{i-1} + \cdots + n_1}{2^l} \; . \tag{6.8}$$

Let us first see why this proves the lemma and then return to justify it. From the above and Equation (6.8) we have

$$\begin{aligned}
\mathbf{P}\left[\mathsf{Col}\right] &\leq \sum_{i=2}^{q} \mathbf{P}\left[\mathsf{Col}_i \mid \mathsf{NoCol}_{i-1}\right] \\
&\leq \sum_{i=2}^{q} \frac{(i-1)n_i + n_{i-1} + \cdots + n_1}{2^l} \\
&= \frac{(q-1)(n_1 + \cdots + n_q)}{2^l} \; .
\end{aligned}$$

How did we do the last sum? The term n_i occurs with weight $i-1$ in the i-th term of the sum, and then with weight 1 in the j-th term of the sum for $j = i+1, \ldots, q$. So its total weight is $(i-1) + (q-i) = q-1$.

It remains to prove Equation (6.8). To get some intuition about it, begin with the cases $i = 1, 2$. When we throw in the first ball, the chance of a collision is zero, since there is no previous row with which to collide, so that is simple. When we throw in the second, what is the chance of a collision? The question is, what is the probability that one of the numbers $r_2 + 1, \ldots, r_2 + n_2$ defined by the second ball is equal to one of the numbers $r_1 + 1, \ldots, r_1 + n_1$ already in the table? View r_1 as fixed. Observe that a collision occurs if and only if $r_1 - n_2 + 1 \leq r_2 \leq r_1 + n_1 - 1$. So there are $(r_1 + n_1 - 1) - (r_1 - n_2 + 1) + 1 = n_1 + n_2 - 1$ choices of r_2 that could yield a collision. This means that $\mathbf{P}\left[\mathsf{Col}_2 \mid \mathsf{NoCol}_1\right] \leq (n_2 + n_1 - 1)/2^l$.

We need to extend this argument as we throw in more balls. So now suppose $i-1$ balls have been thrown in, where $2 \leq i \leq q$, and suppose there is no collision in the first $i-1$ rows of the table. We throw in the i-th ball, and want to know what is the probability that a collision occurs. We are viewing the first $i-1$ rows of the table as fixed, so the question is just what is the probability that one of the numbers defined by r_i equals one of the numbers in the first $i-1$ rows of the table. A little thought shows that the worst case (meaning the case where the probability is the largest) is when the existing $i-1$ rows are well spread-out. We can upper bound the collision probability by reasoning just as above, except that there are $i-1$ different intervals to worry about rather than just one. The i-th row can intersect with the first row, or the second row, or the third, and so on, up to the $(i-1)$-th row. So we get

$$\begin{aligned}
\mathbf{P}\left[\mathsf{Col}_i \mid \mathsf{NoCol}_{i-1}\right] &\leq \frac{(n_i + n_1 - 1) + (n_i + n_2 - 1) + \cdots + (n_i + n_{i-1} - 1)}{2^l} \\
&= \frac{(i-1)n_i + n_{i-1} + \cdots + n_1 - (i-1)}{2^l} \; ,
\end{aligned}$$

and Equation (6.8) follows by just dropping the negative term in the above. ▮

Let us now extend the proof of Lemma 6.21 to prove Lemma 6.22.

Proof of Lemma 6.22: Recall that the idea of the proof of Lemma 6.21 was that when g is a random function, its value on successive counter values yields a one-time pad. This holds whenever g is applied on some set of distinct values. In the counter case, the inputs to g are always distinct. In the randomized case they may not be distinct. The approach is to consider the event that they are distinct, and say that in that case the adversary has no advantage; and on the other hand, while it may have a large advantage in the other case, that case does not happen often. We now flush all this out in more detail.

The adversary makes some number q of oracle queries. Let $(M_{i,0}, M_{i,1})$ be the i-th query, and let n_i be the number of blocks in $M_{i,0}$. (This is the same as the number of blocks in $M_{i,1}$.) Let $M_{i,c}[j]$ be the value of the j-th L-bit block of $M_{i,c}$ for $c \in \{0,1\}$. Let C_i be the response returned by the oracle to query $(M_{i,0}, M_{i,1})$. It consists of $n_i + 1$ blocks, the first block being the l-bit binary representation of the random integer $r_i \in \{0, 1, \ldots, 2^l - 1\}$ chosen by $\mathcal{E}_g(M_{i,b})$, where b is the challenge bit underlying the $\mathcal{E}_g(\mathrm{LR}(\cdot,\cdot,b))$ oracle, and the other blocks are denoted $C_i[1] \ldots C_i[n_i]$. Pictorially:

$$\begin{array}{rcl}
M_{1,b} & = & M_{1,b}[1]M_{1,b}[1]\ldots M_{1,b}[n_1] \\
C_1 & = & \mathsf{NtS}_l(r_1)C_1[1]\ldots C_1[n_1] \\
\\
M_{2,b} & = & M_{2,b}[1]M_{2,b}[2]\ldots M_{2,b}[n_2] \\
C_2 & = & \mathsf{NtS}_l(r_2)C_2[1]\ldots C_2[n_2] \\
\\
\vdots & \vdots & \\
\\
M_{q,b} & = & M_{q,b}[1]M_{q,b}[2]\ldots M_{q,b}[n_q] \\
C_q & = & \mathsf{NtS}_l(r_q)C_q[1]\ldots C_q[n_q]
\end{array}$$

Let NoCol be the event that the following $n_1 + \cdots + n_q$ values are all distinct:

$$\begin{array}{cccc}
r_1+1, & r_1+2, & \cdots, & r_1+n_1 \\
r_2+1, & r_2+2, & \cdots, & r_2+n_2 \\
\vdots & & & \vdots \\
r_q+1, & r_q+2, & \cdots, & r_q+n_q
\end{array}$$

Let Col be the complement of the event NoCol, meaning the event that the above table contains at least two values that are the same. It is useful for the analysis to introduce the following shorthand:

$$\begin{array}{rcl}
\mathbf{P}_0\left[\cdot\right] & = & \text{The probability of event "}\cdot\text{" in world 0} \\
\mathbf{P}_0\left[\cdot\right] & = & \text{The probability of event "}\cdot\text{" in world 1 .}
\end{array}$$

We will use the following three claims, which are proved later. The first claim says that the probability of a collision in the above table does not depend on which world we are in.

Claim 1: $\mathbf{P}_1\left[\mathsf{Col}\right] = \mathbf{P}_0\left[\mathsf{Col}\right]$. $\Box$

The second claim says that A has zero advantage in winning the left-or-right game in the case that no collisions occur in the table. Namely, its probability of outputting one is identical in these two world under the assumption that no collisions have occurred in the values in the table.

Claim 2: $\mathbf{P}_0\left[A = 1 \mid \mathsf{NoCol}\right] = \mathbf{P}_1\left[A = 1 \mid \mathsf{NoCol}\right]$. $\Box$

We can say nothing about the advantage of A if a collision does occur in the table. That is, it might be big. However, it will suffice to know that the probability of a collision is small. Since we already know that this probability is the same in both worlds (Claim 1) we bound it just in world 0:

Claim 3: $\mathbf{P}_0\left[\mathsf{Col}\right] \le \dfrac{\mu(q-1)}{L2^l}$. $\Box$

Let us see how these put together complete the proof of the lemma, and then go back and prove them.

Proof of Lemma given Claims: It is a simple conditioning argument:

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}[\mathsf{Rand}^{l\to L}],A}$$

$$
\begin{aligned}
&= \mathbf{P}_1[A=1] - \mathbf{P}_0[A=1] \\
&= \mathbf{P}_1[A=1 \mid \mathsf{Col}] \cdot \mathbf{P}_1[\mathsf{Col}] + \mathbf{P}_1[A=1 \mid \mathsf{NoCol}] \cdot \mathbf{P}_1[\mathsf{NoCol}] \\
&\qquad - \mathbf{P}_0[A=1 \mid \mathsf{Col}] \cdot \mathbf{P}_0[\mathsf{Col}] - \mathbf{P}_0[A=1 \mid \mathsf{NoCol}] \cdot \mathbf{P}_0[\mathsf{NoCol}]
\end{aligned}
$$

Using Claim 1 and Claim 2, the above equals

$$
\begin{aligned}
&= (\mathbf{P}_1[A=1 \mid \mathsf{Col}] - \mathbf{P}_0[A=1 \mid \mathsf{Col}]) \cdot \mathbf{P}_0[\mathsf{Col}] \\
&\leq \mathbf{P}_0[\mathsf{Col}] .
\end{aligned}
$$

In the last step we simply bounded the parenthesized expression by 1. Now apply Claim 3, and we are done. □

It remains to prove the three claims.

Proof of Claim 1: The event NoCol depends only on the random values $r_1, \ldots, r_q$ chosen by the encryption algorithm $\mathcal{E}_g(\cdot)$. These choices, however, are made in exactly the same way in both worlds. The difference in the two worlds is what message is encrypted, not how the random values are chosen. □

Proof of Claim 2: Given the event NoCol, we have that, in either game, the function g is evaluated at a new point each time it is invoked. (Here we use the assumption that $\mu < L2^l$, since otherwise there may be wraparound in even a single query.) Thus the output is randomly and uniformly distributed over $\{0,1\}^L$, independently of anything else. That means the reasoning from the counter-based scheme as given in Lemma 6.21 applies. Namely we observe that according to the scheme

$$
C_i[j] \quad = \quad g(\mathsf{NtS}_l(r_i + j)) \oplus \begin{cases} M_{i,1}[j] & \text{if we are in world 1} \\ M_{i,0}[j] & \text{if we are in world 0.} \end{cases}
$$

Thus each cipher block is a message block XORed with a random value. A consequence of this is that each cipher block has a distribution that is independent of any previous cipher blocks and of the messages. □

Proof of Claim 3: This follows from Lemma 6.23. We simply note that $n_1 + \cdots + n_q = \mu/L$. □

This concludes the proof. ▮

6.9 Security of CBC against chosen-plaintext attack

CBC encryption, presented in Scheme 6.4, is the most popular mode. We looked at the CTR modes first because they are easier to analyze. Indeed, we will not present the (more complex) analysis of CBC mode encryption here, but we will state the result. The proof can be found in [20].

Theorem 6.24 [**20**] Suppose Let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding CBC symmetric encryption scheme as described in Scheme 6.4. Then for any t, q, μ we have

$$
\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t,q,\mu) \quad \leq \quad 2 \cdot \mathbf{Adv}^{\text{prf}}_{F}(t,q',lq') \; + \; \frac{2\mu^2}{l^2 2^l} \; ,
$$

where $q' = \mu/L$. ▮

Notice that if all messages are of n blocks then $\mu = nql$ so the additive term above is $O(n^2q^2/2^l)$.

So what about we try to improve this using counters? We can do counters with CBC too! But this is no good. It is a nice exercise to try to find the attack. Note this is true even for variations in which the counter is incremented by the number of message blocks rather than by just one per message.

How about if the counter is encrypted? Then these attacks appear to vanish, but birthdays are back. So we don't really win out over the random IV scheme. In fact the two become very similar identical: $f(c)$ where c is a counter is exactly a random number. (They are not identical because c may also arise somewhere in the plaintext.)

In analyzing this, see how to think about these schemes via the modelling we have been doing. Ie think of f as a random function. In that world the random CBC and encrypted counter modes are similar.

6.10 Indistinguishability under chosen-ciphertext attack

Above, we have considered privacy under chosen-plaintext attack. Sometimes we want to consider privacy when the adversary is capable of mounting a stronger type of attack, namely a chosen-ciphertext attack. In this type of attack, an adversary has access to a decryption oracle. It can feed this oracle a ciphertext, and get back the corresponding plaintext.

How might such a situation arise? We could imagine that an adversary at some point gains temporary access to the equipment performing decryption. It can feed the equipment ciphertexts and see what plaintexts emerge. (We assume it cannot directly extract the key from the equipment, however.)

If an adversary has access to a decryption oracle, security at first seems moot, since after all it can decrypt anything it wants. To create a meaningful notion of security, we put a restriction on the use of the decryption oracle. To see what this is, let us look closer at the formalization. As in the case of chosen-plaintext attacks, we consider two worlds:

World 0: The adversary is provided the oracle $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,0))$ as well as the oracle $\mathcal{D}_K(\cdot)$.

World 1: The adversary is provided the oracle $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,1))$ as well as the oracle $\mathcal{D}_K(\cdot)$.

The adversary's goal is the same as in the case of chosen-plaintext attacks: it wants to figure out which world it is in. There is one easy way to do this. Namely, query the lr-encryption oracle on two distinct, equal length messages M_0, M_1 to get back a ciphertext C, and now call the decryption oracle on C. If the message returned by the decryption oracle is M_0 then the adversary is in World 0, and if the message returned by the decryption oracle is M_1 then the adversary is in World 1. The restriction we impose is simply that this call to the decryption oracle is not allowed. More generally, call a query C to the decryption oracle *illegitimate* if C was previously been returned by the lr-encryption oracle. Then, only legitimate queries are allowed. In the formalization below, the experiment simply returns 0 if the adversary makes an illegitimate query. (We clarify that a query C is legitimate if C is returned by the lr-encryption oracle *after* C was queried to the decryption oracle. It is only when C was previously output by the lr-encryption oracle that it is illegitimate.)

This restriction still leaves the adversary with a lot of power. Typically, a successful chosen-ciphertext attack proceeds by taking a ciphertext C returned by the lr-encryption oracle, modifying it into a related ciphertext C', and querying the decryption oracle with C'. The attacker seeks to create C' in such a way that its decryption tells the attacker what was the message underlying M. We will see this illustrated in Section 6.11 below.

The model we are considering here might seem quite artificial. If an adversary has access to a decryption oracle, how can we prevent it from calling the decryption oracle on certain messages? The restriction might arise due to the adversary's having access to the decryption equipment for a limited period of time. We imagine that after it has lost access to the decryption equipment, it sees some ciphertexts, and we are capturing the security of these ciphertexts in the face of previous access to the decryption oracle. Further motivation for the model will emerge when we see how encryption schemes are used in protocols. We will see that when an encryption scheme is used in an authenticated key exchange protocol, the adversary effectively has the ability to mount chosen-ciphertext attacks of the type we are discussing. For now let us just provide the definition and exercise it.

Definition 6.25 Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme, let $b \in \{0,1\}$, and let A be an algorithm that has access to two oracles and returns a bit. We consider the following experiment:

Experiment $\mathbf{Exp}^{\text{ind-cca-}b}_{\mathcal{SE},A}$
  $K \stackrel{R}{\leftarrow} \mathcal{K}$
  $d \leftarrow A^{\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))\,,\,\mathcal{D}_K(\cdot)}$
  If A queried $\mathcal{D}_K(\cdot)$ on a ciphertext previously returned by $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))$
  Then return 0
  Else return d

The *ind-cca-advantage* of A is defined as

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE},A} \quad = \quad \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cca-1}}_{\mathcal{SE},A} = 1\right] - \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cca-0}}_{\mathcal{SE},A} = 1\right].$$

For any $t, q_e, \mu_e, q_d, \mu_d$ we define the *ind-cca-advantage* of $\mathcal{SE}$ via

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE}}(t, q_e, \mu_e, q_d, \mu_d) \quad = \quad \max_A \left\{ \mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE},A} \right\}$$

where the maximum is over all A having time-complexity t, making to the lr-encryption oracle at most q_e queries the sum of whose lengths is at most μ_e bits, and making to the decryption oracle at most q_d queries the sum of whose lengths is at most μ_d bits. ∎

The conventions with regard to resource measures are the same as those used in the case of chosen-plaintext attacks. In particular, the length of a query M_0, M_1 to the lr-encryption oracle is the length is defined as the length of M_0, and the time-complexity is the execution time of the entire experiment plus the size of the code of the adversary.

We consider an encryption scheme to be "secure against chosen-ciphertext attack" if a "reasonable" adversary cannot obtain "significant" advantage in distinguishing the cases $b = 0$ and $b = 1$ given access to the oracles, where reasonable reflects its resource usage. The technical notion is called indistinguishability under chosen-ciphertext attack, denoted IND-CCA.

6.11 Example chosen-ciphertext attacks

Chosen-ciphertext attacks are powerful enough to break all the standard modes of operation, even those like CTR and CBC that (as we will see later) are secure against chosen-plaintext attack. The one-time pad scheme is also vulnerable to a chosen-ciphertext attack: perfect security only takes into account chosen-plaintext attacks. Let us now illustrate a few chosen-ciphertext attacks.

6.11.1 Attack on CTR

Let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the associated R-CTR symmetric encryption scheme as described in Scheme 6.5. The weakness of the scheme that makes it susceptible to a chosen-ciphertext attack is the following. Say $C[0]C[1]$ is a ciphertext of some L-bit message M, and we flip bit i of $C[1]$, resulting in a new ciphertext $C[0]C'[1]$. Let M' be the message obtained by decrypting the new ciphertext. Then M' equals M with the i-th bit flipped. (You should check that you understand why by looking at Scheme 6.5.) Thus, by making a decryption oracle query of $C[0]C'[1]$ one can learn M' and thus M. In the following, we show how this idea can be applied to break the scheme in our model by figuring out in which world an adversary has been placed.

Proposition 6.26 Let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a family of functions and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding R-CTR symmetric encryption scheme as described in Scheme 6.5. Then

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE}}(t, 1, L, 1, l + L) \quad = \quad 1$$

for $t = O(l + L)$ plus the time for one application of F. ∎

The advantage of this adversary is 1 even though it uses hardly any resources: just one query to each oracle. That is clearly an indication that the scheme is insecure.

Proof of Proposition 6.26: We will present an adversary algorithm A, having time-complexity t, making 1 query to its lr-encryption oracle, this query being of length L, making 1 query to its decryption oracle, this query being of length $l + L$, and having

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE},A} \;=\; 1\,.$$

The Proposition follows.

Remember the the lr-encryption oracle $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))$ takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b. The goal of A is to determine the value of b. Our adversary works like this:

Adversary $A^{\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))\,,\,\mathcal{D}_K(\cdot)}$
  $M_0 \leftarrow 0^L$; $M_1 \leftarrow 1^L$
  $C[0]C[1] \leftarrow \mathcal{E}_K(\mathrm{LR}(M_0, M_1, b))$
  $C'[1] \leftarrow C[1]\oplus 1^L$; $C' \leftarrow C[0]C'[1]$
  $M \leftarrow \mathcal{D}_K(C')$
  If $M = M_0$ then return 1 else return 0

The adversary's single lr-encryption oracle query is the pair of distinct messages M_0, M_1, each one block long. It is returned a ciphertext $C[0]C[1]$. It flips the bits of $C[1]$ to get $C'[1]$ and then feeds the ciphertext $C[0]C'[1]$ to the decryption oracle. It bets on World 1 if it gets back M_0, and otherwise on World 0. It is important that $C[0]C'[1] \neq C[0]C[1]$, so the decryption oracle query is legitimate. Now, we claim that

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cca-1}}_{\mathcal{SE},A} = 1\right] \;=\; 1$$

$$\mathbf{P}\left[\mathbf{Exp}^{\text{ind-cca-0}}_{\mathcal{SE},A} = 1\right] \;=\; 0\,.$$

Hence $\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE},A} = 1 - 0 = 1$. And A achieved this advantage by making just one lr-encryption oracle query, whose length, which as per our conventions is just the length of M_0, is L bits, and just one decryption oracle query, whose length is $l + L$ bits. So $\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE}}(t, 1, L, 1, l + L) = 1$.

Why are the two equations claimed above true? You have to return to the definitions of the quantities in question, as well as the description of the scheme itself, and walk it through. In World 1, meaning when $b = 1$, let $C[0]C[1]$ denote the ciphertext returned by the lr-encryption oracle, and let $R = \mathsf{StN}(C[0])$. Then

$$C[1] \;=\; F_K(\mathsf{NtS}_l(R+1))\oplus M_1 \;=\; F_K(\mathsf{NtS}_l(R+1))\oplus 1^L\,.$$

Now notice that

$$\begin{aligned}
M &= \mathcal{D}_K(C[0]C'[1]) \\
&= F_K(\mathsf{NtS}_l(R+1))\oplus C'[1] \\
&= F_K(\mathsf{NtS}_l(R+1))\oplus C[1]\oplus 1^L \\
&= F_K(\mathsf{NtS}_l(R+1))\oplus(F_K(\mathsf{NtS}_l(R+1))\oplus 1^L)\oplus 1^L \\
&= 0^L \\
&= M_0\,.
\end{aligned}$$

Thus, the decryption oracle will return M_0, and thus A will return 1. In World 0, meaning when $b = 0$, let $C[0]C[1]$ denote the ciphertext returned by the lr-encryption oracle, and let $R = \mathsf{StN}(C[0])$. Then

$$C[1] \;=\; F_K(\mathsf{NtS}_l(R+1)) \oplus M_0 \;=\; F_K(\mathsf{NtS}_l(R+1)) \oplus 0^L \;.$$

Now notice that

$$\begin{aligned}
M &= \mathcal{D}_K(C[0]C'[1]) \\
&= F_K(\mathsf{NtS}_l(R+1)) \oplus C'[1] \\
&= F_K(\mathsf{NtS}_l(R+1)) \oplus C[1] \oplus 1^L \\
&= F_K(\mathsf{NtS}_l(R+1)) \oplus (F_K(\mathsf{NtS}_l(R+1)) \oplus 0^L) \oplus 1^L \\
&= 1^L \\
&= M_1 \;.
\end{aligned}$$

Thus, the decryption oracle will return M_1, and thus A will return 0, meaning will return 1 with probability zero. ∎

An attack on C-CTR (cf. Scheme 6.6) is similar, and is left to the reader.

6.11.2 Attack on CBC

Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the associated CBC symmetric encryption scheme as described in Scheme 6.4. The weakness of the scheme that makes it susceptible to a chosen-ciphertext attack is the following. Say $C[0]C[1]$ is a ciphertext of some l-bit message M, and we flip bit i of the IV $C[0]$, resulting in a new ciphertext $C'[0]C[1]$. Let M' be the message obtained by decrypting the new ciphertext. Then M' equals M with the i-th bit flipped. (You should check that you understand why by looking at Scheme 6.4.) Thus, by making a decryption oracle query of $C'[0]C[1]$ one can learn M' and thus M. In the following, we show how this idea can be applied to break the scheme in our model by figuring out in which world an adversary has been placed.

Proposition 6.27 Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be the corresponding CBC symmetric encryption scheme as described in Scheme 6.4. Then

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE}}(t, 1, l, 1, 2l) \;=\; 1$$

for $t = O(l)$ plus the time for one application of F. ∎

The advantage of this adversary is 1 even though it uses hardly any resources: just one query to each oracle. That is clearly an indication that the scheme is insecure.

Proof of Proposition 6.27: We will present an adversary algorithm A, having time-complexity t, making 1 query to its lr-encryption oracle, this query being of length l, making 1 query to its decryption oracle, this query being of length $2l$, and having

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE},A} \;=\; 1 \;.$$

The Proposition follows.

Remember the the lr-encryption oracle $\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))$ takes input a pair of messages, and returns an encryption of either the left or the right message in the pair, depending on the value of b. The goal of A is to determine the value of b. Our adversary works like this:

Adversary $A^{\mathcal{E}_K(\mathrm{LR}(\cdot,\cdot,b))\,,\,\mathcal{D}_K(\cdot)}$
$M_0 \leftarrow 0^l$; $M_1 \leftarrow 1^l$
$C[0]C[1] \leftarrow \mathcal{E}_K(\mathrm{LR}(M_0, M_1, b))$
$C'[0] \leftarrow C[0] \oplus 1^L$; $C' \leftarrow C'[0]C[1]$
$M \leftarrow \mathcal{D}_K(C')$
If $M = M_0$ then return 1 else return 0

The adversary's single lr-encryption oracle query is the pair of distinct messages M_0, M_1, each one block long. It is returned a ciphertext $C[0]C[1]$. It flips the bits of the IV $C[0]$ to get a new IV $C'[0]$ and then feeds the ciphertext $C'[0]C[1]$ to the decryption oracle. It bets on World 1 if it gets back M_0, and otherwise on World 0. It is important that $C'[0]C[1] \neq C[0]C[1]$, so the decryption oracle query is legitimate. Now, we claim that

$$\begin{aligned} \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cca-1}}_{\mathcal{SE},A} = 1\right] &= 1 \\ \mathbf{P}\left[\mathbf{Exp}^{\text{ind-cca-0}}_{\mathcal{SE},A} = 1\right] &= 0\,. \end{aligned}$$

Hence $\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE},A} = 1 - 0 = 1$. And A achieved this advantage by making just one lr-encryption oracle query, whose length, which as per our conventions is just the length of M_0, is l bits, and just one decryption oracle query, whose length is $2l$ bits. So $\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE}}(t, 1, l, 1, 2l) = 1$.

Why are the two equations claimed above true? You have to return to the definitions of the quantities in question, as well as the description of the scheme itself, and walk it through. In World 1, meaning when $b = 1$, the lr-encryption oracle returns $C[0]C[1]$ with

$$C[1] \;=\; E_K(C[0] \oplus M_1) \;=\; E_K(C[0] \oplus 1^l)\,.$$

Now notice that

$$\begin{aligned} M &= \mathcal{D}_K(C'[0]C[1]) \\ &= E_K^{-1}(C[1]) \oplus C'[0] \\ &= E_K^{-1}(E_K(C[0] \oplus 1^l)) \oplus C'[0] \\ &= (C[0] \oplus 1^l) \oplus C'[0] \\ &= (C[0] \oplus 1^l) \oplus (C[0] \oplus 1^l) \\ &= 0^l \\ &= M_0\,. \end{aligned}$$

Thus, the decryption oracle will return M_0, and thus A will return 1. In World 0, meaning when $b = 0$, the lr-encryption oracle returns $C[0]C[1]$ with

$$C[1] \;=\; E_K(C[0] \oplus M_0) \;=\; E_K(C[0] \oplus 0^l)\,.$$

Now notice that

$$\begin{aligned} M &= \mathcal{D}_K(C'[0]C[1]) \\ &= E_K^{-1}(C[1]) \oplus C'[0] \\ &= E_K^{-1}(E_K(C[0] \oplus 0^l)) \oplus C'[0] \\ &= (C[0] \oplus 0^l) \oplus C'[0] \\ &= (C[0] \oplus 0^l) \oplus (C[0] \oplus 1^l) \end{aligned}$$

$$= 1^l$$
$$= M_1 .$$

Thus, the decryption oracle will return M_1, and thus A will return 0, meaning will return 1 with probability zero. ∎

6.12 Other methods for symmetric encryption

6.12.1 Generic encryption with pseudorandom functions

There is a general way to encrypt with pseudorandom functions. Suppose you want to encrypt m bit messages. (Think of m as large.) Suppose we have a pseudorandom function family F in which each key K specifies a function F_K mapping l bits to m bits, for some fixed but quite large value l. Then we can encrypt M via $\mathcal{E}_K(M) = (r, F_K(r) \oplus M)$ for random r. We decrypt (r, C) by computing $M = F_K(r) \oplus C$. This is the method of [92].

Theorem 6.28 [92] Suppose F is a pseudorandom function family with output length m. Then the scheme $(\mathcal{E}, \mathcal{D})$ define above is a secure private key encryption scheme for m-bit messages. ∎

The difference between this and the CBC and XOR methods is that in the latter, we only needed a PRF mapping l bits to l bits for some fixed l independent of the message length. One way to get such a PRF is to use DES or some other block cipher. Thus the CBC and XOR methods result in efficient encryption. To use the general scheme we have just defined we need to constructing PRFs that map l bits to m bits for large m.

There are several approaches to constructing "large" PRFs, depending on the efficiency one wants and what assumptions one wants to make. We have seen in Chapter 5 that pseudorandom function families can be built given one-way functions. Thus we could go this way, but it is quite inefficient. Alternatively, we could try to build these length extending PRFs out of given fixed length PRFs. See Section 5.11.

6.12.2 Encryption with pseudorandom bit generators

A pseudorandom bit generator is a deterministic function G which takes a k-bit seed and produces a $p(k) > k$ bit sequence of bits that looks pseudorandom. These object were defined and studied in Chapter 3. Recall the property they have is that no efficient algorithm can distinguish between a random $p(k)$ bit string and the string $G(K)$ with random K.

Recall the one-time pad encryption scheme: we just XOR the message bits with the pad bits. The problem is we run out of pad bits very soon. Pseudorandom bit generators provide probably the most natural way to get around this. If G is a pseudorandom bit generator and K is the k-bit shared key, the parties implicitly share the long sequence $G(K)$. Now, XOR message bits with the bits of $G(K)$. Never use an output bit of $G(K)$ more than once. Since we can stretch to any polynomial length, we have enough bits to encrypt.

More precisely, the parties maintain a counter N, initially 0. Let $G_i(K)$ denote the i-th bit of the output of $G(K)$. Let M be the message to encrypt. Let M_i be its i-th bit, and let n be its length. The sender computes $C_i = G_{N+i}(K) \oplus M_i$ for $i = 1, \ldots, n$ and lets $C = C_1 \ldots C_n$ be the ciphertext. This is transmitted to the receiver. Now the parties update the counter via $N \leftarrow N + n$. The total number of bits that can be encrypted is the number $p(k)$ of bits output by the generator. One can show, using the definition of PRBGs, that this works:

Theorem 6.29 If G is a secure pseudorandom bit generator then the above is a secure encryption scheme. ∎

One seeming disadvantage of using a PRBG is that the parties must maintain a common, synchronized counter, since both need to know where they are in the sequence $G(K)$. (Note that the schemes we have discussed above avoid this. Although some of the schemes above may optionally use a counter instead of a random value, this counter is not a synchronized one: the sender maintains a counter, but the receiver does not, and doesn't care that the sender thinks of counters.) To get around this, we might have the sender send the current counter value N (in the clear) with each message. If authentication is being used, the value N should be authenticated.

The more major disadvantage is that the pseudorandom sequence $G(K)$ may not have random access. To produce the i-th bit one may have to start from the beginning and produce all bits up to the i-th one. (This means the time to encrypt M depends on the number and length of message encrypted in the past, not a desirable feature.) Alternatively the sequence $G(K)$ may be pre-computed and stored, but this uses a lot of storage. Whether this drawback exists or not depends of course on the choice of PRBG G.

So how do we get pseudorandom bit generators? We saw some number theoretic constructions in Chapter 3. These are less efficient than block cipher based methods, but are based on different kinds of assumptions which might be preferable. More importantly, though, these constructions have the drawback that random access is not possible. Alternatively, one could build pseudorandom bit generators out of finite PRFs. This can be done so that random access is possible. However the resulting encryption scheme ends up being not too different from the XOR scheme with a counter so it isn't clear it is worth a separate discussion.

6.12.3 Encryption with one-way functions

We saw in Chapter 3 that pseudorandom bit generators exist if one-way functions exist [109]. It is also known that given any secure private key encryption scheme one can construct a one-way function [110]. Thus we have the following.

Theorem 6.30 There exists a secure private key encryption scheme if and only if there exists a one-way function. ∎

We will see later that the existence of secure public key encryption schemes requires different kinds of assumptions, namely the existence of primitives with "trapdoors."

6.13 Historical Notes

The pioneering work on the theory of encryption is that of Goldwasser and Micali [98], with refinements by [141, 91]. This body of work is however in the asymmetric (ie. public key) setting, and uses the asymptotic framework of polynomial-time adversaries and negligible success probabilities. The treatment of symmetric encryption we are using is from citeBellareRo:symmetric. In particular Definition 6.1 and the concrete security framework are from [20]. The analysis of the CTR mode encryption schemes, as given in Theorems 6.17 and 6.18, is also from [20].

6.14 Exercises and Problems

Problem 6.31 Formalize a notion of security against key-recovery for symmstric encryption schemes, and prove an analogue of Proposition 6.16. ∎

Problem 6.32 Let $l \geq 1$ and $m \geq 2$ be integers, and let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a given symmetric encryption scheme whose associated plaintext space is $\{0,1\}^l$, meaning one can safely encrypt only messages of length l. In order to be able to encrypt longer messages, we define a new symmetric encryption scheme $\mathcal{SE}^{(m)} = (\mathcal{K}, \mathcal{E}^{(m)}, \mathcal{D}^{(m)})$ having the same key generation algorithm as that of $\mathcal{SE}$, plaintext space $\{0,1\}^{lm}$, and encryption and decryption algorithms as follows:

Algorithm $\mathcal{E}_K^{(m)}(M)$	Algorithm $\mathcal{D}_K^{(m)}(C)$

```
Algorithm E_K^(m)(M)                     Algorithm D_K^(m)(C)
    Parse M as M[1]...M[m]                   Parse C as (C[1],...,C[m])
    For i = 1 to m do                        For i = 1 to m do
        C[i] ← E_K(M[i])                         M[i] ← D_K(C[i])
    EndFor                                       If M[i] = ⊥ then return ⊥
    C ← (C[1],...,C[m])                      EndFor
    Return C                                 M ← M[1]...M[m]
                                             Return M
```

Here M is lm bits long. For encryption, M is broken into a sequence of blocks $M = M[1]\ldots M[m]$, each block being l-bits long. For decryption, C is parsed as a sequence of m strings, each c bits long, where c denotes the length of a ciphertext in scheme $\mathcal{SE}$. If any component ciphertexts $C[i]$ is invalid (meaning $\mathcal{D}_K$ returns $\perp$ for it) then the entire ciphertext $(C[1],\ldots,C[m])$ is declared invalid.

(a) Show that

$$\mathbf{Adv}^{\text{ind-cca}}_{\mathcal{SE}^{(m)}}(t,1,lm,1,cm) \quad = \quad 1$$

for some small t that you must specify.

(b) Show that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}^{(m)}}(t,q,lmq) \quad \le \quad \mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t,mq,lmq)$$

for any t,q.

Part **(a)** says that $\mathcal{SE}^{(m)}$ is insecure against chosen-ciphertext attack. Note this is true regardless of the security properties of $\mathcal{SE}$, which may itself be secure against chosen-ciphertext attack. Part **(b)** says that if $\mathcal{SE}$ is secure against chosen-plaintext attack, then so is $\mathcal{SE}^{(m)}$. ∎

Problem 6.33 Let $E\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher. Operating it in CBC mode with IV a counter yields a stateful symmetric encryption scheme, $\mathcal{SE} = (\mathcal{K},\mathcal{E},\mathcal{D})$. The key generation algorithm simply returns a random key for the block cipher, meaning it picks a random k-bit string key and returns it, so that the key space is $\{0,1\}^k$. The encryption and decryption algorithms are as follows:

```
Algorithm E_K(M)                              Algorithm D_K(C)
    If |M| < l then return ⊥                      If |C| < 2l then return ⊥
    If |M| mod l ≠ 0 then return ⊥                If |C| mod l ≠ 0 then return ⊥
    Parse M as M[1]...M[n]                        Parse C as C[0]C[1]...C[n]
    C[0] ←R NtS_l(ctr)                            For i = 1,...,n do
    For i = 1,...,n do                                M[i] ← E_K^{-1}(C[i])⊕C[i−1]
        C[i] ← F_K(C[i−1]⊕M[i])                   EndFor
    EndFor                                        M ← M[1]...M[n]
    C ← C[0]C[1]...C[n]                           Return M
    ctr ← ctr + n
    Return C
```

Parsing M means that we divide it into l-bit blocks and let $M[i]$ denote the i-th such block. In parsing C we also divide it into l-bit blocks, but this time the blocks are numbered starting at 0. The encrypter maintains a counter ctr, initially zero. The IV is $C[0]$, which is chosen to be the l-bit binary representation of the integer ctr. The counter is updated by the encryption algorithm as indicated.

Show that $\mathcal{SE}$ is insecure against chosen-plaintext attack. Namely, present a lower bound on $\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t,q,\mu)$ for certain specific, small values of t,q,μ that you will specify, along with a certain large (ie. close to one) value of the lower bound itself. Prove your claim correct by presenting and analyzing the corresponding adversary.

Your attack can assume that the initial value of the counter used by the LR-encryption oracle is is zero. (This corresponds to a setting in which the attacker is present right from the moment the encrypter starts using the scheme.) Once you have solved the problem in this setting, however, try to also find an attack in which the initial value of the counter used by the LR-encryption oracle is not known to the attacker. (This corresponds to the more realistic setting in which the attacker enters the picture after the scheme has been in use for some time.) ▮

Problem 6.34 Let $P: \{0,1\}^k \times \{0,1\}^{2l} \to \{0,1\}^{2l}$ be a family of permutations. We define a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$. The key generation algorithm simply returns a random key for P, meaning it picks a random k-bit string key and returns it, so that the key space is $\{0,1\}^k$. The message space is $\{0,1\}^{ln}$ where $n > 1$ is some fixed, given integer. The encryption and decryption algorithms are as follows:

Algorithm $\mathcal{E}_K(M)$	Algorithm $\mathcal{D}_K(C)$
If $\lvert M\rvert \neq nl$ then return $\perp$	If $\lvert C\rvert \neq 2nl$ then return $\perp$
Parse M as $M[1]\ldots M[n]$	Parse C as $C[1]\ldots C[n]$
For $i = 1, \ldots, n$ do	For $i = 1, \ldots, n$ do
$R[i] \stackrel{R}{\leftarrow} \{0,1\}^l$; $C[i] \leftarrow P_K(R[i] \Vert M[i])$	$R[i] \Vert M[i] \leftarrow P_K^{-1}(C[i])$
Return $C[1]\ldots C[n]$	Return $M[1]\ldots M[n]$

Show that this scheme is secure against chosen-plaintext attack as long as P is a secure PRP. More precisely show that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q, lnq) \quad \leq \quad 2 \cdot \mathbf{Adv}^{\text{prp-cpa}}_{P}(t', q') + \frac{n^2 q^2}{2^l} \, ,$$

where you must specify the values of t', q' as functions of t, q, l, n.

Hint: Proceed in analogy to the analysis of CTR mode encryption done above. First analyze the scheme which uses in place of P the family Perm^{2l} of random permutations on $2l$ bits. Then turn to the scheme using the given PRP P. ▮

Problem 6.35 Let $l \geq 64$ be an integer, and let $P: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a pseudorandom permutation. We define a symmetric encryption scheme $\mathcal{S}$ as follows. The key is a randomly chosen k-bit string K, meaning a key for the PRP. The encryption and decryption algorithms are as follows:

Algorithm $\mathcal{E}_K(x_1 \ldots x_n)$	Algorithm $\mathcal{D}_K(\langle r\rangle y_1 \ldots y_n)$
$r \stackrel{R}{\leftarrow} \{0, 1, \ldots, 2^l - 1\}$	For $i = 1, \ldots, n$ do
For $i = 1, \ldots, n$ do	$x_i \leftarrow P_K^{-1}(y_i) \oplus \langle r + i\rangle$
$y_i \leftarrow P_K(\langle r + i\rangle \oplus x_i)$	End For
End For	Return $x_1 \ldots x_n$
Return $\langle r\rangle y_1 \ldots y_n$	

Here the encryption algorithm takes as input a message x of length a multiple of l, which it views as divided into l bit blocks, $x = x_1 \ldots x_n$. It returns a string y of length $l(n+1)$, computed as shown. The decryption algorithm takes y to return x. Here "+" denotes addition modulo 2^l, and $\langle j\rangle$ denotes the binary representation of integer j as an l-bit string.

Show that this scheme is insecure. More precisely, show that

$$\mathbf{Adv}^{\text{ind-cpa}}_{\mathcal{SE}}(t, q, \mu) \quad \geq \quad \frac{1}{3} \, ,$$

where t, q, μ are values that you will specify, and should be as small as possible. ▮

CHAPTER 7

Public-key encryption

The idea of a public-key cryptosystem (PKC) was proposed by Diffie and Hellman in their pioneering paper [68] in 1976. Their revolutionary idea was to enable secure message exchange between sender and receiver without ever having to meet in advance to agree on a common secret key. They proposed the concept of a trapdoor function and how it can be used to achieve a public-key cryptosystem. Shortly there after Rivest, Shamir and Adelman proposed the first candidate trapdoor function, the RSA. The story of modern cryptography followed.

The setup for a public-key cryptosystem is of a network of users $u_1 \cdots u_n$ rather than an single pair of users. Each user u in the network has a pair of keys $< P_u, S_u >$ associated with him, the *public key* P_u which is published under the users name in a "public directory" accessible for everyone to read, and the private-key S_u which is known only to u. The pairs of keys are generated by running a *key-generation* algorithm. To send a secret message m to u everyone in the network uses the *same* exact method, which involves looking up P_u, computing $E(P_u, m)$ where E is a public encryption algorithm, and sending the resulting ciphertext c to u. Upon receiving ciphertext c, user u can decrypt by looking up his private key S_u and computing $D(S_u, c)$ where D is a public decryption algorithm. Clearly, for this to work we need that $D(S_u, E(P_u, m)) = m$.

A particular PKC is thus defined by a triplet of public algorithms (G, E, D), the key generation, encryption, and decryption algorithms.

7.1 Definition of Public-Key Encryption

We now formally define a public-key encryption scheme. For now the definition will say nothing about we mean by "security" of a scheme (which is the subject of much discussion in subsequent sections).

Definition 7.1 A *public-key encryption scheme* is a triple, (G, E, D), of probabilistic polynomial-time algorithms satisfying the following conditions

(1) key generation algorithm : a probabilistic expected polynomial-time algorithm G, which, on input 1^k (the security parameter) produces a pair (e, d) where e is called the public key , and d is the corresponding private key. (Notation: $(e, d) \in G(1^k)$). We will also refer to the pair (e, d) a pair of *encryption/decryption* keys.

(2) An encryption algorithm: a probabilistic polynomial time algorithm E which takes as input a security parameter 1^k, a public-key e from the range of $G(1^k)$ and string $m \in \{0,1\}^k$ called the *message*, and produces as output string $c \in \{0,1\}^*$ called the *ciphertext*. (We use the notation $c \in E(1^k, e, m)$ to denote c being an encryption of message m using key e with security parameter k. When clear, we use shorthand $c \in E_e(m)$, or $c \in E(m)$.)

(3) A decryption algorithm: a probabilistic polynomial time algorithm D that takes as inputs a security parameter 1^k, a private-key d from the range of $G(1^k)$, and a ciphertext c from the range of $E(1^k, e, m)$, and produces as output a string $m' \in \{0,1\}^*$, such that for every pair (e,d) in the range of $G(1^k)$, for every m, for every $c \in D(1^k, e, m)$, the $prob(D(1^k, d, c) \neq m')$ is negligible.

(4) Furthermore, this system is "secure" (see Definition 7.3).

∎

How to use this definition. To use a public-key encryption scheme (G, E, D) with security parameter 1^k, user A runs $G(1^k)$ to obtain a pair (e,d) of encryption/decryption keys. User A then "publishes" e in a public file, and keeps private d. If anyone wants to send A a message, then need to lookup e and compute $E(1^k, e, m)$. Upon receipt of $c \in E(1^k, e, m)$, A computes message $m = D(1^k, d, c)$.

Comments on the Definitions

Comment 0: Note that essentially there is no difference between the definition of a private-key encryption scheme and the definition of a public-key encryption scheme at this point. We could have defined a private key encryption scheme to have one key e for encryption and a different key d for decryption. The difference between the two definitions comes up in the security definition. In a public-key encryption scheme the adversary or "breaking algorithm" is given e (the *public key*) as an additional input; whereas in private-key scheme e is not given to the adversary (thus without loss of generality one may assume that $e = d$).

Comment 1: At this stage, encryption using a key of length k is defined only for messages of length k; generalization is postponed to Convention 7.1.

Comment 2: Note that as algorithm G is polynomial time, the length of its output (e,d) (or e in the private-key encryption case) is bounded by a polynomial in k. On the other hand, since k also serves as the "security parameter", k must be polynomial in $|d|$ (or $|e|$ in the private-key encryption case) in which case "polynomial in k" is equivalent to "polynomial in $|d|$".

Comment 3: Condition (3) in Definition 7.7 and 7.1 may be relaxed so that inequality may occur with negligible probability. For simplicity, we chose to adopt here the more conservative requirement.

Comment 4: We have allowed the encryption algorithm in both of the above definitions to be probabilistic. Namely, there can be many cyphertexts corresponding to the same message. In the simple (informal) example of a public-key encryption scheme based on a trapdoor function outlined in the introduction, every message has a unique corresponding ciphertext. That is too restrictive as, for example, if E is deterministic, the same inputs would always produce the same outputs, an undesirable characteristic.

Comment 5: We allowed D to be a probabilistic algorithms. This may conceivably allow the consideration of encryption schemes which may offer higher security ([46]). Accordingly, we may relax the requirement that $\forall m, D(E(m)) = m$ to hold only with high probability.

Conventions Regarding Definitions

Messages of length not equal to k (the length of the encryption key) are encrypted by breaking them into blocks of length k and possibly padding the last block. We extend the notation so that

$$E_e(\alpha_1 \cdots \alpha_l \alpha_{l+1}) = E_e(\alpha_1) \cdots E_e(\alpha_l) \cdot E_e(\alpha_{l+1} p)$$

where $|\alpha_1| = \cdots = |\alpha_l| = k$, $|\alpha_{l+1}| \leq k$, and p is some standard padding of length $k - |\alpha_{l+1}|$.

The above convention may be interpreted in two ways. First, it waves the extremely restricting convention by which the encryption scheme can be used only to encrypt messages of length equal to the length of the key. Second, it allows to reduce the security of encrypting many messages using the same key to the security of encrypting a single message.

The next convention regarding encryption schemes introduces a breach of security: namely, the length of the cleartext is always revealed by encryption schemes which follow this convention. However, as we show in a latter section some information about the length of the cleartext must be leaked by any encryption scheme.

The encryption algorithm maps messages of the same length to cryptograms of the same length.

7.2 Simple Examples of PKC: The Trapdoor Function Model

A collection of trapdoor functions, discussed at length in the chapter on one-way functions and trapdoor functions, has been defined as $F = \{f_i : D_i \to D_i\}_{i \in I}$ where $D_i \subseteq \{0,1\}^{|i|}$, and I is a set of indices. Recall that $\forall i$, f_i was easy to compute, but hard to invert; and $\forall i$, there existed t_i such that given t_i and $f_i(x)$, $f_i(x)$ could be inverted in polynomial time.

Diffie and Hellman suggested using the supposed existence of trapdoor functions to implement Public Key Cryptosystems as follows.

(1) The generator G on security parameter 1^k outputs pairs (f, t_f) where f is a trapdoor function and t_f its associated trapdoor information.

(2) For every message $m \in M$, $E(f, m) = f(m)$.

(3) Given $c \in E(f, m)$ and t_f, $D(t_f, c) = f^{-1}(c) = f^{-1}(f(m)) = m$.

7.2.1 Problems with the Trapdoor Function Model

There are several immediate problems which come up in using the trapdoor function model for public key encryption.

We summarize briefly the main problems which will be elaborated on in the next few sections.

(1) *Special Message Spaces.* The fact that f is a trapdoor function doesn't imply that inverting $f(x)$ when x is *special* is hard. Namely, suppose that the set of messages that we would like to send is drawn from a highly structured message space such as the English language, or more simply $M = \{0, 1\}$, it may be easy to invert $f(m)$. In fact, it is always easy to distinguish between $f(0)$ and $f(1)$).

(2) *Partial Information.* The fact that f is a one-way or trapdoor function doesn't necessarily imply that $f(x)$ hide all information about x. Even a bit of leakage many be too much for some applications. For example, for candidate one-way function $f(p, g, x) = g^x \bmod p$ where p is prime and g is a generator, the least significant bit of x is always easily computable from $f(x)$. For RSA function $f(n, l, x) = x^l \bmod n$, the Jacobi symbol $\mathbf{J}_n(x) = \mathbf{J}_n(x^l \bmod n)$. Namely, the Jacobi symbol of x is easy to compute from $f(n, l, x)$ – this was observed by Lipton[130] who used this fast to crack a protocol for Mental poker by Shamir Rivest and Adleman[185]. See below. Moreover, In fact, for any one-way function f, information such as "the parity of $f(m)$" about m is always easy to compute from $f(m)$. See below.

(3) *Relationship between Encrypted Messages* Clearly, we may be sending messages which are related to each other in the course of a communication. Some examples are: sending the same secret message to several recipients, or sending the same message (or slight variants) many times. It is thus desirable and sometimes essential that such dependencies remain secret. In the trapdoor function model, it is trivial to see that sending the same message twice is always detectable. More serious problems were noted by several researchers, most notably by Håstad who shows [106] that if RSA with an exponent l is used, and the same message (or known linear combinations of the same message) is send to l recipients, then the message can be computed by an adversary.

7.2.2 Problems with Deterministic Encryption in General

The above problems are actually shared by any public-key cryptosystem in which the encryption algorithm is deterministic.

It is obvious for problems 1 and 3 above. It is easy to show also for problem 3 as follows. Let E is any deterministic encryption algorithm, we can extract partial information by using something similar to the following predicate:

$$P(x) = \left\{ \begin{array}{l} 1 \text{ if } E(x) \text{ even} \\ 0 \text{ if } E(x) \text{ odd} \end{array} \right\}$$

It is clear that we can easily compute this predicate since all we have to do is take the low bit of $E(x)$. Unless $E(x)$ is always even or always odd for all the x's in the message space, we have obtained partial information about x. If $E(x)$ is always even or odd, the low bit of $E(x)$ contains no information. But, some other bit of $E(x)$ must contain some information otherwise the message space is composed of only one message in which case we have total information. Then, simply use that bit instead of the lowest bit and we have a partial information obtaining predicate.

7.2.3 The RSA Cryptosystem

In 1977 Shamir, Rivest and Adelman proposed the first implementation of trapdoor function, the RSA function, [170]. We refer the reader to chapter 2, in particular sections 2.2.5 and Section 2.17 for a thorough treatment of the RSA trapdoor function.

Here, let us examine the use of the RSA trapdoor function for the purpose of encryption in the straight forward manner proposed by Diffie and Hellman. We will show that it will not satisfy the kind of security which we desire. We will later see that a probabilistic variant will do the job.

Recall the definition of RSA trapdoor function 2.17. Let p, q denote primes, $n = pq$, $Z_n^* = \{1 \leq x \leq n, (x,n) = 1\}$ the multiplicative group whose cardinality is $\varphi(n) = (p-1)(q-1)$, and $e \in Z_{p-1}$ relatively prime to $\varphi(n)$. Our set of indices will be $I = \{< n, e > \text{ such that } n = pq\ |p| = |q|\}$ and the trapdoor associated with the particular index $< n, e >$ be $t_{<n,e>} = d$ such that $ed = 1 \mod \phi(n)$. Let $RSA = \{RSA_{<n,e>} : Z_n^* \to Z_n^*\}_{<n,e> \in I}$ where

$$RSA_{<n,e>}(x) = x^e \bmod n$$

Sparse Message Spaces

We showed that the RSA function has some nice properties that seem especially good for use as a PKC. For example, we showed for a given pair $< n, e >$, it is either hard to invert $RSA_{<n,e>}$ for all but a negligible fraction of x's in Z_n^*, or easy to invert $RSA_{<n,e>}(x)\ \forall x, x \in \mathbf{Z}^*$. Does this mean that the RSA cryptosystem is difficult to break for almost all messages if factoring integers is hard? The answer is negative.

Suppose that the message space M we are interested in is the English language. Then, let $M_k = \{0,1\}^k$ where $m \in M_k$ is an English sentence. Compared to the entire space, the set of English sentences is quite small. For example, $\frac{|M_k|}{|\mathbf{Z}_n^*|} \leq \frac{1}{2^{\sqrt{n}}}$. Thus it is possible that $f_{n,e}(x)$ is easy to invert for all $x \in M_k$, even if the factorization problem is hard. In other words, English sentences are highly structured; it might well be the case that our function can be easily inverted on all such inputs. Clearly, we would ultimately like our encryption schemes to be secure for all types of message spaces, including English text.

Partial Information about RSA

What partial information about x can be computed from $RSA_{<n,e>}(x)$.

We showed in the chapter on one-way and trapdoor functions, that indeed some bits such as the least significant bit and most significant bit of RSA are very well hidden. This is the good news.

Unfortunately, in some cases very subtle leakage of partial information can defeat the whole purpose of encryption. We present a "cute" example of this shown by Lipton shortly after RSA was invented.

An Example: Mental Poker (SRA '76): Mental Poker is a protocol by which two parties each of whom distrusts the other can deal each other cards from a deck without either being able to cheat. The protocol for A to deal B a card goes like this:

(1) A and B agree on a set $X = \{x_1, \ldots, x_{52}\}, x_i \in Z_n^*$, of random numbers where $n = pq$, p and q prime and known to both A and B. These numbers represent the deck of cards, x_i representing the ith card in the deck.

(2) A picks s such that $(s, \varphi(n)) = 1$, and t such that $st \equiv 1 \pmod{\varphi(n)}$ secretly. B does the same for e and f. (I.e., $ef \equiv 1 \pmod{\varphi(n)}$)

(3) A calculates $x_i^s \bmod n$ for $i = 1 \ldots 52$, shuffles the numbers, and sends them to B.

(4) B calculates $(x_i^s \bmod n)^e \bmod n$ for $i = 1 \ldots 52$, shuffles the numbers, and sends them to A.

(5) A calculates $((x_i^s \bmod n)^e \bmod n)^t \bmod n = x_i^e \bmod n$ for $i = 1 \ldots 52$. A then chooses a card randomly (I.e., picks x_j^e where $j \in [1 \ldots 52]$) and sends it to B.

(6) B then takes $(x_j^e \bmod n)^d \bmod n = x_j \bmod n$. This is the card B has been dealt.

Why it works: Note that so long as no partial information can be obtained from the RSA trapdoor function, neither A nor B can influence in any way the probably of B getting any given card. A is unable to give B bad cards and likewise B can not deal himself good cards. This follows from the fact that encrypting the cards is analogous to placing each of them in boxes locked with padlocks. So long as a card is locked in a box with a padlock of the other player's on it, nothing can be told about it and it is indistinguishable from the other locked boxes.

When B gets the deck in step 3, he has no idea which card is which and thus is unable to influence which card he is dealt. However, A can still tell them apart since its A's padlocks that are on the boxes. To prevent A from being able to influence the cards, B then puts his own locks on the boxes as well and shuffles the deck. Now A also can not tell the cards apart so when he is forced to make his choice, he is forced to just deal a random card. Thus, the two players in spite of distrusting each other can play poker.

How to extract partial information from the RSA function: The protocol fails, however, because it is possible to extract partial information from the RSA function and thus determine to some degree of accuracy which cards are which and hence influence the outcome of the draw. One way to do this is by computing the Jacobi symbol since $(\mathbf{J}_n(x_i)) = (\mathbf{J}_n(x_i^s))$ since s is odd. Thus, since half of the x_i's have a Jacobi symbol of 1 on the average since they are random numbers in Z_n^*, we can extract roughly one bit of information from each of the cards. In order to influence the draw in our favor, we simply determine whether or not the cards with a Jacobi symbol of 1 or the cards with a Jacobi symbol of -1 are better for us and then draw only from that set of cards.

One's immediate reaction to this, of course, is simply to modify the protocol so that in step 1 only numbers with say a Jacobi symbol of 1 are chosen. Then no information will be gained by computing the Jacobi symbol. However, this is no guarantee that some other more clever predicate does not exist which can still extract partial information and indeed such functions must exist by the very nature of trapdoor functions.

Low exponent attacks

Let the exponent be $e = 3$. We saw that any exponent relatively prime to $\varphi(N)$ is OK, and we can easily choose $N = pq$ so that 3 is relatively prime to $(p-1)(q-1) = \varphi(N)$. This is a popular choice for performance reasons. Encryption is now fast. And we saw that RSA is still (assumed) one-way under this choice.

So encryption of m is now $m^3 \bmod N$. Here is an interesting attack illustrating the weaknesses of RSA encryption, due to Coppersmith, Franklin, Patarin and Reiter [58]. Suppose I encrypt m and then $m+1$. I claim you can recover m. We have ciphertexts:

$$\begin{aligned} c_1 &= m^3 \\ c_2 &= (m+1)^3 = m^3 + 3m + 3m^2 + 1 = c_1 + 3m + 3m^2 + 1 \end{aligned}$$

Now lets try to solve for m. Perhaps the first thought that springs to mind is that we have a quadratic equation for m. But taking square roots is hard, so we don't know how to solve it that way. It turns out the following works:

$$\frac{c_2 + 2c_1 - 1}{c_2 - c_1 + 2} = \frac{(m+1)^3 + 2m^3 - 1}{(m+1)^3 - m^3 + 2} = \frac{3m^3 + 3m^2 + 3m}{3m^2 + 3m + 3} = m \ .$$

This can be generalized. First, you can generalize to messages m and $\alpha m + \beta$ for known α, β. Second, it works for exponents greater than 3. The attack then runs in time $O(e^2)$ so it is feasible for small exponents. Finally, it can work for k messages related by a higher degree polynomial.

These are the kinds of attacks we most definitely would like to avoid.

7.2.4 Rabin's Public key Cryptosystem

Recall Rabin's trapdoor function from Chapter 2.

$$f_n(m) \equiv m^2 \mod n$$

where n is the product of two large primes, p and q. Once again, this function can yield another example of a trapdoor/public key cryptosystem except that f_n is not as permutation but a 4-to-1 function. An inverse of $f_n(m)$:

$$f_n^{-1}(m^2) = x \text{ such that } x^2 = m^2 \mod n$$

However, in practice, when we invert Rabin's function, we do not simply want any square root of the encrypted message, but the correct one of the four that was meant to be sent by the sender and would be meaningful to the intended recipient. So, we need to add a constraint to uniquely identify the root x which must be output by the decryption algorithm on $f_n(m^2)$ such as find x such that $x^2 = m^2 \mod n$, and $x \in S$ where S is a property for which it is quite unlikely that there exists two roots $m, x \in S$. What could S be? Well if the message space M is sparse in $\mathbf{Z}_n^*$ (which would usually be the case), then S may be simply M. In such case it is unlikely that there exists $m \neq m' \in M$ such that $m'^2 = m^2 \mod n$. (If M is not sparse, S may be the all x whose last 20 digits are r for some random r. Then to send m in secrecy, $(f_n(m') = f_n(2^{20}m + r), r)$ need be sent.)

Recall, that earlier in the class, we had shown that inverting Rabin's function is as hard as factoring. Namely, we had shown that inverting Rabin's function for ϵ of the $m^2 \in \mathbf{Z}_n^*$'s implies the ability to factor. The proof went as follows:

- Suppose there existed a black box that on inputs x^2 responded with a y such that $x^2 = y^2 \mod n$. Then, to factor n, choose an i at random from $\mathcal{Z}_n^*$ and give as input $i^2 \mod n$ to the black box. If the box responds with a y, such that $y \neq \pm i$, then we can indeed factor n by computing $gcd(i \pm y, n)$. In the case that $y = \pm i$, we have gained no information, so repeat.

If we think of this black box as a decoding algorithm for the public key system based on Rabin's function used to encrypt messages in message space M, can we conclude that *if it is possible to decrypt the public key system $f_n(m)$ for $m \in M$, then it is possible to factor n?*

If the message space M is sparse, then the answer is **no.** Why? for the black box (above) to be of any use we need to feed it with an $f_n(i)$ for which there exists an y such that $y \in M$ and $y \neq i$. The probability that such y exists is about $\frac{|M|}{|\mathbf{Z}_n^*|}$, which may be exponentially small.

If the message space M is not sparse, we run into another problem. Rabin's scheme would not be secure in the presence of an active adversary who is capable of a chosen ciphertext attack. This is easy to see again using the above proof that inverting Rabin's function is as hard as factoring. Temporary access to a decoding algorithm for Rabin's public key encryption for message in M, is the same as having access to the black box of the above proof. The adversary chooses i at random and feeds the decoding algorithm with $f_n(i)$. If the adversary gets back y such that $y^2 = i^2 \bmod n$, (again, $i \neq \pm y$), factor n, and obtain the secret key. If M is not sparse this will be the case after trying a polynomial number of i's. From here on, the adversary would be able to decrypt any ciphertext with the aid of the secret key and without the need for a black box.

Therefore, either Rabin's scheme is not equivalent to factoring, which is the case when inverting on a sparse message space, or (when M is not sparse) it is insecure before a chosen ciphertext adversary.

7.2.5 Knapsacks

A number of public-key cryptosystems have been proposed which are based on the *knapsack* (or — more properly — the *subset sum*) problem: given a vector $\mathbf{a} = (a_1, a_2, \ldots, a_n)$ of integers, and a target value C, to determine if there is a length-n vector $\mathbf{x}$ of zeroes and ones such that $\mathbf{a} \cdot \mathbf{x} = C$. This problem is NP-complete [86].

To use the knapsack problem as the basis for a public-key cryptosystem, you create a public key by creating a knapsack vector $\mathbf{a}$, and publish that as your public key. Someone else can send you the encryption of a message M (where M is a length-n bit vector), by sending you the value of the inner product $C = M \cdot \mathbf{a}$. Clearly, to decrypt this ciphertext is an instance of the knapsack problem. To make this problem easy for you, you need to build in hidden structure (that is, a trapdoor) into the knapsack so that the encryption operation becomes one-to-one and so that you can decrypt a received ciphertext easily. It seems, however, that the problem of solving knapsacks containing a trapdoor is *not* NP-complete, so that the difficulty of breaking such a knapsack is no longer related to the $P = NP$ question.

In fact, history has not been kind to knapsack schemes; most of them have been broken by extremely clever analysis and the use of the powerful L^3 algorithm [127] for working in lattices. See [137, 180, 182, 2, 184, 123, 44, 151].

Some knapsack or knapsack-like schemes are still unbroken. The Chor-Rivest scheme [56], and the multiplicative versions of the knapsack [137] are examples. McEliece has a knapsack-like public-key cryptosystem based on error-correcting codes [136]. This scheme has not been broken, and was the first scheme to use randomization in the encryption process.

We are now ready to introduce what is required from a *secure* Public Key Cryptosystem.

7.3 Defining Security

Brain storming about what it means to be secure brings immediately to mind several desirable properties. Let us start with the the minimal requirement and build up.

First and foremost the private key should not be recoverable from seeing the public key. Secondly, with high probability for any message space, messages should not be entirely recovered from seeing their encrypted form and the public file. Thirdly, we may want that in fact no useful information can be computed about messages from their encrypted form. Fourthly, we do not want the adversary to be able to compute any useful facts about traffic of messages, such as recognize that two messages of identical content were sent, nor would we want her probability of successfully deciphering a message to increase if the time of delivery or relationship to previous encrypted messages were made known to her.

In short, it would be desirable for the encryption scheme to be the mathematical analogy of opaque envelopes containing a piece of paper on which the message is written. The envelopes should be such that all legal senders can fill it, but only the legal recipient can open it.

We must answer a few questions:

- How can "opaque envelopes" be captured in a precise mathematical definition?
- Are "opaque envelopes" achievable mathematically?

Several definitions of security attempting to capture the "opaque envelope" analogy have been proposed. All definitions proposed so far have been shown to be equivalent. We describe two of them and show they are equivalent.

7.3.1 Definition of Security: Polynomial Indistinguishability

Informally, we say that an encryption scheme is polynomial time indistinguishable if no adversary can find even two messages, whose encryptions he can distinguish between. If we recall the envelope analogy, this translates to saying says that we cannot tell two envelopes apart.

Definition 7.2 We say that a Public Key Cryptosystem (G, E, D) is *polynomial time indistinguishable* if for every PPT M, A, and for every polynomial Q, $\forall$ sufficiently large k

$$Pr(A(1^k, e, m_0, m_1, c) = m \mid (e,d) \stackrel{R}{\leftarrow} G(1^k) \ ; \ \{m_0, m_1\} \stackrel{R}{\leftarrow} M(1^k) \ ; \ m \stackrel{R}{\leftarrow} \{m_0, m_1\} \ ; \ c \stackrel{R}{\leftarrow} E(e,m))$$
$$< \frac{1}{2} + \frac{1}{Q(k)} \tag{7.1}$$

∎

In other words, it is impossible in polynomial in k time to find two messages m_0, m_1 such that a polynomial time algorithm can distinguish between $c \in E(e, m_0)$ and $c \in E(e, m_1)$.

Remarks about the definition:

(1) We remark that a stronger form of security would be: the above holding $\forall m_0, m_1$, (not only those which can be found in polynomial time by running $M(1^k)$). Such security can be shown in a non-uniform model, or when the messages are chosen before the keys, and thus can not involve any information about the secret keys themselves.

(2) In the case of private-key encryption scheme, the definition changes ever so slightly. The encryption key e is not given to algorithm A.

(3) Note that any encryption scheme in which the encryption algorithm E is deterministic immediately fails to pass this security requirement. (e.g given f, m_0, m_1 and $c \in \{f(m_1), f(m_0)\}$ it is trivial to decide whether $c = f(m_0)$ or $c = f(m_1)$.

(4) Note that even if the adversary know that the messages being encrypted is one of two, he still cannot tell the distributions of ciphertext of one message apart from the other.

7.3.2 Another Definition: Semantic Security

Consider the following two games. Let $h : M \to \{0,1\}^*$, where M is a message space in which we can sample in polynomial time, or equivalently, a probabilistic polynomial time algorithm M that takes as input 1^k and generates a message $m \in \{0,1\}^k$, and $h(m)$ is some information about the message (for example, let be such that $h(m) = 1$ if m has the letter 'e' in it, then $V = \{0,1\}$).

- Game 1: I tell the adversary that I am about to choose $m \in M(1^k)$ and, ask her to guess h(m).
- Game 2: I tell the adversary $\alpha \in E(m)$, for some $m \in M(1^k)$ and once again, ask her to guess h(m).

In both of the above cases we may assume that the adversary knows the message space algorithm M and the public key P.

In the first game, the adversary only knows that a message m is about to be chosen. In addition to this fact, the adversary of the Game 2 sees the actual ciphertext itself. For all types of message spaces, semantic security will essentially require that the probability of the adversary winning Game 1 to be about the same as her probability of winning Game 2. Namely, that the adversary should not gain any advantage or information from having seen the ciphertext resulting from our encryption algorithm.

Said differently, this definition will require that for all probability distributions over the message space, no partial information about the message can be computed from the ciphertext. This requirement is reminiscent of Shannon's perfect security definition – with respect to a computationally bounded adversary.

Definition 7.3 We say that an encryption scheme (G, E, D) is *semantically secure* if for all PPT algorithms M and A, functions h, polynomials Q there is a PPT B such that for sufficiently large k,

$$\Pr(A(1^k, c, e) = h(m) \mid (e, d) \stackrel{R}{\leftarrow} G(1^k) \ ; \ m \stackrel{R}{\leftarrow} M(1^k) \ ; \ c \stackrel{R}{\leftarrow} E(e, m))$$
$$\leq \Pr(B(1^k) = h(m) \mid m \stackrel{R}{\leftarrow} M(1^k)) + \frac{1}{Q(k)} \tag{7.2}$$

∎

Here, Game 1 is represented by PTM B, and Game 2 by PTM A. Again, this can only hold true when the encryption algorithm is a probabilistic one which selects one of many possible encodings for a message; otherwise, if E were deterministic, and $M = \{0, 1\}$, then any adversary would have 100% chance of guessing correctly $h(m)$ for $m \in M$ by simply testing whether $E(0) = c$ or $E(1) = c$.

Theorem 7.4 A Public Key Cryptosystem passes Indistinguishable Security if and only if it passes Semantic Security. ∎

7.4 Probabilistic Public Key Encryption

We turn now to showing how to actually build a public key encryption scheme which is polynomial time indistinguishable.

In order to do so, we must abandon the trapdoor function PKC model and deterministic algorithms of encryption all together, in favor of probabilistic encryption algorithm. The probabilistic encryption algorithm which we will construct will still assume the existence of trapdoor functions and use them as a primitive building block.

The key to the construction is to first answer a simpler problem: How to securely encrypt single bits. We show two ways to approach this problem. The first is based on trapdoor predicates as discussed in Section 2.5, and the second is based on hard core predicates as discussed in Section 2.4.

7.4.1 Encrypting Single Bits: Trapdoor Predicates

To encrypt single bits, the notion of one-way and trapdoor predicates was introduced by [98]. It later turned out to be also quite useful for protocol design. We refer the reader to section 2.5 for a general treatment of this subject. Here we look at its use for encryption.

The Idea: Briefly, a one-way predicate, is a Boolean function which is hard to compute in a very strong sense. Namely, an adversary cannot compute the predicate value better than by taking a random guess. Yet, it is possible to sample the domain of the predicate for elements for which the predicate evaluates to 0 and to 1. A trapdoor predicate possesses the extra feature that there exists some trapdoor information that enables the computation of the predicate. We can construct examples of collection of trapdoor predicates based on the intractability of factoring, RSA inversion and the difficulty of distinguishing quadratic residues from non-residues.

Now, given a collection of trapdoor predicates exist, we use them to set up a cryptosystem for one bit encryption as follows. Every user A chooses and publishes a random trapdoor predicate, keeping secret the corresponding trapdoor information. To send A a one bit message m, any other user chooses at random an element in the domain of the trapdoor predicate for which the predicate evaluates to m. To decrypt, A uses his trapdoor information to compute the value of predicate on the domain element it receives. Note, that this is a probabilistic encryption with many possible cyphertexts for 0 as well as 1, where essentially an adversary cannot distinguish between an encoding of 0 and an encoding of 1.

Recall, the formal definition of trapdoor predicates 2.59.

Let I be a set of indices and for $i \in I$ let D_i be finite. A *collection of trapdoor predicates* is a set $B = \{B_i : D_i \to \{0,1\}\}_{i \in I}$ satisfying the following conditions. Let $D_i^v = \{x \in D_i, B_i(x) = v$.

1. There exists a polynomial p and a PTM S_1 which on input 1^k finds pairs (i, t_i) where $i \in I \cap \{0,1\}^k$ and $|t_i| < p(k)$ The information t_i is referred to as the trapdoor of i.
2. There exists a PTM S_2 which on input $i \in I$ and $v \in \{0,1\}$ outputs $x \in D_i$ at random such that $B_i(x) = v$.
3. There exists a PTM A_1 such that for $i \in I$ and trapdoor t_i, $x \in D_i$ $A_1(i, t_i, x) = B_i(x)$.
4. For every PPT A there exists a negligible ν_A such that $\forall$ k large enough
$$\mathbf{P}\left[z \neq v \ : \ i \stackrel{R}{\leftarrow} I \cap \{0,1\}^k \ ; \ v \stackrel{R}{\leftarrow} \{0,1\} \ ; \ x \stackrel{R}{\leftarrow} D_i^v \ ; \ z \leftarrow A(i,x) \right] \leq \nu_A(k)$$

Definition 7.5 Assume that B is a collection of trapdoor predicates. We can now define a public key cryptosystem $(G, E, D)_B$ for sending single bit messages as follows:

- Key generation algorithm: $G(1^k)$ chooses (i, t_i) (public key is then i and private key is t_i). This is doable by running algorithm S_1.
- Encryption algorithm: Let $m \in \{0,1\}$ be the message. Encryption algorithm $E(i, e)$ selects $x \in D_i^m$. (The ciphertext is thus x). This is doable by running algorithm S_2.
- Decryption algorithm: $D(c, t_i)$ computes $B_i(c)$. This is doable using A_1 given the trapdoor information.

∎

It is clear from the definition of a set of trapdoor predicates, that all of the above operations can be done in expected polynomial time and that messages can indeed be sent this way. It follows immediately from the definition of trapdoor predicates than indeed this system is polynomially indistinguishable when restricted to one bit message spaces.

7.4.2 Encrypting Single Bits: Hard Core Predicates

Alternatively, you may take the following perhaps simpler approach, starting directly with trapdoor functions and using their hard core predicates. For a detailed discussion of trapdoor functions and hard core predicates for them see section Section 2.59. The discussion here assumes such knowledge.

Recall that a collection of trapdoor permutations is a set $F = \{f_i : D_i \longrightarrow D_i\}_{i \in I}$ such that:

1. $S_1(1^k)$ samples (i, t_i) where $i \in I$, $|i| = k$ and $|t_i| < p(k)$ for some polynomial p.

2. $S_2(i)$ samples $x \in D_i$.

3. $\exists$ PTM A_1 such that $A_1(i,x) = f_i(x)$.

4. $\Pr[A(i, f_i(x)) \in f_i^{-1}(f_i(x))] < \frac{1}{Q(k)}$ $\forall$ PTM A, $\forall Q$, $\forall k > k_0$.

5. $\exists$ PTM A_2 such that $A_2(i, t_i, f_i(x)) = x$, $\forall x \in D_i$, $i \in I$.

Further, let $B_i(x)$ be hard core for $f_i(x)$. Recall that the existence of F implies the existence of $F\prime$ that has a hard core predicate. So, for notational simplicity assume that $F = F\prime$. Also recall that for the RSA collection of trapdoor functions, LSB is a collection of hard core predicate the LSB.

Definition 7.6 Given a collection F with hard core predicates B, define public key cryptosystem $(G, E, D)_B$ for sending a single bit as follows:

- Key generation algorithm: $G(1^k)$ chooses pair $< i, t_i >$ by running $S_1(1^k)$. (for RSA, $G(1^k)$ chooses $< n, e >, d$ such that n is an RSA modulus, and $ed = 1 \bmod \phi(n)$.)
- Encryption algorithm: $E(i, m)$ chooses at random an $x \in D_i$ such that $B_i(x) = m$, and output as a ciphertext $f_i(x)$. Using the Goldreich Levin construction of a hard core predicate, simply choose x, r such that the inner product of x and r is m and output $f(x) \circ r$. (for RSA, to encrypt bit m, choose at random an $x \in Z_n^*$ such that $LSB_{<n,e>}(x) = m$ and output as a ciphertext $RSA_{<n,e>}(x)$.)
- Decryption algorithm: To decrypt $c = f_i(x)$, given i and t_i, the decryption algorithm $D(t_i, c)$ compute $B_i(f_i^{-1}(c)) = B_i(x) = m$. Using the Goldreich Levin construction this amounts to given $c = f_i(x) \circ r$ to computing the inner product of x and r. (for RSA, to decrypt c, given n, e and d, compute the $LSB((RSA_{<n,e>}(x))^d)$ = least significant bit of x.)

∎

7.4.3 General Probabilistic Encryption

How should we encrypt arbitrary length messages?

The first answer is to simply encrypt each bit individually using one of the above methods. as above. Before considering whether this is wise from an efficiency point of view, we need to argue that it indeed will produce a encryption scheme which is polynomial time indistinguishable. This requires reflection, as even through every bit individually is secure, it can be the case that say that some predicate computed on all the bits is easily computable, such as the exclusive or of the bits. This turns out luckily not to be the case, but requires proof.

We now provide construction and proof.

Definition 7.7 We define a probabilistic encryption based on trapdoor collection F with hard core bit B to be $PE = (G, E, D)$ where:

- $G(1^k)$ chooses (i, t_i) by running $S_1(1^k)$ (Public key is i, private key is t_i).
- Let $m = m_1 \ldots m_k$ where $m_j \in \{0, 1\}$ be the message.
  $E(i, m)$ encrypts m as follows:
  Choose $x_j \in_R D_i$ such that $B_i(x_j) = m_j$ for $j = 1, \ldots, k$.
  Output $c = f_i(x_1) \ldots f_i(x_k)$.
- Let $c = y_1 \ldots y_k$ where $y_i \in D_i$ be the cyph ertext.
  $D(t_i, c)$ decrypts c as follows:
  Compute $m_j = B_i(f_i^{-1}(y_j))$ for $j = 1, \ldots, k$.
  Output $m = m_1 \ldots m_k$.

▮

Claim 7.8 If F is a collection of trapdoor permutations then the probabilistic encryption $PE = (G, E, D)$ is indistinguishably secure. ▮

Proof: Suppose that (G, E, D) is not indistinguishably secure. Then there is a polynomial Q, a PTM A and a message space algorithm M such that for infinitely many k, $\exists m_0, m_1 \in M(1^k)$ with,

$$\Pr[A(1^k, i, m_0, m_1, c) = j | m_j \in \{m_0, m_1\}, c \in E(i, m_j))] > \frac{1}{2} + \frac{1}{Q(k)}$$

where the probability is taken over the coin tosses of A, $(i, t_i) \in G(1^k)$, the coin tosses of E and $m_j \in \{m_0, m_1\}$. In other words, A says 0 more often when c is an encryption of m_0 and says 1 more often when c is an encryption of m_1.

Define distributions $D_j = E(i, s_j)$ for $j = 0, 1, \ldots, k$ where $k = |m_0| = |m_1|$ and such that $s_0 = m_0, s_k = m_1$ and s_j differs from s_{j+1} in precisely 1 bit.

Let $P_j = \Pr[A(1^k, i, m_0, m_1, c) = 1 | c \in D_j = E(i, s_j)]$.

Then $\frac{1}{2} + \frac{1}{Q(k)} < \Pr[A \text{ chooses } j \text{ correctly}] = (1 - P_0)(\frac{1}{2}) + P_k(\frac{1}{2})$.

Hence, $P_k - P_0 > \frac{2}{Q(k)}$ and since $\sum_{j=0}^{k-1}(P_{j+1} - P_j) = P_k - P_0$, $\exists j$ such that $P_{j+1} - P_j > \frac{2}{Q(k)k}$.

Now, consider the following algorithm B which takes input $i, f_i(y)$ and outputs 0 or 1. Assume that s_j and s_{j+1} differ in the l^{th} bit; that is, $s_{j,l} \neq s_{j+1,l}$ or, equivalently, $s_{j+1} = \bar{s}_j$.

B runs as follows on input $i, f_i(y)$:

(1) Choose $y_1, \ldots, y_k$ such that $B_i(y_r) = s_{j,r}$ for $r = 1, \ldots, k$.

(2) Let $c = f_i(y_1), \ldots, f_i(y), \ldots, f_i(y_k)$ where $f_i(y)$ has replaced $f_i(y_l)$ in the l^{th} block.

(3) If $A(1^k, i, , m_0, m_1, c) = 0$ then output $s_{j,l}$.
If $A(1^k, i, , m_0, m_1, c) = 0$ then output $s_{j+1,l} = \bar{s}_{j,l}$.

Note that $c \in E(i, s_j)$ if $B_i(y) = s_{j,l}$ and $c \in E(i, s_{j+1})$ if $B_i(y) = s_{j+1,l}$.

Thus, in step 3 of algorithm B, outputting $s_{j,l}$ corresponds to A predicting that c is an encoding of s_j; in other words, c is an encoding of the string nearest to m_0.

Claim. $\Pr[B(i, f_i(y)) = B_i(y)] > \frac{1}{2} + \frac{1}{Q(k)k}$

Proof:

$$\begin{aligned}
\Pr[B(i, f_i(y)) = B_i(y)] &= \Pr[A(1^k, i, m_0, m_1, c) = 0 | c \in E(i, s_j)] \Pr[c \in E(i, s_j)] \\
&\quad + \Pr[A(1^k, i, m_0, m_1, c) = 1 | c \in E(i, s_{j+1})] \Pr[c \in E(i, s_{j+1})] \\
&\geq (1 - P_j)(\tfrac{1}{2}) + (P_{j+1})(\tfrac{1}{2}) \\
&= \tfrac{1}{2} + \tfrac{1}{2}(P_{j+1} - P_j) \\
&> \tfrac{1}{2} + \tfrac{1}{Q(k)k} \ \square
\end{aligned}$$

Thus, B will predict $B_i(y)$ given i, $f_i(y)$ with probability better than $\frac{1}{2} + \frac{1}{Q(k)k}$. This contradicts the assumption that $B_i(y)$ is hard core for $f_i(y)$.

Hence, the probabilistic encryption $PE = (G, E, D)$ is indistinguishably secure.

∎

In fact, the probabilistic encryption $PE = (G, E, D)$ is also semantically secure. This follows from the fact that semantic and indistinguishable security are equivalent.

7.4.4 Efficient Probabilistic Encryption

How efficient are the probabilistic schemes? In the schemes described so far, the ciphertext is longer than the cleartext by a factor proportional to the security parameter. However, it has been shown [35, 39] using later ideas on pseudo-random number generation how to start with trapdoor functions and build a probabilistic encryption scheme that is polynomial-time secure for which the ciphertext is longer than the cleartext by only an additive factor. The most efficient probabilistic encryption scheme is due to Blum and Goldwasser [39] and is comparable with the RSA deterministic encryption scheme in speed and data expansion. Recall, that private-key encryption seemed to be much more efficient. Indeed, in practice the public-key methods are often used to transmit a secret session key between two participants which have never met, and subsequently the secret session key is used in conjunction with a private-key encryption method.

We first describe a probabilistic public key cryptosystem based on any trapdoor function collection which suffers only from a small additive bandwidth expansion.

As in the previous probabilistic encryption PE, we begin with a collection of trapdoor permutations $F = \{f_i : D_i \to D_i\}$ with hard core predicates $B = \{B_i : D_i \to \{0,1\}$. For this section, we consider that $D_i \subseteq \{0,1\}^k$, where $k = |i|$.

Then EPE $= (G, E, D)$ is our PKC based on F with:

Key Generation: $G(1^k) = S_1(1^k) = (i, t_i)$. The public key is i, and the secret key is t_i.

Encryption Algorithm: To encrypt m, $E(i, m)$ runs as follows, where $l = |m|$:

(1) Choose $r \in D_i$ at random.

(2) Compute $f_i(r), f_i^2(r), \ldots, f_i^l(r)$.

(3) Let $p = B_i(r)B_i(f_i(r))B_i(f_i^2(r)) \ldots B_i(f_i^{l-1}(r))$.

(4) Output the ciphertext $c = (p \oplus m, f_i^l(r))$.

Decryption Algorithm: To decrypt a ciphertext $c = (m', a)$, $D(t_i, c)$ runs as follows, where $l = |m'|$:

(1) Compute r such that $f_i^l(r) = a$. We can do this since we can invert f_i using the trapdoor information, t_i, and this r is unique since f_i is a permutation.

(2) Compute the pad as above for encryption: $p = B_i(r)B_i(f_i(r)) \ldots B_i(f_i^{l-1}(r))$.

(3) Output decrypted message $m = m' \oplus p$.

To consider the efficiency of this scheme, we note that the channel bandwidth is $|c| = |m| + k$, where k is the security parameter as defined above. This is a significant improvement over the $|m| \cdot k$ bandwidth achieved by the scheme proposed in the previous lecture, allowing improvement in security with only minimal increase in bandwidth.

If C_{i1} is the cost of computing f_i, and C_{i2} is the cost of computing f_i^{-1} given t_i, then the cost of encryption is $|m| \cdot C_{i1}$, and the cost of decryption is $|m| \cdot C_{i2}$, assuming that the cost of computing B_i is negligible.

Another interesting point is that for all functions currently conjectured to be trapdoor, even with t_i, it is still easier to compute f_i than f_i^{-1}, that is, $C_{i1} < C_{i2}$, though of course, both are polynomial in $k = |i|$. Thus in EPE, if it is possible to compute f_i^{-l} more efficiently than as l compositions of f_i^{-1}, then computing

$r = f_i^{-l}(a)$, and then computing $f_i(r), f_i^2(r), \ldots, f_i^{l-1}(r)$ may reduce the overall cost of decryption. The following implementation demonstrates this.

7.4.5 An implementation of EPE with cost equal to the cost of RSA

In this section, we consider a particular implementation of EPE as efficient as RSA. This uses for F a subset of Rabin's trapdoor functions which were introduced in Lecture 5. Recall that we can reduce Rabin's functions to permutations if we only consider the Blum primes, and restrict the domain to the set of quadratic residues. In fact, we will restrict our attention to primes of the form $p \equiv 7 \bmod 8$.[1]

Let $\mathcal{N} = \{n | n = pq; |p| = |q|; p, q \equiv 7 \bmod 8\}$. Then let $F = \{f_n : D_n \longrightarrow D_n\}_{n \in \mathcal{N}}$, where $f_n(x) \equiv x^2 \bmod n$, and $D_n = Q_n = \{y | y \equiv x^2 \bmod n\}$. Because $p, q \equiv 3 \bmod 4$, we have that f_n is a permutation on D_n. $B_n(x)$ is the least significant bit (LSB) of x, which is a hard core bit if and only if factoring is difficult, i.e., the Factoring Assumption from Lecture 5 is true. (This fact was stated, but not proven, in Lecture 7.)

Then consider the EPE (G, E, D), with:

Generation: $G(1^k) = (n, (p, q))$ where $pq = n \in \mathcal{N}$, and $|n| = k$. Thus n is the public key, and (p,q) is the secret key.

Encryption: $E(n, m)$, where $l = |m|$ (exactly as in general case above):

(1) Choose $r \in Q_n$ randomly.

(2) Compute $r^2, r^4, r^8, \ldots, r^{2^l} \pmod n$.

(3) Let $p = \text{LSB}(r)\text{LSB}(r^2)\text{LSB}(r^4)\ldots\text{LSB}(r^{2^{l-1}})$.

(4) Output $c = (m \oplus p, r^{2^l} \bmod n)$.

The cost of encrypting is $O(k^2 \cdot l)$.

Decryption: $D((p, q), c)$, where $c = (m', a), l = |m'|$ (as in general case above):

(1) Compute r such that $r^{2^l} \equiv a \bmod n$.

(2) Compute $p = \text{LSB}(r)\text{LSB}(r^2)\text{LSB}(r^4)\ldots\text{LSB}(r^{2^{l-1}})$.

(3) Output $m = m' \oplus p$.

Since $p, q \equiv 7 \bmod 8$, we have $p = 8t + 7$ and $q = 8s + 7$ for some integers s, t. Recall from Lecture 3 that if p is prime, the *Legendre symbol* $\mathbf{J}_p(a) = a^{\frac{p-1}{2}} \equiv 1 \bmod p$ if and only if $a \in Q_p$, Since $a \in Q_n$, we also have $a \in Q_p$. Thus we can compute

$$a \equiv a \cdot a^{\frac{p-1}{2}} \equiv a^{1+4t+3} \equiv (a^{2t+2})^2 \pmod p,$$

yielding, $\sqrt{a} \equiv a^{2t+2} \bmod p$. Furthermore, $a^{2t+2} = (a^{t+1})^2 \in Q_p$, so we can do this repeatedly to find $r_p \equiv \sqrt[2^l]{a} \equiv a^{(2t+2)^l} \bmod p$. (This is why we require $p \equiv 7 \bmod 8$.) Analogously, we can find $r_q \equiv \sqrt[2^l]{a} \equiv a^{(2s+2)^l} \bmod q$, and using the *Chinese Remainder Theorem* (Lecture 5), we can find $r \equiv \sqrt[2^l]{a} \bmod n$. The cost of decrypting in this fashion is $O(k^3 \cdot l)$.

However, we can also compute r directly by computing $u = (2t + 2)^l$ and $v = (2s + 2)^l$ first, and in fact, if the length of the messages is known ahead of time, we can compute u and v off-line. In any event, the cost of decrypting then is simply the cost of computing $a^u \bmod p$ and $a^v \bmod q$, using the Chinese Remainder Theorem, and then computing p given r, just as when encrypting. This comes out to $O(k^3 + k^2 \cdot l) = O(k^3)$ if $l = O(k)$.

[1] More recent results indicate that this additional restriction may not be necessary.

EPE Passes Indistinguishable Security

We wish to show that EPE also passes indistinguishable security. To do this we use the notion of *pseudo-random number generators* (PSRG) introduced in the chapter on pseudo random number generation. Note that PSRG(r, i) $= f_i^l(r) \circ B_i(r) B_i(f_i(r)) B_i(f_i^2(r)) \dots B_i(f_i^{l-1}(r)) = a \circ p$ where p and a are generated while encrypting messages, ($\circ$ is the concatenation operator.) is a pseudo-random number generator. Indeed, this is the construction we used to prove the existence of PSRGs, given *one-way permutations*.

Certainly if the pad p were completely random, it would be impossible to decrypt the message since $m' = m \oplus p$ maps m' to a random string for any m. Since p is pseudo-random, it appears random to any PTM without further information i.e., the trapdoor t_i. However, the adversary does know $a = f_i^l(r)$, and we have to show that it cannot use this to compute p.

More precisely, we note that if there exists a PTM A that can distinguish between $(m \oplus p) \circ a$ and $(m \oplus R) \circ a$ where R is a completely random string from $\{0,1\}^l$, then it can distinguish between $p \circ a$ and $R \circ a$. We can use this then, as a statistical test to check whether a given string is a possible output of PSRG, which contradicts the claim that PSRG is pseudo-random, and thus the claim that f_i is one-way. It is left as an exercise to express this formally.

7.4.6 Practical RSA based encryption: OAEP

Consider a sender who holds a k-bit to k-bit trapdoor permutation f and wants to transmit a message x to a receiver who holds the inverse permutation f^{-1}. We concentrate on the case which arises most often in cryptographic practice, where $n = |x|$ is at least a little smaller than k. Think of f as the RSA function.

Encryption schemes used in practice have the following properties: encryption requires just one computation of f; decryption requires just one computation of f^{-1}; the length of the enciphered text should be precisely k; and the length n of the text x that can be encrypted is close to k. Examples of schemes achieving these conditions are [173, 113].

Unfortunately, these are heuristic schemes. A provably secure scheme would be preferable. We have now seen several provably good asymmetric (i.e. public key) encryption schemes. The most efficient is the Blum-Goldwasser scheme [39]. But, unfortunately, it still doesn't match the heuristic schemes in efficiency. Accordingly, practioners are continuing to prefer the heuristic constructions.

This section presents a scheme called the OAEP (Optimal Asymmetric Encryption Padding) which can fill the gap. It was designed by Bellare and Rogaway [23]. It meets the practical constraints but at the same time has a security that can be reasonably justified, in the following sense. The scheme can be proven secure assuming some underlying hash functions are ideal. Formally, the hash functions are modeled as random oracles. In implementation, the hash functions are derived from cryptographic hash functions.

This random oracle model represents a practical compromise under which we can get efficiency with reasonable security assurances. See [15] for a full discussion of this approach.

RSA-OAEP is currently included in several standards and draft standards and is implemented in various systems. In particular, it is the RSA PKCS#1 v2 encryption standard and is also in the IEEE P1363/P1363a draft standards. It is also used in the SET (Secure Electronic Transactions) protocol of Visa and Mastercard.

Simple embedding schemes and OAEP features

The heuristic schemes invariably take the following form: one (probabilistically, invertibly) embeds x into a string r_x and then takes the encryption of x to be $f(r_x)$.[2] Let's call such a process a *simple-embedding scheme*. We will take as our goal to construct provably-good simple-embedding schemes which allow n to be close to k.

[2] It is well-known that a naive embedding like $r_x = x$ is no good: besides the usual deficiencies of any deterministic encryption, f being a trapdoor permutation does not mean that $f(x)$ conceals all the interesting properties of x. Indeed it was exactly such considerations that helped inspire ideas like semantic security [98] and hardcore bits [40, 201].

The best known example of a simple embedding scheme is the RSA PKCS #1 standard. Its design is however ad hoc; standard assumptions on the trapdoor permutation there (RSA) do not imply the security of the scheme. In fact, the scheme succumbs to chosen ciphertext attack [34]. The OAEP scheme we discuss below is just as efficient as the RSA PKCS #1 scheme, but resists such attacks. Moreover, this resistance is backed by proofs of security. The new version of the RSA PKCS#1 standard, namely v2, uses OAEP.

OAEP is a simple embedding scheme that is bit-optimal (i.e., the length of the string x that can be encrypted by $f(r_x)$ is almost k). It is proven secure assuming the underlying hash functions are ideal. It is shown in [23] that RSA-OAEP achieves semantic security (as defined by [98]). It is shown in [84] (building on [188]) that it also achieves a notion called "plaintext-aware encryption" defined in [23, 21]. The latter notion is very strong, and in particular it is shown in [21] that semantic security plus plaintext awareness implies "ambitious" goals like chosen-ciphertext security and non-malleability [71] in the ideal-hash model.

Now we briefly describe the basic scheme and its properties. We refer the reader to [23] for full descriptions and to [23, 84] for proofs of security.

The scheme

Recall k is the security parameter, f mapping k-bits to k-bits is the trapdoor permutation. Let k_0 be chosen such that the adversary's running time is significantly smaller than 2^{k_0} steps. We fix the length of the message to encrypt as let $n = k - k_0 - k_1$ (shorter messages can be suitably padded to this length). The scheme makes use of a "generator" $G\colon \{0,1\}^{k_0} \to \{0,1\}^{n+k_1}$ and a "hash function" $H\colon \{0,1\}^{n+k_1} \to \{0,1\}^{k_0}$. To encrypt $x \in \{0,1\}^n$ choose a random k_0-bit r and set

$$\mathcal{E}^{G,H}(x) \quad = \quad f(x0^{k_1} \oplus G(r) \| r \oplus H(x0^{k_1} \oplus G(r))).$$

The decryption $\mathcal{D}^{G,H}$ is defined as follows. Apply f^{-1} to the ciphertext to get a string of the form $a\|b$ with $|a| = k - k_0$ and $|b| = k_0$. Compute $r = H(a) \oplus b$ and $y = G(r) \oplus a$. If the last k_1 bits of y are not all zero then reject; else output the first n bits of y as the plaintext.

The use of the redundancy (the 0^{k_1} term and the check for it in decryption) is in order to provide plaintext awareness.

Efficiency

The function f can be set to any candidate trapdoor permutation such as RSA [170] or modular squaring [164, 35]. In such a case the time for computing G and H is negligible compared to the time for computing f, f^{-1}. Thus complexity is discussed only in terms of f, f^{-1} computations. In this light the scheme requires just a single application of f to encrypt, a single application of f^{-1} to decrypt, and the length of the ciphertext is k (as long as $k \geq n + k_0 + k_1$).

The ideal hash function paradigm

As we indicated above, when proving security we take G, H to be random, and when we want a concrete scheme, G, H are instantiated by primitives derived from a cryptographic hash function. In this regard we are following the paradigm of [15] who argue that even though results which assume an ideal hash function do not provide provable security with respect to the standard model of computation, assuming an ideal hash function and doing proofs with respect to it provides much greater assurance benefit than purely *ad. hoc.* protocol design. We refer the reader to that paper for further discussion of the meaningfulness, motivation and history of this ideal hash approach.

Exact security

We want the results to be meaningful for practice. In particular, this means we should be able to say meaningful things about the security of the schemes for specific values of the security parameter (e.g., $k =$

512). This demands not only that we avoid asymptotics and address security "exactly," but also that we strive for security reductions which are as efficient as possible.

Thus the theorem proving the security of the basic scheme quantifies the resources and success probability of a potential adversary: let her run for time t, make $q_{\rm gen}$ queries of G and $q_{\rm hash}$ queries of H, and suppose she could "break" the encryption with advantage ϵ. It then provides an algorithm M and numbers t', ϵ' such that M inverts the underlying trapdoor permutation f in time t' with probability ϵ'. The strength of the result is in the values of t', ϵ' which are specified as functions of $t, q_{\rm gen}, q_{\rm hash}, \epsilon$ and the underlying scheme parameters k, k_0, n $(k = k_0 + n)$. Now a user with some idea of the (assumed) strength of a particular f (e.g., RSA on 512 bits) can get an idea of the resources necessary to break our encryption scheme. See [23] for more details.

OAEP achieves semantic security for any trapdoor function f, as shown in [23]. It achieves plaintext awareness, and thus security against chosen-ciphertext attack, when f is RSA, as shown in [84].

7.4.7 Enhancements

An enhancement to OAEP, made by Johnson and Matyas [114], is to use as redundancy, instead of the 0^{k_1} above, a hash of information associated to the key. This version of OAEP is proposed in the ANSI X9.44 draft standard.

7.5 Exploring Active Adversaries

Until now we have focused mostly on *passive* adversaries. But what happens if the adversaries are *active*? This gives rise to various stronger-than-semantic notions of security such as non-malleability [71], security against chosen ciphertext attack, and plaintext awareness [23, 21]. See [21] for a classification of these notions and discussion of relations among them.

In particular, we consider security against *chosen ciphertext attack*. In this model, we assume that our adversary has temporary access to the decoding equipment, and can use it to decrypt some cyphertexts that it chooses. Afterwards, the adversary sees the ciphertext it wants to decrypt without any further access to the decoding equipment. Notice that this is different from simply being able to generate pairs of messages and cyphertexts, as the adversary was always capable of doing that by simply encrypting messages of its choice. In this case, the adversary gets to choose the ciphertext and get the corresponding message from the decoding equipment.

We saw in previous sections that such an adversary could completely break Rabin's scheme. It is not known whether any of the other schemes discussed for PKC are secure in the presence of this adversary. However, attempts to provably defend against such an adversary have been made.

One idea is to put checks into the decoding equipment so that it will not decrypt cyphertexts unless it has evidence that someone knew the message (i.e., that the ciphertext was not just generated without knowledge of what the message being encoded was). We might think that a simple way to do this would be to require two distinct encodings of the same message, as it is unlikely that an adversary could find two separate encodings of the same message without knowing the message itself. Thus a ciphertext would be (α_1, α_2) where α_1, α_2 are chosen randomly from the encryptions of m.

Unfortunately, this doesn't work because if the decoding equipment fails to decrypt the ciphertext, the adversary would still gain some knowledge, i.e., that α_1 and α_2 do not encrypt the same message. For example, in the *probabilistic encryption* scheme proposed last lecture, an adversary may wish to learn the hard-core bit $B_i(y)$ for some unknown y, where it has $f_i(y)$. Given decoding equipment with the protection described above, the adversary could still discover this bit as follows:

(1) Pick $m \in \mathcal{M}(1^l)$, the message space, and let b be the last bit of m.

(2) Pick $\alpha_1 \in E(i, m)$ randomly and independently.

(3) Recall that $\alpha_1 = (f_i(x_1), f_i(x_2), \ldots, f_i(x_l))$, with x_j chosen randomly from D_i for $j = 1, 2, \ldots, l$. Let $\alpha_2 = (f_i(x_1), \ldots, f_i(x_{l-1}), f_i(y))$.

(4) Use the decoding equipment on $c = (\alpha_1, \alpha_2)$. If it answers m, then $B_i(y) = b$. If it doesn't decrypt c, then $B_i(y) = \overline{b}$.

What is done instead uses the notion of *Non-Interactive Zero-Knowledge Proofs* (NIZK) [41, 146]. The idea is that anyone can check a NIZK to see that it is correct, but no knowledge can be extracted from it about what is being proved, except that it is correct. Shamir and Lapidot have shown that if trapdoor functions exist, then NIZKs exist. Then a ciphertext will consist of three parts: two distinct encodings α_1, α_2 of the message, and a NIZK that α_1 and α_2 encrypt the same message. Then the decoding equipment will simply refuse to decrypt any ciphertext with an invalid NIZK, and this refusal to decrypt will not give the adversary any new knowledge, since it already knew that the proof was invalid.

The practical importance of chosen ciphertext attack is illustrated in the recent attack of Bleichenbacher on the RSA PKCS #1 encryption standard, which has received a lot of attention. Bleichenbacher [34] shows how to break the scheme under a chosen ciphertext attack. One should note that the OAEP scheme discussed in Section 7.4.6 above is immune to such attacks.

Chapter 8

Message authentication

A message authentication scheme enables parties in possession of a shared secret key to achieve the goal of data integrity. This is the second main goal of private-key cryptography.

8.1 Introduction

8.1.1 The problem

Suppose you receive a communication that purports to come from a certain entity, call it S. Here S might be one of many different types of entities: for example a person, or a corporation, or a network address. You may know that it is S that purports to send this communication for several reasons. For example, S's identifier could be attached to the communication. The identifier here is a public identity that is known to belong to S: for example, if S is a person or corporation, typically just the name of the person or corporation; if a network address, the address itself. Or, it may be that from the context in which the communication is taking place you are expecting the communication to be from a certain known entity S.

In many such settings, security requires that the receiver have confidence that the communicated data does originate with the claimed sender. This is necessary to implement access control, so that services and information are provided to the intended parties. The risk is that an attacker will "impersonate" S. It will send a communication with S's identity attached, so that the receiver is lead to believe the communication is from S. This can have various undesirable consequences. Examples of the damage caused abound; here are a few that one might consider.

An on-line stock broker S replies to a quote request by sending the value of a certain stock, but an adversary modifies the transmission, changing the dollar value of the quote. The person who requested the quote receives incorrect information and could be lead to make a financially detrimental action. This applies to any data being obtained from a database: its value lies in its authenticity as vouched for by the database service provider. Or consider S needing to send data of only two kinds, say "buy" and "sell", or "fire" and "don't fire". This might be encoded in a single bit, and if an adversary flips this bit, the wrong action is taken. Or consider electronic banking. S sends its bank a message asking that \$200 be transferred from her account to A's account. A might play the role of adversary and change the sum to \$2,000.

In fact the authenticity of data transmitted across a network can be even more important to security than privacy of the data when it comes to enabling network applications and commerce.

This ability to send data purporting to be from a source it is not requires an *active attack* on the part of the adversary. That is, the adversary must have the means to modify transmitted communications or introduce new ones. These abilities depend on the setting. It may be hard to introduce data into a dedicated phone line, but not on a network like the Internet. It would be advisable to assume adversaries do have such abilities.

The authentication problem is very different from the encryption problem. We are not worried about secrecy of the data; let the data be in the clear. We are worried about the adversary modifying it.

8.1.2 Encryption does not provide data integrity

We know how to encrypt data so as to provide privacy. Something often suggested (and done) is to encrypt to provide data integrity, as follows. Fix a symmetric encryption scheme $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$, and let parties S and B share a key K for this scheme. When S wants to send a message M to B, she encrypts it, transferring a ciphertext C generated via $C \stackrel{R}{\leftarrow} \mathcal{E}_K(M)$. B decrypts it, recovering $\mathcal{D}_K(C)$.

The argument that this provides data integrity is as follows. Suppose S transmits, as in the above example, a message M to its bank B asking that \$200 be transferred from S's account to A's account. A wants to change the \$200 to \$2,000. If M is sent in the clear, A can easily modify it. But if M is encrypted so that ciphertext C is sent, how is A to modify C so as to make B recover the modified message M'? It does not know the key K, so cannot encrypt the modified message M'. The privacy of the message appears to make tampering difficult.

This argument is fallacious. To see the flaws let's first look at a counter-example and then the issues. Consider, say the randomized CTR scheme, using some block cipher F, say RC6. We proved in the chapter on symmetric encryption that this was a secure encryption scheme assuming RC6 is a pseudorandom function. For simplicity say that the message M above is a single 128 bit block, containing account information for the parties involved, plus a field for the dollar amount. To be concrete, the last 16 bits of the 128-bit block hold the dollar amount encoded as a 16-bit binary number. (So the amount must be at most \$65,535.) Thus, the last 16 bits of M are 0000000011001000, the binary representation of the integer 200. We assume that A is aware that the dollar amount in this electronic check is \$200; this information is not secret. Now recall that under randomized CTR encryption the ciphertext transmitted by S has the form $C = \langle r \rangle y$ where $y = F_K(\langle r+1 \rangle) \oplus M$. A's attack is as follows. It gets $C = \langle r \rangle y$ and sets $y' = y \oplus 0^{112}0000011100001000$. It sets $C' = \langle r \rangle y'$ and forwards C' to B. B will decrypt this, so that it recovers the message $F_K(\langle r+1 \rangle) \oplus y'$. Denoting it by M', its value is

$$
\begin{aligned}
M' &= F_K(\langle r+1 \rangle) \oplus y' \\
&= F_K(\langle r+1 \rangle) \oplus y \oplus 0^{112}0000011100001000 \\
&= M \oplus 0^{112}0000011100001000 \\
&= M_{\text{prefix}}0000011111000000
\end{aligned}
$$

where M_{prefix} is the first 112 bits of the original message M. Notice that the last 16 bits of M' is the binary representation of the integer 2000, while the first 112 bits of M' are equal to those of M. So the end result is that the bank B will be misled into executing the transaction that S requested except that the dollar amount has been changed from 200 to 2000.

There are many possible reactions to this counter-example, some sound and some unsound. Let's take a look at them.

What you should conclude from this is that encryption does not provide data integrity. With hindsight, it is pretty clear. The fact that data is encrypted need not prevent an adversary from being able to make the receiver recover data different from that which the sender had intended, for many reasons. First, the data, or some part of it, might not be private at all. For example, above, some information about M was known to A: as the recipient of the money, A can be assumed to know that the amount will be \$200, a sum probably agreed upon beforehand. However, even when the data is not known a priori, an adversary can make the receiver recover something incorrect. For example with the randomized CTR scheme, an adversary can effectively flip an bit in the message M. Even if it does not know what is the value of the original bit, damage can be caused by flipping it to the opposite value. Another possibility is for the adversary to simply transmit some string C. In many encryption schemes, including CTR and CBC encryption, C will decrypt to something, call it M. The adversary may have no idea what M will be, but we should still view it as wrong that the receiver accepts M as being sent by S when in fact it wasn't.

Now here is another possible reaction to the above counter-example: CTR mode encryption is bad, since it permits the above attack. So one should not use this mode. Let's use CBC instead; there you can't flip message bits by flipping ciphertext bits.

This is an unsound reaction to the counter-example. Nonetheless it is not only often voiced, but even printed. Why is it unsound? Because the point is not the specific attack on CTR, but rather to recognize the disparity in *goals*. There is simply no reason to expect encryption to provide integrity. Encryption was not designed to solve the integrity problem. The way to address this problem is to first pin down precisely what is the problem, and then seek a solution. Nonetheless there are many existing systems, and places in the literature, where encryption and authentication are confused, and where the former is assumed to provide the latter.

It turns out that CBC encryption can also be attacked from the integrity point of view, again leading to claims in some places that it is not a good encryption mechanism. Faulting an encryption scheme for not providing authenticity is like faulting a screwdriver because you could not cut vegetables with it. There is no reason to expect a tool to solve a problem it was not designed to solve.

It is sometimes suggested that one should "encrypt with redundancy" to provide data integrity. That is, the sender S pads the data with some known, fixed string, for example 128 bits of zeros, before encrypting it. The receiver decrypts the ciphertext and checks whether the decrypted string ends with 128 zeros. If not, the receiver rejects the transmission as unauthentic; else it outputs the rest of the string as the actual data. This too can fail in general; for example it is easy to see that with CTR mode encryption, an attack just like the above applies. It can be attacked under CBC encryption too.

Good cryptographic design is *goal oriented*. One must first understand and formalize the goal. Only then does one have the basis on which to design and evaluate potential solutions. Accordingly, our next step will be to come up with a definition of message authentication schemes and their security.

8.2 Message authentication schemes

In the private key setting, the primitive used to provide data integrity is a message authentication scheme. This is a scheme specified by three algorithms: a key generation algorithm $\mathcal{K}$; a tagging algorithm $\mathcal{T}$ and a verification algorithm $\mathcal{V}$. The sender and receiver are assumed to be in possession of a key K generated via $\mathcal{K}$ and not known to the adversary. When the sender wants to send M in an authenticated way to B, she computes a tag σ for M as a function of M and the secret key K shared between the sender and receiver, in a manner specified by the tagging algorithm; namely, she sets $\sigma \leftarrow \mathcal{T}_K(M)$. This tag accompanies the message in transmission; that is, S transmits the pair M, σ to B. (Notice that the message is sent in the clear. Also notice the transmission is longer than the original message by the length of the tag σ.) Upon receiving a transmission M', σ' purporting to be from S, the receiver B verifies the authenticity of the tag by using the specified verification procedure, which depends on the message, tag, and shared key. Namely he computes $\mathcal{V}_K(M', \sigma')$, whose value is a bit. If this value is 1, it is read as saying the data is authentic, and so B accepts it as coming from S. Else it discards the data as unauthentic.

As we have discussed before, there are many ways in which the receiver might know that the transmission purports to be from S. For example, S's identity might accompany the transmission, or the communication may be taking place in a context where the receiver is already expecting to be interacting with S. Accordingly, we do not address this issue explicitly in the model, preferring to leave it as being attended to "out of band".

A viable scheme of course requires some security properties. But these are not our concern now. First we want to pin down what constitutes a specification of a scheme, so that we know what are the kinds of objects whose security we want to assess. Let us now summarize the above in a definition.

Definition 8.1 A *message authentication scheme* $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ consists of three algorithms, as follows:

- The *key generation* algorithm $\mathcal{K}$ is a randomized algorithm that returns a key K; we write $K \stackrel{R}{\leftarrow} \mathcal{K}$
- The *tagging* algorithm $\mathcal{T}$ is a (possibly randomized) algorithm that takes the key K and a message M to return a *tag* σ; we write $\sigma \leftarrow \mathcal{T}_K(M)$
- The *verification* algorithm $\mathcal{V}$ is a deterministic algorithm that takes the key K, a message M, and a candidate tag σ for M to return a bit; we write $d \leftarrow \mathcal{V}_K(M, \sigma)$.

Associated to the scheme is a *message space* Plaintexts from which M is allowed to be drawn. We require that $\mathcal{V}_K(M, \mathcal{T}_K(M)) = 1$ for all $M \in$ Plaintexts. ▮

The last part of the definition says that tags that were correctly generated will pass the verification test. This simply ensures that authentic data will be accepted by the receiver.

The tagging algorithm might be randomized, meaning internally flip coins and use these coins to determine its output. In this case, there may be many correct tags associated to a single message M. The algorithm might also be stateful, for example making use of a counter that is maintained by the sender. In that case the tagging algorithm will access the counter as a global variable, updating it as necessary.

Unlike encryption schemes, whose encryption algorithms must be either randomized or stateful for the scheme to be secure, a deterministic, stateless tagging algorithm is not only possible, but common, and in that case we refer to the message authentication scheme as deterministic. In this case, verification can be performed by computing the correct tag and checking that the transmitted tag equals the correct one. That is, the verification algorithm is simply the following:

Algorithm $\mathcal{V}_K(M, \sigma)$
 $\sigma' \leftarrow \mathcal{T}_K(M)$
 If $\sigma = \sigma'$ then return 1 else return 0

Hence when the tagging algorithm is deterministic, the verification algorithm need not be explicitly specified; it is understood that it is the above.

Example 8.2 Let $F: \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a block cipher. The CBC MAC with base family F is the (deterministic) message authentication scheme in which the tag of a message is the last block of ciphertext obtained by processing the message in CBC mode with zero IV. In more detail, the scheme is $\mathcal{CBC}\text{-}\mathcal{MAC}^F = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ with the algorithms defined as follows. We let $\mathcal{K}$ be the algorithm which picks a k-bit key K by flipping k coins and returning their outcome. The message M input to the algorithms below must have length a multiple of l bits.

Algorithm $\mathcal{T}_K(M)$ Divide M into l bit blocks, $M = x_1 \ldots x_n$ $y_0 \stackrel{R}{\leftarrow} 0^l$ For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(y_{i-1} \oplus x_i)$ Return y_n	Algorithm $\mathcal{V}_K(M, \sigma)$ Divide M into l bit blocks, $M = x_1 \ldots x_n$ $y_0 \stackrel{R}{\leftarrow} 0^l$ For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(y_{i-1} \oplus x_i)$ If $y_n = \sigma$ then return 1 else return 0

Since the tagging algorithm is deterministic, the verification algorithm simply checks whether or not σ is the correct tag, as discussed above.

The choice of message space is important for the security of the CBC MAC as we will see later. If messages of varying length are allowed the scheme is insecure, but if the length of the messages is restricted to some single pre-specified and fixed value, the scheme is secure. ▮

8.3 A notion of security

We will first try to build up some intuition about what properties a message authentication scheme should have to call it "secure", and then distill a formal definition of security.

8.3.1 Issues in security

The goal that we seek to achieve in using a message authentication scheme is to be able to detect any attempt by the adversary to modify the transmitted data. What we are afraid of is that the adversary can produce messages that the receiver accepts as coming from the legitimate sender S when in fact S did not send this message. That is, A can produce M, σ such that $\mathcal{V}_K(M, \sigma) = 1$, but M did not originate with the sender S. This is called a *forgery*.

Perhaps the first question one might ask is whether one should try to gauge the value of the forgery to the adversary, for example by asking what is the content of the message. For example, say the messages are expected to have certain formats, and the forgery is just a random string. Should this really be viewed as a forgery? The answer is *yes*. We have seen this general issue before. It would be unwise to make assumptions about how the messages are formatted or interpreted. Good protocol design means the security is guaranteed no matter what is the application. Accordingly we view the adversary as successful if she produces M, σ such that the sender never authenticated M but $\mathcal{V}_K(M, \sigma) = 1$.

In some discussions of security in the literature, the adversary's goal is viewed as being to recover the shared secret key K. Certainly if she could do this, it would be a disaster, since she could forge anything. It is important to understand, however, that she might be able to forge without recovering the key. Thus, we are here making the notion of adversarial success more liberal: what counts is forgery, not key recovery. So a secure scheme has a stronger guarantee of security.

In making forgeries we must consider various attacks, of increasing severity. The simplest setting is that the sender has not yet sent any transmissions, and the adversary may simply attempt to concoct a pair M, σ which passes the verification test, namely such that $\mathcal{V}_K(M, \sigma) = 1$. This is called a no-message attack.

However, the adversary also has the ability to see what is transmitted. She might try to make forgeries based on this information. So suppose the sender sends the transmission M, σ consisting of some message M and its legitimate (correct) tag σ. The receiver will certainly accept this. At once, a simple attack comes to mind. The adversary can just copy this transmission. That is, she stores M, σ, and at some later time re-transmits it. If the receiver accepted it the first time, he will do so again. This is called a replay attack. Is this a valid forgery? In real life it probably should be so considered. Say the first message was "Transfer \$1000 from my account to the account of B." B suddenly sees a way of enriching herself. She keeps replaying this message and her bank balance increases.

It is important to protect against replay attacks. But for the moment we will not try to do this. We will say that a replay is NOT a valid forgery. To be valid, a forgery must be of a message M which was *not* transmitted by the sender. We will later see that we can achieve security against replay by addition of time stamps or counters to any normal message authentication mechanism. At this point, separating the issues results in a cleaner problem and allows greater *modularity* in protocol design. Namely we will cut up the problem into logical parts and solve them one by one. Accordingly from now on don't regard replay as a valid attack.

So if the adversary wants to be successful, it must somehow use the valid transmission M, σ to concoct a pair M', σ' such that $M \neq M'$ but $\mathcal{V}_K(M', \sigma') = 1$. If she can do this, we say she is successful. Thus, we have a very *liberal* notion of adversarial success. So when we say a scheme is secure, it is secure in a very strong sense.

We have allowed the adversary to see an example message. Of course, it may see more than one, and forgery must still be hard. We expect that the adversary's ability to forge will increase as it sees more examples of legitimately authenticated data, so that as usual we expect the notion of security to be quantitative, with the adversary's success probability being a function of of the number q of legitimate pairs seen.

In many settings, it is possible for the adversary to influence the choice of legitimate messages being tagged. In the worst case, we can think the adversary herself chooses these messages. This is called a chosen plaintext attack. At first glance a chosen plaintext attack may seem unrealistically strong. There are two arguments against this view. First, we will see examples where such an attack is quite realistic. Second, recall our general principles. We want to design schemes which are secure in any usage. This requires that we make "worst case" notions of security in which if we err, it is on the side of caution, allowing the adversary as much power as possible. Since eventually we will be able to design schemes that meet such stringent notions of security, we only gain by the process.

One instance of a chosen-message attack is a setting in which S is forwarding to B data that it receives from C, and authenticating that data based on a key K shared between S and B, in the process. If C wants to play an adversarial role, C can choose the data as it wishes, and then see the corresponding tags as sent by S to B. Other scenarios are also possible.

In summary, we want a notion of security to capture the following. We allow an adversary to mount a chosen-message attack on the sender, obtaining from the sender correct tags of messages of the adversary's choice. Then, the adversary attempts a forgery, and is declared successful if the forgery is valid (meaning accepted by the receiver) and the message in it was never authenticated by the sender.

8.3.2 A notion of security

Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be an arbitrary message authentication scheme. Our goal is to formalize a measure a insecurity against forgery under chosen-message attack for this scheme. As discussed above, we envision a chosen-message attack mounted on the sender, and then a forgery attempt directed at the receiver. In formalizing this we begin by distilling the key aspects of the model. There is no need, in fact, to think explicitly of the sender and receiver as animate entities. The result of the adversary requesting the sender to authenticate a message M is that the adversary obtains a tag σ generated via $\sigma \leftarrow \mathcal{T}_K(M)$, where K is the key shared between sender and receiver. Thus, we may as well simplify the situation and think of the adversary as having oracle access to the algorithm $\mathcal{T}_K(\cdot)$. It can query this oracle at any point M in the message space and obtain the result. Correspondingly we eliminate the receiver from the picture and focus only on the verification process. The adversary will eventually output a pair M, σ and it is a valid forgery as long as $\mathcal{V}_K(M, \sigma) = 1$ and M was never a query to the tagging oracle.

Note the key K is not directly given to the adversary, and neither are any random choices or counter used by the tagging algorithm; the adversary sees only the generated tag σ. However σ is a function of the key and the random choices or counter, so it might provide information about these, to an extent that depends on the scheme. If the tag allows the adversary to infer the key, the scheme will certainly be insecure.

The adversary's actions are thus viewed as divided into two phases. The first is a "learning" phase in which it is given oracle access to $\mathcal{T}_K(\cdot)$, where K was a priori chosen at random according to $\mathcal{K}$. It can query this oracle up to q times, in any manner it pleases, as long as all the queries are messages in the underlying message space $\mathsf{Plaintexts}$ associated to the scheme. Once this phase is over, it enters a "forgery" phases, in which it outputs a pair (M, σ). The adversary is declared successful if $\mathcal{V}_K(M, \sigma) = 1$ and M was never a query made by the adversary to the tagging oracle. Associated to any adversary A is thus a success probability. (The probability is over the choice of key K, any probabilistic choices that $\mathcal{T}$ might make, and the probabilistic choices, if any, that A makes.) The insecurity of the scheme is the success probability of the "cleverest" possible adversary, amongst all adversaries restricted in their resources to some fixed amount. We choose as resources the running time of the adversary, the number of queries it makes, and the total bit-length of all queries combined plus the bit-length of the output message M in the forgery.

Formally we define the "experiment of running the adversary" A in an attack on scheme $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ as the following.

Experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{MA},A}$
  Let $K \stackrel{R}{\leftarrow} \mathcal{K}$
  Let $(M, \sigma) \leftarrow A^{\mathcal{T}_K(\cdot)}$
  If $\mathcal{V}_K(M, \sigma) = 1$ and M was not a query of A to its oracle
  Then return 1 else return 0

Definition 8.3 Let $\mathcal{MA} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be a message authentication scheme, and let A be an adversary that has access to an oracle. Let $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA},A}$ be the probability that experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{MA},A}$ returns 1. Then for any t, q, μ let

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}}(t, q, \mu) \quad = \quad \max_{A} \{ \mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA},A} \}$$

where the maximum is over all A running in time t, making at most q oracle queries, and such that the sum of the lengths of all oracle queries plus the length of the message M in the output forgery is at most μ bits. ∎

In practice, the queries correspond to tagged messages sent by the legitimate sender, and it would make sense that getting these examples is more expensive than just computing on one's own. That is, we would expect q to be smaller than t. That is why q, μ are resources separate from t.

8.3.3 Using the definition: Some examples

Let us examine some example message authentication schemes and use the definition to assess their strengths and weaknesses. We fix a PRF $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$. Our first scheme $\mathcal{MA}_1 = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ works like this–

Algorithm $\mathcal{T}_K(M)$	Algorithm $\mathcal{V}_K(M, \sigma)$
Divide M into l bit blocks, $M = x_1 \ldots x_n$	Divide M into l bit blocks, $M = x_1 \ldots x_n$
For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(x_i)$	For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(x_i)$
$\sigma \leftarrow y_1 \oplus \cdots \oplus y_n$	$\sigma' \leftarrow y_1 \oplus \cdots \oplus y_n$
Return σ	If $\sigma = \sigma'$ then return 1 else return 0

Now let us try to assess the security of this message authentication scheme.

Suppose the adversary wants to forge the tag of a certain given message M. A priori it is unclear this can be done. The adversary is not in possession of the secret key K, so cannot compute F_K and hence will have a hard time computing σ. However, remember that the notion of security we have defined says that the adversary is successful as long as it can produce a correct tag for *some* message, not necessarily a given one. We now note that even without a chosen-message attack (in fact without seeing any examples of correctly tagged data) the adversary can do this. It can choose a message M consisting of two equal blocks, say $M = x \| x$ where x is some l-bit string, set $\sigma \leftarrow 0^l$, and output M, σ. Notice that $\mathcal{V}_K(M, \sigma) = 1$ because $F_K(x) \oplus F_K(x) = 0^l = \sigma$. So the adversary is successful. In more detail, the adversary is:

Adversary $A_1^{\mathcal{T}_K(\cdot)}$
  Let x be some l-bit string
  Let $M \leftarrow x \| x$
  Let $\sigma \leftarrow 0^l$
  Return (M, σ)

Then $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}_1, A_1} = 1$. Furthermore A_1 makes no oracle queries, uses $t = O(l)$ time, and outputs an l-bit message in its forgery, so we have shown that

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}_1}(t, 0, l) \quad = \quad 1 .$$

That is, the scheme $\mathcal{MA}_1$ is totally insecure.

There are many other attacks. For example we note that if $\sigma = F_K(M_1) \oplus F_K(M_2)$ is the tag of $M_1 M_2$ then σ is also the correct tag of $M_2 M_1$. So it is possible, given the tag of a message, to forge the tag of a new message formed by permuting the blocks of the old message. We leave it to the reader to specify the corresponding adversary and compute its advantage.

Let us now try to strengthen the scheme to avoid these attacks. Instead of applying F_K to a data block, we will first prefix the data block with its index. To do this we pick some parameter m with $1 \leq m \leq l - 1$, and write the index as an m-bit string. The message authentication scheme $\mathcal{MA}_1 = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ looks like this:

Algorithm $\mathcal{T}_K(M)$ Divide M into $l-m$ bit blocks, $M = x_1 \dots x_n$ For $i = 1, \dots, n$ do $y_i \leftarrow F_K(\langle i \rangle \| x_i)$ $\sigma \leftarrow y_1 \oplus \cdots \oplus y_n$ Return σ	Algorithm $\mathcal{V}_K(M, \sigma)$ Divide M into $l-m$ bit blocks, $M = x_1 \dots x_n$ For $i = 1, \dots, n$ do $y_i \leftarrow F_K(\langle i \rangle \| x_i)$ $\sigma' \leftarrow y_1 \oplus \cdots \oplus y_n$ If $\sigma = \sigma'$ then return 1 else return 0

As the code indicates, we divide M into smaller blocks: not of size l, but of size $l-m$. Then we prefix the i-th message block with the value i itself, the block index, written in binary. Above $\langle i \rangle$ denotes the integer i written as a binary string of m bits. It is to this padded block that we apply F_K before taking the XOR.

Note that encoding of the block index i as an m-bit string is only possible if $i < 2^m$. This means that we cannot authenticate a message M having more than 2^m blocks. That is, the message space is confined to strings of length at most $(l-m)(2^m - 1)$, and, for simplicity, of length a multiple of $l-m$ bits. However this is hardly a restriction in practice since a reasonable value of m, like $m = 32$, is large enough that typical messages fall in the message space, and since l is typically at least 64, we have at least 32 bits left for the data itself.

Anyway, the question we are really concerned with is the security. Has this improved with respect to $\mathcal{MA}_1$? Begin by noticing that the attacks we found on $\mathcal{MA}_1$ no longer work. For example take the adversary A_1 above. (It needs a minor modification to make sense in the new setting, namely the chosen block x should not be of length l but of length $l-m$. Consider this modification made.) What is its success probability when viewed as an adversary attacking $\mathcal{MA}_2$? The question amounts to asking what is the chance that $\mathcal{V}_K(M, \sigma) = 1$ where $\mathcal{V}$ is the verification algorithm of our amended scheme and M, σ is the output of A_1. The verification algorithm will compute $\sigma' = F_K(\langle 1 \rangle \| x) \oplus F_K(\langle 2 \rangle \| x)$ and test whether this equals 0^l, the value of σ output by A. This happens only when

$$F_K(\langle 1 \rangle \| x) \;=\; F_K(\langle 2 \rangle \| x) \;,$$

and this is rather unlikely. For example if we are using a block cipher it never happens because F_K is a permutation. Even when F is not a block cipher, this event has very low probability as long as F is a good PRF; specifically, $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}_2, A_1}$ is at most $\mathbf{Adv}^{\text{prf}}_F(t, 2)$ where $t = O(l)$. (A reader might make sure they see why this bound is true.) So the attack has very low success probability.

Similar arguments show that the second attack discussed above, namely that based on permuting of message blocks, also has low success against the new scheme. Why? In the new scheme

$$\begin{aligned}
\mathcal{T}_K(M_1 M_2) &= F_K(\langle 1 \rangle \| M_1) \oplus F_K(\langle 2 \rangle \| M_2) \\
\mathcal{T}_K(M_2 M_1) &= F_K(\langle 1 \rangle \| M_2) \oplus F_K(\langle 2 \rangle \| M_1) \;.
\end{aligned}$$

These are unlikely to be equal for the same reasons discussed above. As an exercise, a reader might upper bound the probability that these values are equal in terms of the value of the insecurity of F at appropriate parameter values.

However, $\mathcal{MA}_2$ is still insecure. The attacks however require a more non-trivial usage of the chosen-message attacking ability. The adversary will query the tagging oracle at several related points and combine the responses into the tag of a new message. We call it A_2–

Adversary $A_2^{\mathcal{T}_K(\cdot)}$
  Let x_1, x_1' be distinct, $l-m$ bit strings, and let x_2, x_2' be distinct $l-m$ bit strings
  $\sigma_1 \leftarrow \mathcal{T}_K(x_1 x_2)$; $\sigma_2 \leftarrow \mathcal{T}_K(x_1 x_2')$; $\sigma_3 \leftarrow \mathcal{T}_K(x_1' x_2)$
  $\sigma \leftarrow \sigma_1 \oplus \sigma_2 \oplus \sigma_3$
  Return $(x_1' x_2', \sigma)$

We claim that $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}_2, A_2} = 1$. Why? This requires two things. First that $\mathcal{V}_K(x_1' x_2', \sigma) = 1$, and second that $x_1' x_2'$ was never a query to $\mathcal{T}_K(\cdot)$ in the above code. The latter is true because we insisted above that $x_1 \neq x_1'$ and $x_2 \neq x_2'$, which together mean that $x_1' x_2' \notin \{x_1 x_2, x_1 x_2, x_1' x_2\}$. So now let us check the first claim. We use the definition of the tagging algorithm to see that

$$\begin{aligned} \sigma_1 &= F_K(\langle 1\rangle \| x_1) \oplus F_K(\langle 2\rangle \| x_2) \\ \sigma_2 &= F_K(\langle 1\rangle \| x_1) \oplus F_K(\langle 2\rangle \| x_2') \\ \sigma_3 &= F_K(\langle 1\rangle \| x_1') \oplus F_K(\langle 2\rangle \| x_2) \,. \end{aligned}$$

Now look how A_2 defined σ and do the computation; due to cancellations we get

$$\begin{aligned} \sigma &= \sigma_1 \oplus \sigma_2 \oplus \sigma_3 \\ &= F_K(\langle 1\rangle \| x_1') \oplus F_K(\langle 2\rangle \| x_2') \,. \end{aligned}$$

This is indeed the correct tag of $x_1'x_2'$, meaning the value σ' that $\mathcal{V}_K(x_1'x_2', \sigma)$ would compute, so the latter algorithm returns 1, as claimed. In summary we have shown that

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA}_2}(t, 3, 4(l-m)) \quad = \quad 1 \,,$$

where $t = O(l)$. So the scheme $\mathcal{MA}_2$ is also totally insecure.

Later we will see how a slight modification of the above actually yields a secure scheme. For the moment however we want to stress a feature of the above attacks. Namely that these attacks *did not cryptanalyze the PRF F*. The cryptanalysis of the message authentication schemes did not care anything about the structure of F; whether it was DES, RC6, or anything else. They found weaknesses in the message authentication schemes themselves. In particular, the attacks work just as well when F_K is a random function, or a "perfect" cipher. This illustrates again the point we have been making, about the distinction between a tool (here the PRF) and its usage. We need to make better usage of the tool, and in fact to tie the security of the scheme to that of the underlying tool in such a way that attacks like those illustrated here are provably impossible under the assumption that the tool is secure.

8.4 The XOR schemes

We now consider a family of message authentication schemes called XOR MACs due to [11], and show that they are secure.

8.4.1 The schemes

The schemes use a similar paradigm to the second example scheme discussed above. A certain block size l is given. We choose a parameter m, this time $1 \le m \le l-2$. We view the message M as being divided into blocks of size $l-m-1$, and denote the i-th block by x_i. We denote by $\langle i\rangle$ the encoding of integer i as an m-bit binary string. We assume that $|M| \le (l-m-1)(2^m-1)$ so that the index of any block can be written as an m-bit string.

Both to define the schemes and to analyze their security it is helpful to first introduce an auxiliary function which we call XOR-Tag$^?(\cdot,\cdot)$. It takes an oracle for a function $f\colon \{0,1\}^l \to \{0,1\}^L$. It also takes two inputs. The first is an $l-1$ bit string which we call s, and whose role will emerge later. The second is the message M discussed above. It processes these inputs using f as indicated below and returns a value we call τ.

Algorithm XOR-Tag$^f(s, M)$
  Divide M into $l-m-1$ bit blocks, $M = x_1 \ldots x_n$
  $y_0 \leftarrow f(0\|s)$
  For $i = 1, \ldots, n$ do $y_i \leftarrow f(1\|\langle i\rangle\|x_i)$
  $\tau \leftarrow y_0 \oplus y_1 \oplus \cdots \oplus y_n$
  Return τ

Our auxiliary function applies f at $n+1$ points, each of these points being an l-bit string. The first point is $0\|s$. Namely we prefix the $(l-1)$-bit string s with a zero bit, which brings the total to l-bits, and then apply f to get y_0. The other n points on which f is applied are all prefixed with the bit 1. That is followed by an encoding of the block index and then the data block itself, for a total length of $1+m+(l-m-1)=l$ bits.

We are now ready to describe the scheme. We fix a family of functions $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$, and the key for the message authentication scheme is simply a k-bit key K for the family F, which specifies a specific function F_K. The parties will use F_K in the role of f above. There are actually two versions of the scheme. One is deterministic and stateful, making use of a global counter; the other is stateless and randomized. The difference is only in how s is chosen. We begin with the counter version. Here the sender maintains an integer counter ctr, initially 0. We denote by $\langle ctr \rangle$ its encoding as an $l-1$ bit integer. (The counter thus ranges from 0 to $2^{l-1}-1$. Note that when i is a block index, $\langle i \rangle$ also denotes its binary encoding, but as an m bit string, so that the notation $\langle\cdot\rangle$ is a bit overloaded in that the length of the string returned depends on the context of its argument, but hopefully this will not cause too much confusion.) The *counter-based XOR MAC* scheme using F, denoted $\mathcal{C}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \mathcal{T}, \mathcal{V})$, works as follows–

Algorithm $\mathcal{T}_K(M)$	Algorithm $\mathcal{V}_K(M,\sigma)$
$\tau \leftarrow \text{XOR-Tag}^{F_K}(\langle ctr \rangle, M)$	Parse σ as (s,τ)
$\sigma \leftarrow (\langle ctr \rangle, \tau)$	$\tau' \leftarrow \text{XOR-Tag}^{F_K}(s, M)$
$ctr \leftarrow ctr + 1$	If $\tau = \tau'$ then return 1 else return 0
Return σ	

In other words, the tag for message $M = x_1 \ldots x_n$ is a pair consisting of the current counter value ctr encoded in binary, and the subtag τ, where

$$\tau \quad = \quad F_K(0\|\langle ctr \rangle) \oplus F_K(1\|\langle 1 \rangle\|x_1) \oplus \cdots \oplus F_K(1\|\langle n \rangle\|x_n)\ .$$

To verify the received tag $\sigma = (\langle ctr \rangle, \tau)$ the verification algorithm recomputes the correct subtag, calling it τ', as a function of the given counter value, and then checks that this subtag matches the one provided in σ.

The randomized version of the scheme, namely the *randomized XOR MAC* scheme using F, is denoted $\mathcal{R}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \mathcal{T}, \mathcal{V})$. It simply substitutes the counter with a random $(l-1)$-bit value chosen anew at each application of the tagging algorithm. In more detail, the algorithms work as follows–

Algorithm $\mathcal{T}_K(M)$	Algorithm $\mathcal{V}_K(M,\sigma)$
$r \stackrel{R}{\leftarrow} \{0,1\}^{l-1}$	Parse σ as (r,τ)
$\tau \leftarrow \text{XOR-Tag}^{F_K}(r, M)$	$\tau' \leftarrow \text{XOR-Tag}^{F_K}(r, M)$
$\sigma \leftarrow (r, \tau)$	If $\tau = \tau'$ then return 1 else return 0
Return σ	

In other words, the tag for message $M = x_1 \ldots x_n$ is a pair consisting of a random value r and the subtag τ where

$$\tau \quad = \quad F_K(0\|r) \oplus F_K(1\|\langle 1 \rangle\|x_1) \oplus \cdots \oplus F_K(1\|\langle n \rangle\|x_n)\ .$$

To verify the received tag $\sigma = (r, \tau)$ the verification algorithm recomputes the correct subtag, calling it τ', as a function of the given value r, and then checks that this subtag matches the one provided in σ.

8.4.2 Security considerations

Before we consider security it is important to clarify one thing about possible forgeries. Recall that a forgery is a pair M, σ consisting of a message M and a value σ which purports to be a valid tag for M. In the XOR schemes, a tag σ is a pair (s, τ) where s is an $(l-1)$-bit string and τ is an L-bit string. Now, we know that the tagging algorithm itself generates the first component in a very specific way. For concreteness, take the counter-based XOR scheme; here s is the value of a counter, and thus for legitimately tagged messages, a value that never repeats from one message to the next. (We assume no more than 2^{l-1} messages are authenticated so that the counter does not wrap around.) This does *not* mean that the adversary is forced to use a counter value in the role of s in its attempted forgery $M, (s, \tau)$. The adversary is free to try any value of s, and in particular to re-use an already used counter value. Remember that the adversary's goal is to get the verification algorithm to accept the pair $M, (s, \tau)$, subject only to the constraint that M was not a query to the tagging oracle. Look at the code of the verification algorithm: it does not in any way reflect knowledge of s as a counter, or try to check any counter-related property of s. In fact the verification algorithm does not maintain a counter at all; it is stateless. So there is nothing to constrain an adversary to use the value of the counter in its attempted forgery.

A similar situation holds with respect to the randomized version of the XOR scheme. Although the legitimate party chooses r at random, so that legitimate tags have random values of r as the first component of their tags, the adversary can attempt a forgery $M, (r, \tau)$ in which r is quite non-random; the adversary gets to choose r and can set it to whatever it wants. This freedom on the part of the adversary must be remembered in analyzing the schemes.

To get some intuition about the security of these schemes, it is helpful to return to the attack that we used to break the example scheme $\mathcal{MA}_2$ in the previous section, and see that it fails here. We will look at the attack in the context of the counter-based XOR scheme. Remember that the attack, specified by adversary A_2 above, requested the tags of three related messages and XORed the returned values to get the tag of the third message, exploiting commonality between the values to get cancellations. For the same three messages under the new scheme, let us look at the subtags returned by the tagging oracle. They are:

$$\begin{aligned}
\text{XOR-Tag}^{F_K}(\langle 0\rangle, x_1x_2) &= F_K(0\|\langle 0\rangle) \oplus F_K(1\|\langle 1\rangle\|x_1) \oplus F_K(1\|\langle 2\rangle\|x_2) \\
\text{XOR-Tag}^{F_K}(\langle 1\rangle, x_1x_2') &= F_K(0\|\langle 1\rangle) \oplus F_K(1\|\langle 1\rangle\|x_1) \oplus F_K(1\|\langle 2\rangle\|x_2') \\
\text{XOR-Tag}^{F_K}(\langle 2\rangle, x_1'x_2) &= F_K(0\|\langle 2\rangle) \oplus F_K(1\|\langle 1\rangle\|x_1') \oplus F_K(1\|\langle 2\rangle\|x_2)\ .
\end{aligned}$$

Summing these three values yields a mess, something that does not look like the subtag of any message, because the values corresponding to the counter don't cancel. So this attack does not work.

Is there another attack? It seems hard to come up with one, but that does not mean much; maybe the attack is quite clever. This is the point where the kind of approach we have been developing, namely provable security, can be instrumental. We will see that the XOR schemes can be proven secure under the assumption that the family F is a PRF. This means that simple attacks like the above do not exist. And our confidence in this stems from much more than an inability to find the attacks; it stems from our confidence that the underlying family F is itself secure.

8.4.3 Results on the security of the XOR schemes

We state the theorems that summarize the security of the schemes, beginning with the counter-based scheme. We call the (nteger) parameter m in the scheme the block-indexing parameter in the following. We also let $\mathsf{Plaintexts}(l, m)$ denote the set of all strings M such that the length of M is $n \cdot (l - m - 1)$ for some integer n in the range $1 \leq n \leq 2^m - 1$; this is the message space for the XOR message authentication schemes.

The theorem below has (what should by now be) a familiar format. It upper bounds the insecurity of the counter-based XOR message authentication scheme in terms of the insecurity of the underlying PRF F. In other words, it upper bounds the maximum (over all strategies an adversary might try) of the probability that the adversary can break the XOR scheme (namely, successfully forge a correct tag for an as yet un-authenticated message), and the upper bound is in terms of the (assumed known) maximum ability to break the PRF F that underlies the scheme. It is another example of the kind of "punch line" we strive towards: a guarantee that there is simply no attack against a scheme, no matter how clever, as long as we know that the underlying tool is good. In particular we are assured that attacks like those we have seen above on our example schemes will not work against this scheme.

Also as usual, the bounds are quantitative, so that we can use them to assess the amount of security we will get when using some specific PRF (say a block cipher) in the role of F. The bounds are rather good: we see that the chance of breaking the message authentication scheme is hardly more than that of breaking the PRF.

Theorem 8.4 **[11]** Suppose F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ is a PRF, and let $\mathcal{C}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be the corresponding counter-based XOR message authentication scheme as described above, with block-indexing parameter $m \leq l-2$ and message space $\mathsf{Plaintexts}(l,m)$. Then for any t, q, μ with $q < 2^{l-1}$ we have

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{C}\text{-}\mathcal{XOR}^F}(t,q,\mu) \quad \leq \quad \mathbf{Adv}^{\text{prf}}_F(t',q') + 2^{-L} \ ,$$

where $t' = t + O(\mu)$ and $q' = q + 1 + \mu/(l-m-1)$. ▮

The result for the randomized version of the scheme is similar except for accruing an extra term. This time, there is a "collision probability" type term of $q^2/2^l$ in the upper bound, indicating that we are unable to rule out a breaking probability of this magnitude regardless of the quality of the PRF. We will see later that this is an inherent feature of the scheme, which is subject to a sort of birthday attack.

Theorem 8.5 **[11]** Suppose F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ is a PRF, and let $\mathcal{R}\text{-}\mathcal{XOR}^F = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ be the corresponding randomized XOR message authentication scheme as described above, with block-indexing parameter $m \leq l-2$ and message space $\mathsf{Plaintexts}(l,m)$. Then for any t, q, μ

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{R}\text{-}\mathcal{XOR}^F}(t,q,\mu) \quad \leq \quad \mathbf{Adv}^{\text{prf}}_F(t',q') + \frac{q^2}{2^l} + 2^{-L} \ ,$$

where $t' = t + O(\mu)$ and $q' = q + 1 + \mu/(l-m-1)$. ▮

We will not prove these results at this time. We will be able to do this later when we have developed some more technology.

8.5 Pseudorandom functions make good MACs

A general method for designing MACs is to make use of the fact that any pseudorandom function is in fact a MAC. The reduction is due to [92, 93] and the concrete security analysis that follows is from [12]. It shows that the reduction is almost tight—security hardly degrades at all. This relation means that to prove the security of the CBC MAC as a MAC it is enough to show that the CBC transform preserves pseudorandomness. For simplicity the domain of the MAC is restricted to strings of length exactly d for some integer d.

Theorem 8.6 Let MAC: $\mathsf{Keys}MAC \times \{0,1\}^d \to \{0,1\}^s$ be a family of functions, and let $q, t \geq 1$ be integers. Then

$$\mathbf{Adv}^{\text{uf-cma}}_{MAC}(t,q,dq) \quad \leq \quad \mathbf{Adv}^{\text{prf}}_{MAC}(t',q) + \frac{1}{2^s} \tag{8.1}$$

where $t' = t + O(s+d)$. ▮

The constant hidden in the O-notation depends only on details of the model of computation. It is a small constant; one should think of $t' \approx t$.

Proof: Let A by any forger attacking the message authentication code MAC. Assume the oracle in Experiment Forge(MAC, A) is invoked at most q times, and the "running time" of A is at most t, these quantities being measured as discussed in Definition 8.3. We design a distinguisher B_A for MAC versus $\mathsf{Rand}^{d \to s}$ such that

$$\mathbf{Adv}^{\text{prf}}_{MAC}(B_A) \quad \geq \quad \mathbf{Adv}^{\text{uf-cma}}_{MAC,A} - \frac{1}{2^s} \ . \tag{8.2}$$

Moreover B will run in time t' and make at most q queries to its oracle, with the time measured as discussed in Definition 5.4. This implies Equation (8.1) because

$$\begin{aligned}
\mathbf{Adv}^{\text{uf-cma}}_{MAC}(t,q,dq) &= \max_A \left\{ \mathbf{Adv}^{\text{uf-cma}}_{MAC,A} \right\} \\
&\leq \max_A \left\{ \mathbf{Adv}^{\text{prf}}_{MAC}(B_A) + 2^{-s} \right\} \\
&= \max_A \left\{ \mathbf{Adv}^{\text{prf}}_{MAC}(B_A) \right\} + 2^{-s} \\
&\leq \max_B \left\{ \mathbf{Adv}^{\text{prf}}_{MAC}(B) \right\} + 2^{-s} \\
&= \mathbf{Adv}^{\text{prf}}_{MAC}(t',q) + 2^{-s} \,.
\end{aligned}$$

Above the first equality is by the definition of the insecurity function in Definition 8.3. The following inequality uses Equation (8.2). Next we simplify using properties of the maximum, and conclude by using the definition of the insecurity function as per Definition 5.4.

So it remains to design B_A such that Equation (8.2) is true. Remember that B_A is given an oracle for a function $f\colon \{0,1\}^d \to \{0,1\}^s$. It will run A, providing it an environment in which A's oracle queries are answered by B_A. When A finally outputs its forgery, B_A checks whether it is correct, and if so bets that f must have been an instance of the family MAC rather than a random function.

By assumption the oracle in Experiment Forge(MAC, A) is invoked at most q times, and for simplicity we assume it is exactly q. This means that the number of queries made by A to its oracle is $q-1$. Here now is the code implementing B_A.

Distinguisher B_A^f
    For $i = 1,\ldots,q-1$ do
        When A asks its oracle some query, M_i, answer with $f(M_i)$
    End For
    A outputs (M,σ)
    $\sigma' \leftarrow f(M)$
    If $\sigma = \sigma'$ and $M \notin \{M_1,\ldots,M_{q-1}\}$
        then return 1 else return 0

Here B_A initializes A with some random sequence of coins and starts running it. When A makes its first oracle query M_1, algorithm B_A pauses and computes $f(M_1)$ using its own oracle f. The value $f(M_1)$ is returned to A and the execution of the latter continues in this way until all its oracle queries are answered. Now A will output its forgery (M,σ). B_A verifies the forgery, and if it is correct, returns 1.

We now proceed to the analysis. We claim that

$$\mathbf{P}\left[B_A^f = 1 \;:\; f \stackrel{R}{\leftarrow} MAC \right] \;=\; \mathbf{Adv}^{\text{uf-cma}}_{MAC,A} \tag{8.3}$$

$$\mathbf{P}\left[B_A^f = 1 \;:\; f \stackrel{R}{\leftarrow} \mathsf{Rand}^{d\to s} \right] \;\leq\; \frac{1}{2^s}\,. \tag{8.4}$$

Subtracting, we get Equation (8.2), and from the code it is evident that B_A makes q oracle queries. Taking into account our conventions about the running times referring to that of the entire experiment it is also true that the running time of B_A is $t + O(d+s)$. So it remains to justify the two equations above.

In the first case f is an instance of MAC, so that the simulated environment that B_A is providing for A is exactly that of experiment Forge(MAC, A). Since B_A returns 1 exactly when A makes a successful forgery, we have Equation (8.3).

In the second case, A is running in an environment that is alien to it, namely one where a random function is being used to compute MACs. We have no idea what A will do in this environment, but no matter what, we know that the probability that $\sigma = f(M)$ is 2^{-s}, because f is a random function, as long as A did not query M of its oracle. Equation (8.4) follows. ∎

8.6 The CBC MAC

The most popular message authentication code in use is the CBC (Cipher Block Chaining) MAC. Let $f\colon \{0,1\}^l \to \{0,1\}^l$ be a function. Let $f^{(n)}\colon \{0,1\}^{nl} \to \{0,1\}^l$ be the function which on input $x_1 \ldots x_n$ outputs y_n where $y_i = f(y_{i-1} \oplus x_i)$ and $y_0 = 0^l$. If F is a finite family of functions with input length l and output length l then let $F^{(n)}$ denote the family of functions in which the function indexed by key K is $F_K^{(n)}$. The new family has input length nl and output length l and is called the CBC of F.

When F is DES, we have the CBC construction used in practice, called the DES CBC MAC. This construction is both a US and an International Standard, extensively used in the banking and financial sectors. Its security is worth investigation.

8.6.1 Security of the CBC MAC

We discussed the CBC construction in Example 8.2 and Section 5.11.1, and noted in Theorem 5.20 that if F is a PRF family then so is $F^{(n)}$. From Theorem 8.6 we can now conclude that the CBC construction makes a good MAC as long as the underlying functions are pseudorandom.

Theorem 8.7 **[12]** Let $l, m \geq 1$ and $q, t \geq 1$ be integers such that $qm \leq 2^{(l+1)/2}$. Let $F\colon \mathsf{Keys}F \times \{0,1\}^l \to \{0,1\}^l$ be a family of functions. Then

$$\begin{aligned} \mathbf{Adv}^{\text{uf-cma}}_{F^{(m)}}(t, q, mql) &\leq \mathbf{Adv}^{\text{prf}}_{F}(t', q') + \frac{3q^2m^2 + 2}{2^{l+1}} \\ &\leq \mathbf{Adv}^{\text{prp-cpa}}_{F}(t', q') + \frac{2q^2m^2 + 1}{2^{l}} \end{aligned}$$

where $q' = mq$ and $t' = t + O(mql)$.

∎

In particular, if $F =$ DES we have an assessment of the strength of the DES CBC MAC in terms of the strength of DES as a PRF. Unfortunately as we discussed in Section 5.6.2, DES is not too strong as a PRF. We would be better of with stronger block ciphers.

8.6.2 Birthday attack on the CBC MAC

The basic idea behind the attack, due to Preneel and Van Oorschott [163] and (independently) to Krawczyk, is that internal collisions can be exploited for forgery. The attacks presented in [163] are analyzed assuming the underlying functions are random, meaning the family to which the CBC-MAC transform is applied is $\mathsf{Rand}^{l \to l}$ or Perm^l. Here we do not make such an assumption. This attack is from [12] and works for any family of permutations. The randomness in the attack (which is the source of birthday collisions) comes from coin tosses of the forger only. This makes the attack more general. (We focus on the case of permutations because in practice the CBC-MAC is usually based on a block cipher.)

Proposition 8.8 Let l, m, q be integers such that $1 \leq q \leq 2^{(l+1)/2}$ and $m \geq 2$. Let $F\colon \mathsf{Keys}F \times \{0,1\}^l \to \{0,1\}^l$ be a family of permutations. Then there is a forger A making $q+1$ oracle queries, running for time

$O(lmq\ \log q)$ and achieving

$$\mathbf{Adv}^{\text{uf-cma}}_{F^{(m)},A} \quad \geq \quad 0.3 \cdot \frac{q(q-1)}{2^l} \ .$$

As a consequence for $q \geq 2$

$$\mathbf{Adv}^{\text{uf-cma}}_{F^{(m)}}(t, q, qml) \quad \geq \quad 0.3 \cdot \frac{(q-1)(q-2)}{2^l} \ .$$

▮

The time assessment here puts the cost of an oracle call at one unit.

Comparing the above to Theorem 8.7 we see that the upper bound is tight to within a factor of the square of the number of message blocks.

We now proceed to the proof. We begin with a couple of lemmas. The first lemma considers a slight variant of the usual birthday problem and shows that the "collision probability" is still the same as that of the usual birthday problem.

Lemma 8.9 Let l, q be integers such that $1 \leq q \leq 2^{(l+1)/2}$. Fix $b_1, \ldots, b_q \in \{0,1\}^l$. Then

$$\mathbf{P}\left[\, \exists\, i, j \text{ such that } i \neq j \text{ and } b_i \oplus r_i = b_j \oplus r_j \ : \ r_1, \ldots, r_q \stackrel{R}{\leftarrow} \{0,1\}^l \,\right] \quad \geq \quad 0.3 \cdot \frac{q(q-1)}{2^l} \ .$$

▮

Proof: This is just like throwing q balls into $N = 2^l$ bins and lower bounding the collision probability, except that things are "shifted" a bit: the bin assigned to the i-th ball is $r_i \oplus b_i$ rather than r_i as we would usually imagine. But with b_i fixed, if r_i is uniformly distributed, so is $r_i \oplus b_i$. So the probabilities are the same as in the standard birthday problem of Appendix A.1. ▮

The first part of the following lemma states an obvious property of the CBC-MAC transform. The item of real interest is the second part of the lemma, which says that in the case where the underlying function is a permutation, the CBC-MAC transform has the property that output collisions occur if and only if input collisions occur. This is crucial to the attack we will present later.

Lemma 8.10 Let $l, m \geq 2$ be integers and $f\colon \{0,1\}^l \to \{0,1\}^l$ a function. Suppose $\alpha_1 \cdots \alpha_m$ and $\beta_1 \cdots \beta_m$ in $\{0,1\}^{ml}$ are such that $\alpha_k = \beta_k$ for $k = 3, \ldots, m$. Then

$$f(\alpha_1) \oplus \alpha_2 = f(\beta_1) \oplus \beta_2 \qquad \Rightarrow \qquad f^{(m)}(\alpha_1 \cdots \alpha_m) = f^{(m)}(\beta_1 \cdots \beta_m) \ .$$

If f is a permutation then, in addition, the converse is true:

$$f^{(m)}(\alpha_1 \cdots \alpha_m) = f^{(m)}(\beta_1 \cdots \beta_m) \qquad \Rightarrow \qquad f(\alpha_1) \oplus \alpha_2 = f(\beta_1) \oplus \beta_2 \ .$$

▮

Proof: The first part follows from the definition of $f^{(m)}$. For the second part let f^{-1} denote the inverse of the permutation f. The CBC-MAC computation is easily unraveled using f^{-1}. Thus the procedure

$$y_m \leftarrow f^{(m)}(\alpha_1 \cdots \alpha_m) \ ; \quad \text{For } k = m \text{ downto } 3 \text{ do } y_{k-1} \leftarrow f^{-1}(y_k) \oplus \alpha_k \text{ End For } \ ; \quad \text{Return } f^{-1}(y_2)$$

returns $f(\alpha_1) \oplus \alpha_2$, while the procedure

$$y_m \leftarrow f^{(m)}(\beta_1 \cdots \beta_m) \ ; \quad \text{For } k = m \text{ downto } 3 \text{ do } y_{k-1} \leftarrow f^{-1}(y_k) \oplus \beta_k \text{ End For } \ ; \quad \text{Return } f^{-1}(y_2)$$

returns $f(\beta_1)\oplus\beta_2$. But the procedures have the same value of y_m by assumption and we know that $\alpha_k = \beta_k$ for $k = 3, \ldots, m$, so the procedures return the same thing. ∎

Proof of Proposition 8.8: Before presenting the forger let us discuss the idea.

The forger A has an oracle $g = f^{(m)}$ where f is an instance of F. The strategy of the forger is to make q queries all of which agree in the last $m-2$ blocks. The first blocks of these queries are all distinct but fixed. The second blocks, however, are random and independent across the queries. Denoting the first block of query n by a_n and the second block as r_n, the forger hopes to have $i \neq j$ such that $f(a_i)\oplus r_i = f(a_j)\oplus r_j$. The probability of this happening is lower bounded by Lemma 8.9, but simply knowing the event happens with some probability is not enough; the forger needs to detect its happening. Lemma 8.10 enables us to say that this internal collision happens iff the output MAC values for these queries are equal. (This is true because f is a permutation.) We then observe that if the second blocks of the two colliding queries are modified by the xor to both of some value a, the resulting queries still collide. The forger can thus forge by modifying the second blocks in this way, obtaining the MAC of one of the modified queries using the second, and outputting it as the MAC of the second modified query.

The forger is presented in detail below. It makes use of a subroutine *Find* that given a sequence $\sigma_1, \ldots, \sigma_q$ of values returns a pair (i, j) such that $\sigma_i = \sigma_j$ if such a pair exists, and otherwise returns $(0, 0)$.

Forger A^g
  Let $a_1, \ldots, a_q$ be distinct l-bit strings
  For $i = 1, \ldots, q$ do $r_i \stackrel{R}{\leftarrow} \{0,1\}^l$
  For $i = 1, \ldots, q$ do
    $x_{i,1} \leftarrow a_i$; $x_{i,2} \leftarrow r_i$
    For $k = 3, \ldots, m$ do $x_{i,k} \leftarrow 0^l$
    $X_i \leftarrow x_{i,1} \ldots x_{i,m}$
    $\sigma_i \leftarrow g(X_i)$
  End For
  $(i, j) \leftarrow \mathit{Find}(\sigma_1, \ldots, \sigma_q)$
  If $(i, j) = (0, 0)$ then abort
  Else
    Let a be any l-bit string different from 0^l
    $x'_{i,2} \leftarrow x_{i,2}\oplus a$; $x'_{j,2} \leftarrow x_{j,2}\oplus a$
    $X'_i \leftarrow x_{i,1}x'_{i,2}x_{i,3}\cdots x_{i,m}$; $X'_j \leftarrow x_{j,1}x'_{j,2}x_{j,3}\cdots x_{j,m}$
    $\sigma'_i \leftarrow g(X'_i)$
    Return (X'_j, σ'_i)
  End If

To estimate the probability of success, suppose $g = f^{(m)}$ where f is an instance of F. Let (i, j) be the pair of values returned by the *Find* subroutine. Assume $(i, j) \neq (0, 0)$. Then we know that

$$f^{(m)}(x_{i,1}\cdots x_{i,m}) = f^{(m)}(x_{j,1}\cdots x_{j,m}) \ .$$

By assumption f is a permutation and by design $x_{i,k} = x_{j,k}$ for $k = 3, \ldots, m$. The second part of Lemma 8.10 then implies that $f(a_i)\oplus r_i = f(a_j)\oplus r_j$. Adding a to both sides we get $f(a_i)\oplus(r_i\oplus a) = f(a_j)\oplus(r_j\oplus a)$. In other words, $f(a_i)\oplus x'_{i,2} = f(a_j)\oplus x'_{j,2}$. The first part of Lemma 8.10 then implies that $f^{(m)}(X'_i) = f^{(m)}(X'_j)$. Thus σ'_i is a correct MAC of X'_j. Furthermore we claim that X'_j is new, meaning was not queried of the g oracle. Since $a_1, \ldots, a_q$ are distinct, the only thing we have to worry about is that $X'_j = X_j$, but this is ruled out because $a \neq 0^l$.

We have just argued that if the *Find* subroutine returns $(i, j) \neq (0, 0)$ then the forger is successful, so the success probability is the probability that $(i, j) \neq (0, 0)$. This happens whenever here is a collision amongst the q values $\sigma_1, \ldots, \sigma_q$. Lemma 8.10 tells us however that there is a collision in these values if and only if there is a collision amongst the q values $f(a_1) \oplus r_1, \ldots, f(a_q) \oplus r_q$. The probability is over the random choices of $r_1, \ldots, r_q$. By Lemma 8.9 the probability of the latter is lower bounded by the quantity claimed in the Proposition. We conclude the theorem by noting that, with a simple implementation of FindCol (say using a balanced binary search tree scheme) the running time is as claimed. ▮

8.6.3 Length Variability

For simplicity, let us assume throughout this section that strings to be authenticated have length which is a multiple of l bits. This restriction is easy to dispense with by using simple and well-known padding methods: for example, always append a "1" and then append the minimal number of 0's to make the string a multiple of l bits.

The CBC MAC does not directly give a method to authenticate messages of variable input lengths. In fact, it is easy to "break" the CBC MAC construction if the length of strings is allowed to vary. You are asked to do this in a problem at the end of this chapter. Try it; it is a good exercise in MACs!

One possible attempt to authenticate messages of varying lengths is to append to each string $x = x_1 \cdots x_m$ the number m, properly encoded as the final l-bit block, and then CBC MAC the resulting string $m+1$ blocks. (Of course this imposes a restriction that $m < 2^l$, not likely to be a serious concern.) We define $f_a^*(x_1 \cdots x_m) = f_a^{(m+1)}(x_1 \cdots x_m \ m)$.

We show that f^* is not a secure MAC. Take arbitrary l-bit words b, b' and c, where $b \neq b'$. It is easy to check that given

(1) $t_b = f^*(b)$,

(2) $t_{b'} = f^*(b')$, and

(3) $t_{b1c} = f^*(b \| 1 \| c)$

the adversary has in hand $f^*(b' \| 1 \| t_b \oplus t_{b'} \oplus c)$ —the authentication tag of a string she has not asked about before—since this is precisely t_{b1c}.

Despite the failure of the above method there are many suitable ways to obtain a PRF that is good on variable input lengths. We mention three. In each, let F be a finite function family from and to l-bit strings. Let $x = x_1 \cdots x_m$ be the message to which we will apply f_a:

(1) *Input-length key separation.* Set $f_a^*(x) = f_{a_m}^{(m)}(x)$, where $a_m = f_a(m)$.

(2) *Length-prepending.* Set $f_a^*(x) = f_a^{(m+1)}(m \| x)$.

(3) *Encrypt last block.* Set $f_{a_1 a_2}^*(x) = f_{a_2}(f_{a_1}^{(m)}(x))$.

The first two methods are from [12]. The last method appears in an informational Annex of [112], and has now been analyzed by Petrank and Rackoff [156]. It is the most attractive method of the bunch, since the length of x is not needed until the end of the computation, facilitating on-line MAC computation.

8.7 Universal hash based MACs

Today the most effective paradigm for fast message authentication is based on the use of "almost universal hash functions". The design of these hash functions receives much attention and has resulted in some very fast ones [32], so that universal hash based MACs are the fastest MACs around. Let us begin by describing the tool, and then seeing how it can be used for message authentication.

8.7.1 Almost universal hash functions

Let $H\colon \mathsf{Keys}(H) \times \mathsf{Dom}(H) \to \{0,1\}^L$ be a family of functions. We think of them as hash functions because the domain $\mathsf{Dom}(H)$ of any individual function H_K is typically large, being the message space of the desired message authentication scheme.

Definition 8.11 Let $H\colon \mathsf{Keys}(H) \times \mathsf{Dom}(H) \to \{0,1\}^L$ be a family of functions. Let

$$\mathbf{Adv}^{\mathrm{uh}} H \quad = \quad \max_{a_1,a_2} \left\{ \mathbf{P}\left[H_K(a_1) = H_K(a_2) \;:\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \right] \right\} ,$$

the maximum being over all *distinct* points $a_1, a_2 \in \mathsf{Dom}(H)$. ▮

The smaller the value of $\mathbf{Adv}^{\mathrm{uh}} H$, the better the quality of H as an almost universal function. We say that H is a *universal hash function* if $\mathbf{Adv}^{\mathrm{uh}} H = 2^{-L}$. (We will see later that this is the lowest possible value of the insecurity.)

A stronger property is almost xor-universality.

Definition 8.12 Let $H\colon \mathsf{Keys}(H) \times \mathsf{Dom}(H) \to \{0,1\}^L$ be a family of functions. Let

$$\mathbf{Adv}^{\mathrm{xuh}} H \quad = \quad \max_{a_1,a_2,b} \left\{ \mathbf{P}\left[H_K(a_1) \oplus H_K(a_2) = b \;:\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \right] \right\} ,$$

the maximum being over all *distinct* points $a_1, a_2 \in \mathsf{Dom}(H)$ and all strings $b \in \{0,1\}^L$. ▮

The smaller the value of $\mathbf{Adv}^{\mathrm{xuh}} H$, the better the quality of H as an almost xor-universal function. We say that H is a *xor-universal hash function* if $\mathbf{Adv}^{\mathrm{xuh}} H = 2^{-L}$. (We will see later that this is the lowest possible value of the insecurity.)

Almost xor-universality is a stronger requierement than almost universality.

Proposition 8.13 Let $H\colon \mathsf{Keys}(H) \times \mathsf{Dom}(H) \to \{0,1\}^L$ be a family of functions. Then

$$\mathbf{Adv}^{\mathrm{uh}} H \le \mathbf{Adv}^{\mathrm{xuh}} H .$$

▮

Proof: Setting $b = 0^L$ in Definition 8.12 yields the quantity of Definition 8.11. ▮

The simplest example is the family of all functions.

Proposition 8.14 The family $\mathsf{Rand}^{l \to L}$ of all functions of l-bits to L-bits is universal and xor-universal, meaning

$$\mathbf{Adv}^{\mathrm{uh}} \mathsf{Rand}^{l \to L} = \mathbf{Adv}^{\mathrm{xuh}} \mathsf{Rand}^{l \to L} = 2^{-L} .$$

▮

Proof: By Proposition 8.13 we need only show that $\mathbf{Adv}^{\mathrm{xuh}} \mathsf{Rand}^{l \to L} = 2^{-L}$. With distinct $a_1, a_2 \in \{0,1\}^l$, and $b \in \{0,1\}^L$ fixed, we clearly have

$$\mathbf{P}\left[h(a_1) \oplus h(a_2) = b \;:\; h \stackrel{R}{\leftarrow} \mathsf{Rand}^{l \to L} \right] \quad = \quad 2^{-L}$$

because h is a random function. ▮

Another source of examples is polynomials over finite fields.

Example 8.15 Identify $\{0,1\}^l$ with $\mathrm{GF}(2^l)$, the finite field of 2^l elements. We fix an irreducible, degree l polynomial over GF(2) so as to be able to do arithmatic over the field. The hash function H we define takes as key a pair α, β of points in $\{0,1\}^l$ such that $\alpha \neq 0$. The domain is $\{0,1\}^l$ and the range is $\{0,1\}^L$ where $L \leq l$. We define the function by

$$H_{\alpha,\beta}(x) \quad = \quad [\alpha x + \beta]_{1\ldots L} \; .$$

That is, with key α, β and input $x \in \{0,1\}^l$, first compute, in the finite field, the value $\alpha x + \beta$. View this as an l-bit string, and output the first L bits of it. ▌

Proposition 8.16 The family $H\colon \mathsf{Keys}(H) \times \{0,1\}^l \to \{0,1\}^L$ defined above, where $L \leq l$ and $\mathsf{Keys}(H)$ is the set of all pairs (a,b) of l-bit strings such that $a \neq 0$, is a xor-universal hash function. ▌

Proof: We need to show that $\mathbf{Adv}H = 2^{-L}$. Accordingly fix $a_1, a_2 \in \{0,1\}^l$ such that $a_1 \neq a_2$, and fix $b \in \{0,1\}^L$. Fix any key for the function, meaning any $\alpha \neq 0$ and any β. Notice that $y = \alpha x + \beta$ iff $x = \alpha^{-1}(y - \beta)$. (The arithmatic here is over the finite field, and we are using the assumption that $\alpha \neq 0$.) This means that the map of $\mathrm{GF}(2^l)$ to $\mathrm{GF}(2^l)$ given by $x \mapsto \alpha x + \beta$ is a permutation. The proposition follows from this. ▌

For the following it is useful to have some terminology. Fix any two points a_1, a_2 in the domain $\mathsf{Dom}(H)$ of the family, the only restriction on them being that they are not allowed to be equal. Also fix a point b in the range $\{0,1\}^L$ of the family. With H fixed, we can associate to these three points a probability

$$\begin{aligned} \mathrm{UHColPr}_H(a_1, a_2, b) &= \mathbf{P}\left[\, H_K(a_1) \oplus H_K(a_2) = b \;:\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \,\right] \\ &= \mathbf{P}\left[\, h(a_1) \oplus h(a_2) = b \;:\; h \stackrel{R}{\leftarrow} H \,\right] \, , \end{aligned}$$

the two expressions above being equal by definition.

It is useful to interpret the almost xor-universal measure in another, more dynamic way. Imagine that the choice of the points a_1, a_2, b is made by an adversary. This adversary C knows that H is the target family. It clunks along for a while and then outputs some distinct values $a_1, a_2 \in \mathsf{Dom}(H)$, and a value $b \in \{0,1\}^L$. Now a key K is chosen at random, defining the function $H_K\colon \mathsf{Dom}(H) \to \{0,1\}^L$, and we test whether or not $H_K(a_1) \oplus H_K(a_2) = b$. If so, the adversary C wins. We denote the probability that the adversary wins by $\mathbf{Adv}^{\mathrm{xuh}}H, C$. We then claim that this probability is at most $\mathbf{Adv}^{\mathrm{xuh}}H$.

The reason is that there is a single best strategy for the adversary, namely to choose points a_1, a_2, b which maximize the probability $\mathrm{UHColPr}_H(a_1, a_2, b)$ defined above. This should be relatively clear, at least for the case when the adversary is deterministic. But the claim is true even when the adversary is probabilistic, meaning that the triple of points it outputs can be different depending on its own coin tosses. (In such a case, the probability defining $\mathbf{Adv}^{\mathrm{xuh}}H, C$ is taken over the choice of K and also the coin tosses of C.) We justify this claim in Proposition 8.17 below. We thus have two, equivalent ways of thinking about $\mathbf{Adv}^{\mathrm{xuh}}H$, one more "static" and the other more "dynamic". Depending on the setting, we may benefit more from one view than another.

Before stating and proving Proposition 8.17, however, let us emphasize some features of this notion. A key feature of the game is that the steps must follow a particular order: *first* the adversary chooses points a_1, a_2, b, *then* K is chosen at random and the function H_K is defined. The adversary is *not* allowed to choose a_1, a_2, b as a function of K; it must first commit to them, and then there is some probability of its winning the game.

This notion differs from others we have considered in that there is no computational restriction on the adversary. Namely, it can run for as long as it likes in deciding how to choose a_1, a_2, b, and the security condition is true nonetheless. Thus, it is a purely information theoretic notion.

Here now is the promised bound.

Proposition 8.17 Let $H\mathsf{Keys}(H) \times \mathsf{Dom}(H) \to \{0,1\}^L$ be a family of functions and C a (possibly probabilistic) algorithm that outputs a triple a_1, a_2, b such that a_1, a_2 are distinct points in $\mathsf{Dom}(H)$ and $b \in \{0,1\}^L$. Then

$$\mathbf{Adv}^{\mathrm{xuh}} H, C \quad \leq \quad \mathbf{Adv}^{\mathrm{xuh}} H \; .$$

▮

Proof: Remember that to say C is probabilistic means that it has as an auxiliary input a sequence ρ of random bits of some length r, and uses them in its computation. Depending on the value of r, the output triple of C will change. We can denote by $a_1(\rho), a_2(\rho), b(\rho)$ the triple that C outputs when its coins are ρ. For any particular value of ρ it is clear from Definition 8.11 that

$$\begin{aligned}
& \mathbf{P}\left[\, H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho) \;:\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \,\right] \\
\leq \quad & \max_{a_1, a_2, b} \{\, \mathbf{P}\left[\, H_K(a_1) \oplus H_K(a_2) = b \;:\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \,\right] \} \\
= \quad & \mathbf{Adv}^{\mathrm{xuh}} H \; .
\end{aligned}$$

Using this we get

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{xuh}} H, C \quad &= \quad \mathbf{P}\left[\, H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho) \;:\; \rho \stackrel{R}{\leftarrow} \{0,1\}^r \;;\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \,\right] \\
&= \quad \sum_{\rho \in \{0,1\}^r} \mathbf{P}\left[\, H_K(a_1(\rho)) \oplus H_K(a_2(\rho)) = b(\rho) \;:\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \,\right] \cdot 2^{-r} \\
&\leq \quad \sum_{\rho \in \{0,1\}^r} \mathbf{Adv}^{\mathrm{xuh}} H \cdot 2^{-r} \\
&= \quad \mathbf{Adv}^{\mathrm{xuh}} H \; .
\end{aligned}$$

The first equality is by definition of $\mathbf{Adv}^{\mathrm{xuh}} H, C$. In the second line we used the fact that the coins of C are chosen at random from the set of all strings of length r. In the third line, we used the above observation. ▮

How low can $\mathbf{Adv}^{\mathrm{xuh}} H$ go? We claim that the lowest possible value is 2^{-L}, the value achieved by a xor-universal family. The following justifies this claim.

Proposition 8.18 Let $H\mathsf{Keys}(H) \times \mathsf{Dom}(H) \to \{0,1\}^L$ be a family of functions. Then

$$\mathbf{Adv}^{\mathrm{xuh}} H \quad \geq \quad 2^{-L} \; .$$

▮

Proof: Fix two distinct points $a_1, a_2 \in \mathsf{Dom}(H)$, and for any fixed key $K \in \mathsf{Keys}(H)$ let

$$c(K) \quad = \quad \mathbf{P}\left[\, H_K(a_1) \oplus H_K(a_2) = b \;:\; b \stackrel{R}{\leftarrow} \{0,1\}^L \,\right] \; .$$

Then $c(K) = 2^{-L}$. Why? With K, a_1, a_2 all fixed, $H_K(a_1) \oplus H_K(a_2)$ is some fixed value, call it b'. The above is then just asking what is the probability that $b = b'$ if we pick b at random, and this of course is 2^{-L}.

Now consider the adversary C that picks b at random from $\{0,1\}^L$ and outputs the triple a_1, a_2, b. (Note this adversary is probabilistic, because of its random choice of b.) Then

$$\begin{aligned}
\mathbf{Adv}^{\mathrm{xuh}} H, C &= \mathbf{P}\left[\, H_K(a_1) \oplus H_K(a_2) = b \;:\; b \stackrel{R}{\leftarrow} \{0,1\}^L \;;\; K \stackrel{R}{\leftarrow} \mathsf{Keys}(H) \,\right] \\
&= \sum_{K \in \mathsf{Keys}(H)} c(K) \cdot \mathbf{P}\left[\, K' = K \;:\; K' \leftarrow \mathsf{Keys}(H) \,\right] \\
&= \sum_{K \in \mathsf{Keys}(H)} 2^{-L} \cdot \mathbf{P}\left[\, K' = K \;:\; K' \leftarrow \mathsf{Keys}(H) \,\right] \\
&= 2^{-L} \cdot 1\,.
\end{aligned}$$

Thus we have been able to present an adversary C such that $\mathbf{Adv}^{\mathrm{xuh}} H, C = 2^{-L}$. From Proposition 8.17 it follows that $\mathbf{Adv}^{\mathrm{xuh}} H \geq 2^{-L}$. ∎

8.7.2 MACing using UH functions

Let $H\colon \mathsf{Keys}(H) \times \mathsf{Plaintexts} \to \{0,1\}^L$ be a family of hash functions, and let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a PRF. We associate to them the *universal hash based MAC*. The key will be a pair of strings, K_1, K_2, where the first subkey is for H and the second is for F. (We call them the *hashing* and *masking* keys respectively.) The message is first hashed to a string x using H_{K_1}, and this value is then "encrypted" by applying $F_{K_2}(s)$ to yield a value σ which is the tag. Here now is the full description of the scheme, $\mathcal{UHM}^{H,F} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$–

Algorithm $\mathcal{T}_{K_1,K_2}(M)$	Algorithm $\mathcal{V}_{K_1,K_2}(M,\sigma)$
$x \leftarrow H_{K_1}(M)$ $\sigma \leftarrow F_{K_2}(x)$ Return σ	$x \leftarrow H_{K_1}(M)$ $\sigma' \leftarrow F_{K_2}(x)$ If $\sigma = \sigma'$ then return 1 else return 0

Lemma 8.19 Let $H\colon \mathsf{Keys}(H) \times \mathsf{Plaintexts} \to \{0,1\}^L$ be a family of functions, and A an adversary attacking the message authentication scheme $\mathcal{UHM}^{H,\mathsf{Rand}^{l \to L}}$. Then for any q, μ we have

$$\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{UHM}^{H,\mathsf{Rand}^{l\to L}},A} \;\leq\; \frac{q(q-1)}{2} \cdot \mathbf{Adv}^{\mathrm{uh}} H\,.$$

∎

Theorem 8.20 Let $H\colon \mathsf{Keys}(H) \times \mathsf{Plaintexts} \to \{0,1\}^L$ be a family of functions, and let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a PRF. Then for any t, q, μ we have

$$\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{UHM}^{H,F}}(t,q,\mu) \;\leq\; \frac{q(q-1)}{2} \cdot \mathbf{Adv}^{\mathrm{uh}} H + \mathbf{Adv}^{\mathrm{prf}}_F(t', q+1)$$

where $t' = t + O(\mu)$. ∎

8.7.3 MACing using XUH functions

Let $H\colon \mathsf{Keys}(H) \times \mathsf{Plaintexts} \to \{0,1\}^L$ be a family of hash functions, and let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a PRF. We associate to them the *xor-universal hash based MACs*. There are two such MACs; one stateful (using counters) and deterministic, the other stateless and randomized. The key will be a pair of strings, K_1, K_2, where the first subkey is for H and the second is for F. (We call them the *hashing* and *masking* keys respectively.) In both cases, the basic paradigm is the same. The message is first hashed to a string x

using H_{K_1}, and this value is then "encrypted" by XORing with $F_{K_2}(s)$ to yield a value τ, where s is some point chosen by the sender. The tag contains τ, but also s so as to permit verification. The difference in the two version is in how s is selected. In the counter version it is a counter, and in the randomized version a random number chosen anew with each application of the tagging algorithm.

Here now is the full description of the counter-based version of the scheme, $\mathcal{C}\text{-}\mathcal{UHM}^{H,F} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$–

Algorithm $\mathcal{T}_{K_1,K_2}(M)$	Algorithm $\mathcal{V}_{K_1,K_2}(M,\sigma)$
$x \leftarrow H_{K_1}(M)$	Parse σ as (s,τ)
$\tau \leftarrow F_{K_2}(ctr) \oplus x$	$x' \leftarrow F_{K_2}(s) \oplus \tau$
$\sigma \leftarrow (ctr, \tau)$	$x \leftarrow H_{K_1}(M)$
$ctr \leftarrow ctr + 1$	If $x = x'$ then return 1 else return 0
Return σ	

The randomized version $\mathcal{R}\text{-}\mathcal{UHM}^{H,F} = (\mathcal{K}, \mathcal{T}, \mathcal{V})$ is like this–

Algorithm $\mathcal{T}_{K_1,K_2}(M)$	Algorithm $\mathcal{V}_{K_1,K_2}(M,\sigma)$
$x \leftarrow H_{K_1}(M)$	Parse σ as (s,τ)
$r \stackrel{R}{\leftarrow} \{0,1\}^l$	$x' \leftarrow F_{K_2}(s) \oplus \tau$
$\tau \leftarrow F_{K_2}(r) \oplus x$	$x \leftarrow H_{K_1}(M)$
$\sigma \leftarrow (r, \tau)$	If $x = x'$ then return 1 else return 0
Return σ	

Lemma 8.21 Let $H\colon \mathsf{Keys}(H) \times \mathsf{Plaintexts} \to \{0,1\}^L$ be a family of functions, and A an adversary attacking the message authentication scheme $\mathcal{C}\text{-}\mathcal{UHM}^{H,\mathsf{Rand}^{l\to L}}$. Then for any q, μ with $q < 2^l$ we have

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{C}\text{-}\mathcal{UHM}^{H,\mathsf{Rand}^{l\to L}},A} \;\leq\; \mathbf{Adv}^{\text{xuh}} H \,.$$

∎

Proof of Lemma 8.21: The adversary A makes a sequence $M_1, \ldots, M_q$ of queries to its $\mathcal{T}_{K_1,K_2}(\cdot)$ oracle, and these are answered according to the above scheme. Pictorially:

$$\begin{array}{ccccc} M_1 & \Longrightarrow & \sigma_1 & = & (s_1, \tau_1) \\ M_2 & \Longrightarrow & \sigma_2 & = & (s_2, \tau_2) \\ \vdots & & \vdots & & \vdots \\ M_q & \Longrightarrow & \sigma_q & = & (s_q, \tau_q) \end{array}$$

Here $s_i = \langle i-1 \rangle$ is simply the (binary representation of the) counter value, and $\tau_i = f(s_i) \oplus h(M_i)$, where $h = H_{K_1}$ is the hash function instance in use, and $f = \mathsf{Rand}^{l\to L}_{K_2}$ is the random function specified by the second key. Following this chosen-message attack, A outputs a pair M, σ where $\sigma = (s, \tau)$. We may assume wlog that $M \notin \{M_1, \ldots, M_q\}$. We know that A will be considered successful if $\mathcal{V}_{K_1,K_2}(M,\sigma) = 1$. We wish to upper bound the probability of this event.

Let New be the event that $s \notin \{s_1, \ldots, s_q\}$, and Old the complement event, namely that $s = s_i$ for some value of $i \in \{1, \ldots, q\}$. Let $\mathbf{P}[\cdot]$ denote the probability of event "·" in the experiment ForgeExp($\mathcal{C}\text{-}\mathcal{UHM}^{H,\mathsf{Rand}^{l\to L}}, A$). We consider

$$\begin{aligned} p_1 &= \mathbf{P}\left[\mathcal{V}_{K_1,K_2}(M,\sigma) = 1 \mid \mathsf{Old}\right] \\ p_2 &= \mathbf{P}\left[\mathcal{V}_{K_1,K_2}(M,\sigma) = 1 \mid \mathsf{New}\right] \\ q &= \mathbf{P}\left[\mathsf{New}\right]. \end{aligned}$$

We will use the following two claims.

Claim 1: $p_1 \leq \mathbf{Adv}^{\mathrm{uh}} H$.

Claim 2: $p_2 \leq 2^{-L}$.

We will prove these claims later. Let us first check that they yield the desired result:

$$\begin{aligned}
\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{C}\text{-}\mathcal{UHM}^{H,\mathsf{Rand}^{l\to L}},A} &= \mathbf{P}\left[\mathcal{V}_{K_1,K_2}(M,\sigma)=1\right] \\
&= p_1 q + p_2(1-q) \\
&\leq \mathbf{Adv}^{\mathrm{uh}} H \cdot q + 2^{-L} \cdot (1-q) \\
&\leq \mathbf{Adv}^{\mathrm{uh}} H \cdot q + \mathbf{Adv}^{\mathrm{uh}} H \cdot (1-q) \\
&\leq \mathbf{Adv}^{\mathrm{uh}} H \,.
\end{aligned}$$

The first line is simply by definition of the success probability. The second line is obtained by conditioning. In the third line we used the claims. In the fourth line we used Proposition 8.18.

It remains to prove the claims. We begin with the second.

Proof of Claim 2: Since the queries of the adversary did not result in the function f being evaluted on the point s, the value $f(s)$ is uniformly distributed from the point of view of A. Or, remember the dynamic view of random functions; we can imagine that f gets specified only as it is queried. Since the tagging oracle (as invoked by A) has not applied f at s, we can imagine that the coins to determine $f(s)$ are tossed after the forgery is created. With that view it is clear that

$$p_2 \;=\; \mathbf{P}\left[f(s) \oplus h(M) = \tau\right] \;=\; 2^{-L} \,.$$

Note that here we did not use anything about the hash function; the claim is true due only to the randomness of f. $\square$

Proof of Claim 2:

Adversary C
    Initialize counter ctr to 0
    For $i = 1, \ldots, q$ do
        $A \to M_i$
        $\tau_i \stackrel{R}{\leftarrow} \{0,1\}^L$; $s_i \leftarrow \langle ctr \rangle$; $\sigma_i \leftarrow (s_i, \tau_i)$
        $A \leftarrow \sigma_i$; $ctr \leftarrow ctr + 1$
    $A \to M, \sigma$
    Parse σ as (s, τ)
    If $s \notin \{s_1, \ldots, s_q\}$ then FAIL
    Else let i be such that $s = s_i$
    Let $b \leftarrow \tau_i \oplus \tau$ and return M, M_i, b

We claim that $\mathbf{Adv}^{\mathrm{uh}} H, C = p_1$.

▮

Theorem 8.22 Let H: $\mathsf{Keys}(H) \times \mathsf{Plaintexts} \to \{0,1\}^L$ be a family of functions, and let F: $\{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a PRF. Then for any t, q, μ we have

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{C}\text{-}\mathcal{UHM}^{H,F}}(t,q,\mu) \;\leq\; \mathbf{Adv}^{\mathrm{xuh}} H + \mathbf{Adv}^{\mathrm{prf}}_F(t', q+1)$$

where $t' = t + O(\mu)$. ▮

8.8 MACing with cryptographic hash functions

Recently there has been a surge of interest in MACing using *only* cryptographic hash functions like MD5 or SHA. It is easy to see why. The popular hash functions like MD5 and SHA-1 are faster than block ciphers in software implementation; these software implementations are readily and freely available; and the functions are not subject to the export restriction rules of the USA and other countries.

The more difficult question is how best to do it. These hash functions were not originally designed to be used for message authentication. (One of many difficulties is that hash functions are not keyed primitives, ie. do not accommodate naturally the notion of secret key.) So special care must be taken in using them to this end.

A variety of constructions have been proposed and analyzed. (See Tsudik [195] for an early description of such constructions and Touch [194] for a list of Internet protocols that use this approach. Preneel and van Oorschot [163, 162] survey existing constructions and point out to some of their properties and weaknesses; in particular, they present a detailed description of the effect of birthday attacks on iterated constructions. They also present a heuristic construction, the MDx-MAC, based on these findings. Kaliski and Robshaw [115] discuss and compare various constructions. Performance issues are discussed in [194, 11].) Recently, one construction seems to be gaining acceptance. This is the HMAC construction of [18]. In particular HMAC was recently chosen as the mandatory to implement authentication transform for Internet security protocols and for this purpose is described in an Internet RFC [122].

8.8.1 The HMAC construction

Let H be the hash function. For simplicity of description we may assume H to be MD5 or SHA-1; however the construction and analysis can be applied to other functions as well (see below). H takes inputs of any length and produces l-bit output ($l = 128$ for MD5 and $l = 160$ for SHA-1). Let Text denote the data to which the MAC function is to be applied and let K be the message authentication secret key shared by the two parties. (It should not be larger than 64 bytes, the size of a hashing block, and, if shorter, zeros are appended to bring its length to exactly 64 bytes.) We further define two fixed and different 64 byte strings ipad and opad as follows (the "i" and "o" are mnemonics for inner and outer):

ipad = the byte 0x36 repeated 64 times
opad = the byte 0x5C repeated 64 times.

The function HMAC takes the key K and Text, and produces $\mathrm{HMAC}_K(\mathrm{Text}) =$

$$H(K \oplus \mathsf{opad}, H(K \oplus \mathsf{ipad}, \mathrm{Text})) .$$

Namely,

(1) Append zeros to the end of K to create a 64 byte string

(2) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with ipad

(3) Append the data stream Text to the 64 byte string resulting from step (2)

(4) Apply H to the stream generated in step (3)

(5) XOR (bitwise exclusive-OR) the 64 byte string computed in step (1) with opad

(6) Append the H result from step (4) to the 64 byte string resulting from step (5)

(7) Apply H to the stream generated in step (6) and output the result

The recommended length of the key is at least l bits. A longer key does not add significantly to the security of the function, although it may be advisable if the randomness of the key is considered weak.

HMAC optionally allows truncation of the final output say to 80 bits.

As a result we get a simple and efficient construction. The overall cost for authenticating a stream Text is close to that of hashing that stream, especially as Text gets large. Furthermore, the hashing of the padded keys can be precomputed for even improved efficiency.

Note HMAC uses the hash function H as a black box. No modifications to the code for H are required to implement HMAC. This makes it easy to use library code for H, and also makes it easy to replace a particular hash function, such as MD5, with another, such as SHA-1, should the need to do this arise.

8.8.2 Security of HMAC

The advantage of HMAC is that its security can be justified given some reasonable assumptions about the strength of the underlying hash function.

The assumptions on the security of the hash function should not be too strong, since after all not enough confidence has been gathered in current candidates like MD5 or SHA. (In particular, we now know that MD5 is not collision-resistant [70]. We will discuss the MD5 case later.) In fact, the weaker the assumed security properties of the hash function, the stronger the resultant MAC construction.

We make assumptions that reflect the more standard existing usages of the hash function. The properties we require are mainly a certain kind of *weak* collision-freeness and some limited "unpredictability." What is shown is that if the hash function function has these properties the MAC is secure; the *only* way the MAC could fail is if the hash function fails.

The analysis of [18] applies to hash functions of the iterated type, a class that includes MD5 and SHA, and consists of hash functions built by iterating applications of a compression function CF according to the procedure of Merkle [138] and Damgård [63]. (In this construction a l-bit initial variable IV is fixed, and the output of H on text x is computed by breaking x into 512 bit blocks and hashing in stages using CF, in a simple way that the reader can find described in many places, e.g. [115].) Roughly what [18] say is that an attacker who can forge the HMAC function can, with the same effort (time and collected information), break the underlying hash function in one of the following ways:

(1) The attacker finds collisions in the hash function even when the IV is random and secret, or

(2) The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker. (That is, the attacker is successful in forging with respect to the application of the compression function secretly keyed and viewed as a MAC on fixed length messages.)

The feasibility of any of these attacks would contradict some of our basic assumptions about the cryptographic strength of these hash functions. Success in the first of the above attacks means success in finding collisions, the prevention of which is the main design goal of cryptographic hash functions, and thus can usually be assumed hard to do. But in fact, even more is true: success in the first attack above is even *harder* than finding collisions in the hash function, because collisions when the IV is secret (as is the case here) is far more difficult than finding collisions in the plain (fixed IV) hash function. This is because the former requires interaction with the legitimate user of the function (in order to generate pairs of input/outputs from the function), and disallows the parallelism of traditional birthday attacks. Thus, even if the hash function is not collision-free in the traditional sense, our schemes could be secure.

Some "randomness" of hash functions is assumed in their usage for key generation and as pseudo-random generators. (For example the designers of SHA suggested that SHA be used for this purpose [81].) Randomness of the function is also used as a design methodology towards achieving collision-resistance. The success of the second attack above would imply that these randomness properties of the hash functions are very poor.

It is important to realize that these results are guided by the desire to have simple to state assumptions and a simple analysis. In reality, the construction is even stronger than the analyses indicates, in the sense that even were the hash functions found not to meet the stated assumptions, the schemes might be secure. For example, even the weak collision resistance property is an overkill, because in actuality, in our constructions, the attacker must find collisions in the keyed function without seeing any outputs of this function, which is significantly harder.

The later remark is relevant to the recently discovered collision attacks on MD5 [70]. While these attacks could be adapted to attack the weak collision-resistance property of MD5, they do not seem to lead to a breaking of HMAC even when used with MD5.

8.8.3 Resistance to known attacks

As shown in [163, 19], birthday attacks, that are the basis to finding collisions in cryptographic hash functions, can be applied to attack also keyed MAC schemes based on iterated functions (including also CBC-MAC, and other schemes). These attacks apply to most (or all) of the proposed hash-based constructions of MACs. In particular, they constitute the best known forgery attacks against the HMAC construction. Consideration of these attacks is important since they strongly improve on naive exhaustive search attacks. However, their practical relevance against these functions is negligible given the typical hash lengths like 128 or 160. Indeed, these attacks require the collection of the MAC value (for a given key) on about $2^{l/2}$ messages (where l is the length of the hash output). For values of $l \geq 128$ the attack becomes totally infeasible. In contrast to the birthday attack on key-less hash functions, the new attacks require interaction with the key owner to produce the MAC values on a huge number of messages, and then allow for no parallelization. For example, when using MD5 such an attack would require the authentication of 2^{64} blocks (or 2^{73} bits) of data using the same key. On a 1 Gbit/sec communication link, one would need 250,000 years to process all the data required by such an attack. This is in sharp contrast to birthday attacks on key-less hash functions which allow for far more efficient and close-to-realistic attacks [196].

8.9 Minimizing assumptions for MACs

As with the other primitives of private key cryptography, the existence of secure message authentication schemes is equivalent to the existence of one-way functions. That one-way functions yield message authentication schemes follows from Theorem 5.19 and Theorem 8.6. The other direction is [110]. In summary:

Theorem 8.23 There exists a secure message authentication scheme for message space $\{0,1\}^*$ if and only if there exists a one-way function. ▮

8.10 Problems and exercises

Problem 8.24 Let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^l$ be a PRF. Recall that the CBC MAC based on F is the message authentication scheme $\mathcal{MA}$ whose tagging and verifying algorithms are as follows:

Algorithm $\mathcal{T}_K(x_1 \ldots x_n)$	Algorithm $\mathcal{V}_K(x_1 \ldots x_n, \sigma)$
$y_0 \leftarrow 0^l$	$y_0 \leftarrow 0^l$
For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(y_{i-1} \oplus x_i)$	For $i = 1, \ldots, n$ do $y_i \leftarrow F_K(y_{i-1} \oplus x_i)$
Return y_n	If $y_n = \sigma$ then return 1 else return 0

Let the message space be the set of all strings x whose length is a multiple of l bits. (Meaning the number of message blocks n may vary in the above.) Show that the scheme is insecure over this message space. Namely present an adversary A attacking the scheme using time t, making q oracle queries, these totalling μ bits, and achieving $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{MA},A} = 1$, where t, q, μ are some small values that you will specify. ▮

CHAPTER 9

Digital signatures

The notion of a *digital signature* may prove to be one of the most fundamental and useful inventions of modern cryptography. A signature scheme provides a way for each user to *sign* messages so that the signatures can later be *verified* by anyone else. More specifically, each user can create a matched pair of private and public keys so that only he can create a signature for a message (using his private key), but anyone can verify the signature for the message (using the signer's public key). The verifier can convince himself that the message contents have not been altered since the message was signed. Also, the signer can not later repudiate having signed the message, since no one but the signer possesses his private key.

By analogy with the paper world, where one might sign a letter and seal it in an envelope, one can sign an electronic message using one's private key, and then *seal* the result by encrypting it with the recipient's public key. The recipient can perform the inverse operations of opening the letter and verifying the signature using his private key and the sender's public key, respectively. These applications of public-key technology to electronic mail are quite widespread today already.

If the directory of public keys is accessed over the network, one needs to protect the users from being sent fraudulent messages purporting to be public keys from the directory. An elegant solution is the use of a *certificate* – a copy of a user's public key digitally signed by the public key directory manager or other trusted party. If user A keeps locally a copy of the public key of the directory manager, he can validate all the signed communications from the public-key directory and avoid being tricked into using fraudulent keys. Moreover, each user can transmit the certificate for his public key with any message he signs, thus removing the need for a central directory and allowing one to verify signed messages with no information other than the directory manager's public key. Some of the protocol issues involved in such a network organization, are discussed in the section on key distribution in these lecture notes.

9.1 The Ingredients of Digital Signatures

A *digital signature scheme* within the public key framework, is defined as a triple of algorithms (G, σ, V) such that

- Key generation algorithm G is a probabilistic, polynomial-time algorithm which on input a security parameter 1^k, produces pairs (P, S) where P is called a public key and S a secret key. (We use the notation $(P, S) \in G(1^k)$ indicates that the pair (P, S) is produced by the algorithm G.)
- Signing algorithm σ is a probabilistic polynomial time algorithm which is given a security parameter 1^k, a secret key S in range $G(1^k)$, and a message $m \in \{0,1\}^k$ and produces as output string s which we

call the *signature of* m. (We use notation $s \in \sigma(1^k, S, m)$ if the signing algorithm is probabilistic and $s = \sigma(1^k, S, m)$ otherwise. As a shorthand when the context is clear, the secret key may be omitted and we will write $s \in \sigma(S, m)$ to mean meaning that s is the signature of message m.)

- Verification algorithm V is a probabilistic polynomial time algorithm which given a public key P, a digital signature s, and a message m, returns 1 (i.e "true") or 0 (i.e "false") to indicate whether or not the signature is valid. We require that $V(P, s, m) = 1$ if $s \in \sigma(m)$ and 0 otherwise. (We may omit the public key and abbreviate $V(P, s, m)$ as $V(s, m)$ to indicate verifying signature s of message m when the context is clear.)
- The final characteristic of a digital signature system is its security against a probabilistic polynomial-time forger. We delay this definition to later.

Note that if V is probabilistic, we can relax the requirement on V to accept valid signatures and reject invalid signatures with high probability for all messages m, all sufficiently large security parameters k, and all pairs of keys $(P, S) \in G(1^k)$. The probability is taken over the coins of V and S. Note also that the message to be signed may be plain text or encrypted, because the message space of the digital signature system can be any subset of $\{0,1\}^*$.

9.2 Digital Signatures: the Trapdoor Function Model

Diffie and Hellman [68] propose that with a public key cryptosystem (G, E, D) based on the trapdoor function model, user A can sign any message M by appending as a digital signature $D(M) = f^{-1}(M)$ to M where f is A's trapdoor public function for which A alone knows the corresponding trapdoor information. Anyone can check the validity of this signature using A's public key from the public directory, since $E(D(M)) = f^{-1}(f(M))$. Note also that this signature becomes invalid if the message is changed, so that A is protected against modifications after he has signed the message, and the person examining the signature can be sure that the message he has received that was originally signed by A.

Thus, in their original proposal Diffie and Hellman linked the two tasks of encryption and digital signatures. We, however, **separate** these two tasks. It turns out that just as some cryptographic schemes are suited for encryption but not signatures, many proposals have been made for *signature-only* schemes which achieve higher security.

The RSA public-key cryptosystem which falls in the Diffie and Hellman paradigm allows one to implement digital signatures in a straightforward manner. The private exponent d now becomes the *signing exponent*, and the signature of a message M i which falls in the Diffie and Hellman paradigm s now the quantity $M^d \bmod n$. Anyone can verify that this signature is valid using the corresponding public *verification exponent* e by checking the identity $M = (M^d)^e \bmod n$. If this equation holds, then the signature M^d must have been created from M by the possessor of the corresponding signing exponent d. (Actually, it is possible that the reverse happened and that the "message" M was computed from the "signature" M^d using the verification equation and the public exponent e. However, such a message is likely to be unintelligible. In practice, this problem is easily avoided by always signing $f(M)$ instead of M, where f is a standard public one-way function.)

Cast in our notation for digital signature schemes, the Diffie-Hellman proposal is the following triple of algorithms (G, σ, V):

- Key Generation: $G(1^k)$ picks pairs (f_i, t_i) from F where $i \in I \cap \{0,1\}^k$.
- Signing Algorithm: $\sigma(1^k, f_i, t_i, m)$ outputs $f_i^{-1}(m)$.
- Verification Algorithm: $V(f_i, s, m)$ outputs 1 if $f_i(s) = m$ and 0 otherwise.

We will consider the security of this proposal and others. We first define security for digital signatures.

9.3 Defining and Proving Security for Signature Schemes

A theoretical treatment of digital signatures security was started by Goldwasser, Micali and Yao in [103] and continued in [101, 14, 145, 171, 74].

9.3.1 Attacks Against Digital Signatures

We distinguish three basic kinds of attacks, listed below in the order of increasing severity.

- *Key-Only Attack:* In this attack the adversary knows only the public key of the signer and therefore only has the capability of checking the validity of signatures of messages given to him.
- *Known Signature Attack:* The adversary knows the public key of the signer and has seen message/signature pairs chosen and produced by the legal signer. In reality, this the minimum an adversary can do.
- *Chosen Message Attack:* The adversary is allowed to ask the signer to sign a number of messages of the adversary's choice. The choice of these messages may depend on previously obtained signatures. For example, one may think of a notary public who signs documents on demand.

For a finer subdivision of the adversary's possible attacks see [101].

What does it mean to successfully forge a signature?

We distinguish several levels of success for an adversary, listed below in the order of increasing success for the adversary.

- *Existential Forgery:* The adversary succeeds in forging the signature of one message, not necessarily of his choice.
- *Selective Forgery:* The adversary succeeds in forging the signature of some message of his choice.
- *Universal Forgery:* The adversary, although unable to find the secret key of the The forger, is able to forge the signature of any message.
- *Total Break :* The adversary can compute the signer's secret key.

Clearly, different levels of security may be required for different applications. Sometimes, it may suffice to show that an adversary who is capable of a known signature attack can not succeed in selective forgery, while for other applications (for example when the signer is a notary-public or a tax-return preparer) it may be required that an adversary capable of a chosen signature attack can not succeed even at existential forgery with non-negligible probability.

The security that we will aim at, in these notes are that with high probability a polynomial time adversary would not be able to even existentially forge in the presence of a chosen message attack.

We say that *a digital signature is secure* if an enemy who can use the real signer as "an oracle" can not in time polynomial in the size of the public key forge a signature for any message whose signature was not obtained from the real signer. Formally, let B be a black box which maps messages m to valid signatures, i.e , $V(P, B(m), m) = 1$ for all messages m. Let the forging algorithm F on input the public key P have access to B, denoted as $F^B(P)$. The forging algorithm runs in two stages: it first launches a chosen message attack, and then outputs a "new forgery" which is defined to be any message-signature pair such that the message was not signed before and that signature is valid. We require that for all forging algorithms F, for all polynomials Q, for all sufficiently large k, $Prob(V(P,s,m) = 1 : (P,S) \stackrel{R}{\leftarrow} G(1^k) \ ; \ (m,s) \stackrel{R}{\leftarrow} F^B(P)) \leq \frac{1}{Q(k)}$.

The probability is taken over the choice of the keys $(P, S) \in G(1^k)$, the coin tosses of the forgery algorithm F, and the coins of B.

Diffie and Hellman's original proposal does not meet this strict definition of security; it is possible to existentially forge with just the public information: Choose an s at random. Apply the public key to s to produce $m = f(s)$. Now s is a valid signature of m.

Many digital signature systems have been proposed. For a fairly exhaustive list we refer to the paper [101] handed out.

We examine the security of three systems here.

9.3.2 The RSA Digital Signature Scheme

The first example is based on the RSA cryptosystem.

The public key is a pair of numbers (n, e) where n is the product of two large primes and e is relatively prime to $\phi(n)$, and the secret key is d such that $ed = 1 \bmod \phi(n)$. Signing is to compute $\sigma(m) = m^d modn$. Verifying is to raise the signature to the power e and compare it to the original message.

Claim 9.1 RSA is universallly forgable under a chosen-message attack. (alternatively, existentially forgable under known message attack) ∎

Proof: If we are able to produce signatures for two messages, the signature of the the product of the two messages is the product of the signatures. Let $m1$ and $m2$ be the two messages. Generate signatures for these messages with the black box: $\sigma(m1) = m1^d \bmod n$, $\sigma(m2) = m2^d \bmod n$. Now we can produce the signature for the product of these two messages: $\sigma(m1m2) = (m1m2)^d = m1^d m2^d = \sigma(m1)\sigma(m2) \bmod n$

To produce a signature for a message m, begin by choosing a random number $r \in 2n^*$. Now define $m1$ and $m2$ as follows: $m1 = mr \bmod n$, and $m2 = r^{-1} \bmod n$ Using the strategy above, we can find a signature for the product of these messages, which is the original message m, as follows: $m1m2 = (mr)r^{-1} = m$. ∎

9.3.3 El Gamal's Scheme

This digital signature system security relies on the difficulty of solving a a problem called the Diffie-Hellman-key-exchange (DHKE)problem, which is related to the discrete log problem. The DHKE problem is on input a prime p, a generator g, and $g^y, g^x \in Z_p^*$, compute output $g^{xy} \bmod p$. The best way currently known to solve the DHKE is to first solve the discrete log problem. Whether computing a discrete log is as hard as the Diffie-Hellman problem is currently an open question.

The following digital signature scheme is probabilistic. A close variant of it called DSS has been endorsed as a national standard.

Idea of the scheme:

- Public key: A triple (y, p, g), where $y = g^x \bmod p$, p is prime and g is a generator for Z_p^*.
- Secret key: x such that $y = g^x \bmod p$.
- Signing: The signature of message m is a pair (r, s) such that $0 \neq r, s \neq p-1$ and $g^m = y^r r^s \bmod p$.
- Verifying: Check that $g^m = y^r r^s \bmod p$ actually holds.

In order to generate a pair (r, s) which constitutes a signature, the signer begins by choosing a random number k such that $0 \neq k \neq p-1$ and $GCD(k, p-1) = 1$. Let $r = g^k (modp)$. Now we want to compute an s

such that $g^m = y^r r^s = g^{xr+ks} mod p$. In terms of the exponents, this relationship is $m = xr + ks(mod p - 1)$. Hence $s = (m - xr)^{k^{-1}} mod p - 1$. The signature of m is the pair (r, s).

Clearly, If an attacker could solve the discrete logarithm problem, he could break the scheme completely by computing the secret key x from the information in the public file. Moreover, if an attacker finds k for one message, he can solve the discrete logarithm problem, so the pseudo random number generator employed to generate k's has to be of superior quality.

Claim 9.2 This scheme is existentially forgable in the presence of a known message attack. ▮

Exercise.

Note on a key exchange protocol based on discrete log: It is interesting to note that it is possible for two people to exchange a secret key without prior secret meeting using the DL problem which is not known to yield a trapdoor function. This can be done by Persons A and B agree on a prime p and a generator g. Person A chooses a secret number x and sends $g^x(mod p)$ to B. Person B chooses a secret number y and sends $g^y(mod p)$ to A. Now each user can readily compute $g^{xy}(mod p)$; let this be the shared secret key. It is not known if computing xy is as difficult as DLP.

9.3.4 Rabin's Scheme

Rabin [164] proposed a method where the signature for a message M was essentially the square root of M, modulo n, the product of two large primes. Since the ability to take square roots is provably equivalent to the ability to factor n, an adversary should not be able to forge any signatures unless he can factor n. For our purpose let's consider the variant of it when $n = pq$ and $p = q = 3 \bmod 4$, so that the signature is uniquely determined.

This argument assumes that the adversary only has access to the public key containing the modulus n of the signer. An enemy may break this scheme with an *active* attack by asking the real signer to sign $M = x^2 \bmod n$, where x has been chosen randomly. If the signer agrees and produces a square root y of M, there is half a chance that $\gcd(n, x - y)$ will yield a nontrivial factor of n — the signer has thus betrayed his own secrets! Although Rabin proposed some practical techniques for circumventing this problem, they have the effect of eliminating the constructive reduction of factoring to forgery.

Let us look at this in some detail.

This digital signature scheme is based on the difficulty of computing square roots modulo a composite number.

- Public key: $n = pq$
- Secret key: primes p, q
- Signing: $s = \sqrt{m} \bmod n$ (assume WLOG that all m are squares)
- Verification: Check that $s^2 = m \bmod n$.

Claim 9.3 This system is existentially forgable with key-only attack. ▮

Proof: Choose a signature and square it to produce a corresponding message. ▮

Claim 9.4 The system is totally breakable in the face of a chosen message attack. ▮

Proof: We know that if we can find two distinct square roots of a message, we can factor the modulus. Choose a value s and let $m = s^2$. Now s is a valid signature of m. Submit m to the black box. There is a one in two chance that it will produce the same signature s. If so, repeat this process. If not, we have both square roots of m and can recover the factors of n. ▮

Security when "Breaking" is Equivalent to Factoring

Given the insecurity of Rabin's scheme in the face of a chosen message attack, one might hypothesize that there exists no secure digital signature system based on factoring. That is, a scheme wherein:

- "Breaking" the scheme is equivalent to factoring.
- The signature scheme is secure against a chosen message attack.

False proof: We assume (1) and show that (2) is impossible. Since the first statement is that "breaking" the scheme is equivalent to factoring, we know that the following reduction must be possible on input of a composite number n.

- Generate a public key P.
- Produce a message m.
- Produce a valid signature $s \in \sigma(P, m)$ using the "breaker" algorithm. (Repeat these three steps up to a polynomial number of times.)
- Factor n.

Conclude that the system must be insecure in the face of a chosen message attack, since we can substitute the CMA for the "breaker" algorithm in step 3. QED

What is wrong with this argument? First, there is only a vague definition of the public information P; it need not contain the number n. Second, the CMA will always produce signatures with respect to fixed public information, whereas in the above reduction it may be necessary to use different public information in every call to the "breaker".

9.4 Probabilistic Signatures

Probabilistic techniques have also been applied to the creation of digital signatures. This approach was pioneered by Goldwasser, Micali, and Yao [103], who presented signature schemes based on the difficulty of factoring and on the difficulty of inverting the RSA function for which it is provably hard for the adversary to existentially forge using a known signature attack.

Goldwasser, Micali, and Rivest [101] have strengthened this result by proposing a signature scheme which is not existentially forgable under a chosen message attack. Their scheme is based on the difficulty of factoring, and more generally on the existence of claw-free trapdoor permutations (that is, pairs f_0, f_1 of trapdoor permutations defined on a common domain for which it is hard to find x, y such that $f_0(x) = f_1(y)$).

The scheme, as originally described,although attractive in theory, is quite inefficient. However, it can be modified to allow more compact signatures, to make no use of memory between signatures other than for the public and secret keys, and even to remove the need of making random choices for every new signature. In particular, Goldreich [90] has made suggestions that make the factoring-based version of this scheme more practical while preserving its security properties.

Bellare and Micali in [14] have shown a digital signature scheme whose security can be based on the existence of any trapdoor permutation (a weaker requirement than claw-freeness). Then Naor and Yung [145] have shown how, starting with any *one-way* permutation, to design a digital signature scheme which is secure against existential forgery by a chosen signature attack. Finally, Rompel [171] has shown how to sign given any one-way function. These works build on an early idea due to Lamport on how to sign a single bit in [125]. The idea is as follows. If f is a one-way function, and Alice has published the two numbers $f(x_0) = y_0$ and $f(x_1) = y_1$, then she can sign the message 0 by releasing x_0 and she can similarly sign the message 1 by releasing the message x_1. Merkle [139] introduced some extensions of this basic idea, involving building a tree of authenticated values whose root is stored in the public key of the signer.

We now proceed to describe in detail some of these theoretical developments.

9.4.1 Claw-free Trap-door Permutations

We introduce the notion of *claw-free trap-door permutations* and show how to construct a signature scheme assuming the existence of a claw-free pair of permutations.

Definition 9.5 [f-claw] Let f_0, f_1 be permutation over a common domain D. We say that (x, y, z) is *f-claw* if $f_0(x) = f_1(y) = z$. $\blacksquare$

Definition 9.6 [A family of claw-free permutations] A family $F = \{f_{0,i}, f_{1,i} : D_i \to D_i\}_{i \in I}$ is called *a family of claw-free trap-door permutations* if:

1. There exists an algorithm G such that $G(1^k)$ outputs two pairs (f_0, t_0), (f_1, t_1) where where t_i is the trapdoor information for f_i.
2. There exists PPT an algorithm that given f_i and $x \in D_i$ computes $f_i(x)$.
3. $\forall$ (inverting algorithm) I, there exists some negligible function ν_I such that for all sufficiently large k, $Prob(f_0(x) = f_1(y) = z : ((f_0, t_0), (f_1, t_1)) \stackrel{R}{\leftarrow} G(1^k)\ ;\ (x, y, z) \stackrel{R}{\leftarrow} I(f_0, f_1)) < \nu_I(k)$

$\blacksquare$

The following observation shows that the existence of a pair of trap-door permutations does not immediately imply the existence of a claw-free permutation. For example, define a family of ("RSA") permutations by

$$f_{0,n}(x) \equiv x^3 \bmod n \quad f_{1,n}(x) \equiv x^5 \bmod n$$

($\gcd(x, n) = 1$, and $\gcd(15, \Phi(n)) = 1$). Since the two functions commute, it is easy to create a claw by choosing w at random and defining $x = f_{1,n}(w), y = f_{0,n}(w)$, and

$$z = f_{0,n}(x) = f_{1,n}(y) = w^{15} \bmod n$$

In general, the following question is

Open Problem 9.7 Does the existence of a family of trap-door permutations imply the existence of a family of claw-free trap-door permutations ?

The converse of the above is clearly true: Given a claw-free permutations generator, it is easy to generate a trap-door permutation. If $\{f_0, f_1\}$ is a pair of claw-free permutations over a common domain, that is, it is computationally infeasible to find a triple x, y, z such that $f_0(x) = f_1(y) = z$, then (f_0, f_0^{-1}) is trap-door. (Otherwise, give the inverting algorithm I, $z = f_1(y)$; z is also distributed uniformly over D, so with non-negligible probability, I can produce $x = f_0^{-1}(z)$. Therefore (x, y, z) is a claw, contradiction.)

9.4.2 Example: Claw-free permutations exists if factoring is hard

Let $n = pq$, where p and q are primes $(p, q \in H_k)$ and $p \equiv 3 \bmod 8$, $q \equiv 7 \bmod 8$. Observe that about 1/16 of odd prime pairs fit this requirement. Let QR_n denote the set of quadratic residues mod$n$.

We first note that:

1. $(\mathbf{J}_n(-1)) = +1$, but $-1 \notin QR_n$

2. $(\mathbf{J}_n(2)) = -1$, (and $2 \notin QR_n$) .

3. $x \in QR_n$ has exactly one square root $y \in QR_n$ (x is a Blum integer), but has four square root $y, -y, w, -w$ in general. Roots $w, -w$ have Jacobi symbol -1, y and $-y$ have Jacobi symbol $+1$.

We now define a family of pairs of functions, and prove, assuming the intractability of factoring, that it is a family of claw-free trap-door permutations over QR_n.
Define, for $x \in QR_n$:

$$f_{0,n}(x) = x^2 \bmod n \quad f_{1,n}(x) = 4x^2 \bmod n$$

It follows from the above notes that the functions $f_{0,n}$, $f_{1,n}$ are permutations of QR_n.
Claim: $\{f_{0,n}, f_{1,n}\}$ is claw-free.

Proof: Suppose that the pair is not claw-free. Assume $x, y \in QR_n$ satisfy

$$x^2 \equiv 4y^2 \bmod n$$

This implies that $(x-2y)(x+2y) \equiv 0 \bmod n$. However, checking the Jacobi symbol of both sides we have:

$$(\mathbf{J}_n(x)) = +1 \quad (\mathbf{J}_n(2y)) = (\frac{y}{n})(\frac{2}{n}) = -1 \quad (\mathbf{J}_n(-2y)) = (\frac{-1}{n}) = -1$$

That is, x is a quadratic residue, but $\pm 2y$ are not. Since $x \not\equiv \pm 2y \bmod n$ $\gcd(x \pm 2y, n)$ will produce a nontrivial factor on n. ∎

9.4.3 How to sign one bit

We first describe the basic building block of the signature scheme: signing one bit.

Let D be the common domain of the claw-free pair $\{f_0, f_1\}$, and assume x is selected randomly in D.

Public	Secret
$x \in D, f_0, f_1$	f_0^{-1}, f_1^{-1}

To sign the bit $b \in \{0,1\}$ let $s = \sigma(b) = f_b^{-1}(x)$.

To verify the signature s, check that $f_b(s) = x$.

Claim 9.8 The above scheme is existentially secured against Chosen Message Attack. ∎

Proof: Suppose, by way of contradiction, that the scheme is not secure. That is, $\exists$ a forging algorithm $F^{CMA}(P)$ that can forge the signature (given the public information); F asks for the signature of b and ($\forall$ polynomial Q and infinitely many k's) can sign $\overline{b}$ correctly with probability $> 1/Q(k)$. To derive the contradiction, we design an algorithm that, given F^{CMA}, can make claws:

input: f_0, f_1.

output: x, y, z, such that $f_0(x) = f_1(y) = z$ (with probability $> 1/Q(k)$).

(1) Select randomly $x \in D$; flip a coin and put in the public file: $z = f_{coin}(x) \in D, f_0, f_1$. (Note that f_0, f_1 are permutations, so z is uniform in D).

(2) Run algorithm $F^{CMA}(P)$:

1. If F asks for signature of $b = \overline{coin}$, go back to *(1)*.
2. If F asks for signature of $b = coin$, answer with $x = f_b^{-1}(f_{coin}(x))$.

(3) By the assumption, F can produce now a signature for $\overline{b}$, $y = f_{\overline{b}}^{-1}(f_{coin}(x))$, i.e. $z = f_b(x) = f_{\overline{b}}(y)$. That is, we have a claw:

∎

9.4.4 How to sign a message

As before, D is the common domain of the claw-free pair $\{f_0, f_1\}$, and x is selected randomly in D.

Public	Secret
$x \in D, f_0, f_1$	f_0^{-1}, f_1^{-1}

For $x \in D$, we sign the *first* message m^1 by:

$$s^1 = \sigma(m^1) = f_{m^1}^{-1}(x)$$

and verify by:

$$V(s^1, m^1) = \begin{cases} 1 & \text{if } f_{m^1}(s^1) = x \\ 0 & \text{otherwise} \end{cases}$$

where, for $m^1 = m_1^1 m_2^1 \dots m_k^1$:

$$f_{m^1}^{-1}(x) = f_{m_k^1}^{-1}(\dots(f_{m_2^1}^{-1}(f_{m_1^1}^{-1}(x))))$$

$$f_{m^1}(x) = f_{m_1^1}(\dots(f_{m_{k-1}^1}(f_{m_k^1}(x))))$$

Clearly f_m is a permutation on D, and is easy to compute. To sign the next message m^2, we apply the new permutation on the previous signature:

$$s^2 = \sigma(m^2) = (f_{m^2}^{-1}(s^1), m^1)$$

and verify by:

$$V(s^2, m^2) = \begin{cases} 1 & \text{if } f_{m^2}(s^2) = s^1 \text{ and } f_{m^1}(s^1) = x \\ 0 & \text{otherwise} \end{cases}$$

Notes:

1. With this scheme, the length of the signature grows linearly with the number of messages signed so far.

2. It is clearly easy to forge signatures for prefix of a message we have already seen. We therefore assume here that we pre-process the messages to be presented in a prefix-free encoding scheme. (i.e no messages is a prefix of another message).

Claim: The scheme is not existentially secure with respect to a Known Message Attack.

Proof: Assume $\exists F(H, P)$ that ($\forall$ polynomial Q and sufficiently large k), given the public information P and the history $H = ((m_1, \sigma(m_1)), \ldots, (m_l, \sigma(m_l)))$, for messages $m_1, m_2, ..., m_l$ selected by running $M(1^k)$, can find a message $\hat{m} \neq m_i$, $(1 \leq i \leq l)$, can produce a signature $\sigma(\hat{m})$ such that

$$Prob\{V(\sigma(\hat{m}), \hat{m}) = 1\} > \frac{1}{Q(k)}$$

where the probability is taken over all public files and coin tosses of F.

We now design an algorithm A that uses F to come up with a claw:

input: f_0, f_1.

output: a, b, c, such that $f_0(a) = f_1(b) = c$ (with probability $> 1/Q(k)$).

(1) Choose $m_1, m_2, \ldots, m_i \in \mathcal{M}(1^k), x \in_R D$. Let $z = f_{m_l}(\ldots(f_{m_1}(x)))$. Let $P = \{f_0, f_1, x\}$ be the public file. (Notice that z is also selected uniformly in D).

(2) Generate the history $H = (m_1, f_{m_1}(z)), \ldots (m_l, (f_{m_l}(\ldots(f_{m_1}(z))))$, Denote $m = m_1 \circ m_2 \circ \ldots \circ m_l$, the string of all messages generated.

(3) Run the forging algorithm $F(H, P)$ to produce $(\hat{m}, \sigma(\hat{m}))$.

(4) With non negligible probability, $\sigma(\hat{m})$ is a valid signature; that is, "walking back" with $f_{\hat{m}}$ from $\sigma(\hat{m})$, according to the history it supplies, will get to x, and therefore must meet the path going back from $\sigma(m_i)$ Let l be the location at which the two paths meet, that is, m agrees with $\hat{m}$ on the first $l-1$ bits, and denote $w = f^{-1}_{m_{l-1}}(\ldots(f^{-1}_{m_0}(z)))$. Assume, w.l.o.g that the $l-th$ bit of m is 0, the $l-th$ bit of $\hat{m}$ is 1, and let u, v be the corresponding $f_0^{-1}(w), f_1^{-1}(w)$. Output (u, v, w).

Clearly (u, v, w) is a claw. Thus, applying the public f_0, f_1 on the output of the forging algorithm F results in a claw, with non-negligible probability; contradiction. ▮

However, this scheme does not seem to be secure against a Chosen Message Attack. At least we do not know how to prove that it is. In the next section we modify it to achieve this.

9.4.5 A secure signature scheme based on claw free permutations

Let D_f be the common domain of the claw-free permutations pair Consider the following scheme, for signing messages $m_i \in \{0,1\}^k$ where $i \in \{1, \cdots, B(k)\}$ and $B(k)$ is a polynomial in k:

Choose two pairs of claw-free permutations, (f_0, f_1) and (g_0, g_1) for which we know $f_0^{-1}, f_1^{-1}, g_0^{-1}, g_1^{-1}$. Choose $X \in D_f$. Let the public key contain $D_f, X, f_0, f_1, g_0, g_1$ and let the secret key contain $f_0^{-1}, f_1^{-1}, g_0^{-1}, g_1^{-1}$.

PK	SK
D_f, X, f_0, f_1	f_0^{-1}, f_1^{-1}
g_0, g_1	g_0^{-1}, g_1^{-1}

Let $\circ$ be the concatenation function and set the history $H_1 = \emptyset$. To sign m_i, for $i \in \{1, \cdots, B(k)\}$:

1. Choose $R_i \in D_g$ uniformly.

2. Set $z_1^i = f^{-1}_{H_i \circ R_i}(X)$.
3. Set $z_2^i = g^{-1}_{m_i}(R_i)$.
4. Set signature $\sigma(m_i) = (z_1^i, z_2^i, H_i)$.
5. Set $H_{i+1} = H_i \circ R_i$.

To verify a message-signature pair (m, s) where $s = (z_1, z_2, H)$,

1. Let $R = g_m(z_2)$.
2. Check that $f_{H \circ R}(z_1) = X$.

If so, then the signature is valid and the verification function $V(m, s) = 1$. Otherwise, $V(m, s) = 0$. This scheme takes advantage of the fact that a new random element z_1^i can be used in place of X for each message so that the forger is unable to gain information by requesting signatures for a polynomial number of messages.

It is clear that the signing and verification procedures can be performed in polynomial time as required. The following theorem also shows that it is secure:

Theorem 9.9 The claw-free permutation signature scheme is existentially secure against CMA if claw-free permutations exist. ▮

Proof: (by contradiction) Suppose not. Then there is a forger $F^{CMA}(f_0, f_1, g_0, g_1, X)$ which consists of the following two stages:

Stage 1: F obtains signatures $\sigma(m_i)$ for up to $B(k)$ messages m_i of its choice.

Stage 2: F outputs $(\hat{m}, \hat{s})$ where $\hat{s} = (\hat{z}_1, \hat{z}_2, \hat{H})$ such that $\hat{m}$ is different than all m_i's requested in stage 1 and $V(\hat{m}, \hat{s}) = 1$.

We show that if such an F did exist, then there would be a PTM A which would:

Input: Uniformly chosen (h_0, h_1, D_h) claw-free such that h_0^{-1} and h_1^{-1} are not known.

Output: Either a h-claw with probability greater than $\frac{1}{Q(k)}$ where $Q(k)$ is a polynomial in k.

This is a contradiction by the definition of h_0 and h_1.

PTM A is based on the fact that when F is successful it does one of the following in stage 2:

Type 1 forgery: Find a g-claw

Type 2 forgery: Find a f-claw

Type 3 forgery: Find $f_0^{-1}(\omega)$ or $f_1^{-1}(\omega)$ for $\omega = z_1^{B(k)}$ the last point in the history provided by the signer

PTM A consists of two PTM's A_1 and A_2 which are run one after the other. A_1 attempts to find an h-claw based on the assumption that F produces a g-claw. A_2 attempts to find a h-claw based on the assumption that F produces a f-claw. Both A_1 and A_2 will use h_0 and h_1 in their public keys. In order to sign a message using h_0 and h_1, these PTM's will compute $v = h_i(R)$ for some $R \in D_h$ and use R as $h_b^{-1}(v)$. Thus, neither A_1 nor A_2 will need to invert h_b when answering F's requests. Note that since h_b is a permutation, v will be random if R is.

Description of A_1:

1. Choose (f_0, f_1, D_f) claw-free such that we know f_0^{-1} and f_1^{-1}. Let the public key contain $D_f, X, f_0, f_1, g_0 = h_0$, and $g_1 = h_1$. Let the secret key contain f_0^{-1} and f_1^{-1}.

PK	SK
D_f, X, f_0, f_1	f_0^{-1}, f_1^{-1}
$g_0 = h_0, g_1 = h_1$	

2. Set history $H_1 = \emptyset$ and run $F(f_0, f_1, g_0, g_1, X)$. When F asks for the signature of a message m_i,
   (a) Choose $z_2^i \in D_g$ at random.
   (b) Set $R_i = g_{m_i}(z_2^i)$.
   (c) Set $z_1^i = f_{H_i \circ R_i}^{-1}(X)$.
   (d) Output (z_1^i, z_2^i, H_i).
   (e) Set $H_{i+1} = H_i \circ R_i$.

   F then outputs $(\hat{m}, \hat{s})$ where $\hat{s} = (\hat{z}_1, \hat{z}_2, \hat{H})$.
3. Test to see that $V(\hat{m}, \hat{s}) = 1$. If not then A_1 fails.
4. Let $\hat{R} = g_{\hat{m}}(\hat{z}_2)$. If $\hat{R} \neq R_i$ for any i, then A_1 fails since F did not produce a type 1 forgery
5. Otherwise, let j be such that $\hat{R} = R_j$. We now have $h_{\hat{m}}(\hat{z}_2) = h_{m_j}(z_2^j) = R_j$. From this we easily obtain a h-claw.

Description of A_2:

1. Choose (g_0, g_1, D_g) claw-free such that we know g_0^{-1} and g_1^{-1}. Let $f_0 = h_0$ and $f_1 = h_1$. Choose $R_1, R_2, \cdots, R_{B(k)} \in D_g$, $c \in \{0, 1\}$ and $z \in D_f$ uniformly and independently. Set $X = f_{R_1 \circ R_2 \circ \cdots \circ R_{B(k)} \circ c}(z)$. Let the public key contain D_f, X, f_0, f_1, g_0 and g_1. Let the secret key contain g_0^{-1} and g_1^{-1}.

PK	SK
D_f, X, g_0, g_1	g_0^{-1}, g_1^{-1}
$f_0 = h_0, f_1 = h_1$	

2. Set history $H_1 = \emptyset$ and run $F(f_0, f_1, g_0, g_1, X)$. When F asks for signature of message m_i,
   (a) Set $z_1^i = f_{R_{i+1} \circ \cdots \circ R_{B(k)}}(X)$.
   (b) Set $z_2^i = g_{m_i}^{-1}(R_i)$.
   (c) Output (z_1^i, z_2^i, H_i).
   (d) Set $H_{i+1} = H_i \circ R_i$.

   F then outputs $(\hat{m}, \hat{s})$ where $\hat{s} = (\hat{z}_1, \hat{z}_2, \hat{H})$.
3. Let $\hat{R} = g_{\hat{m}}(\hat{z}^2)$.
4. There are three possibilities to consider:

   F made type 1 forgery: This means $\hat{H} \circ \hat{R} = H_i$ for some i. In this case A_2 fails.

F made type 2 forgery: There is some first bit in $\hat{H} \circ \hat{R}$ which differs from A_2's final history H_N. As a result, $\hat{H} \circ \hat{R} = H \circ b \circ \hat{S}$ and $H_N = H \circ \bar{b} \circ S$ for some $b \in \{0,1\}$ and strings $H, \hat{S}, S$. From this we obtain $f_b(f_{\hat{S}}(\hat{z}_1)) = f_{\bar{b}}(f_S(z_1^N))$ which provides A_2 with a h-claw.

F made type 3 forgery: $\hat{H} \circ \hat{R} = H_N \circ b \circ S$ for some bit b and string S. Since the bit d chosen by A_2 to follow H_N if another request were made is random, b will be different from d with probability $1/2$. In this case, A_2 will have $h_0^{-1}(h_{H_N}^{-1}(X))$ and $h_1^{-1}(h_{H_N}^{-1}(X))$ providing A_2 with a h-claw.

Suppose that with probability p_1 $F(f_0, f_1, g_0, g_1, X)$ provides a type 1 forgery, with probability p_2 $F(f_0, f_1, g_0, g_1, X)$ provides a type 2 forgery, and with probability p_3 $F(f_0, f_1, g_0, g_1, X)$ provides a type 3 forgery. Since $f_0, f_1, g_0, g_1, h_0, h_1$ are chosen uniformly over claw-free permutations, A_1 will succeed with probability p_1 and A_2 will succeed with probability $p_2 + \frac{p_3}{2}$. Thus, A_1 or A_2 will succeed with probability at least $\max(p_1, p_2 + \frac{p_3}{2}) \geq \frac{1}{3Q(k)}$. ∎

Notes:

1. Unlike the previous scheme, the signature here need not contain all the previous messages signed by the scheme; only the elements $R^i \in D_g$ are attached to the signature.

2. The length of the signature need not be linear with the number of messages signed. It is possible instead of linking the R^i together in a linear fashion, to build a tree structure, where R^1 authenticates R^2 and R^3, and R^2 authenticates R^4 and R^5 and so forth till we construct a full binary tree of depth logarithmic in $B(k)$ where $B(k)$ is a bound on the total number of signatures ever to be signed. Then, relabel the R^j's in the leafs of this tree as $r^1, ..., r^{B(k)}$.

   In the computation of the signature of the i-th message, we let $z_2^i = g_{m^i}^{-1}(r^i)$, and let $z_1^i = f_{r^i}^{-1}(R)$ where R is the father of r^i in the tree of authenticated R's. The signature of the ith message needs to contain then all R's on the path from the leaf r^i to the root, which is only logarithmic in the number of messages ever to be signed.

3. The cost of computing a $f_m^{-1}(x)$ is $|m|$(cost of computing f^{-1}). Next we show that for the implementation of claw-free functions based on factoring, the m factor can be saved.

Example: Efficient way to compute $f_m^{-1}(z)$

As we saw in Example 9.4.2, if factoring is hard, a particular family of trap-door permutations is claw free. Let $n = pq$, where p and q are primes and $p \equiv 3 \bmod 8$, $q \equiv 7 \bmod 8$. for $x \in QR_n$:

$$f_{0,n}(x) = x^2 \bmod n \quad f_{1,n}(x) = 4x^2 \bmod n$$

is this family of claw-free trap-door permutations.

Notation: We write $\sqrt{x} = y$ when that $y^2 = x$ and $y \in QR_n$.

To compute $f_m^{-1}(z)$ we first compute (all computations below are mod$n$):
$f_{00}^{-1}(z) = \sqrt{\sqrt{z}}$
$f_{01}^{-1}(z) = \sqrt{\frac{\sqrt{z}}{4}} = \frac{1}{\sqrt{4}}\sqrt{\sqrt{z}}$
$f_{10}^{-1}(z) = \sqrt{\sqrt{\frac{z}{4}}} = \frac{1}{\sqrt{\sqrt{4}}}\sqrt{\sqrt{z}}$
$f_{11}^{-1}(z) = \sqrt{\frac{1}{4}\sqrt{\frac{z}{4}}}$
Let $i(m)$ be the integer corresponding to the string m reversed. It is easy to see that in the general case we get: $f_m^{-1}(z) = (\frac{z}{4^{i(m)}})^{\frac{1}{2^{|m|}}}$

Now, all we need is to compute the $2^{|m|}$th root mod$n$ once, and this can be done efficiently, by raising to a power mod$\Phi(n)$.

9.4.6 A secure signature scheme based on trapdoor permutations

This section contains the trapdoor permutation signature scheme. We begin by showing the method for signing a single bit b:

1. Choose a trapdoor permutation f for which we know the inverse. Choose $X_0, X_1 \in D_f$ uniformly and independently. Let the public key contain $f, f(X_0)$, and $f(X_1)$. Let the secret key contain X_0 and X_1.

PK	SK
$f, f(X_0), f(X_1)$	X_0, X_1

2. The signature of b, $\sigma(b) = X_b$.

To verify (b, s) simply test $f(s) = f(X_b)$.

The scheme for signing multiple messages, uses the scheme above as a building block. The problem with signing multiple messages is that f cannot be reused. Thus, the trapdoor permutation signature scheme generates and signs a new trapdoor permutation for each message that is signed. The new trapdoor permutation can then be used to sign the next message.

Description of the trapdoor permutation signature scheme:

1. Choose a trapdoor permutation f_1 for which we know the inverse. Choose $\alpha_0^j, \alpha_1^j \in \{0,1\}^k$ for $j \in \{1, \cdots, k\}$ and $\beta_0^j, \beta_1^j \in \{0,1\}^k$ for $j \in \{1, \cdots, K(k)\}$ where $K(k)$ is a polynomial in k uniformly and independently. Let the public key contain f_1 and all α's and β's. Let the secret key contain f_1^{-1}. Let history $H_1 = \emptyset$.

PK	SK
$f_1, \alpha_b^i, \beta_b^j$ for $b \in \{0,1\}, i \in 1, \cdots, k, j \in 1, \cdots, K(k)$	f_1^{-1}

   To sign message $m^i = m_1 m_2 \cdots m_k$:

2. Set $AUTH_{m^i}^{\alpha, f_i} = (f_i^{-1}(\alpha_{m_1}^1), f_i^{-1}(\alpha_{m_2}^2), \cdots, f_1^{-1}(\alpha_{m_k}^k))$. $AUTH_{m^i}^{\alpha, f_i}$ is the signature of m^i using f_i and the α's.
3. Choose a new trapdoor function f_{i+1} such that we know f_{i+1}^{-1}.
4. Set $AUTH_{f_{i+1}}^{\beta, f_i} = (f_i^{-1}(\beta_{f_{i+1,1}}^1), (f_i^{-1}(\beta_{f_{i+1,2}}^2)), \cdots, (f_i^{-1}(\beta_{f_{i+1,k}}^K(k))))$ where $f_{i+1} = f_{i+1,1} \circ f_{i+1,2} \circ \cdots \circ f_{i+1,K(k)}$ is the binary representation of f_{i+1}.
5. The signature of m^i is $\sigma(m^i) = (AUTH_{m^i}^{\alpha, f_i}, AUTH_{f_{i+1}}^{\beta, f_i}, H_i)$. $AUTH_{f_{i+1}}^{\beta, f_i}$ is the signature of f_{i+1} using f_i and β's.
6. Set $H_{i+1} = H_i \circ (AUTH_{m^i}^{\alpha, f_i}, AUTH_{f_{i+1}}^{\beta, f_i})$.

Note: We assume that to describe f_{i+1}, $K(k)$ bits are sufficient.

Theorem 9.10 The trapdoor permutation signature scheme is existentially secure against CMA if trapdoor permutations exist. ▮

Proof: (by contradiction) Suppose not. Then there is a forger F which can request messages of its choice and then forge a message not yet requested with probability at least $\frac{1}{Q(k)}$ where $Q(k)$ is a polynomial in k. We show that if such an F did exist, we could find a PTM A' which would:

Input: Trapdoor function h for which inverse is unknown and $\omega \in \{0,1\}^k$.

Output: $h^{-1}(\omega)$ with probability at least $\frac{1}{Q'(k)}$ where $Q'(k)$ is a polynomial in k. Probability is taken over h's, ω's, coins of A'.

The construction of A' is as follows:

1. A' will attempt to use h as one of its trapdoor permutations in answering a signature request by F. Since A' does not know h^{-1}, it generates an appropriate set of α's and β's as follows: Randomly and uniformly choose $\gamma_b^j, \delta_b^j \in \{0,1\}^k$ for all $b \in \{0,1\}$ and $j \in \{1, \cdots, k\}$. Let $\alpha_b^j = h(\gamma_b^j)$ and $\beta_b^j = h(\delta_b^j)$ for the same range of b and j. Choose $n \in \{1, \cdots, B(k)\}$ uniformly. For the first phase, A' will act very much like a trapdoor permutation signature scheme with one exception. When it is time for A' to choose its one way permutation f_n, it will choose h. If A' were to leave the α's and β's unchanged at this point, it would be able to sign F's request for m^n though it does not know h^{-1}. However A' does change one of the α's or β's, as follows:

2. Randomly choose one of the α's or β's and set it equal to the input ω. Let the public key contain f_1 (this is h if $n = 1$), the α's and β's:

3. Run F using the current scheme. Note that with probability at least $\frac{1}{B(k)}$, F will make at least n message requests. Note also that when F does request a signature for message m^n, A' will be able to sign m^n with probability $1/2$. This is because with probability $1/2$ A' will not have to calculate (using h) the inverse of the α (or β) which was set to ω.

4. With probability $\frac{1}{Q(k)}$, F will successfully output a good forgery $(\hat{m}, \hat{s})$. In order for $\hat{s}$ to be a good forgery it must not only be verifiable, but it must diverge from the history of requests made to A'. With probability at least $\frac{1}{B(k)}$ the forger will choose to diverge from the history precisely at request n. Thus, F will use h as its trapdoor permutation.

5. If this is the case, the probability is $\frac{1}{2(k+K(k))}$ that the forger will invert the α (or β) which was set to ω.

6. If so, A' outputs $h^{-1}(\omega)$.

The probability that A' succeeds is therefore at least $\frac{1}{Q'(k)} = \frac{1}{4(k+K(k))B^2(k)}$ and since $4(k + K(k))B^2(k)$ is a polynomial in k we have a contradiction. ▮

9.5 Concrete security and Practical RSA based signatures

In practice, the most widely employed paradigm for signing with RSA is "hash then decrypt:" First "hash" the message into a domain point of RSA and then decrypt (ie. exponentiate with the RSA decryption exponent). The attraction of this paradigm is clear: signing takes just one RSA decryption, and verification just one RSA encryption. Furthermore it is simple to implement. Thus, in particular, this is the basis of several existing standards.

In this section we analyze this paradigm. We will see that, unfortunately, the security of the standardized schemes cannot be justified under standard assumptions about RSA, even assuming the underlying hash functions are ideal. Schemes with better justified security would be recommended.

We have already seen that such schemes do exist. Unfortunately, none of them match the schemes of the hash then decrypt paradigm in efficiency and simplicity. (See Section 9.5.12 for comparisons). So what can we do?

We present here some schemes that match "hash then decrypt" ones in efficiency but are provably secure assuming we have access to ideal hash functions. (As discussed in Section 7.4.6, this means that formally, the hash functions are modeled as random oracles, and in implementation, the hash functions are derived from cryptographic hash functions. This represents a practical compromise under which we can get efficiency with reasonable security assurances. See [15] for a full discussion of this approach.)

We present and analyze two schemes. The first is the $\mathcal{FDH}$ scheme of [15]. The second is the PSS of [24]. Furthermore we present a scheme called PSS-R which has the feature of *message recovery*. This is a useful way to effectively shorten signature sizes.

Let us now expand on all of the above. We begin by looking at current practice. Then we consider the full domain hash scheme of [15, 24] and discuss its security. Finally we come to PSS and PSS-R, and their exact security.

We present these schemes for RSA. The same can be done for the Rabin scheme.

The materiel of this section is taken largely from [24].

In order to make this section self-contained, we repeat some of the basics of previous parts of this chapter. Still the viewpoint is different, being that of concrete security, so the materiel is not entirely redundant.

9.5.1 Digital signature schemes

In the public key setting, the primitive used to provide data integrity is a digital signature scheme. It is just like a message authentication scheme except for an asymmetry in the key structure. The key sk used to generate tags (in this setting the tags are often called signatures) is different from the key pk used to verify signatures. Furthermore pk is public, in the sense that the adversary knows it too. So while only a signer in possession of the secret key can generate signatures, anyone in possession of the corresponding public key can verify the signatures.

Definition 9.11 A *digital signature scheme* $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ consists of three algorithms, as follows:

- The *key generation* algorithm $\mathcal{K}$ is a randomized algorithm that returns a pair (pk, sk) of keys, the public key and matching secret key, respectively; we write $(pk, sk) \stackrel{R}{\leftarrow} \mathcal{K}$
- The *signing* algorithm $\mathcal{S}$ is a (possibly randomized) algorithm that takes the secret key sk and a message M to return a *tag* or *signature* σ; we write $\sigma \leftarrow \mathcal{S}_{sk}(M)$
- The *verification* algorithm $\mathcal{V}$ is a deterministic algorithm that takes the public key pk, a message M, and a candidate signature σ for M to return a bit; we write $d \leftarrow \mathcal{V}_{pk}(M, \sigma)$.

Associated to each public key pk is a *message space* $\mathsf{Plaintexts}(pk)$ from which M is allowed to be drawn. We require that $\mathcal{V}_{pk}(M, \mathcal{S}_{sk}(M)) = 1$ for all $M \in \mathsf{Plaintexts}(pk)$. ▮ ▮

Let S be an entity that wants to have a digital signature capability. The first step is key generation: S runs $\mathcal{K}$ to generate a pair of keys (pk, sk) for itself. The key generation algorithm is run locally by S. S will produce signatures using sk, and others will verify these signatures using pk. The latter requires that anyone wishing to verify S's signatures must be in possession of this key pk which S has generated. Furthermore, the verifier must be assured that the public key is authentic, meaning really is the key of S and not someone else.

There are various mechanisms used to ensure that a prospective verifier is in possession of an authentic public key of the signer. These usually go under the name of *key management*. Very briefly, here are a few options. S might "hand" its public key to the verifier. More commonly S registers pk in S's name with some trusted server who acts like a public phone book, and anyone wishing to obtain S's public key requests it of the server by sending the server the name of S and getting back the public key. Steps must be taken to ensure that this communication too is authenticated, meaning the verifier is really in communication with the legitimate server, and that the registration process itself is authentic.

In fact key management is a topic in its own right, and needs an in-depth look. We will address it later. For the moment, what is important to grasp is the separation between problems. Namely, the key management processes are not part of the digital signature scheme itself. In constructing and analyzing the security of digital signature schemes, we make the assumption that any prospective verifier is in possession of an authentic copy of the public key of the signer. This assumption is made in what follows.

Once the key structure is in place, S can produce a digital signature on some document M by running $\mathcal{S}_{sk}(M)$ to return a signature σ. The pair (M, σ) is then the authenticated version of the document. Upon receiving a document M' and tag σ' purporting to be from S, a receiver B verifies the authenticity of the signature by using the specified verification procedure, which depends on the message, signature, and public key key. Namely he computes $\mathcal{V}_{pk}(M', \sigma')$, whose value is a bit. If this value is 1, it is read as saying the data is authentic, and so B accepts it as coming from S. Else it discards the data as unauthentic.

A viable scheme of course requires some security properties. But these are not our concern now. First we want to pin down what constitutes a specification of a scheme, so that we know what are the kinds of objects whose security we want to assess.

The last part of the definition says that tags that were correctly generated will pass the verification test. This simply ensures that authentic data will be accepted by the receiver.

The signature algorithm might be randomized, meaning internally flip coins and use these coins to determine its output. In this case, there may be many correct tags associated to a single message M. The algorithm might also be stateful, for example making use of a counter that is maintained by the sender. In that case the signature algorithm will access the counter as a global variable, updating it as necessary.

Unlike encryption schemes, whose encryption algorithms must be either randomized or stateful for the scheme to be secure, a deterministic, stateless signature algorithm is not only possible, but common.

9.5.2 A notion of security

Digital signatures aim to provide the same security property as message authentication schemes; the only change is the more flexible key structure. Accordingly, we can build on our past work in understanding and pinning down a notion of security for message authentication; the one for digital signatures differs only in that the adversary has access to the public key.

The goal of the adversary F is forgery: It wants to produce document M and tag σ such that $\mathcal{V}_{pk}(M, \sigma) = 1$, but M did not originate with the sender S. The adversary is allowed a chosen-message attack in the process of trying to produce forgeries, and the scheme is secure if even after such an attack the adversary has low probability of producing forgeries.

Let $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be an arbitrary digital signature scheme. Our goal is to formalize a measure a insecurity against forgery under chosen-message attack for this scheme. The adversary's actions are viewed as divided into two phases. The first is a "learning" phase in which it is given oracle access to $\mathcal{S}_{sk}(\cdot)$, where (pk, sk) was a priori chosen at random according to $\mathcal{K}$. It can query this oracle up to q times, in any manner it pleases, as long as all the queries are messages in the underlying message space $\mathsf{Plaintexts}(pk)$ associated to this key. Once this phase is over, it enters a "forgery" phases, in which it outputs a pair (M, σ) with $M \in \mathsf{Plaintexts}(pk)$. The adversary is declared successful if $\mathcal{V}_{pk}(M, \sigma) = 1$ and M was never a query made by the adversary to the signing oracle. Associated to any adversary F is thus a success probability. (The probability is over the choice of keys, any probabilistic choices that $\mathcal{S}$ might make, and the probabilistic choices, if any, that F makes.) The insecurity of the scheme is the success probability of the "cleverest" possible adversary, amongst all adversaries restricted in their resources to some fixed amount. We choose as resources the running time of the adversary, the number of queries it makes, and the total bit-length of all queries combined plus the bit-length of the output message M in the forgery.

Formally we define the "experiment of running the adversary" F in an attack on digital signature scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ as the following. Notice that the public key is provided as an input to F.

Experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{DS},F}$
  Let $(pk, sk) \stackrel{R}{\leftarrow} \mathcal{K}$
  Let $(M, \sigma) \leftarrow F^{\mathcal{S}_{sk}(\cdot)}(pk)$
  If $\mathcal{V}_{pk}(M, \sigma) = 1$ and M was not a query of F to its oracle
  Then return 1 else return 0

Definition 9.12 Let $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ be a digital signature scheme, and let F be an adversary that has access to an oracle. Let $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F}$ be the probability that experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{DS},F}$ returns 1. Then for any t, q, μ let

$$\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS}}(t, q, \mu) \quad = \quad \max_F \{ \mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F} \}$$

where the maximum is over all F such that the execution time of experiment $\mathbf{Exp}^{\text{uf-cma}}_{\mathcal{DS},F}$ is at most t, the number of oracle queries made by F is at most q, and the sum of the lengths of all oracle queries plus the length of the message M in the output forgery is at most μ bits. ∎ ∎

In practice, the queries correspond to messages signed by the legitimate sender, and it would make sense that getting these examples is more expensive than just computing on one's own. That is, we would expect q to be smaller than t. That is why q, μ are resources separate from t.

The RSA trapdoor permutation is widely used as the basis for digital signature schemes. Let us see how.

9.5.3 Key generation for RSA systems

We will consider various methods for generating digital signatures based on the presumed one-wayness of the RSA function. While these methods differ in how the signature and verification algorithms operate, they all use the same standard RSA key setup. Namely the public key of a user is a RSA modulus N and an encryption exponent e, where $N = pq$ is the product of two distinct primes, each of length $k/2$, and $\gcd(e, \varphi(N)) = 1$. The corresponding secret is the decryption exponent d where $ed \equiv 1 \pmod{\varphi(N)}$. For signatures it is convenient to put N into the secret key, viewing it as a pair (N, d), even though N is not, of course, really "secret". Here is the key generation algorithm in full:

Algorithm $\mathcal{K}$
  Pick at random two distinct primes p, q each $k/2$ bits long
  $N \leftarrow pq$; $e \stackrel{R}{\leftarrow} Z^*_{\varphi(N)}$; $d \leftarrow e^{-1} \bmod \varphi(N)$
  $pk \leftarrow (N, e)$; $sk \leftarrow (N, d)$
  Return pk, sk

Recall that $\varphi(N) = (p-1)(q-1)$ so that having generated p, q, the above key generation algorithm can compute $\varphi(N)$ and thus is able to complete the rest of the steps which depend on knowledge of this value. The computation of d is done using the extended GCD algorithm.

Recall that the map $\mathcal{RSA}_{N,e}(x) = x^e \bmod N$ is a permutation on Z^*_N with inverse $\mathcal{RSA}^{-1}_{N,d}(y) = y^d \bmod N$. We will make the well-believed assumption that $\mathcal{RSA}$ is one-way.

Below we will consider various signature schemes all of which use the above key generation algorithm and try to build in different ways on the one-wayness of RSA in order to securely sign.

9.5.4 Trapdoor signatures

Trapdoor signatures represent the most direct way in which to attempt to build on the one-wayness of $\mathcal{RSA}$ in order to sign. We believe that the signer, being in possession of the secret key N, d, is the only one who can compute the inverse RSA function $\mathcal{RSA}_{N,d}^{-1}$; for anyone else, knowing only the public key N, d, this task is computationally infeasible. Accordingly, the signer signs a message by performing on it this "hard" operation. This requires that the message be a member of Z_N^*, which, for convenience, is assumed. It is possible to verify a signature by performing the "easy" operation of computing $\mathcal{RSA}_{N,e}$ on the claimed signature and seeing if we get back the message. More precisely, the scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ has the above key generation algorithm, and the following signing and verifying algorithms:

Algorithm $\mathcal{S}_{N,d}(M)$	Algorithm $\mathcal{V}_{N,e}(M,x)$
$x \leftarrow M^d \bmod N$	$M' \leftarrow x^e \bmod N$
Return x	If $M = M'$ then return 1 else return 0

The message space for this public key is $\mathsf{Plaintexts}(N,e) = Z_N^*$, meaning the only allowed messages that the signer can sign are those which are elements of the group Z_N^*. In this scheme we have denoted the signature of M by x.

How secure is this scheme? As we said above, the intuition behind it is that the signing operation should be something only the signer can perform, since computing $\mathcal{RSA}_{N,e}^{-1}(M)$ is hard without knowledge of d. However, what one should remember is that one-wayness is under a very different model and setting than that of security for signatures. One-wayness tells us that if we select M at random and then feed it to an adversary (who knows N, e but not d) and ask the latter to find $x = \mathcal{RSA}_{N,e}^{-1}(M)$, then the adversary will have a hard time succeeding. But the adversary in a signature scheme is not given a random message M on which to forge a signature. Keep in mind that the goal of the adversary is to create a pair (M, x) such that $\mathcal{V}_{N,e}(M,x) = 1$. It does not have to try to imitate the signing algorithm; it must only do something that satisfies the verification algorithm. In particular it is allowed to choose M rather than having to sign a given or random M. It is also allowed to obtain a valid signature on any message other than the M it eventually outputs, via the signing oracle, corresponding in this case to having an oracle for $\mathcal{RSA}_{N,e}^{-1}(\cdot)$. These features make it easy for an adversary to forge signatures.

The simplest forgery strategy is simply to choose the signature first, and define the message as a function of it. That is illustrated by the following forger.

Forger $F^{\mathcal{S}_{N,e}(\cdot)}(N,e)$
  $x \stackrel{R}{\leftarrow} Z_N^*$; $M \leftarrow x^e \bmod N$
  Return (M, x)

This forger makes no queries of its signing oracle, and simply outputs the forgery (M, x) derived as shown. To compute its success probability we note that $\mathcal{V}_{N,e}(M,x) = 1$, because $x^e \bmod N = M$. So $\mathbf{Adv}_{\mathcal{DS},F}^{\text{uf-cma}} = 1$. This implies $\mathbf{Adv}_{\mathcal{DS}}^{\text{uf-cma}}(t, 0, k) = 1$, where $t = O(k^3)$ is the time to do an exponentiation modulo N, which is the computation time of F, and $\mu = k$ because the length of M is k. The values t, q, μ here being very small, we are saying the scheme is totally insecure.

The message M whose signature the above forger managed to forge is random. This is enough to break the scheme as per our definition of security, because we made a very strong definition of security. Actually for this scheme it is possible to even forge the signature of a given message M, but this time one has to use the signing oracle. The attack relies on the multiplicativity of the RSA function.

Forger $F^{\mathcal{S}_{N,e}(\cdot)}(N,e)$
  $M_1 \stackrel{R}{\leftarrow} Z_N^* - \{1, M\}$; $M_2 \leftarrow M M_1^{-1} \bmod N$
  $x_1 \leftarrow \mathcal{S}_{N,e}(M_1)$; $x_2 \leftarrow \mathcal{S}_{N,e}(M_2)$
  $x \leftarrow x_1 x_2 \bmod N$
  Return (M, x)

Given M the forger wants to compute a valid signature x for M. It creates M_1, M_2 as shown, and obtains their signatures x_1, x_2. It then sets $x = x_1 x_2 \bmod N$. Now the verification algorithm will check whether

$x^e \bmod N = M$. But note that

$$x^e \equiv (x_1x_2)^e \equiv x_1^e x_2^e \equiv M_1M_2 \equiv M \pmod{N}.$$

Here we used the multiplicativity of the RSA function and the fact that x_i is a valid signature of M_i for $i = 1, 2$. This means that x is a valid signature of M. Since M_1 is chosen to not be 1 or M, the same is true of M_2, and thus M was not an oracle query of F. So F succeeds with probability one.

These attacks indicate that there is more to signatures than one-wayness of the underlying function.

9.5.5 The hash-then-invert paradigm

Real-world RSA based signature schemes need to surmount the above attacks, and also attend to other impracticalities of the trapdoor setting. In particular, messages are not usually group elements; they are possibly long files, meaning bit strings of arbitrary lengths. Both issues are typically dealt with by pre-processing the given message M via a hash function Hash to yield a "domain point" y, which is then inverted under $\mathcal{RSA}_{N,e}^{-1}$ to yield the actual signature. The hash function $\mathit{Hash}\colon \{0,1\}^* \to Z_N^*$ is public, meaning its description is known, and anyone can compute it. (It may or may not use a key, but if it does, the key is public.) More precisely the scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ has the above key generation algorithm, and the following signing and verifying algorithms:

Algorithm $\mathcal{S}_{N,d}(M)$	Algorithm $\mathcal{V}_{N,e}(M,x)$
$y \leftarrow \mathit{Hash}(M)$	$y \leftarrow \mathit{Hash}(M)$
$x \leftarrow y^d \bmod N$	$y' \leftarrow x^e \bmod N$
Return x	If $y = y'$ then return 1 else return 0

Let us see why this might help resolve the weaknesses of trapdoor signatures, and what requirements security imposes on the hash function.

Let us return to the attacks presented on the trapdoor signature scheme above. Begin with the first forger we presented, who simply set M to $x^e \bmod N$ for some random $x \in Z_N^*$. What is the success probability of this strategy under the hash-then-invert scheme? The forger wins if $x^e \bmod N = \mathit{Hash}(M)$ (rather than merely $x^e \bmod N = M$ as before). The hope is that with a "good" hash function, it is very unlikely that $x^e \bmod N = \mathit{Hash}(M)$.

Consider now the second attack we presented above, which relied on the multiplicativity of the RSA function. For this attack to work under the hash-then-invert scheme, it would have to be true that

$$\mathit{Hash}(M_1) \cdot \mathit{Hash}(M_2) \equiv \mathit{Hash}(M) \pmod{N}. \tag{9.1}$$

Again, with a "good" hash function, we would hope that this is unlikely to be true.

The hash function is thus supposed to "destroy" the algebraic structure that makes attacks like the above possible. How we might find one that does this is something we have not addressed.

While the hash function might prevent some attacks that worked on the trapdoor scheme, its use leads to a new line of attack, based on collisions in the hash function. If an adversary can find two distinct messages M_1, M_2 that hash to the same value, meaning $\mathit{Hash}(M_1) = \mathit{Hash}(M_2)$, then it can easily forge signatures, as follows:

Forger $F^{\mathcal{S}_{N,e}(\cdot)}(N,e)$
  $x_1 \leftarrow \mathcal{S}_{N,e}(M_1)$
  Return (M_2, x_1)

This works because M_1, M_2 have the same signature. Namely because x_1 is a valid signature of M_1, and because M_1, M_2 have the same hash value, we have

$$x_1^e \equiv \mathit{Hash}(M_1) \equiv \mathit{Hash}(M_2) \pmod{N},$$

and this means the verification procedure will accept x_1 as a signature of M_2. Thus, a necessary requirement on the hash function is that it be collision-resistant, meaning it should be computationally infeasible to find distinct values M, M' such that $Hash(M) = Hash(M')$.

Below we will go on to more concrete instantiations of the hash-then-invert paradigm. But before we do that, it is important to try to assess what we have done so far. Above, we have pin-pointed some features of the hash function that are necessary for the security of the signature scheme. Collision-resistance is one. The other requirement is not so well formulated, but roughly we want to destroy algebraic structure in such a way that Equation (9.1), for example, should fail with high probability. Classical design focuses on these attacks and associated features of the hash function, and aims to implement suitable hash functions. But if you have been understanding the approaches and viewpoints we have been endeavoring to develop in this class and notes, you should have a more critical perspective. The key point to note is that what we need is not really to pin-point necessary features of the hash function to prevent certain attacks, but rather to pin-point *sufficient* features of the hash function, namely features sufficient to prevent *all* attacks, even ones that have not yet been conceived. And we have not done this. Of course, pinning down necessary features of the hash function is useful to gather intuition about what sufficient features might be, but it is only that, and we must be careful to not be seduced into thinking that it is enough, that we have identified all the concerns. Practice proves this complacence wrong again and again.

How can we hope to do better? Return to the basic philosophy of provable security. We want assurance that the signature scheme is secure under the assumption that its underlying primitives are secure. Thus we must try to tie the security of the signature scheme to the security of RSA as a one-way function, and some security condition on the hash function. With this in mind, let us proceed to examine some suggested solutions.

9.5.6 The PKCS #1 scheme

RSA corporation has been one of the main sources of software and standards for RSA based cryptography. RSA Labs (now a part of Security Dynamics Corporation) has created a set of standards called PKCS (Public Key Cryptography Standards). PKCS #1 is about signature (and encryption) schemes based on the RSA function. This standard is in wide use, and accordingly it will be illustrative to see what they do.

The standard uses the hash-then-invert paradigm, instantiating *Hash* via a particular hash function *PKCS-Hash*. Before specifying it we need to attend to some implementation issues. So far we have been thinking of the hash function as returning a group element, namely a point in the set Z_N^*. This is necessary because the output of the hash function must be something to which we can apply $\mathcal{RSA}_{N,e}^{-1}$. However, in an implementation, we can only process bit-strings. So in actuality the hash function must return a sequence of bits that we can interpret as an element of Z_N^*. This is not a big deal. The modulus N has length k bits (an example value of k is $k = 1024$) and Z_N^* is a subset of $\{1, \ldots, N\}$, consisting of those elements of $\{1, \ldots, N\}$ which are relatively prime to N. Each element of Z_N^* can thus be written as a k-bit string. So if the hash function returns a k-bit string y, we can interpret it as an element of Z_N^* simply by interpreting it as an integer. Well, almost. There are a couple of caveats. First, the integer must be relatively prime to N; second it must be at most N. The second can be ensured if the high-order bit of the k-bit string y is 0, meaning as an integer y is at most $2^{k-1} - 1 < N$. The first we simply don't worry about. The reason is that very few integers in the range $\{1, \ldots, N\}$ are not relatively prime to N. Indeed, the fraction of such integers in $\{1, \ldots, N\}$ is at most

$$\frac{N - \varphi(N)}{N} = 1 - \frac{(p-1)(q-1)}{pq} = \frac{p+q-1}{pq} < \frac{2^{1+k/2}}{2^{k-1}} = 4 \cdot 2^{-k/2} .$$

We used here the fact that $|p| = |q| = k/2$. Thus with typical modulus sizes like $k = 1024$, the fraction of points non-relatively prime to N is negligible. Not only do we not expect to hit these points by chance, but even the possibility of an adversary doing so on purpose is small because any such point is a multiple of either p or q, and taking the gcd of such a point with N would lead to factoring N, which is assumed computationally infeasible.

These technicalities having been dispensed with, let us proceed to describe the PKCS #1 hash function.

Recall we have already discussed collision-resistant hash functions. Let us fix a function $h: \{0,1\}^* \to \{0,1\}^l$ where $l \geq 128$ and which is "collision-resistant" in the sense that nobody knows how to find any pair of distinct points M, M' such that $h(M) = h(M')$. Currently the role tends to be played by SHA-1, so that $l = 160$. Prior to that it was MD5, which has $l = 128$. The RSA PKCS #1 standard defines

$$\mathit{PKCS\text{-}Hash}(M) \quad = \quad \texttt{0x 00 01 FF FF} \cdots \texttt{FF FF 00} \| h(M) \; .$$

Here $\|$ denotes concatenation, and enough FF-bytes are inserted that the length of $\mathit{PKCS\text{-}Hash}(M)$ is equal to k bits. Note the the first four bits of the hash output are zero, meaning as an integer it is certainly at most N, and by the above thus a group element. Also note that *PKCS-Hash* is collision-resistant simply because h is collision-resistant, so that it fulfills the first of our necessary conditions.

Recall that the signature scheme is exactly that of the hash-then-invert paradigm. For concreteness, let us rewrite the signing and verifying algorithms:

Algorithm $\mathcal{S}_{N,d}(M)$	Algorithm $\mathcal{V}_{N,e}(M,x)$
$y \leftarrow \mathit{PKCS\text{-}Hash}(M)$	$y \leftarrow \mathit{PKCS\text{-}Hash}(M)$
$x \leftarrow y^d \bmod N$	$y' \leftarrow x^e \bmod N$
Return x	If $y = y'$ then return 1 else return 0

Now what about the security of this signature scheme? Our first concern is the kinds of algebraic attacks we saw on trapdoor signatures. As discussed in Section 9.5.5, we would like that relations like Equation (9.1) fail. This we appear to get; it is hard to imagine how $\mathit{PKCS\text{-}Hash}(M_1) \cdot \mathit{PKCS\text{-}Hash}(M_2) \bmod N$ could have the specific structure required to make it look like the PKCS-hash of some message. This isn't a proof that the attack is impossible, of course, but at least it is not evident.

This is the point where our approach departs from the classical attack-based design one. Under the latter, the above scheme is acceptable because known attacks fail. But looking deeper there is cause for concern. The approach we want to take is to see how the desired security of the signature scheme relates to the assumed or understood security of the underlying primitive, in this case the RSA function.

We are assuming $\mathcal{RSA}$ is one-way, meaning it is computationally infeasible to compute $\mathcal{RSA}_{N,e}^{-1}(y)$ for a randomly chosen point $y \in Z_N^*$. On the other hand, the points on which $\mathcal{RSA}_{N,e}^{-1}$ is applied in the signature scheme are those in the range $S = \{ \mathit{PKCS\text{-}Hash}(M) \, : \, M \in \{0,1\}^* \}$ of the PKCS hash function. The size of S is at most 2^l since h outputs l bits and the other bits of $\mathit{PKCS\text{-}Hash}(\cdot)$ are fixed. With SHA-1 this means $|S| \leq 2^{160}$. This may seem like quite a big set, but within the RSA domain Z_N^* it is tiny:

$$\frac{|S|}{|Z_N^*|} \;\leq\; \frac{2^{160}}{2^{1023}} \;=\; \frac{1}{2^{863}} \; .$$

This is the probability with which a point chosen randomly from Z_N^* lands in S. For all practical purposes, it is zero. So $\mathcal{RSA}$ could very well be one-way and still be easy to invert on S, since the chance of a random point landing in S is so tiny. So the security of the PKCS scheme cannot be guaranteed solely under the standard one-wayness assumption on RSA. Note this is true no matter how "good" is the underlying hash function h (in this case SHA-1) which forms the basis for *PKCS-Hash*. The problem is the design of *PKCS-Hash* itself, in particular the padding.

The security of the PKCS signature scheme would require the assumption that $\mathcal{RSA}$ is hard to invert on the set S, a miniscule fraction of its full range. (And even this would be only a necessary, but not sufficient condition for the security of the signature scheme.)

Let us try to clarify and emphasize the view taken here. We are not saying that we know how to attack the PKCS scheme. But we are saying that an absence of known attacks should not be deemed a good reason to be satisfied with the scheme. We can identify "design flaws," such as the way the scheme uses RSA, which is not in accordance with our understanding of the security of RSA as a one-way function. And this is cause for concern.

9.5.7 The FDH scheme

From the above we see that if the hash-then-invert paradigm is to yield a signature scheme whose security can be based on the one-wayness of the $\mathcal{RSA}$ function, it must be that the points y on which $\mathcal{RSA}_{N,e}^{-1}$ is applied in the scheme are random ones. In other words, the output of the hash function must always "look random". Yet, even this only highlights a necessary condition, not (as far as we know) a sufficient one.

We now ask ourselves the following question. Suppose we had a "perfect" hash function *Hash*. In that case, at least, is the hash-then-invert signature scheme secure? To address this we must first decide what is a "perfect" hash function. The answer is quite natural: one that is random, namely returns a random answer to any query except for being consistent with respect to past queries. (We will explain more how this "random oracle" works later, but for the moment let us continue.) So our question becomes: in a model where *Hash* is perfect, can we *prove* that the signature scheme is secure if $\mathcal{RSA}$ is one-way?

This is a basic question indeed. If the hash-then-invert paradigm is in any way viable, we really must be able to prove security in the case the hash function is perfect. Were it not possible to prove security in this model it would be extremely inadvisable to adopt the hash-then-invert paradigm; if it doesn't work for a perfect hash function, how can we expect it to work in any real world setting?

Accordingly, we now focus on this "thought experiment" involving the use of the signature scheme with a perfect hash function. It is a thought experiment because no specific hash function is perfect. Our "hash function" is no longer fixed, it is just a box that flips coins. Yet, this thought experiment has something important to say about the security of our signing paradigm. It is not only a key step in our understanding but will lead us to better concrete schemes as we will see later.

Now let us say more about perfect hash functions. We assume that *Hash* returns a random member of Z_N^* every time it is invoked, except that if twice invoked on the same message, it returns the same thing both times. In other words, it is an instance of a random function with domain $\{0,1\}^*$ and range Z_N^*. We have seen such objects before, when we studied pseudorandomness: remember that we defined pseudorandom functions by considering experiments involving random functions. So the concept is not new. We call *Hash* a random oracle, and denote it by H in this context. It is accessible to all parties, signer, verifiers and adversary, but as an oracle. This means it is only accessible across a specified interface. To compute $H(M)$ a party must make an oracle call. This means it outputs M together with some indication that it wants $H(M)$ back, and an appropriate value is returned. Specifically it can output a pair (hash, M), the first component being merely a formal symbol used to indicate that this is a hash-oracle query. Having output this, the calling algorithm waits for the answer. Once the value $H(M)$ is returned, it continues its execution.

The best way to think about H is as a dynamic process which maintains a table of input-output pairs. Every time a query (hash, M) is made, the process first checks if its table contains a pair of the form (M, y) for some y, and if so, returns y. Else it picks a random y in Z_N^*, puts (M, y) into the table, and returns y as the answer to the oracle query.

We consider the above hash-then-invert signature scheme in the model where the hash function *Hash* is a random oracle H. This is called the Full Domain Hash (FDH) scheme. This scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ has the usual RSA key generation algorithm $\mathcal{K}$ of Section 9.5.3. We write the signing and verifying algorithms as follows:

Algorithm $\mathcal{S}_{N,d}^H(M)$	Algorithm $\mathcal{V}_{N,e}^H(M, x)$
$y \leftarrow H(M)$	$y \leftarrow H(M)$
$x \leftarrow y^d \bmod N$	$y' \leftarrow x^e \bmod N$
Return x	If $y = y'$ then return 1 else return 0

The only change with respect to the way we wrote the algorithms for the generic hash-then-invert scheme of Section 9.5.5 is notational: we write H as a superscript to indicate that it is an oracle accessible only via the specified oracle interface. The instruction $y \leftarrow H(M)$ is implemented by making the query (hash, M) and letting y denote the answer returned, as discussed above.

We now ask ourselves whether the above signature scheme is secure under the assumption that $\mathcal{RSA}$ is one-way. To consider this question we first need to extend our definitions to encompass the new model. The key difference is the the success probability of an adversary is taken over the random choice of H in addition to the random choices previously considered. The forger F as before has access to a signing oracle, but now also has access to H. Furthermore, $\mathcal{S}$ itself has access to H. Let us first write the experiment that measures the success of forger F and then discuss it more.

Experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$
  Let $((N,e),(N,d)) \stackrel{R}{\leftarrow} \mathcal{K}$
  Pick $H\colon \{0,1\}^* \to Z_N^*$ at random
  Let $(M,x) \leftarrow F^{H,\mathcal{S}^H_{N,d}(\cdot)}(N,e)$
  If $\mathcal{V}^H_{N,e}(M,\sigma) = 1$ and M was not a query of F to its signing oracle
  Then return 1 else return 0

The superscript of "ro" to the name of the experiment indicates it is in the random oracle model. For concreteness we have written the experiment specific to the case of RSA based schemes such as FDH-RSA, but it is easily generalized. We begin by picking the RSA public key (N,e) and secret key (N,d) as per the standard RSA key generation algorithm. Next a random hash function is chosen. This choice is best thought of dynamically as discussed above. Don't think of H as chosen all at once, but rather think of the process implementing the table we described, so that random choices are made only at the time the H oracle is called. The forger is given oracle access to H. To model a chosen-message attack it is also given access to a signing oracle $\mathcal{S}^H_{N,d}(\cdot)$ to which it can give any message M and receive a signature, this being $H(M)^d \bmod N$ in the case of FDH-RSA. In order to return a signature, the signing oracle itself must invoke the H oracle, so that there are two ways in which the H oracle might be invoked: either directly by F or indirectly, by $\mathcal{S}^H_{N,d}(\cdot)$ when the latter is invoked by F. After querying its oracles some number of times the forger outputs a message M and candidate signature x for it. We say that F is successful if the verification process would accept M,x, but F never asked the signing oracle to sign M. (F is certainly allowed to make hash query M, and indeed it is hard to imagine how it might hope to succeed in forgery otherwise, but it is not allowed to make sign query M.)

We let $\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{DS},F}$ be the probability that experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS},F)$ returns 1. The notation is the same as we have used before; we will know whether or not we are in the random oracle model from the description of the scheme. Then for any $t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \mu$ let

$$\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{DS}}(t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \mu) \quad = \quad \max_F \{ \mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{DS},F} \} .$$

The resources $t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \mu$ are measured in a specific way as we now describe. Rather than referring to the resources used by the adversary F itself, they measure the resources used by the entire experiment. We first define the *execution time* as the time taken by the entire experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS},F)$. This means it includes the time to compute answers to oracle queries, to generate the keys, and even to verify the forgery. Then t is supposed to upper bound the execution time plus the size of the code of F. The number of sign queries made by F must be at most q_{sig}. In counting hash queries we again look at the entire experiment and ask that the total number of queries to H here be at most q_{hash}. Included in the count are the direct hash queries of F, the indirect hash queries made by the signing oracle, and even the hash query made by the verification algorithm in the last step. (The latter means that q_{hash} is always at least the number of hash queries required for a verification, which for FDH-RSA is one. In fact for FDH-RSA we will have $q_{\mathrm{hash}} \geq q_{\mathrm{sig}} + 1$, something to be kept in mind when interpreting later results.) Finally μ is the sum of the lengths of all messages in sign queries plus the length of the final output message M.

In this setting we claim that the FDH-RSA scheme is secure. The following theorem upper bounds its insecurity solely in terms of the insecurity of $\mathcal{RSA}$ as a one-way function.

Theorem 9.13 [**24**] Let $\mathcal{DS}$ be the FDH-RSA scheme with security parameter k. Then for any t, q_{sig}, μ and any $q_{\mathrm{hash}} \geq 1 + q_{\mathrm{sig}}$ we have

$$\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{DS}}(t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \mu) \quad \leq \quad q_{\mathrm{hash}} \cdot \mathbf{Adv}^{\mathrm{owf}}_{\mathcal{RSA}}(t')$$

where $t' = t + q_{\mathrm{hash}} \cdot O(k^3)$. ▌▌

The theorem says that the only way to forge signatures in the FDH-RSA scheme is to try to invert the $\mathcal{RSA}$ function on random points. There is some loss in security: it might be that the chance of breaking the signature scheme is larger than that of inverting $\mathcal{RSA}$ in comparable time, by a factor of the number of hash queries made in the forging experiment. But we can make $\mathbf{Adv}^{\mathrm{owf}}_{\mathcal{RSA}}(t')$ small enough that even $q_{\mathrm{hash}} \cdot \mathbf{Adv}^{\mathrm{owf}}_{\mathcal{RSA}}(t')$ is small, by choosing a larger modulus size k.

One must remember the caveat: this is in a model where the hash function is random. Yet, even this tells us something, namely that the hash-then-invert paradigm itself is sound, at least for "perfect" hash functions. This puts us in a better position to explore concrete instantiations of the paradigm.

Let us now proceed to the proof of Theorem 9.13. The paradigm is the usual one. Let F be some forger attacking the FDH-RSA scheme. Assume that the resource parameters associated to it are $t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \mu$, measured relative to the experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$ as we discussed above. We will design an inverter I for the $\mathcal{RSA}$ function such that

$$\mathbf{Adv}^{\mathrm{owf}}_{\mathcal{RSA},I} \;\geq\; \frac{\mathbf{Adv}^{\mathrm{uf\text{-}cma}}_{\mathcal{DS},F}}{q_{\mathrm{hash}}} \,. \tag{9.2}$$

Furthermore I will have running time bounded by the value t' given in the theorem statement. Now the theorem follows as usual by some arithmetic and the taking of maximums.

Remember that inverter I takes as input (N, e), describing an instance $\mathcal{RSA}_{N,e}$ of the RSA function, and also a point $y \in Z_N^*$. Its job is to try to output $\mathcal{RSA}^{-1}_{N,e}(y) = y^d \bmod N$, the inverse of y under the RSA function, where d is the decryption exponent corresponding to encryption exponent e. Of course, neither d nor the factorization of N are available to I. The success of I is measured under a random choice of $(N, e), (N, d)$ as given by the standard RSA key generation algorithm, and also a random choice of y from Z_N^*. In order to accomplish its task, I will run F as a subroutine, on input public key N, e, hoping somehow to use F's ability to forger signatures to find $\mathcal{RSA}^{-1}_{N,e}(y)$. Before we discuss how I might hope to use the forger to determine the inverse of point y, we need to take a closer look at what it means to run F as a subroutine.

Recall that F has access to two oracles, and makes calls to them. At any point in its execution it might output (hash, M). It will then wait for a return value, which it interprets as $H(M)$. Once this is received, it continues its execution. Similarly it might output (sign, M) and then wait to receive a value it interprets as $\mathcal{S}^H_{N,d}(M)$. Having got this value, it continues. The important thing to understand is that F, as an algorithm, merely communicates with oracles via an interface. It does not control what these oracles return. You might think of an oracle query like a system call. Think of F as writing an oracle query M at some specific prescribed place in memory. Some process is expected to put in another prescribed place a value that F will take as the answer. F reads what is there, and goes on.

When I executes F, no oracles are actually present. F does not know that. It will at some point make an oracle query, assuming the oracles are present, say query (hash, M). It then waits for an answer. If I wants to run F to completion, it is up to I to provide some answer to F as the answer to this oracle query. F will take whatever it is given and go on executing. If I cannot provide an answer, F will not continue running; it will just sit there, waiting. We have seen this idea of "simulation" before in several proofs: I is creating a "virtual reality" under which F can believe itself to be in its usual environment.

The strategy of I will be to take advantage of its control over responses to oracle queries. It will choose them in strange ways, not quite the way they were chosen in Experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$. Since F is just an algorithm, it processes whatever it receives, and eventually will halt with some output, a claimed forgery (M, x). By clever choices of replies to oracle queries, I will ensure that F is fooled into not knowing that it is not really in $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$, and furthermore x will be the desired inverse of y. Not always, though; I has to be lucky. But it will be lucky often enough.

We begin by consider the case of a very simple forger F. It makes no sign queries and exactly one hash query (hash, M). It then outputs a pair (M, x) as the claimed forgery, the message M being the same in the hash query and the forgery. (In this case we have $q_{\mathrm{sig}} = 0$ and $q_{\mathrm{hash}} = 2$, the last due to the hash query of F and the final verification query in the experiment.) Now if F is successful then x is a valid signature of M, meaning $x^e \equiv H(M) \bmod N$, or, equivalently, $x \equiv H(M)^d \bmod N$. Somehow, F has found the inverse of $H(M)$, the value returned to it as the response to oracle query M. Now remember that I's goal had been to compute $y^d \bmod N$ where y was its given input. A natural thought suggests itself: If F can invert $\mathcal{RSA}_{N,e}$ at $H(M)$, then I will "set" $H(M)$ to y, and thereby obtain the inverse of y under $\mathcal{RSA}_{N,e}$. I can set $H(M)$ in this way because it controls the answers to oracle queries. When F makes query (hash, M), the inverter I will simply return y as the response. If F then outputs a valid forgery (M, x), we have $x = y^d \bmod N$, and I can output x, its job done.

But why would F return a valid forgery when it got y as its response to hash query M? Maybe it will refuse this, saying it will not work on points supplied by an inverter I. But this will not happen. F is simply an algorithm and works on whatever it is given. What is important is solely the distribution of the response. In Experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$ the response to (hash, M) is a random element of Z_N^*. But y has exactly the same distribution, because that is how it is chosen in the experiment defining the success of I in breaking $\mathcal{RSA}$ as a one-way function. So F cannot behave any differently in this virtual reality than it could in its real world; its probability of returning a valid forgery is still $\mathbf{Adv}^{\text{uf-cma}}_{\mathcal{DS},F}$. Thus for this simple F the success probability of the inverter in finding $y^d \bmod N$ is exactly the same as the success probability of F in forging signatures. Equation (9.2) claims less, so we certainly satisfy it.

However, most forgers will not be so obliging as to make no sign queries, and just one hash query consisting of the very message in their forgery. I must be able to handle any forger.

Inverter I will define a pair of subroutines, $H\text{-}Sim$ (called the hash oracle simulator) and $\mathcal{S}\text{-}Sim$ (called the sign oracle simulator) to play the role of the hash and sign oracles respectively. Namely, whenever F makes a query (hash, M) the inverter I will return $H\text{-}Sim(M)$ to F as the answer, and whenever F makes a query (sign, M) the inverter I will return $\mathcal{S}\text{-}Sim(M)$ to F as the answer. (The $\mathcal{S}\text{-}Sim$ routine will additionally invoke $H\text{-}Sim$.) As it executes, I will build up various tables (arrays) that "define" H. For $j = 1, \ldots, q_{\mathrm{hash}}$, the j-th string on which H is called in the experiment (either directly due to a hash query by F, indirectly due to a sign query by F, or due to the final verification query) will be recorded as $Msg[j]$; the response returned by the hash oracle simulator to $Msg[j]$ is stored as $Y[j]$; and if $Msg[j]$ is a sign query then the response returned to F as the "signature" is $X[j]$. Now the question is how I defines all these values.

Suppose the j-th hash query in the experiment arises indirectly, as a result of a sign query $(\mathsf{sign}, Msg[j])$ by F. In Experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$ the forger will be returned $H(Msg[j])^d \bmod N$. If I wants to keep F running it must return something plausible. What could I do? It could attempt to directly mimic the signing process, setting $Y[j]$ to a random value (remember $Y[j]$ plays the role of $H(Msg[j])$) and returning $(Y[j])^d \bmod N$. But it won't be able to compute the latter since it is not in possesion of the secret signing exponent d. The trick, instead, is that I first picks a value $X[j]$ at random in Z_N^* and sets $Y[j] = (X[j])^e \bmod N$. Now it can return $X[j]$ as the answer to the sign query, and this answer is accurate in the sense that the verification relation (which F might check) holds: we have $Y[j] \equiv (X[j])^e \bmod N$.

This leaves a couple of loose ends. One is that we assumed above that I has the liberty of defining $Y[j]$ at the point the sign query was made. But perhaps $Msg[j] = Msg[l]$ for some $l < j$ due to there having been a hash query involving this same message in the past. Then the hash value $Y[j]$ is already defined, as $Y[l]$, and cannot be changed. This can be addressed quite simply however: for any hash query $Msg[l]$, the hash simulator can follow the above strategy of setting the reply $Y[l] = (X[l])^e \bmod N$ at the time the hash query is made, meaning it prepares itself ahead of time for the possibility that $Msg[l]$ is later a sign query. Maybe it will not be, but nothing is lost.

Well, almost. Something is lost, actually. A reader who has managed to stay awake so far may notice that we have solved two problems: how to use F to find $y^d \bmod N$ where y is the input to I, and how to simulate answers to sign and hash queries of F, but that these processes are in conflict. The way we got $y^d \bmod N$ was by returning y as the answer to query (hash, M) where M is the message in the forgery. However, we do not know beforehand which message in a hash query will be the one in the forgery. So it is difficult to know how to answer a hash query $Msg[j]$; do we return y, or do we return $(X[j])^e \bmod N$ for some $X[j]$? If we do the first, we will not be able to answer a sign query with message $Msg[j]$; if we do the second, and if $Msg[j]$ equals the message in the forgery, we will not find the inverse of y. The answer is to take a guess as to which to do. There is some chance that this guess is right, and I succeeds in that case.

Specifically, notice that $Msg[q_{\mathrm{hash}}] = M$ is the message in the forgery by definition since $Msg[q_{\mathrm{hash}}]$ is the message in the final verification query. The message M might occur more than once in the list, but it occurs at least once. Now I will choose a random i in the range $1 \leq i \leq q_{\rm hash}$ and respond by y to hash query $(\mathsf{hash}, Msg[i])$. To all other queries j it will respond by first picking $X[j]$ at random in Z_N^* and setting $H(Msg[j]) = (X[j])^e \bmod N$. The forged message M will equal $Msg[i]$ with probability at least $1/q_{\rm hash}$ and this will imply Equation (9.2). Below we summarize these ideas as a proof of Theorem 9.13.

It is tempting from the above description to suggest that we always choose $i = q_{\rm hash}$, since $Msg[q_{\rm hash}] = M$ by definition. Why won't that work? Because M might also have been equal to $Msg[j]$ for some $j < q_{\rm hash}$, and if we had set $i = q_{\rm hash}$ then at the time we want to return y as the answer to M we find we have already defined $H(M)$ as something else and it is too late to change our minds.

Proof of Theorem 9.13: Let F be some forger attacking the FDH-RSA scheme, with resource parameters $t, q_{\rm sig}, q_{\rm hash}, \mu$, measured relative to the experiment $\mathrm{ForgeExp}^{\rm ro}(\mathcal{DS}, F)$ as we discussed above. We will design an inverter I for the $\mathcal{RSA}$ function such that Equation (9.2) is true and the running time of I is bounded by the value t' given in the theorem statement. The theorem follows.

We first decribe I in terms of two subroutines: a hash oracle simulator $H\text{-}Sim(\cdot)$ and a sign oracle simulator $\mathcal{S}\text{-}Sim(\cdot)$. It maintains three tables, Msg, X and Y, each an array with index in the range from 1 to $q_{\rm hash}$. It picks a random index i. All these are global variables which will be used also be the subroutines. The intended meaning of the array entries is the following, for $j = 1, \ldots, q_{\rm hash}$–

$Msg[j]$	–	The j-th hash query in the experiment
$Y[j]$	–	The reply of the hash oracle simulator to the above, meaning the value playing the role of $H(Msg[j])$
$X[j]$	–	For $j \neq i$, the response to sign query $Msg[j]$, meaning it satisfies $(X[j])^e \equiv Y[j] \pmod N$. For $j = i$ it is the value y.

The code for the inverter is below.

Inverter $I(N, e, y)$
  Initialize arrays $Msg[1 \ldots q_{\rm hash}]$, $X[1 \ldots q_{\rm hash}]$, $Y[1 \ldots q_{\rm hash}]$ to empty
  $j \leftarrow 0$; $i \stackrel{R}{\leftarrow} \{1, \ldots, q_{\rm hash}\}$
  Run F on input N, e
  If F makes oracle query (hash, M)
    then $h \leftarrow H\text{-}Sim(M)$; return h to F as the answer
  If F makes oracle query (sign, M)
    then $x \leftarrow \mathcal{S}\text{-}Sim(M)$; return x to F as the answer
  Until F halts with output (M, x)
  $y' \leftarrow H\text{-}Sim(M)$
  Return x

The inverter responds to oracle queries by using the appropriate subroutines. Once it has the claimed forgery, it makes the corresponding hash query and then returns the signature x.

We now describe the hash oracle simulator. It makes reference to the global variables instantiated in in the main code of I. It takes as argument a value v which is simply some message whose hash is requested either directly by F or by the sign simulator below when the latter is invoked by F.

We will make use of a subroutine *Find* that given an array A, a value v and index m, returns 0 if $v \notin \{A[1], \ldots, A[m]\}$, and else returns the smallest index l such that $v = A[l]$.

Subroutine $H\text{-}Sim(v)$
  $l \leftarrow Find(Msg, v, j)$; $j \leftarrow j + 1$; $Msg[j] \leftarrow v$
  If $l = 0$ then

If $j = i$ then $Y[j] \leftarrow y$
Else $X[j] \stackrel{R}{\leftarrow} Z_N^*$; $Y[j] \leftarrow (X[j])^e \bmod N$
End If
Return $Y[j]$
Else
If $j = i$ then abort
Else $X[j] \leftarrow X[l]$; $Y[j] \leftarrow Y[l]$; Return $Y[j]$
End If
End If

The manner in which the hash queries are answered enables the following sign simulator.

Subroutine $\mathcal{S}\text{-}Sim(M)$
$h \leftarrow H\text{-}Sim(M)$
If $j = i$ then abort
Else return $X[j]$
End If

Inverter I might abort execution due to the "abort" instruction in either subroutine. The first such situation is that the hash oracle simulator is unable to return y as the response to the i-th hash query because this query equals a previously replied to query. The second case is that F asks for the signature of the message which is the i-th hash query, and I cannot provide that since it is hoping the i-th message is the one in the forgery and has returned y as the hash oracle response.

Now we need to lower bound the success probability of I in inverting $\mathcal{RSA}$, namely the quantity

$$\mathbf{Adv}^{\mathrm{owf}}_{\mathcal{RSA},I} \quad = \quad \mathbf{P}\left[x^e \equiv y \pmod N \;:\; ((N,e),(N,d)) \stackrel{R}{\leftarrow} \mathcal{K} \;;\; y \stackrel{R}{\leftarrow} Z_N^* \;;\; x \leftarrow I(N,e,y) \right] .$$

There are a few observations involved in verifying the bound claimed in Equation (9.2). First that the "view" of F at any time at which I has not aborted is the "same" as in Experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$. This means that the answers being returned to F by I are distributed exactly as they would be in the real experiment. Second, F gets no information about the value i that I chooses at random. Now remember that the last hash simulator query made by I is the message M in the forgery, so M is certainly in the array Msg at the end of the execution of I. Let $l = Find(Msg, M, q_{\mathrm{hash}})$ be the first index at which M occurs, meaning $Msg[l] = M$ but no previous message is M. The random choice of i then means that there is a $1/q_{\mathrm{hash}}$ chance that $i = l$, which in turn means that $Y[i] = y$ and the hash oracle simulator won't abort. If x is a correct signature of M we will have $x^e \equiv Y[i] \pmod N$ because $Y[i]$ is $H(M)$ from the point of view of F. So I is successful whenever this happens. ∎

9.5.8 PSS0: A security improvement

The FDH-RSA signature scheme has the attractive security attribute of possessing a proof of security under the assumption that $\mathcal{RSA}$ is a one-way function, albeit in the random oracle model. However the quantitative security as given by Theorem 9.13 could be better. The theorem leaves open the possibility that one could forge signatures with a probability that is q_{hash} times the probability of being able to invert the $\mathcal{RSA}$ function at a random point, the two actions being measured with regard to adversaries with comparable execution time. Since q_{hash} could be quite large, say 2^{60}, there is an appreciable loss in security here. We now present a scheme in which the security relation is much tighter: the probability of signature forgery is not appreciably higher than that of being able to invert $\mathcal{RSA}$ in comparable time.

The scheme is called PSS0, for "probabilistic signature scheme, version 0", to emphasize a key aspect of it, namely that it is randomized: the signing algorithm picks a new random value each time it is invoked and uses that to compute signatures. The scheme $\mathcal{DS} = (\mathcal{K}, \mathcal{S}, \mathcal{V})$ has the usual RSA key generation algorithm $\mathcal{K}$ of Section 9.5.3. Like FDH-RSA it makes use of a public hash function $H\colon \{0,1\}^* \to Z_N^*$ which is modeled as a random oracle. Additonally it has a parameter s which is the length of the random value chosen by the signing algorithm. We write the signing and verifying algorithms as follows:

Algorithm $\mathcal{S}_{N,d}^H(M)$	Algorithm $\mathcal{V}_{N,e}^H(M,\sigma)$
$\quad r \stackrel{R}{\leftarrow} \{0,1\}^s$	$\quad$ Parse σ as (r,x) where $\lvert r\rvert = s$
$\quad y \leftarrow H(r\Vert M)$	$\quad y \leftarrow H(r\Vert M)$
$\quad x \leftarrow y^d \bmod N$	$\quad$ If $x^e \bmod N = y$
$\quad$ Return (r,x)	$\qquad$ Then return 1 else return 0

Obvious "range checks" are for simplicity not written explicitly in the verification code; for example in a real implementation the latter should check that $1 \le x < N$ and $\gcd(x,N) = 1$.

This scheme may still be viewed as being in the "hash-then-invert" paradigm, except that the hash is randomized via a value chosen by the signing algorithm. If you twice sign the same message, you are likely to get different signatures. Notice that random value r must be included in the signature since otherwise it would not be possible to verify the signature. Thus unlike the previous schemes, the signature is not a member of Z_N^*; it is a pair one of whose components is an s-bit string and the other is a member of Z_N^*. The length of the signature is $s+k$ bits, somewhat longer than signatures for deterministic hash-then-invert signature schemes. It will usually suffice to set l to, say, 160, and given that k could be 1024, the length increase may be tolerable.

The success probability of a forger F attacking $\mathcal{DS}$ is measured in the random oracle model, via experiment $\mathrm{ForgeExp}^{\mathrm{ro}}(\mathcal{DS}, F)$. Namely the experiment is the same experiment as in the FDH-RSA case; only the scheme $\mathcal{DS}$ we plug in is now the one above. Accordingly we have the insecurity function associated to the scheme. Now we can summarize the security property of the PSS0 scheme.

Theorem 9.14 [24] Let $\mathcal{DS}$ be the PSS0 scheme with security parameters k and s. Then for any t, q_{sig}, μ and any $q_{\mathrm{hash}} \ge 1 + q_{\mathrm{sig}}$ we have

$$\mathbf{Adv}_{\mathcal{DS}}^{\text{uf-cma}}(t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \mu) \quad \le \quad \mathbf{Adv}_{\mathcal{RSA}}^{\mathrm{owf}}(t') + \frac{(q_{\mathrm{hash}} - 1) \cdot q_{\mathrm{sig}}}{2^s}$$

where $t' = t + q_{\mathrm{hash}} \cdot O(k^3)$. ▌▌

Say $q_{\mathrm{hash}} = 2^{60}$ and $q_{\mathrm{sig}} = 2^{40}$. With $l = 160$ the additive term above is about 2^{-60}, which is very small. So for all practical purposes the additive term can be neglected and the security of the PSS0 signature scheme is tightly related to that of $\mathcal{RSA}$.

We proceed to the proof of Theorem 9.14. Given a foger F attacking $\mathcal{DS}$ in the random oracle model, with resources $t, q_{\mathrm{sig}}, q_{\mathrm{hash}}, \mu$, we construct an inverter I for $\mathcal{RSA}$ such that

$$\mathbf{Adv}_{\mathcal{RSA},I}^{\mathrm{owf}} \;\ge\; \mathbf{Adv}_{\mathcal{DS},F}^{\text{uf-cma}} - \frac{(q_{\mathrm{hash}} - 1) \cdot q_{\mathrm{sig}}}{2^s} \,. \tag{9.3}$$

Furthermore I will have running time bounded by the value t' given in the theorem statement. Now the theorem follows as usual by some arithmetic and the taking of maximums.

The design of I follows the same framework used in the proof of Theorem 9.13. Namely I, on input N, e, y, will execute F on input N, e, and answer F's oracle queries so that F can complete its execution. From the forgery, I will somehow find $y^d \bmod N$. I will respond to hash oracle queries of F via a subroutine $\mathit{H}\text{-}\mathit{Sim}$ called the hash oracle simulator, and will respond to sign queries of F via a subroutine $\mathcal{S}\text{-}\mathit{Sim}$ called the sign oracle simulator. A large part of the design is the design of these subroutines. To get some intuition it is helpful to step back to the proof of Theorem 9.13.

We see that in that proof, the multiplicative factor of q_{hash} in Equation (9.2) came from I's guessing at random a value $i \in \{1, \ldots, q_{\mathrm{hash}}\}$, and hoping that $i = \mathit{Find}(\mathit{Msg}, M, q_{\mathrm{hash}})$ where M is the message in the forgery. That is, it must guess the time at which the message in the forgery is first queried of the hash oracle. The best we can say about the chance of getting this guess right is that it is at least $1/q_{\text{hash}}$. However if we now want I's probability of success to be as in Equation (9.3), we cannot afford to guess the time at which the forgery message is queried of the hash oracle. Yet, we certainly don't know this time in advance. Somehow, I has to be able to take advantage of the forgery to return $y^d \bmod N$ nonetheless.

A simple idea that comes to mind is to return y as the answer to all hash queries. Then certainly a forgery on a queried message yields the desired value $y^d \bmod N$. Consider this strategy for FDH. In that case, two problems arise. First, these answers would then not be random and indpendent, as required for answers to hash queries. Second, if a message in a hash query is later a sign query, I would have no way to answer the sign query. (Remember that I computed its reply to hash query $Msg[j]$ for $j \neq i$ as $(X[j])^e \bmod N$ exactly in order to be able to later return $X[j]$ if $Msg[j]$ showed up as a sign query. But there is a conflict here: I can either do this, or return y, but not both. It has to choose, and in FDH case it chooses at random.)

The first problem is actually easily settled by a small algebraic trick, exploiting what is called the *self-reducibility* of RSA. When I wants to return y as an answer to a hash oracle query $Msg[j]$, it picks a random $X[j]$ in Z_N^* and returns $Y[j] = y \cdot (X[j])^e \bmod N$. The value $X[j]$ is chosen randomly and independently each time. Now the fact that $\mathcal{RSA}_{N,e}$ is a permutation means that all the different $Y[j]$ values are randomly and independently distributed. Furthermore, suppose $(M, (r, x))$ is a forgery for which hash oracle query $r \| M$ has been made and got the reponse $Y[l] = y \cdot (X[l])^e \bmod N$. Then we have $(x \cdot X[l]^{-1})^e \equiv y \pmod{N}$, and thus the inverse of y is $x \cdot X[l]^{-1} \bmod N$.

The second problem however, cannot be resolved for FDH. That is exactly why PSS0 pre-pends the random value r to the message before hashing. This effectively "separates" the two kinds of hash queries: the direct queries of F to the hash oracle, and the indirect queries to the hash oracle arising from the sign oracle. The direct hash oracle queries have the form $r \| M$ for some l-bit string r and some message M. The sign query is just a message M. To answer it, a value r is first chosen at random. But then the value $r \| M$ has low probability of having been a previous hash query. So at the time any new direct hash query is made, I can assume it will never be an indirect hash query, and thus reply via the above trick.

Here now is the full proof.

Proof of Theorem 9.13: Let F be some forger attacking the PSS0 scheme, with resource parameters $t, q_{\text{sig}}, q_{\text{hash}}, \mu$, measured relative to the experiment $\text{ForgeExp}^{\text{ro}}(\mathcal{DS}, F)$ as we discussed above. We will design an inverter I for the $\mathcal{RSA}$ function such that Equation (9.3) is true and the running time of I is bounded by the value t' given in the theorem statement. The theorem follows.

We first decribe I in terms of two subroutines: a hash oracle simulator $H\text{-}Sim(\cdot)$ and a sign oracle simulator $\mathcal{S}\text{-}Sim(\cdot)$. It maintains four tables, R, V, X and Y, each an array with index in the range from 1 to q_{hash}. All these are global variables which will be used also be the subroutines. The intended meaning of the array entries is the following, for $j = 1, \ldots, q_{\text{hash}}$–

$V[j]$	–	The j-th hash query in the experiment, having the form $R[j] \| Msg[j]$
$R[j]$	–	The first l-bits of $V[j]$
$Y[j]$	–	The value playing the role of $H(V[j])$, chosen either by the hash simulator or the sign simulator
$X[j]$	–	If $V[j]$ is a direct hash oracle query of F this satisfies $Y[j] \cdot X[j]^{-e} \equiv y \pmod{N}$. If $V[j]$ is an indirect hash oracle query this satisfies $X[j]^e \equiv Y[j] \pmod{N}$, meaning it is a signature of $Msg[j]$.

Note that we don't actually need to store the array Msg; it is only referred to above in the explanation of terms.

We will make use of a subroutine *Find* that given an array A, a value v and index m, returns 0 if $v \notin \{A[1], \ldots, A[m]\}$, and else returns the smallest index l such that $v = A[l]$.

Inverter $I(N, e, y)$
  Initialize arrays $R[1 \ldots q_{\text{hash}}]$, $V[1 \ldots q_{\text{hash}}]$, $X[1 \ldots q_{\text{hash}}]$, $Y[1 \ldots q_{\text{hash}}]$, to empty
  $j \leftarrow 0$
  Run F on input N, e
  If F makes oracle query (hash, v)
    then $h \leftarrow \textit{H-Sim}(v)$; return h to F as the answer
  If F makes oracle query (sign, M)
    then $\sigma \leftarrow \mathcal{S}\textit{-Sim}(M)$; return σ to F as the answer
  Until F halts with output $(M, (r, x))$
  $y \leftarrow \textit{H-Sim}(r \| M)$; $l \leftarrow \textit{Find}(V, r\|M, q_{\text{hash}})$
  $w \leftarrow x \cdot X[l]^{-1} \bmod N$; Return w

We now describe the hash oracle simulator. It makes reference to the global variables instantiated in in the main code of I. It takes as argument a value v which is assumed to be at least s bits long, meaning of the form $r\|M$ for some s bit strong r. (There is no need to consider hash queries not of this form since they are not relevant to the signature scheme.)

Subroutine $\textit{H-Sim}(v)$
  Parse v as $r\|M$ where $|r| = s$
  $l \leftarrow \textit{Find}(V, v, j)$; $j \leftarrow j + 1$; $R[j] \leftarrow r$; $V[j] \leftarrow v$
  If $l = 0$ then
    $X[j] \stackrel{R}{\leftarrow} Z_N^*$; $Y[j] \leftarrow y \cdot (X[j])^e \bmod N$; Return $Y[j]$
  Else
    $X[j] \leftarrow X[l]$; $Y[j] \leftarrow Y[l]$; Return $Y[j]$
  End If

Every string v queried of the hash oracle is put by this routine into a table V, so that $V[j]$ is the j-th hash oracle query in the execution of F. The following sign simulator does not invoke the hash simulator, but if necessary fills in the necessary tables itself.

Subroutine $\mathcal{S}\textit{-Sim}(M)$
  $r \stackrel{R}{\leftarrow} \{0,1\}^s$
  $l \leftarrow \textit{Find}(R, r, j)$
  If $l \neq 0$ then abort
  Else
    $j \leftarrow j + 1$; $R[j] \leftarrow r$; $V[j] \leftarrow r\|M$; $X[j] \stackrel{R}{\leftarrow} Z_N^*$; $Y[j] \leftarrow (X[j])^e \bmod N$
    Return $X[j]$
  End If

Now we need to lower bound the success probability of I in inverting $\mathcal{RSA}$, namely the quantity

$$\mathbf{Adv}^{\text{owf}}_{\mathcal{RSA},I} \quad = \quad \mathbf{P}\left[\, x^e \equiv y \pmod N \;:\; ((N,e),(N,d)) \stackrel{R}{\leftarrow} \mathcal{K} \;;\; y \stackrel{R}{\leftarrow} Z_N^* \;;\; x \leftarrow I(N,e,y) \,\right] .$$

Inverter I might abort execution due to the "abort" instruction in the sign oracle simulator. This happens if the random value r chosen in the sign oracle simulator is already present in the set $\{R[1], \ldots, R[j]\}$. This set has size at most $q_{\text{hash}} - 1$ at the time of an sign query, so the probability that r falls in it is at most $(q_{\text{hash}} - 1)/2^s$. The sign oracle simulator is invoked at most q_{sig} times, so the probability of abortion at some time in the execution of I is at most $(q_{\text{hash}} - 1)q_{\text{sig}}/2^s$.

The "view" of F at any time at which I has not aborted is the "same" as the view of F in Experiment $\text{ForgeExp}^{\text{ro}}(\mathcal{DS}, F)$. This means that the answers being returned to F by I are distributed exactly as they

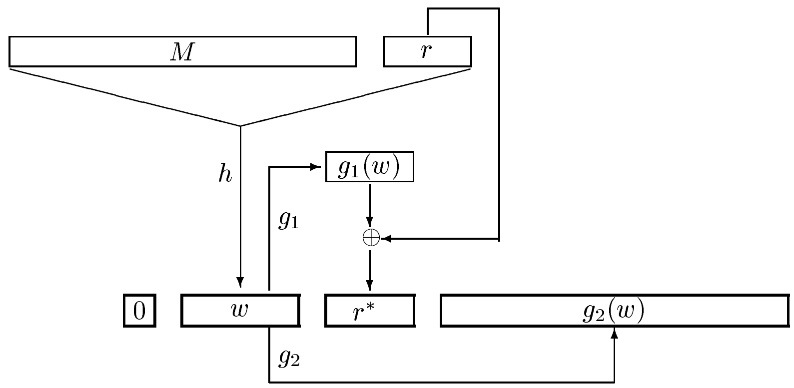

Figure 9.1: PSS: Components of image $y = 0\|w\|r^*\|g_2(w)$ are darkened. The signature of M is $y^d \bmod N$.

would be in the real experiment. Now remember that the last hash simulator query made by I is $r\|M$ where M is the message in the forgery, so $r\|M$ is certainly in the array V at the end of the execution of I. So $l = \mathit{Find}(V, r\|M, q_{\text{hash}}) \neq 0$. We know that $r\|M$ was not put in V by the sign simulator, because F is not allowed to have made sign query M. This means the hash oracle simulator has been invoked on $r\|M$. This means that $Y[l] = y \cdot (X[l])^e \bmod N$ because that is the way the hash oracle simulator chooses its replies. The correctness of the forgery means that $x^e \equiv H(r\|M) \pmod{N}$, and the role of the H value here is played by $Y[l]$, so we get $x^e \equiv Y[l] \equiv y \cdot X[l] \pmod{N}$. Solving this gives $(x \cdot X[l]^{-1})^e \bmod N = y$, and thus the inverter is correct in returning $x \cdot X[l]^{-1} \bmod N$. ▮

9.5.9 The Probabilistic Signature Scheme – PSS

PSS0 obtained improved security over FDH-RSA but at the cost of an increase in signature size. The scheme presented here reduces the signature size, so that it has both high security and the same signature size as FDH-RSA. This is the probabilistic signature scheme (PSS) of [24].

Signature scheme $\mathsf{PSS}[k_0, k_1] = (\mathcal{K}, \mathit{SignPSS}, \mathit{VerifyPSS})$ is parameterized by k_0 and k_1, which are numbers between 1 and k satisfying $k_0 + k_1 \leq k - 1$. To be concrete, the reader may like to imagine $k = 1024$, $k_0 = k_1 = 128$. The scheme has the usual RSA key generation algorithm $\mathcal{K}$ of Section 9.5.3. The signing and verifying algorithms make use of two hash functions. The first, h, called the compressor, maps as $h\colon \{0,1\}^* \to \{0,1\}^{k_1}$ and the second, g, called the generator, maps as $g\colon \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$. (The analysis assumes these to be ideal. In practice they can be implemented in simple ways out of cryptographic hash functions like MD5, as discussed in Appendix 9.5.11.) Let g_1 be the function which on input $w \in \{0,1\}^{k_1}$ returns the first k_0 bits of $g(w)$, and let g_2 be the function which on input $w \in \{0,1\}^{k_1}$ returns the remaining $k - k_0 - k_1 - 1$ bits of $g(w)$. We now describe how to sign and verify. Refer to Figure 9.1 for a picture. We write the signing and verifying algorithms as follows:

Algorithm $\mathit{SignPSS}^{g,h}_{N,d}(M)$	Algorithm $\mathit{VerifyPSS}^{g,h}_{N,e}(M, x)$
$r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$; $w \leftarrow h(M\|r)$	$y \leftarrow x^e \bmod N$
$r^* \leftarrow g_1(w) \oplus r$	Parse y as $b\|w\|r^*\|\gamma$ where
$y \leftarrow 0\|w\|r^*\|g_2(w)$	$\|b\| = 1$, $\|w\| = k_1$, $\|r^*\| = k_0$
$x \leftarrow y^d \bmod N$	$r \leftarrow r^* \oplus g_1(w)$
Return x	If ($h(M\|r) = w$ and $g_2(w) = \gamma$ and $b = 0$)
	Then return 1 else return 0

Obvious "range checks" are for simplicity not written explicitly in the verification code; for example in a real implementation the latter should check that $1 \leq x < N$ and $\gcd(x, N) = 1$.

The step $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$ indicates that the signer picks at random a seed r of k_0 bits. He then concatenates this seed to the message M, effectively "randomizing" the message, and hashes this down, via the "compressing" function, to a k_1 bit string w. Then the generator g is applied to w to yield a k_0 bit string $r^* = g_1(w)$ and a $k - k_0 - k_1 - 1$ bit string $g_2(w)$. The first is used to "mask" the k_0-bit seed r, resulting in the masked seed r^*. Now $w \| r^*$ is pre-pended with a 0 bit and appended with $g_2(w)$ to create the image point y which is decrypted under the RSA function to define the signature. (The 0-bit is to guarantee that y is in Z_N^*.)

Notice that a new seed is chosen for each message. In particular, a given message has many possible signatures, depending on the value of r chosen by the signer.

Given (M, x), the verifier first computes $y = x^e \bmod N$ and recovers r^*, w, r. These are used to check that y was correctly constructed, and the verifier only accepts if all the checks succeed.

Note the efficiency of the scheme is as claimed. Signing takes one application of h, one application of g, and one RSA decryption, while verification takes one application of h, one application of g, and one RSA encryption.

The following theorem proves the security of the PSS based on the one-wayness of RSA. The relation between the two securities is pretty much the same as that for PSS0 that we saw in Theorem 9.14, meaning essentially tight, and much tighter than the one we saw for the $\mathcal{FDH}$ scheme. This time however it was achieved without increase in signature size.

Theorem 9.15 [24] Let $\mathcal{DS}$ be the PSS scheme with security parameters k_0 and k_1. Then for any $t, q_{\rm sig}, \mu$ and any $q_{\rm hash} \geq 1 + q_{\rm sig}$ we have

$$\mathbf{Adv}^{\rm uf\text{-}cma}_{\mathcal{DS}}(t, q_{\rm sig}, q_{\rm hash}, \mu) \quad \leq \quad \mathbf{Adv}^{\rm owf}_{\mathcal{RSA}}(t') + [3(q_{\rm hash} - 1)^2] \cdot (2^{-k_0} + 2^{-k_1}) ,$$

where $t' = t + q_{\rm hash} \cdot k_0 \cdot O(k^3)$. ∎ ∎

The proof is in [24]. It extends the proof of Theorem 9.14 given above.

9.5.10 Signing with Message Recovery – PSS-R

Message recovery. In a standard signature scheme the signer transmits the message M in the clear, attaching to it the signature x. In a scheme which provides message recovery, only an "enhanced signature" τ is transmitted. The goal is to save on the bandwidth for a signed message: we want the length of this enhanced signature to be smaller than $|M| + |x|$. (In particular, when M is short, we would like the length of τ to be k, the signature length.) The verifier recovers the message M from the enhanced signature and checks authenticity at the same time.

We accomplish this by "folding" part of the message into the signature in such a way that it is "recoverable" by the verifier. When the length n of M is small, we can in fact fold the entire message into the signature, so that only a k bit quantity is transmitted. In the scheme below, if the security parameter is $k = 1024$, we can fold up to 767 message bits into the signature.

Definition. Formally, the key generation and signing algorithms are as before, but $\mathcal{V}$ is replaced by *Recover*, which takes *pk* and x and returns $\mathit{Recover}_{pk}(x) \in \{0,1\}^* \cup \{\mathsf{REJECT}\}$. The distinguished point REJECT is used to indicate that the recipient rejected the signature; a return value of $M \in \{0,1\}^*$ indicates that the verifier accepts the message M as authentic. The formulation of security is the same except for what it means for the forger to be *successful*: it should provide an x such that $\mathit{Recover}_{pk}(x) = M \in \{0,1\}^*$, where M was not a previous signing query. We demand that if x is produced via $x \leftarrow \mathcal{S}_{sk}(M)$ then $\mathit{Recover}_{pk}(x) = M$.

A simple variant of PSS achieves message recovery. We now describe that scheme and its security.

The scheme. The scheme $\mathsf{PSS\text{-}R}[k_0, k_1] = (\mathcal{K}, \mathit{SignPSSR}, \mathit{RecPSSR})$ is parameterized by k_0 and k_1, as before. The key generation algorithm is $\mathcal{K}$, the same as before. As with PSS, the signing and verifying algorithms depend on hash functions $h \colon \{0,1\}^* \to \{0,1\}^{k_1}$ and $g \colon \{0,1\}^{k_1} \to \{0,1\}^{k-k_1-1}$, and we use the same g_1 and g_2 notation. For simplicity of explication, we assume that the messages to be signed have length $n = k - k_0 - k_1 - 1$. (Suggested choices of parameters are $k = 1024$, $k_0 = k_1 = 128$ and $n = 767$.) In this case, we produce "enhanced signatures" of only k bits from which the verifier can recover the n-bit message

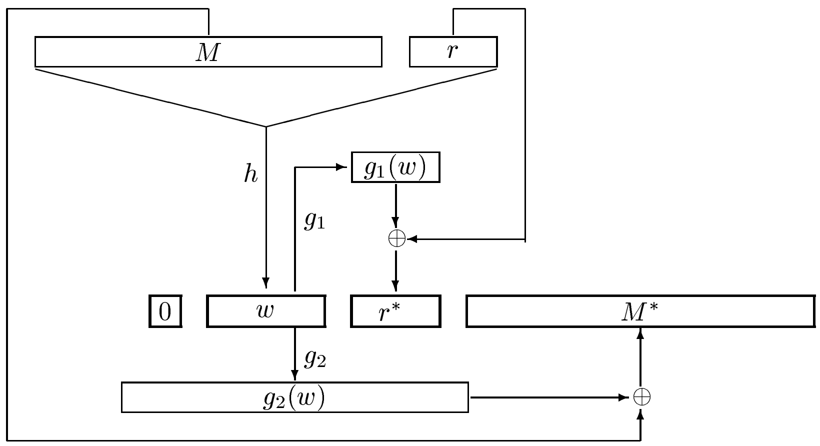

Figure 9.2: PSS-R: Components of image $y = 0\|w\|r^*\|M^*$ are darkened.

and simultaneously check authenticity. Signature generation and verification proceed as follows. Refer to Figure 9.2 for a picture.

Algorithm $SignPSSR^{g,h}_{N,d}(M)$ $r \stackrel{R}{\leftarrow} \{0,1\}^{k_0}$; $w \leftarrow h(M\|r)$ $r^* \leftarrow g_1(w) \oplus r$ $M^* \leftarrow g_2(w) \oplus M$ $y \leftarrow 0\|w\|r^*\|M^*$ $x \leftarrow y^d \bmod N$ Return x	Algorithm $RecPSSR^{g,h}_{N,e}(x)$ $y \leftarrow x^e \bmod N$ Parse y as $b\|w\|r^*\|M^*$ where $\|b\| = 1$, $\|w\| = k_1$, $\|r^*\| = k_0$ $r \leftarrow r^* \oplus g_1(w)$ $M \leftarrow M^* \oplus g_2(w)$ If ($h(M\|r) = w$ and $b = 0$) Then return M else return REJECT

The difference in *SignPSSR* with respect to *SignPSS* is that the last part of y is not $g_2(w)$. Instead, $g_2(w)$ is used to "mask" the message, and the masked message M^* is the last part of the image point y.

The above is easily adapted to handle messages of arbitrary length. A fully-specified scheme would use about $\min\{k,\ n + k_0 + k_1 + 16\}$ bits.

Security. The security of PSS-R is the same as for PSS.

Theorem 9.16 [24] Let $\mathcal{DS}$ be the PSS with recovery scheme with security parameters k_0 and k_1. Then for any $t, q_{\rm sig}, \mu$ and any $q_{\rm hash} \geq 1 + q_{\rm sig}$ we have

$$\mathbf{Adv}^{\rm uf\text{-}cma}_{\mathcal{DS}}(t, q_{\rm sig}, q_{\rm hash}, \mu) \quad \leq \quad \mathbf{Adv}^{\rm owf}_{\mathcal{RSA}}(t') + [3(q_{\rm hash} - 1)^2] \cdot (2^{-k_0} + 2^{-k_1}) ,$$

where $t' = t + q_{\rm hash} \cdot k_0 \cdot O(k^3)$. ▌▌

The proof of this theorem is very similar to that of Theorem 9.15.

9.5.11 How to implement the hash functions

In the PSS we need a concrete hash function h with output length some given number k_1. Typically we will construct h from some cryptographic hash function H such as H = MD5 or H = SHA-1. Ways to do this have been discussed before in [15, 23]. For completeness we quickly summarize some of these possibilities. The simplest is to define $h(x)$ as the appropriate-length prefix of

$$H(\texttt{const}.\langle 0\rangle.x)\|H(\texttt{const}.\langle 1\rangle.x)\|H(\texttt{const}.\langle 2\rangle.x)\|\cdots \ .$$

The constant `const` should be unique to h; to make another hash function, g, simply select a different constant.

9.5.12 Comparison with other schemes

We have already discussed the PKCS standards [173, 174] and the ISO standard [1] and seen that their security cannot be justified based on the assumption that RSA is trapdoor one-way. Other standards, such as [9], are similar to [173], and the same statement applies.

The schemes we discuss in the remainder of this section do not use the hash-then-decrypt paradigm.

Signature schemes whose security can be provably based on the RSA assumption include [101, 14, 145, 171, 74]. The major plus of these works is that they do not use an ideal hash function (random oracle) model— the provable security is in the standard sense. On the other hand, the security reductions are quite loose for each of those schemes. On the efficiency front, the efficiency of the schemes of [101, 14, 145, 171] is too poor to seriously consider them for practice. The Dwork-Naor scheme [74], on the other hand, is computationally quite efficient, taking two to six RSA computations, although there is some storage overhead and the signatures are longer than a single RSA modulus. This scheme is the best current choice if one is willing to allow some extra computation and storage, and one wants well-justified security *without* assuming an ideal hash function.

Back among signature schemes which assume an ideal hash, a great many have been proposed, based on the hardness of factoring or other assumptions. Most of these schemes are derived from identification schemes, as was first done by [79]. Some of these methods are provable (in the ideal hash model), some not. In some of the proven schemes exact security is analyzed; usually it is not. In no case that we know of is the security tight. The efficiency varies. The computational requirements are often lower than a hash-then-decrypt RSA signature, although key sizes are typically larger.

Finally we note related new work. Pointcheval and Stern [159] consider the provable security of signatures in the random oracle model and show that a modified version of the El Gamal scheme [85], as well as the Schnorr [178] scheme, can be proven secure. (And the scheme of [79] can be proven secure against attacks in which there are no signature queries.) But they don't consider exact security. An interesting question is to consider, and possibly improve, the exact security of their reductions (making, if necessary, modifications to the schemes).

More recently, some quite simple RSA based signature schemes have appeared that have a proof of security based on a stronger and less standard assumption about RSA, but which do not rely on random oracles [87, 61].

9.6 Threshold Signature Schemes

Using a threshold signature scheme, digital signatures can be produced by a group of players rather than by one party. In contrast to the regular signature schemes where the signer is a single entity which holds the secret key, in threshold signature schemes the secret key is shared by a group of n players. In order to produce a valid signature on a given message m, individual players produce their *partial signatures* on that message, and then combine them into a full signature on m. A distributed signature scheme achieves threshold $t < n$, if no coalition of t (or less) players can produce a new valid signature, even after the system has produced many signatures on different messages. A signature resulting from a threshold signature scheme is the same as if it was produced by a single signer possessing the full secret signature key. In particular, the validity of this signature can be verified by anyone who has the corresponding unique public verification key. In other words, the fact that the signature was produced in a distributed fashion is transparent to the recipient of the signature.

Threshold signatures are motivated both by the need that arises in some organizations to have a group of employees agree on a given message (or a document) before signing it, as well as by the need to protect signature keys from the attack of internal and external adversaries. The latter becomes increasingly important with the actual deployment of public key systems in practice. The signing power of some entities, (e.g., a government agency, a bank, a certification authority) inevitably invites attackers to try and "steal" this power. The goal of a threshold signature scheme is twofold: To increase the availability of the signing agency, and at the same time to increase the protection against forgery by making it harder for the adversary to learn the secret signature key. Notice that in particular, the threshold approach rules out the naive solution based on traditional secret sharing (see Chapter 11), where the secret key is shared in a group but reconstructed by a *single* player each time that a signature is to be produced. Such protocol would contradict the requirement that no t (or less) players can ever produce a new valid signature. In threshold schemes, multiple signatures are produced without an exposure or an explicit reconstruction of the secret key.

Threshold signatures are part of a general approach known as *threshold cryptography.* This approach has received considerable attention in the literature; we refer the reader to [66] for a survey of the work in this area. Particular examples of solutions to threshold signatures schemes can be found in [65, 177] for RSA and in [105] for ElGamal-type of signatures.

A threshold signature scheme is called robust if not only t or less players cannot produce a valid signature, but also cannot *prevent* the remaining players from computing a signature on their own. A robust scheme basically foils possible denial of service attacks on the part of corrupted servers. The solutions mentined above are *not* robust. In this chapter we will concentrate on robust schemes. We will not go into technical details. The goal of this section is to present the reader with the relevant notions and point to the sources in the literature.

In the following we will refer to the signing servers with the letters $P_1, \ldots, P_n$.

9.6.1 Key Generation for a Threshold Scheme

The task of generating a key for a threshold signature scheme is more complicated than when we are in the presence of a single signer. Indeed we must generate a public key PK whose matching secret key SK is shared in some form among the servers $P_1, \ldots, P_n$.

A way of doing this is to have some trusted *dealer* who generates a key pair (PK, SK) for the given signature scheme, makes PK public and shares SK among the P_i's using a secret sharing protocol (see Chapter 11.) However notice that such a key generation mechanisms contradicts the requirement that no single entity should be able to sign, as now the dealer knows the secret key SK and he is able to sign on his own. This is why people have been trying to avoid the use of such a dealer during the key generation phase.

For the case of discrete-log based signature schemes, this quest has been successful. Robust threshold signature schemes for the El Gamal, Schnorr and DSS signature schemes (see [85, 178, 81]) can be found in [49, 153, 89], all using underlying results of Feldman and Pedersen [78, 154, 155].

Yet, in some cases the dealer solution is the best we can do. For example, if the underlying signature scheme is RSA, then we do not know how to generate a key in a shared form without the use of a dealer.

9.6.2 The Signature Protocol

Once the key is generated and in some way shared among the servers $P_1, \ldots, P_n$ we need a signature protocol.

The idea is that on input a message M, the servers will engage in some form of communication that will allow them to compute a signature σ for M, without revealing the secret key. Such protocol should not leak any information beyond such signature σ. Also in order to obtain the *robustness* property, such protocols should correctly compute the signature even if up to t servers P_i's are corrupted and behave in *any* way during the protocol. If possible the computation required by a server P_i to sign in this distributed manner should be comparable to the effort required if P_i were signing on his own. Interaction should be reduced to a minimum

For El Gamal-like schemes robust threshold signature schemes can be found in [49, 153]. The specific case of DSS turned out to be very difficult to handle. The best solution is in [89].

RSA turned out to be even less amenable to the construction of robust schemes. A somewhat inefficient solution (requires much more computation and a lot of interaction between servers) can be found in [82]. A very efficient and non-interactive solution was independently proposed in [88].

CHAPTER 10

Key distribution

We have looked extensively at encryption and data authentication and seen lots of ways to design schemes for these tasks. We must now address one of the assumptions underlying these schemes. This is the assumption that the parties have available certain kinds of keys.

This chapter examines various methods for key distribution and key management. A good deal of our effort will be expended in understanding the most important practical problem in the area, namely session key distribution.

Let us begin with the classic secret key exchange protocol of Diffie and Hellman.

10.1 Diffie Hellman secret key exchange

Suppose Alice and Bob have no keys (shared or public), and want to come up with a joint key which they would use for private key cryptography. The Diffie-Hellman (DH) secret key exchange (SKE) protocol [68] enables them to do just this.

10.1.1 The protocol

We fix a prime p and a generator $g \in Z_p^*$. These are public, and known not only to all parties but also to the adversary E.

- A picks $x \in Z_{p-1}$ at random and lets $X = g^x \bmod p$. She sends X to B
- B picks $y \in Z_{p-1}$ at random and lets $Y = g^y \bmod p$. He sends Y to A.

Now notice that

$$X^y = (g^x)^y = g^{xy} = (g^y)^x = Y^x ,$$

the operations being in the group Z_p^*. Let's call this common quantity K. The crucial fact is that *both* parties can compute it! Namely A computes Y^x, which is K, and B computes X^y, which is also K, and now they have a shared key.

10.1.2 Security against eavesdropping: The DH problem

Is this secure? Consider an adversary that is sitting on the wire and sees the flows that go by. She wants to compute K. What she sees is X and Y. But she knows neither x nor y. How could she get K? The natural attack is to find either x or y (either will do!) from which she can easily compute K. However, notice that computing x given X is just the discrete logarithm problem in Z_p^*, which is widely believed to be intractable (for suitable choices of the prime p). Similarly for computing y from Y. Accordingly, we would be justified in having some confidence that this attack would fail.

A number of issues now arise. The first is that computing discrete logarithms is not the only possible attack to try to recover K from X, Y. Perhaps there are others. To examine this issue, let us formulate the computational problem the adversary is trying to solve. It is the following:

The DH Problem: Given g^x and g^y for x, y chosen at random from Z_{p-1}, compute g^{xy}.

Thus the question is how hard is this problem? We saw that if the discrete logarithm problem in Z_p^* is easy then so is the DH problem; ie. if we can compute discrete logs we can solve the DH problem. Is the converse true? That is, if we can solve the DH problem, can we compute discrete logarithms? This remains an open question. To date it seems possible that there is some clever approach to solving the DH problem without computing discrete logarithms. However, no such approach has been found. The best known algorithm for the DH problem is to compute the discrete logarithm of either X or Y. This has lead cryptographers to believe that the DH problem, although not known to be equivalent to the discrete logarithm one, is still a computationally hard problem, and that as a result the DH secret key exchange is secure in the sense that a computationally bounded adversary can't compute the key K shared by the parties.

The DH Assumption: The DH problem is computationally intractable.

These days the size of the prime p is recommended to be at least 512 bits and preferably 1024. As we have already seen, in order to make sure the discrete logarithm problem modulo p is intractable, $p-1$ should have at least one large factor. In practice we often take $p = 2q + 1$ for some large prime q.

The relationship between the DH problem and the discrete logarithm problem is the subject of much investigation. See for example Maurer [135].

10.1.3 The DH cryptosystem

The DH secret key exchange gives rise to a very convenient public key cryptosystem. A party A will choose as its secret key a random point $x \in Z_{p-1}$, and let $X = g^x$ be its public key. Now if party B wants to privately send A a message M, it would proceed as follows.

First, the parties agree on a private key cryptosystem $(\mathcal{E}, \mathcal{D})$ (cf. Chapter 6). For concreteness assume it is a DES based cryptosystem, so that it needs a 56 bit key. Now B picks y at random from Z_{p-1} and computes the DH key $K = X^y = g^{xy}$. From this, he extracts a 56 bit key a for the private key cryptosystem according to some fixed convention, for example by letting a be the first 56 bits of K. He now encrypts the plaintext M under a using the private key cryptosystem to get the ciphertext $C = \mathcal{E}_a(M)$, and transmits the pair (Y, C) where $Y = g^y$.

A receives (Y, C). Using her secret key x she can compute the DH key $K = Y^x = g^{xy}$, and thus recover a. Now she can decrypt the ciphertext C according to the private key cryptosystem, via $M = \mathcal{D}_a(C)$, and thus recover the plaintext M.

Intuitively, the security would lie in the fact that the adversary is unable to compute K and hence a. This, however, is not quite right, and brings us to the issue of the bit security of the DH key.

10.1.4 Bit security of the DH key

Above the first 56 bits of the key $K = g^{xy}$ is used as the key to a private key cryptosystem. What we know (are willing to assume) is that given g^x, g^y the adversary cannot recover K. This is not enough to make the usage of K as the key to the private key cryptosystem secure. What if the adversary were able to recover the first 56 bits of K, but not all of K? Then certainly the above cryptosystem would be insecure. Yet, having the first 56 bits of K may not enable one to find K, so that we have not contradicted the DH assumption.

This is an issue we have seen before in many contexts, for example with one-way functions and with encryption. It is the problem of partial information. If f is one-way it means given $f(x)$ I can't find x; it doesn't mean I can't find some bits of x. Similarly, here, that we can't compute K doesn't mean we can't compute some bits of K.

Indeed, it turns out that computing the last bit (ie. LSB) of $K = g^{xy}$ given g^x, g^y is easy. To date there do not seem to be other detectable losses of partial information. Nonetheless it would be unwise to just use some subset of bits of the DH key as the key to a private key cryptosystem. Assuming that these bits are secure is a much stronger assumption than the DH assumption.

So what could we do? In practice, we might hash the DH key K to get a symmetric key a. For example, applying a cryptographic hash function like SHA-1 to K yields 160 bits that might have better "randomness" properties than the DH key. Now use the first 56 bits of this if you need a DES key.

However, while the above may be a good heuristic in practice, it can't be validated without very strong assumptions on the randomness properties of the hash function. One possibility that can be validated is to extract hard bits from the DH key via an analogue of Theorem 2.49. Namely, let r be a random string of length $|p|$ and let b be the dot product of K and r. Then predicting b given g^x, g^y is infeasible if computing $K = g^{xy}$ given g^x, g^y is infeasible. The drawback of this approach is that one gets very few bits. To get 56 bits one would need to exchange several DH keys and get a few bits from each.

We saw in Chapter 2 that for certain one way functions we can present hardcore predicates, the prediction of which can be reduced to the problem of inverting the function itself. A theorem like that for the DH key would be nice, and would indicate how to extract bits to use for a symmetric key. Recently results of this kind have been proved by Boneh and Venkatesan [42].

10.1.5 The lack of authenticity

At first glance, the DH secret key exchange might appear to solve in one stroke the entire problem of getting keys to do cryptography. If A wants to share a key with B, they can just do a DH key exchange to get one, and then use private key cryptography.

Don't do it. The problem is *authenticity*. The security of the DH key is against a *passive* adversary, or *eavesdropper*. It is assumed that the adversary will recover the transmitted data but not try to inject data on the line. In practice, of course, this is an untenable assumption. It is quite easy to inject messages on networks, and hackers can mount active attacks.

What damage can this do? Here is what the adversary does. She calls up B and simply plays the role of A. That is, she claims to be A, who is someone with whom B would like to share a key, and then executes the DH protocol like A would. Namely she picks x at random and sends $X = g^x$ to B. B returns $Y = g^y$ and now B and the adversary share the key $K = g^{xy}$. But B thinks the key is shared with A. He might encrypt confidential data using K, and then the adversary would recover this data.

Thus in the realistic model of an active adversary, the DH key exchange is of no direct use. The real problem is to exchange a key in an authenticated manner. It is this that we now turn to.

However, we remark that while the DH key exchange is not a solution, by itself, to the key distribution problem in the presence of an active adversary, it is a useful tool. We will see how to use it in conjunction with other tools we will develop to add to session key distribution protocols nice features like "forward secrecy."

10.2 Session key distribution

Assume now we are in the presence of an active adversary. The adversary can inject messages on the line and alter messages sent by legitimate parties, in addition to eavesdropping on their communications. We want to get shared keys.

A little thought will make it clear that if the legitimate parties have no information the adversary does not know, it will not be possible for them to exchange a key the adversary does not know. This is because the adversary can just impersonate one party to another, like in the attack on DH above. Thus, in order to get off the ground, the legitimate parties need an "information advantage." This is some information, pre-distributed over a trusted channel, which the adversary does not know, and which enables them to securely exchange keys in the future.

We now discuss various ways to realize this information advantage, and the session key distribution problems to which they give rise. Then we explain the problem in more depth. We largely follow [16, 17].

10.2.1 Trust models and key distribution problems

What forms might the information advantage take? There are various different *trust models* and corresponding key distribution problems.

The three party model

This model seems to have been first mentioned by Needham and Schroeder [148]. It has since been popularized largely by the *Kerberos* system [193].

In this model there is a trusted party called the *authentication server*, and denoted S. Each party A in the system has a key K_A which it shares with the server. This is a private key between these two parties, not known to any other party. When two parties A, B, sharing, respectively, keys K_A and K_B with S, want to engage in a communication session, a three party protocol will be executed, involving A, B and S. The result will be to issue a common key K to A and B. They can then use this key to encrypt or authenticate the data they send each other.

The distributed key is supposed to be a secure session key. When the parties have completed their communication session, they will discard the key K. If later they should desire another communication session, the three party protocol will be re-executed to get a new, fresh session key.

What kinds of security properties should this distributed session key have? We will look at this question in depth later. It is an important issue, since, as we will see, session key distribution protocols must resist a variety of novel attacks.

The two party asymmetric model

When public key cryptography can be used, the authentication server's active role can be eliminated. In this trust model, the assumption is that A has the public key pk_B of B, and B has the public key pk_A of A. These keys are assumed authentic. That is, A is assured the key he holds is really the public key of B and not someone else, and analogously for B.[1]

Now, suppose A and B want to engage in a secure communication session. The problem we want to consider is how they can get a shared, private and authentic session key based on the public keys, via a two party protocol.

[1] How is this situation arrived at? That isn't a problem we really want to discuss yet: it falls under the issue of key management and will be discussed later. But, briefly, what we will have is trusted servers which provide public, certified directories of users and their public keys. The server maintains for each user identity a public key, and provides this upon request to any other user, with a signature of the server that serves as a certificate of authenticity. Barring this directory service, however, the server plays no active role.

Questions pertaining to what exactly is the problem, and why, may arise here. We already know that we can authenticate and encrypt data with public keys. That is, the parties already have the means to secure communication. So why do they need a shared session key?

There are several reasons. One is that private key cryptography, at least under current technology, is considerable more efficient than public key cryptography. The second, however, probably more important, is that it is convenient to have *session* keys. They allow one to associate a key uniquely to a session. This is an advantage for the following reasons.

Keys actually used to encrypt or authenticate data get greater exposure. They are used by applications in ways that may not be known, or controllable, beforehand. In particular, an application might mis-use a key, or expose it. This might (or might not) compromise the current session, but we would not like it to compromise the long lived secret key and thus other uses and sessions. Similarly, the long lived secret key of a user A (namely the secret key sk_A corresponding to her public key pk_A) may be stored in protected hardware and accessed only via a special interface, while the session key lies on a more exposed machine.

The two party symmetric model

Probably the simplest model is of two parties who already share a long lived key. Each time they wish to engage in a communication session they will run a protocol to derive a session key.

Again, the motivation is the convenience and security advantages of session keys. We stress the main one. A host of applications might be run by the users, all wanting keys for one reason or another. We don't want to make assumptions about how they use the key. Some might use it in ways that are secure for their own purposes but compromise the key globally. In order for this not to affect the global security, we assign each run of each application a separate session key.

10.2.2 History of session key distribution

Although session key distribution is an old problem, it is only recently that a cryptographically sound treatment of it, in the "provable security" or "reductionist" tradition that these lecture notes are describing, has emerged [16, 17]. Via this approach we now have models in which to discuss and prove correct protocols, and several protocols proven secure under standard cryptographic assumptions.

The history prior to this was troubled. Session key distribution is an area in which a large number of papers are published, proposing protocols to solve the problem. However, many of them are later broken, or suffer from discernible design flaws.

The problem is deceptively simple. It is easy to propose protocols in which subtle security problems later emerge.

In the three party case, Needham and Schroeder [148] describe a number of candidate protocols. They had prophetically ended their paper with a warning on this approach, saying that "protocols such as those developed here are prone to extremely subtle errors that are unlikely to be detected in normal operations. The need for techniques to verify the correctness of such protocols is great . . .". Evidence of the authors' claim came unexpectedly when a bug was pointed out in their own "Protocol 1" (Denning and Sacco, [64]).[2] Many related protocols were eventually to suffer the same fate.

As a result of a long history of such attacks there is finally a general consensus that session key distribution is not a goal adequately addressed by giving a protocol for which the authors can find no attacks.

A large body of work, beginning with Burrows, Abadi and Needham [45], aims to improve on this situation via the use of special-purpose logics. The aim is to demonstrate a lack of "reasoning problems" in a protocol being analyzed. The technique has helped to find errors in various protocols, but a proof that a protocol is "logically correct" does not imply that it is is right (once its abstract cryptographic operations are instantiated). Indeed

[2] Insofar as there were no formal statements of what this protocol was supposed to do, it is not entirely fair to call it buggy; but the authors themselves regarded the protocol as having a problem worthy of fixing [149].

it is easy to come up with concrete protocols which are logically correct but blatantly insecure.

Examining the work on the session key distribution problem, one finds that the bulk of it is divorced from basic cryptographic principles. For example one find over and over again a confusion between data encryption and data authentication. The most prevalent problem is a lack of specification of what exactly is the problem that one is trying to solve. There is no *model* of adversarial capabilities, or definition of security.

Influential works in this area were Bird et. al. [31] and Diffie et. al. [69]. In particular the former pointed to new classes of attacks, called "interleaving attacks," which they used to break existing protocols, and they suggested a protocol (2PP) defeated by none of the interleaving attacks they considered. Building on this, Bellare and Rogaway provide a model and a definition of security for two party symmetric session key distribution [16] and for three party session key distribution [17], just like we have for primitives like encryption and signatures. Based on this they derive protocols whose security can be proven based on standard cryptographic assumptions. It turns out the protocols are efficient too.

Now other well justified protocols are also emerging. For example, the SKEME protocol of Krawczyk [121] is an elegant and multi-purpose two party session key distribution protocol directed at fulfilling the key distribution needs of Internet security protocols. Even more recently, a proven-secure protocol for session key distribution in smart cards was developed by Shoup and Rubin [189].

10.2.3 An informal description of the problem

We normally think of a party in a protocol as being devoted to that protocol alone; it is not doing doing other things alongside. The main element of novelty in session key distribution is that parties may simultaneously maintain multiple sessions. A party has multiple *instances*. It is these instances that are the logical endpoints of a session, not the party itself.

We let $\{P_1, \ldots, P_N\}$ denote the parties in the distributed system. As discussed above, a given pair of players, P_i and P_j may simultaneously maintain multiple sessions (each with its own session key). Thus it is not really P_i and P_j which form the logical endpoints of a secure session; instead, it is an *instance* $\Pi^s_{i,j}$ of P_i and an *instance* $\Pi^t_{j,i}$ of P_j. We emphasize instances as a central aspect of the session key distribution problem, and one of the things that makes session key distribution different from many other problems.

It is the goal of a *session-key distribution protocol* to provide $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ with a session key $\sigma^{s,t}_{i,j}$ to protect their session. Instances $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ must come up with this key without knowledge of s, t, or whatever other instances may currently exist in the distributed system.

An active adversary attacks the network. She controls all the communication among the players: she can deliver messages out of order and to unintended recipients, concoct messages entirely of her own choosing, and start up entirely new instances of players. Furthermore, she can mount various attacks on session keys which we will discuss.

10.2.4 Issues in security

Ultimately, what we want to say is that the adversary cannot compromise a session key exchanged between a pair of instances of the legitimate parties. We must worry about two (related) issues: authentication, and key secrecy. The first means, roughly, that when an instance of i accepts B then it must have been "talking to" an instance of j. The second, roughly, means that if $\Pi^s_{i,j}$ and $\Pi^t_{j,i}$ share a session key then this key must be secure.

It is an important requirement on session keys that the key of one session be independent of another. This is because we cannot make assumptions about how a session key will be used in an application. It might end up exposing it, and we want this not to compromise other session keys. We model this in a worst case way by allowing the adversary to expose session keys at will. Then we will say that a key shared between partners who are unexposed must remain secure even if keys of other sessions are exposed.

One of the most important issues is what is meant by security of the key. The way it has traditionally been viewed is that the key is secure if the adversary cannot compute it. We have by now, however, seen time and again, in a variety of settings, that this is not the right notion of secrecy. We must also prevent partial information from leaking. (Examples of why this is important for session keys are easy to find, analogous to the many examples we have seen previously illustrating this issue.) Accordingly, the definitions ask that a session key be unpredictable in the sense of a probabilistically encrypted message.

We note that insufficient protection of the session key is a flaw that is present in all session key distribution protocols of which we are aware barring those of [16, 17]. In fact, this insecurity is often built in by a desire to have a property that is called "key confirmation." In order to "confirm" that it has received the session key, one party might encrypt a fixed message with it, and its ability to do so correctly is read by the other party as evidence that it has the right session key. But this reveals partial information about the key. It might seem un-important, but one can find examples of usages of the session key which are rendered insecure by this kind of key confirmation. In fact "key confirmation," if needed at all, can be achieved in other ways.

10.2.5 Entity authentication versus key distribution

The goal of the key distributions we are considering is for the parties to simultaneously authenticate one another and come into possession of a secure, shared session key. There are several ways one might interpret the notion of authentication.

The literature has considered two ways. The first is authentication in a very strong sense, considered in [16] for the two party case. This has been relaxed to a weaker notion in [17], considered for the three party case. The weaker notion for the two party case is still under research and development.

Which to prefer depends on the setting. The approach we will follow here is to follow the existing literature. Namely we will consider the stronger notion for the two party setting, and the weaker one for the three party setting. It may perhaps be more correct to use the weaker notion throughout, and in a future version of these notes we would hope to do so; the situation at present is simply that the formalizations of the weaker notion for the two party case have not yet appeared.

10.3 Authenticated key exchanges

We begin by looking at the two party case, both symmetric and asymmetric. We look at providing authentic exchange of a session key, meaning the parties want to authenticate one another and simultaneously come into possession of a shared secret session key. The formal model, and the definition of what constitutes a secure authenticated session key distribution protocol, are provided in [16]. Here we will only describe some protocols.

First however let us note some conventions. We assume the parties want to come into possession of a l-bit, random shared secret key, eg. $l = 56$. (More generally we could distribute a key from some arbitrary samplable distribution, but for simplicity let's stick to what is after all the most common case.) The session key will be denoted by α.

Whenever a party A sends a flow to another party B, it is understood that her identity A accompanies the flow, so that B knows who the flow purports to come from. (This has nothing to do with cryptography or security: it is just a service of the communication medium. Note this identity is not secured: the adversary can change it. If the parties want the claim secured, it is their responsibility to use cryptography to this end, and will see how they do this.)

10.3.1 The symmetric case

Let K be the (long-lived) key shared between the parties. We fix a private key encryption scheme $(\mathcal{E}, \mathcal{D})$ and a private key message authentication scheme $(\mathcal{T}, \mathcal{V})$. The key K is divided into two parts, K^e and K^m, the first to be used for encryption and the second for message authentication. The protocol, called

Authenticated Key Exchange Protocol 1, is depicted in Figure 10.1, and a more complete description of the flows follows.

$$A \xrightarrow{\quad R_A \quad} B$$

$$\xleftarrow{\quad R_B \| \mathcal{E}_{K^e}(\alpha) \| \mathcal{T}_{K^m}(B\|A\|R_A\|R_B\|\mathcal{E}_{K^e}(\alpha)) \quad}$$

$$\xrightarrow{\quad \mathcal{T}_{K^m}(A\|R_B) \quad}$$

Figure 10.1: Protocol AKEP1: Session key distribution in symmetric setting.

Here is a more complete description of the flows:

(1) A picks at random a string R_A and sends it to B

(2) B picks at random a string R_B. She also picks at random an l-bit session key α. She encrypts it under K^e to produce the ciphertext $C = \mathcal{E}_{K^e}(\alpha)$. She now computes the tag $\mu = \mathcal{T}_{K^m}(B\|A\|R_A\|R_B\|C)$. She sends R_B, C, μ to A.

(3) A verifies that $\mathcal{V}_{K^m}(B\|A\|R_A\|R_B\|C, \mu) = 1$. If this is the case she computes the tag $\mathcal{T}_{K^m}(A\|R_B)$ and sends it to B. She also decrypts C via $\alpha = \mathcal{D}_{K^e}(C)$ to recover the session key.

(4) B verifies the last tag and accepts (outputting session key α) if the last tag was valid.

Remark 10.1 Notice that both encryption and message authentication are used. As we mentioned above, one of the commonly found fallacies in session key distribution protocols is to try to use encryption to provide authentication. One should really use a message authentication code. ▮

Remark 10.2 It is important that the encryption scheme $(\mathcal{E}, \mathcal{D})$ used above be secure in the sense we have discussed in Chapter 6. Recall in particular this means it is probabilistic. A single plaintext has many possible ciphertexts, depending on the probabilistic choices made by the encryption algorithms. These probabilistic choices are made by S when the latter encrypts the session key, independently for the two encryptions it performs. This is a crucial element in the security of the session key. ▮

These remarks apply also to the protocols that follow, appropriately modified, of course, to reflect a change in setting. We will not repeat the remarks.

10.3.2 The asymmetric case

We will be using public key cryptography. Specifically, we will be using both public key encryption and digital signatures.

Fix a public key encryption scheme, and let $\mathcal{E}, \mathcal{D}$ denote, respectively, the encryption and the decryption algorithms for this scheme. The former takes a public encryption key pk^e and message to return a ciphertext, and the latter takes the secret decryption key sk^e and ciphertext to return the plaintext. This scheme should be secure in the sense we have discussed in Chapter 7.

Fix a digital signature scheme, and let $\mathcal{S}, \mathcal{V}$ denote, respectively, the signature and verification algorithms for this scheme. The former takes a secret signing key sk^d and message to return a signature, and the latter takes the public verification key pk^d, message, and candidate signature to return an indication of whether or not the signature is valid. This scheme should be secure in the sense we have discussed in Chapter 9.

Every user I in the system has a public key pk_I which is in fact a pair of public keys, $pk_I = (pk_I^e, pk_I^d)$, one for the encryption scheme and the other for the signature scheme. These keys are known to all other users and the adversary. However, the user keeps privately the corresponding secret keys. Namely he holds $sk_I = (sk_I^e, sk_I^d)$ and nobody else knows these keys.

Recall the model is that A has B's public key pk_B and B has A's public key pk_A. The protocol for the parties to get a joint, shared secret key α is depicted in Figure 10.2, and a more complete explanation follows.

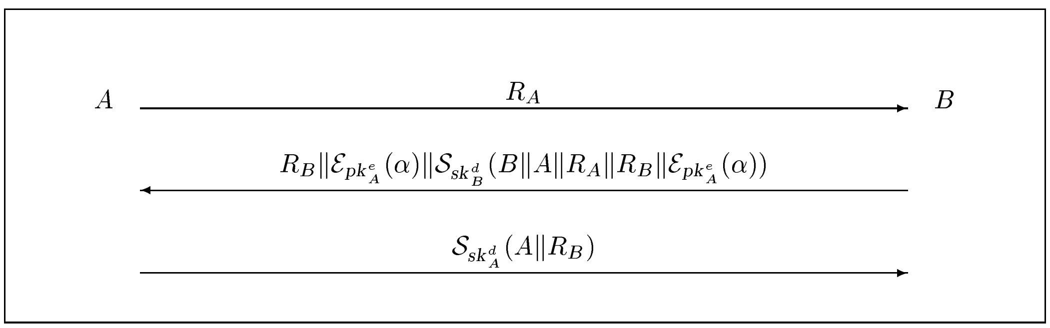

Figure 10.2: Protocol for exchange of symmetric key in asymmetric setting.

Here is a more complete description of the flows:

(1) A picks at random a string R_A and sends it to B

(2) B picks at random a string R_B. She also picks at random an l-bit session key α. She encrypts it under A's public key pk_A^e to produce the ciphertext $C = \mathcal{E}_{pk_A^e}(\alpha)$. She now computes the signature $\mu = \mathcal{S}_{sk_B^d}(A \| R_A \| R_B \| C)$, under her secret signing key sk_B^d. She sends R_B, C, μ to A.

(3) A verifies that $\mathcal{V}_{pk_B^d}(A \| R_A \| R_B \| C, \mu) = 1$. If this is the case she computes the signature $\mathcal{S}_{sk_A^d}(R_B)$ and sends it to B. She also decrypts C via $\alpha = \mathcal{D}_{sk_A^e}(C)$ to recover the session key.

(4) B verifies the last signature and accepts (outputting session key α) if the last signature was valid.

10.4 Three party session key distribution

Fix a private key encryption scheme $(\mathcal{E}, \mathcal{D})$ which is secure in the sense discussed in Chapter 6. Also fix a message authentication scheme $(\mathcal{T}, \mathcal{V})$ which is secure in the sense discussed in Chapter 8. The key K_I shared between the server S and party I is a pair (K_I^e, K_I^m) of keys, one a key for the encryption scheme and the other a key for the message authentication scheme. We now consider parties A, B, whose keys K_A and K_B, respectively have this form. A terse representation of the protocol of [17] is is given in Figure 10.3, and a more complete explanation follows.

Here now is a more complete description of the flows and accompanying computations:

(1) In Step 1, party A chooses a random challenge R_A and sends it to B.

Flow 1.	$A \to B$:	R_A
Flow 2.	$B \to S$:	$R_A \| R_B$
Flow 3A.	$S \to A$:	$\mathcal{E}_{K_A^e}(\alpha) \;\|\; \mathcal{T}_{K_A^m}(A\|B\|R_A\|\mathcal{E}_{K_A^e}(\alpha))$
Flow 3B.	$S \to B$:	$\mathcal{E}_{K_B^e}(\alpha) \;\|\; \mathcal{T}_{K_B^m}(A\|B\|R_B\|\mathcal{E}_{K_B^e}(\alpha))$

Figure 10.3: Three party session key distribution protocol.

(2) In Step 2, party B chooses a random challenge R_B and sends $R_A \| R_B$ to S.

(3) In Step 3, S picks a random l-bit session key α which he will distribute. Then S encrypts this session key under each of the parties' shared keys. Namely he computes the ciphertexts $\alpha_A = \mathcal{E}_{K_A^e}(\alpha)$ and $\alpha_B = \mathcal{E}_{K_B^e}(\alpha)$. Then S computes $\mu_A = \mathcal{T}_{K_A^m}(A\|B\|R_A\|\alpha_A)$ and $\mu_B = \mathcal{T}_{K_B^m}(A\|B\|R_B\|\alpha_B)$. In flow 3A (resp. 3B) S sends A (resp. B) the message $\alpha_A \| \mu_A$ (resp. $\alpha_B \| \mu_B$).

(4) In Step 4A (resp. 4B) Party A (resp. B) receives a message $\alpha'_A \| \mu'_A$ (resp. $\alpha'_B \| \mu'_B$) and accepts, with session key $\mathcal{D}_{K_A^e}(\alpha'_A)$ (resp. $\mathcal{D}_{K_B^e}(\alpha'_B)$), if and only if $\mathcal{V}_{K_A^m}(A\|B\|R_A\|\alpha'_A, \mu'_A) = 1$ (resp. $\mathcal{V}_{K_B^m}(A\|B\| R_B\|\alpha'_B, \mu'_A) = 1$).

Remark 10.3 This protocol has four flows. Typically, the three party key distribution protocols you will see in the literature have five. In fact, four suffices. ∎

10.5 Forward secrecy

Forward secrecy is an extra security property that a session key can have and which seems very desirable.

Consider, for concreteness, the protocol of Figure 10.2 for exchange of a symmetric key in the asymmetric setting. Suppose A and B have run this protocol and exchanged a session key α, and used it to encrypt data. Suppose the adversary recorded the transcript of this exchange. This means she has in her possession $C = \mathcal{E}_{pk_A^e}(\alpha)$, the encrypted session key, and also any ciphertexts encrypted under α that the parties may have transmitted, call them $C_1, C_2, \ldots$. Since the session key distribution protocol is secure, the information she has doesn't give her anything; certainly she does not learn the session key α.

Now that session is over. But now suppose, for some reason, the *long lived* key of A is exposed. Meaning the adversary, somehow, gets hold of $sk_A = (sk_A^e, sk_A^d)$.

Certainly, the adversary can compromise all future sessions of A. Yet in practice we would expect that A would soon realize her secret information is lost and revoke her public key $pk_A = (pk_A^e, pk_A^d)$ to mitigate the damage. However, there is another issue. The adversary now has sk^e and can decrypt the ciphertext C to get α. Using this, she can decrypt $C_1, C_2, \ldots$ and thereby read the confidential data that the parties sent in the past session.

This does not contradict the security of the basic session key distribution protocol which assumed that the adversary does not gain access to the long-lived keys. But we might ask for a new and stronger property. Namely that even if the adversary got the long-lived keys, at least *past* sessions would not be compromised. This is called *forward secrecy.*

Forward secrecy can be accomplished via the Diffie-Hellman key exchange with which we began this chapter. Let us give a protocol. We do so in the asymmetric, two party setting; analogous protocols can be given in the other settings. The protocol we give is an extension of the STS protocol of [69]. It is depicted in Figure 10.4 and a more complete explanation follows.

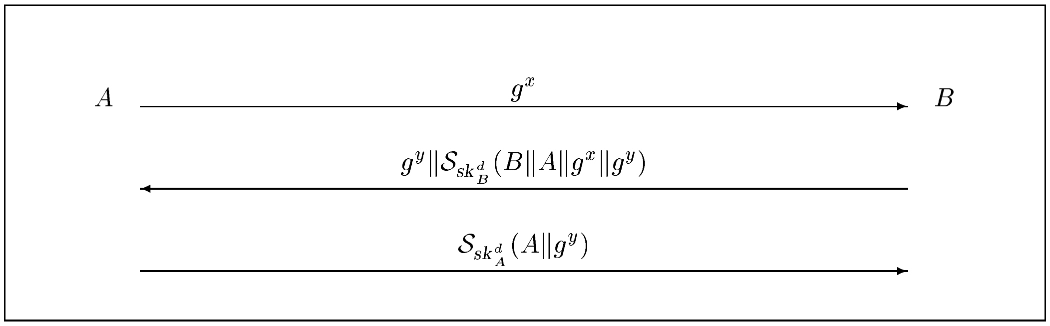

Figure 10.4: Protocol for exchange of symmetric key with forward secrecy.

Here is a more complete description of the flows:

(1) A picks at random a string x, computes $X = g^x$, and sends it to B

(2) B picks at random a string y and lets $Y = g^y$. She now computes the signature $\mu = \mathcal{S}_{sk_B^d}(A\|X\|Y)$, under her secret signing key sk_B^d. She sends Y, μ to A.

(3) A verifies that $\mathcal{V}_{pk_B^d}(A\|X\|Y, \mu) = 1$. If this is the case she computes the signature $\mathcal{S}_{sk_A^d}(Y)$ and sends it to B. She also decrypts outputs the DH key $g^{xy} = Y^x$ as the session key.

(4) B verifies the last signature and accepts (outputting session key $g^{xy} = X^y$) if the last signature was valid.

The use of the DH secret key exchange protocol here is intriguing. Is that the *only* way to get forward secrecy? It turns out it is. Bellare and Rogaway have noted that secret key exchange is not only sufficient but also necessary for the forward secrecy property [22].

As we noted in Section 10.1.4, the DH key is not by itself a good key because we cannot guarantee bit security. Accordingly, the session key in the above should actually be set to, say, $H(g^{xy})$ rather than g^{xy} itself, for a "good" hash function H.

Chapter 11

Protocols

Classical cryptography is concerned with the problem of security communication between users by providing privacy and authenticity. The need for an underlying infrastructure for key management leads naturally into the topic of key distribution. For many years this is all there was to cryptography.

One of the major contributions of modern cryptography has been the development of advanced *protocols*. These protocols enable users to electronically solve many real world problems, play games, and accomplish all kinds of intriguing and very general distributed tasks. Amongst these are zero-knowledge proofs, secure distributed computing, and voting protocols. The goal of this chapter is to give a brief introduction to this area.

11.1 Some two party protocols

We make reference to some number theoretic facts in Section C.6.

11.1.1 Oblivious transfer

This protocol was invented by M. Rabin [166].

An *oblivious transfer* is an unusual protocol wherein Alice transfers a secret bit m to Alice in such a way that the bit is transferred to Bob with probability 1/2;Bob knows when he gets the bit, but Alice doesn't know whether it was transferred or not.

This strange-sounding protocol has a number of useful applications (see, for example [166, 29]). In fact, Kilian has shown [119] that the ability to perform oblivious transfers is a sufficiently strong primitive to enable *any* two-party protocol to be performed.

The following implementation for oblivious transfer has been proposed in the literature (related ideas due to Rabin and Blum.)

(1) Alice picks two primes p, q at random and multiplies them to produce the modulus $N = pq$. She encrypts the message m under this modulus in some standard way, having the property that if you know p, q then you can decrypt, else you can't. She sends N and the ciphertext C to Bob.

(2) Bob picks $a \in Z_N^*$ at random and sends $w = a^2 \bmod N$ to Alice.

(3) Alice computes the four square roots $x, -x, y, -y$ of w, picks one at random and sends it back to Bob

(4) If Bob got back the root which is not $\pm a$ he can factor N and recover m. Else he can't.

And Alice doesn't know which happened since a was random.

It is fairly clear that there is no way for A to cheat in this protocol, since A does not know which square root of z B knows, as x was chosen at random. On first sight it looks like B cannot get anything either, since he only obtains a square root of a random square. However, a formal proof of this fact is not known. It is not clear whether B can cheat or not. For example, if B chooses a particular value of z instead of choosing x at random and setting $z = x^2 \pmod{n}$, then this may lead to an advantage in factoring n. It is conceivable, for example, that knowing a square root of (n-1)/2 mod n (or some other special value) could allow B to factor n. Thus condition ii) is satisfied, but we can't prove whether or not the first condition is satisfied.

If we had a method by which B could **prove** to A that he indeed followed the protocol and choose x at random without revealing what x is, the protocol could be modified to provably work. We will see in a later section on zero-knowledge proofs on how such proofs can be done.

There is another form of OT called 1 out of 2 OT. Here Alice has two secrets, m_0 and m_1. Bob has a selection bit c. At the end of the protocol, Bob gets b_c and Alice still does not know c. See [77].

11.1.2 Simultaneous contract signing

Alice and Bob want to sign the contract, but only if the other person does as well. That is, neither wants to be left in the position of being the only one who signs. Thus, if Alice signs first, she is worried Bob will then not sign, and vice versa. (Maybe easier to think of having two contracts, the first promising something to Alice, the second to Bob. It is a trade. Obviously, each wants the other one to sign.) This problem was proposed in [77].

One approach is that Alice signs the first letter of her name and sends the contract to Bob. He does likewise, and sends it back. And so on. Assume their names have the same length. Then this makes some progress towards a solution. Of course the problem is the person who must go last can stop. But you can make this a negligible difference. For example, not a letter at a time, but a few millimeters of the letter at a time. No party is ever much ahead of the other. If at some point they both stop, they both are at about the same point.

Electronically, we are exchanging strings, which are digital signatures of the contract. Alice has signed it to produce σ_A and Bob has signed it to produce σ_B. Now they exchange these strings a bit at a time, each time sending one more bit.

There is a problem with this. What if one person does not send the signature, but just some garbage string? The other will not know until the end. Even, Goldreich and Lempel [77] show how oblivious transfer can be used to fix this.

Alice creates L_A which is the signature of the contract together with the phrase "this is my signature of the left half of the contract." Similarly she creates R_A which is the signature of the contract together with the phrase "this is my signature of the right half of the contract." Similarly, Bob creates L_B and R_B.

Also Alice picks two DES keys, K_A^L and K_A^R, and encrypts L, R respectively to produce C_A^L and C_A^R. Similarly for Bob, replacing As by Bs.

The contract is considered signed if you have both halves of the other person's signature.

All the ciphertexts are sent to the other party.

Alice 1 out of two OTs (K_A^L, K_A^R) to Bob with the latter choosing a random selection bit, and vice versa. Say Bob gets K_A^L and Alice gets K_B^R.

Alice and Bob send each the first bits of both DES keys. Keep repeating until all bits of all keys are sent. In this phase, if a party catches a mistake in the bits corresponding to the key it already has, it aborts, else it continues.

11.1.3 Bit Commitment

Bob wants Alice to commit to some value, say a bid, so that she can't change this at a later time as a function of other things. On the other hand, Alice does not want Bob to know, at this time, what is the value she is committing to, but will open it up later, at the right time.

Alice makes up an "electronic safe." She has a key to it. She puts the value in the safe and sends the safe to Bob. The latter can't open it to extract the contents. This is a committal. Later, Alice will decommit by sending the key. Now Bob can open it. What must be true is that Alice can't produce a safe having two keys, such that either open it, and when you look inside, you see different values.

One way to implement this is via collision-free hashing. To commit to x Alice sends $yH(x)$. From this Bob can't figure out x, since H is one-way. To decommit, Alice sends x and Bob checks that $H(x) = y$. But Alice can't find $x' \neq x$ such that $H(x') = y$, so can't cheat.

This however has poor bit security. You can fix it with hardcore bits.

Another way is to use quadratic residues. First, we fix a particular number $y \in Z_N^*$ which is known to be a non-residue. Commit to a 0 by sending a random square mod N, namely x^2, and to a 1 by sending a random non-square mod N, in the form yx^2. The QRA says Bob can't tell which is which. To decommit, reveal x in either case.

Notice the QR commitment scheme is secure even against a sender who has unbounded computing power. But not the receiver.

Can you do the opposite? Yes, use discrete logarithms. Let p be a known prime, $g \in Z_p^*$ a known generator of Z_p^*, and $s \in Z_p^*$ a known element of unknown discrete logarithm, namely $\log_g(s)$ is not known. Commit to 0 by picking x at random and sending $y = g^x$; to a 1 by sending sg^x. Notice that to the receiver, each is a random element of the range. But what if the sender could create a y that could be opened both ways? It would have the discrete logarithm of s.

Commitment schemes are useful for lots of things. In particular, ZK proofs, but also coin flipping.

11.1.4 Coin flipping in a well

Blum [36] has proposed the problem of *coin flipping over the telephone*. Alice and Bob want a fair, common, coin. They want to take a random choice, but neither should be able to dictate it. Heads Alice wins, and tails Bob wins.

What if Bob says, "I'll flip the coin and send you the value." No good. Bob will just flip to win. They must both influence the value.

Here is a thought. Alice picks a random bit a and sends it to Bob, and Bob picks a random bit b and send it to Alice, and the value of the coin is $a \oplus b$. The problem is who goes first. If Alice goes first, Bob will choose b to make the coin whatever he wants. Not fair.

So what Alice does is first commit to her coin. She sends $y = \text{Committ}(a)$ to Bob. Now Bob can't make b a function of a. He sends back b, in the clear. Alice may want to make a a function of b, but it is too late since a is committed to. She decommits, and the coin is $a \oplus b$.

11.1.5 Oblivious circuit evaluation

Alice and Bob want to know which of them is older. But neither wants to reveal their age. (Which means they also don't want the reveal the age difference, since from this and their own age, each gets the other's age too!) They just want a single bit to pop out, pointing to the older one.

Sometimes called the Millionaires problem, with the values being the earning of each millionaire.

In general, the problem is that Alice has an input x_A and Bob has an input x_B and they want to compute $f(x_A, x_B)$ where f is some known function, for example $f(x_A, x_B) = 1$ if $x_A \geq x_B$ and 0 otherwise. They want to compute it obliviously, so that at the end of the game they both have the value $v = f(x_A, x_B)$ but neither knows anything else.

There are protocols for this task, and they are quite complex. We refer the reader to [10, 43]

11.1.6 Simultaneous Secret Exchange Protocol

This has been studied in [37, 198, 132, 203].

The protocol given here is an example of a protocol that seems to work at first glance, but is in actuality open to cheating for similar reasons that the above oblivious transfer protocol was open to cheating. The common input consists of $1^k, \alpha \in E_{n_A}(s_A), \beta \in E_{n_B}(s_B), n_A$, and n_B, where n_A and n_B are each products of two equal size primes congruent to 3 mod 4; E_{n_A} (E_{n_B}) are the same encryption as in the oblivious transfer protocol above with respect to n_A (n_B respectively). A's private input has in it the prime factorization $n_A = p_A q_A$ of n_A and B's contains the same for n_B. What we want is for A and B to be able to figure out s_B and s_A at the "same time". We assume equal computing power and knowledge of algorithms. The suggested protocol of Blum [37] follows .

Step 1: A picks $a_1, a_2, ..., a_K$ at random in $Z^*_{n_B}$ and then computes $b_i = a_i^2 (mod\ n_B)$ for $1 \leq i \leq k$. B picks $w_1, w_2, ..., w_k$ at random in Z_{n_B} and then computes $x_i = w_{i^*}$ (mod n_A) for $1 \leq i \leq k$.

Step 2: A sends all the b_i's to B and B sends all the x_i's to A.

Step 3: For each x_i A computes y_i and z_i such that $y_i^2 = z_i^2 = x_i$ (mod n_A) but $y_i \neq \pm z_i$ mod n_B. (Note: either y_i or z_i equals $\pm w_i$.) For each b_i, B computes c_i and d_i with similar restrictions. (Note: either c_i or d_i equal $\pm a_i$.

Step 4: While $1 \leq j \leq k$ A sends B the jth significant bit of y_i and z_i for $1 \leq i \leq k$. B sends A the jth significant bit of c_i and d_i for $1 \leq i \leq k$.

Step 5: After completing the above loop, A (and B) figure out the factorization of n_B (and n_A) with the information obtained in Step 4. (A computes $gcd(c_i - d_i, n_B)$ for each i and B computes gcd $(y_i - z_i, n_A)$ for each i. Using this information, they figure out s_B and s_A by decrypting α and β.

Why are k numbers chosen rather that just one? This is to prevent the following type of cheating on the A and B's behalf. Suppose only one x was sent to A. A could figure out y and z and then send the jth significant bits of y and a junk string to B in Step 4, hoping that $y = \pm w$ and A will not notice that junk is being sent. If $y = \pm w$ then B has no way of knowing that A is cheating until the last step, at which time A has all the information he needs to find s_B, but B has not gained any new information to find s_A. So A can cheat with a 50% chance of success. If, on the other hand, k different x's are sent to A, A has an exponentially vanishing chance of successfully cheating in this fashion. Namely $Prob(y_i = \pm w_i \forall i) \leq (\frac{1}{2})^k$.

Unfortunately, Shamir, and Håstad pointed out a way to successfully cheat at this protocol. If, instead of choosing the w_i's at random, A chooses w_1 at random, sets $x_1 = w_1^2$ (mod n_B), and then sets $x_i = x_1/2^{i-1}$ (mod n_B), then after one iteration of Step 4, A has all of the information that he needs to factor n_B by the reduction of [102]. So, a seemingly good protocol fails, since B has no way to check whether A chose the x_is at random independently from each as specified in the protocol or not. Note: that this problem is similar to the problem which arose in the oblivious transfer protocol and can be corrected if A and B could check that each other was following the protocol.

11.2 Zero-Knowledge Protocols

The previous sections listed a number of cryptographic protocol applications and some problems they suffer from. In this section we review the theory that has been developed to prove that these protocols are secure, and to design protocols that are "provably secure by construction". The key idea is to reduce the general problem of two-party protocols to a simpler problem: How can A prove to B that x is in a language L so that no more knowledge than $x \in L$ is revealed. If this could be done for any $L \in NP$ A could prove to B that he followed the protocol steps. We proceed to define the loose terms "interactive proof" (or "proof by a protocol") and "zero knowledge".

11.2.1 Interactive Proof-Systems(IP)

Before defining notion of interactive proof-systems, we define the notion of interactive Turing machine.

Definition 11.1 An *interactive Turing machine (ITM)* is a Turing machine with a read-only input tape, a read-only random tape, a read/write worktape, a read-only communication tape, a write-only communication tape, and a write-only output tape. The random tape contains an infinite sequence of bits which can be thought of as the outcome of unbiased coin tosses, this tape can be scanned only from left to right. We say that an interactive machine *flips a coin* to mean that it reads the next bit from its random tape. The contents of the write-only communication tape can be thought of as *messages sent* by the machine; while the contents of the read-only communication tape can be thought of as *messages received* by the machine. ■

Definition 11.2 An *interactive protocol* is an ordered pair of ITMs (A, B) which share the same input tape; B's write-only communication tape is A's read-only communication tape and vice versa. The machines take turns in being active with B being active first. During its active stage, the machine first performs some internal computation based on the contents of its tapes, and second writes a string on its write-only communication tape. The i^{th} *message* of $A(B)$ is the string $A(B)$ writes on its write-only communication tape in i^{th} stage. At this point, the machine is deactivated and the other machine becomes active, unless the protocol has terminated. Either machine can terminate the protocol, by not sending any message in its active stage. Machine B *accepts* (or rejects) the input by entering an accept (or reject) state and terminating the protocol. The first member of the pair, A, is a computationally unbounded Turing machine. The *computation time* of machine B is defined as the sum of B's computation time during its active stages, and it is bounded by a polynomial in the length of the input string. ■

Definition 11.3 Let $L \in \{0,1\}^*$ We say that

L has an *interactive proof-system* if $\exists$ ITM V *s.t.*

1. $\exists\, ITM\ P\ s.t\ (P, V)$ is an interactive protocol and $\forall x \in L\ s.t\ |x|$ is sufficiently large the $prob(V\ \ accepts) > \frac{2}{3}$ (when probabilities are taken over coin tosses of V and P).
2. $\forall\, ITM\ P\ \ s.t\ (P, V)$ is an interactive protocol $\forall x \notin L\ s.t.\ |x|$ is sufficiently large $Prob(V\ \ accepts) > \frac{1}{3}$ (when probabilities are taken over coin tosses of V and P's).

■

Note that it does not suffice to require that the verifier cannot be fooled by the predetermined prover (such a mild condition would have presupposed that the "prover" is a trusted oracle). **NP** is a special case of interactive proofs, where the interaction is trivial and the verifier tosses no coins.

We say that (P, V) (for which condition 1 holds) is an *interactive proof-system* for L.

Define $IP = \{L \mid L$ has interactive proof$\}$.

11.2.2 Examples

Notation

Throughout the lecture notes, whenever an interactive protocol is demonstrated, we let $B \longrightarrow A$: denote an active stage of machine B, in the end of which B sends A a message. Similarly, $A \longrightarrow B$: denotes an active stage of machine A.

Example 1: (From Number Theory)

$$
\begin{array}{lll}
\text{Let } Z_n^* & = & \{x < n, ; (x,n) = 1\} \\
QR & = & \{(x,n) \mid x < n, (x,n) \text{ and } \exists y \;\; s.t\; y^2 \equiv x \bmod n\} \\
QNR & = & \{(x,n) \mid x < n, (x,n) \text{ and } \not\exists y \;\; s.t\; y^2 \equiv x \;\bmod n\}
\end{array}
$$

We demonstrate an interactive proof-system for QNR.

On input (x,n) to interactive protocol (A,B):

$B \longrightarrow A$: B sends to A the list $w_1 \cdots w_k$ where $k = \mid n \mid$ and

$$
w_i = \begin{cases} z_i^2 \bmod n & \text{if } b_i = 1 \\ x \cdot z_i^2 \bmod n & \text{if } b_i = 0 \end{cases}
$$

where B selected $z_i \in Z_n^*, b_i \in \{0,1\}$ at random.

$A \longrightarrow B$: A sends to B the list $c_1 \cdots c_k$ $s.t.$

$$
c_i = \begin{cases} 1 & \text{if } w_i \text{ is a quadratic residue mod } n \\ 0 & \text{otherwise} \end{cases}
$$

B accepts iff $\forall_{1 \le i \le k},\; c_i = b_i$

B interprets $b_i = c_i$ as evidence that $(x,n) \in QRN$;while $b_i \neq c_i$ leads him to reject.

We claim that (A,B) is an interactive proof-system for QNR. If $(x,n) \in QNR$, then w_i is a quadratic residue mod$n$ iff $b_i = 1$. Thus, the all powerful A can easily compute whether w_i is a quadratic residue mod n or not, compute c_i correctly and make B accept with probability 1. If $(x,n) \notin QNR$ and $(x,n) \in QR$ then w_i is a random quadratic residue mod$n$ regardless of whether $b_i = 0$ or 1. Thus, the probability that A (no matter how powerful he is) can send c_i $s.t$ $c_i = b_i$, is bounded by $\frac{1}{2}$ for each i and probability that B accepts is at most $(\frac{1}{2})^k$.

Example 2: (From Graph Theory)

To illustrate the definition of an interactive proof, we present an interactive proof for *Graph Non-Isomorphism*. The input is a pair of graphs G_1 and G_2, and one is required to prove that there exists no 1-1 edge-invariant mapping of the vertices of the first graph to the vertices of the second graph. (A mapping π from the vertices of G_1 to the vertices G_2 is *edge-invariant* if the nodes v and u are adjacent in G_1 iff the nodes $\pi(v)$ and $\pi(u)$ are adjacent in G_2.) It is interesting to note that no short NP-proofs are known for this problem; namely Graph Non-isomorphism is *not known* to be in **NP**.

The interactive proof (A,B) on input (G_1, G_2) proceeds as follows:

$B \longrightarrow A$: B chooses at random one of the two input graphs, G_{α_i}

where $\alpha_i \in \{1,2\}$. B creates a random isomorphic copy of G_{α_i} and sends it to A. (This is repeated k times, for $1 \le i \le k$, with independent random choices.)

$A \longrightarrow B$: A sends B $\beta_i \in \{1,2\}$ for all $1 \le i \le k$.

B accepts iff $\beta_i = \alpha_i$ for all $1 \le i \le k$.

B interprets $\beta_i = \alpha_i$ as evidence that the graphs are not isomorphic; while $\beta_i \neq \alpha_i$ leads him to reject.

If the two graphs are not isomorphic, the prover has no difficulty to always answer correctly (i.e., a β equal to α), and the verifier will accept. If the two graphs are isomorphic, it is impossible to distinguish a random isomorphic copy of the first from a random isomorphic copy of the second, and the probability that the prover answers correctly to one "query" is at most $\frac{1}{2}$. The probability that the prover answers correctly all k queries is $\leq (\frac{1}{2})^k$.

11.2.3 Zero-Knowledge

Now that we have extended the notion of what is an efficient proof-system, we address the question of how much "knowledge" need to be transferred in order to convince a polynomial-time bounded verifier, of the truth of a proposition. What do we mean by "knowledge"? For example, consider SAT, the NP-complete language of satisfiable sentences of propositional calculus. The most obvious proof-system is one in which on logical formula F the prover gives the verifier a satisfying assignment I, which the verifier can check in polynomial time. If finding this assignment I by himself would take the verifier more than polynomial time (which is the case if $P \neq NP$), we say that the verifier gains additional knowledge to the mere fact that $F \in SAT$.

Goldwasser, Micali and Rackoff [99] make this notion precise. They call an interactive proof-system for language L zero-knowledge if $\forall x \in L$ whatever the verifier can compute after participating in the interaction with the prover, could have been computed in polynomial time on the input x alone by a probabilistic polynomial time Turing machine.

We give the technical definition of zero-knowledge proof-systems and its variants in section 11.2.4, and briefly mention a few interesting results shown in this area.

11.2.4 Definitions

Let (A, B) be an interactive protocol. Let *view* be a random variable denoting the verifier view during the protocol on input x. Namely, for fixed sequence of coin tosses for A and B, *view* is the sequences of messages exchanged between verifier and prover, in addition to the string of coin tosses that the verifier used. The string h denotes any private input that the verifier may have with the only restriction that its length is bounded by a polynomial in the length of the common input. (*view* is distributed over both A's and B's coin tosses).

We say that (A, B) is *perfect zero-knowledge* for L if there exists a probabilistic, polynomial time Turing machine M $s.t$ $\forall x \in L$, for all $a > 0$, for all strings h such that $|h| < |x|^a$, the random variable $M(x, h)$ and *view* are identically distributed. ($M(x, h)$ is distributed over the coin tosses of M on inputs x and h).

We say that (A, B) is *statistically zero-knowledge* for L if there exists a probabilistic polynomial time Turing machine M $s.t$ $\forall x \in L$, for all $a > 0$, for all strings h such that $|h| < |x|^a$,

$$\sum_{\alpha} |prob(M(x,h) = \alpha) - prob(view = \alpha)| < \frac{1}{|x|^c}$$

for all constants $c > 0$ and sufficiently large $|x|$.

Intuitively the way to think of statistically zero-knowledge protocols, is that an infinite power "examiner" who is given only polynomially large samples of $\{M(x,h)|M$'s coin tosses $\}$ and $\{view\ |A$'s and B's coin tosses$\}$ can't tell the two sets apart.

Finally, we say that a protocol (A, B) is computationally zero-knowledge if a probabilistic polynomial time bounded "examiner" given a polynomial number of samples from the above sets can not tell them apart. Formally,

We say that (A, B) is *computationally zero-knowledge* for L if $\exists$ probabilistic, polynomial time Turing machine M $s.t$ $\forall$ polynomial size circuit families $C = \{C_{|x|}\}$, $\forall$ constants $a, d > 0$, for all sufficiently large $|x|$ s.t $x \in L$, and for all strings h such that $|h| < |x|^a$,

$$prob(C_{|x|}(\alpha) = 1 | \alpha \text{ random in } M(x,h)) - prob(C_{|x|}(\alpha) = 1 | \alpha \text{ random in } view(x))| < \frac{1}{|x|^d}$$

We say that L has *(computational/statistical/perfect) zero-knowledge proof-system* if

1. $\exists$ interactive proof-system (A, B) for L.
2. $\forall$ ITM's B', interactive protocol (A, B') is (computational/statistical/perfect) zero-knowledge for L.

Clearly,the last definition is the most general of the three. We thus let $KC[0] = \{L | L$ has computational zero-knowledge proof-system$\}$.

11.2.5 If there exists one way functions, then NP is in KC[0]

By far, the most important result obtained about zero-knowledge is by Goldreich, Micali and Wigderson [95]. They show the following result.

Theorem[95]: if there exist (non-unifrom) polynomial-time indistinguishable encryption scheme then every NP language has a computational zero-knowledge interactive proof-system.

The non uniformity condition is necessary for technical reasons (i.e the encryption scheme should be secure against non-uniform adversary. see section 3.7). The latest assumption under which such encryption scheme exists is the existence of one-way functions (with respect to non-uniform adversary) by results of Imagliazzo-Levin-Luby and Naor.

The proof outline is to show a zero-knowledge proof system for an NP-complete language, graph three colorability. We outline the protocol here. Suppose the prover wish to convince the verifier that a certain input graph is three-colorable, without revealing to the verifier the coloring that the prover knows. The prover can do so in a sequence of $|E|^2$ stages, each of which goes as follows.

- The prover switchs the three colors at random (e.g., switching all red nodes to blue, all blue nodes to yellow, and all yellow nodes to red).
- The prover encrypts the color of each node, using a different probabilistic encryption scheme for each node, and show the verifier all these encryptions, together with the correspondence indicating which ciphertext goes with which vertex.
- The verifier selects an edge of the graph at random.
- The prover reveals the decryptions of the colors of the two nodes that are incident to this edge by revealing the corresponding decryption keys.
- The verifier confirms that the decryptions are proper, and that the two endpoints of the edge are colored with two different but legal colors.

(any private probabilistic encryption scheme which is polynomial time indistinguishable will work here) If the graph is indeed three-colorable (and the prover know the coloring), then the verifier will never detect any edge being incorrectly labeled. However, if the graph is not three-colorable, then there is a chance of at least $|E|^{-1}$ on each stage that the prover will be caught trying to fool the verifier. The chance that the prover could fool the verifier for $|E|^2$ stages without being caught is exponentially small.

Note that the history of our communications—in the case that the graph is three-colorable—consists of the concatenation of the messages sent during each stage. It is possible to prove (on the assumption that secure encryption is possible) that the probability distribution defined over these histories by our set of possible interactions is indistinguishable in polynomial time from a distribution that the verifier can create on these histories by itself, without the provers participation. This fact means that the verifier gains zero (additional) knowledge from the protocol, other than the fact that the graph is three-colorable.

The proof that graph three-colorability has such a zero-knowledge interactive proof system can be used to prove that every language in NP has such a zero-knowledge proof system.

11.2.6 Applications to User Identification

Zero knowledge proofs provide a revolutionary new way to realize passwords [100, 79]. The idea is for every user to store a statement of a theorem in his publicly readable directory, the proof of which only he knows. Upon login, the user engages in a zero-knowledge proof of the correctness of the theorem. If the proof is convincing, access permission is granted. This guarantees that even an adversary who overhears the zero-knowledge proof can not learn enough to gain unauthorized access. This is a novel property which can not be achieved with traditional password mechanisms. Fiat and Shamir [79] have developed variations on some of the previously proposed zero-knowledge protocols [100] which are quite efficient and particularly useful for user identification and passwords.

11.3 Multi Party protocols

In a typical multi-party protocol problem, a number of parties wish to coordinate their activities to achieve some goal, even though some (sufficiently small) subset of them may have been corrupted by an adversary. The protocol should guarantee that the "good" parties are able to achieve the goal even though the corrupted parties send misleading information or otherwise maliciously misbehave in an attempt to prevent the good parties from succeeding.

11.3.1 Secret sharing

Secret Sharing protocols were invented independently by Blakley and Shamir [33, 181]. In the multi-party setting, secret sharing is a fundamental protocol and tool.

The basic idea is protection of privacy of information by distribution. Say you have a key to an important system. You are afraid you might loose it, so you want to give it to someone else. But no single person can be trusted with the key. Not just because that person may become untrustworthy, but because the place they keep the key may be compromised. So the key is shared amongst a bunch of people.

Let's call the key the secret s. A way to share it amongst five people is split it up as $s = s_1 \oplus \cdots \oplus s_5$ and give s_i to person i. No one person can figure out s. Even more, no four people can to it: it takes all five. If they all get together they can recover s. (Once that is done, they may discard it, ie it may be a one time key! Because now everyone knows it.)

We call s_i a share. Who creates the shares? The original holder of s. Sometimes it is one of the n players, sometimes not. We call this person the dealer.

Notice that s_i must be given *privately* to the i-th player. If other players see it, then, of course, this doesn't work.

We may want something more flexible. Say we have n people. We want that any $t+1$ of them can recover the secret but no t of them can find out anything about it, for some parameter t. For example, say $n = 5$ and $t = 2$. Any three of your friends can open your system, but no two of them can. This is better since above if one of them looses their share, the system can't be opened.

Shamir's idea is to use polynomials [181]. Let F be a finite field, like Z_p^*. A degree t polynomial is of the form $f(x) = a_0 + a_1 x + \cdots + a_t x^t$ for coefficients $a_0, \ldots, a_t \in F$. It has $t+1$ terms, not t! One more term than the degree. Polynomials have the following nice properties:

- Interpolation: Given $t+1$ points on the polynomial, namely $(x_1, y_1), \ldots, (x_{t+1}, y_{t+1})$ where $x_1, \ldots, x_{t=1}$ are distinct and $y_i = f(x_i)$, it is possible to find $a_0, \ldots, a_t$. The algorithm to do this is called interpolation. You can find it in many books.
- Secrecy: Given any t points on the polynomial, namely $(x_1, y_1), \ldots, (x_t, y_t)$ where $x_1, \ldots, x_t$ are distinct and $y_i = f(x_i)$, one can't figure out anything about a_0. More precisely, for any value v, the number of polynomials satisfying these t constraints does not depend on v. (In fact there is exactly one of them.)

These makes them a tool for secret sharing. Associate to each player i a point $x_i \in F$, these points being all distinct. (So $|F| \geq n$). To share secret s, the dealer picks $a_1, \ldots, a_t$ at random, sets $a_0 = s$ and forms the polynomial $f(x) = a_0 + a_1 x + \cdots + a_t x^t$. Now he computes $s_i = f(x_i)$ and sends this *privately* to player i. Now if $t+1$ players get together they can figure out f and hence s; any set of at most t players can't figure out anything about s.

11.3.2 Verifiable Secret Sharing

Shamir's scheme suffers from two problems. If the dealer of the secret is dishonest, he can give pieces which when put together do not uniquely define a secret. Secondly, if some of the players are dishonest, at the reconstruction stage they may provide other players with different pieces than they received and again cause an incorrect secret to be reconstructed.

Chor, Goldwasser, Micali, and Awerbuch [55] have observed the above problems and showed how to achieve secret sharing based on the intractability of factoring which does not suffer from the above problems. They call the new protocol *verifiable secret sharing* since now every party can verify that the piece of the secret he received is indeed a proper piece. Their protocol tolerated up to $O(\log n)$ colluders. Benaloh [26], and others [95, 78] showed how to achieve verifiable secret sharing if any one-way function exists which tolerates a minority of colluders. In [25] it has been recently shown how to achieve verifiable secret sharing against a third of colluders using error correcting codes, without making cryptographic assumptions. This was improved to a minority of colluders in [167].

11.3.3 Anonymous Transactions

Chaum has advocated the use of *anonymous transactions* as a way of protecting individuals from the maintenance by "Big Brother" of a database listing all their transactions, and proposes using *digital pseudonyms* to do so. Using pseudonyms, individuals can enter into electronic transactions with assurance that the transactions can not be later traced to the individual. However, since the individual is anonymous, the other party may wish assurance that the individual is authorized to enter into the transaction, or is able to pay. [50, 53].

11.3.4 Multiparty Ping-Pong Protocols

One way of demonstrating that a cryptographic protocol is secure is to show that the primitive operations that each party performs can not be composed to reveal any secret information.

Consider a simple example due to Dolev and Yao [73] involving the use of public keys. Alice sends a message M to Bob, encrypting it with his public key, so that the ciphertext C is $E_B(M)$ where E_B is Bob's public encryption key. Then Bob "echos" the message back to Alice, encrypting it with Alice's public key, so that the ciphertext returned is $C' = E_A(M)$. This completes the description of the protocol.

Is this secure? Since the message M is encrypted on both trips, it is clearly infeasible for a *passive* eavesdropper to learn M. However, an *active* eavesdropper X can defeat this protocol. Here's how: the eavesdropper X overhears the previous conversation, and records the ciphertext $C = E_B(M)$. Later, X starts up a conversation with Bob using this protocol, and sends Bob the encrypted message $E_B(M)$ that he has recorded. Now Bob dutifully returns to X the ciphertext $E_X(M)$, which gives X the message M he desires!

The moral is that an adversary may be able to "cut and paste" various pieces of the protocol together to break the system, where each "piece" is an elementary transaction performed by a legitimate party during the protocol, or a step that the adversary can perform himself.

It is sometimes possible to *prove* that a protocol is invulnerable to this style of attack. Dolev and Yao [73] pioneered this style of proof; additional work was performed by Dolev, Even, and Karp [72], Yao [202], and Even and Goldreich [76]. In other cases a modification of the protocol can eliminate or alleviate the danger; see [168] as an example of this approach against the danger of an adversary "inserting himself into the middle" of a public-key exchange protocol.

11.3.5 Multiparty Protocols When Most Parties are Honest

Goldreich, Micali, and Wigderson [95] have shown how to "compile" a protocol designed for honest parties into one which will still work correctly even if some number less than half of the players try to "cheat". While the protocol for the honest parties may involve the disclosure of secrets, at the end of the compiled protocol none of the parties know any more than what they knew originally, plus whatever information is disclosed as the "official output" of the protocol. Their compiler correctness and privacy is based on the existence of trapdoor functions.

Ben-Or, Goldwasser and Wigderson [25] and Chaum, Crépeau, and Damgàrd [51] go one step further. They assume secret communication between pairs of users as a primitive. Making no intractability assumption, they show a "compiler" which, given a description (e.g., a polynomial time algorithm or circuit) of any polynomial time function f, produces a protocol which always computes the function correctly and guarantees that no additional information to the function value is leaked to dishonest players . The "compiler" withstands up to 1/3 of the parties acting dishonestly in a manner directed by a worst-case unbounded-computation-time adversary.

These "master theorems" promise to be very powerful tool in the future design of secure protocols.

11.4 Electronic Elections

Electronic Elections can be considered the typical example of secure multiparty computations. The general instance of such a problem is that there are m people, each of them with their own private input x_i and we want to compute the result of a n-ary function f over such values, without revealing them.

In the case of electronic elections the parties are the voters, their input a binary value, the function being computed is just a simple sum and the result is the tally.

In general, these are the properties that we would like our Election Protocols to have:

1. Only authorized voters can vote.
2. No one can vote more than once.
3. Secrecy of votes is maintained.
4. No one can duplicate anyone else's vote.
5. The tally is computed correctly.
6. Anybody should be able to check 5.
7. The protocol should be fault-tolerant, meaning it should be able to work even in the presence of a number of "bad" parties.
8. It should be impossible to coerce a voter into revealing how she voted (e.g. vote-buying)

Usually in in election protocols it is not desirable to involve all the voters V_i in the computation process. So we assume that there are n government centers $C_1, \ldots, C_n$ whose task is to collect votes and compute the tally.

11.4.1 The Merritt Election Protocol

Consider the following scheme by Michael Merritt [140].

Each center C_i publishes a public key E_i and keeps secret the corresponding secret key. In order to cast her vote v_j, each voter V_j chooses a random number s_j and computes,

$$E_1(E_2(\dots E_n(v_j, s_j))) = y_{n+1,j} \tag{11.1}$$

(The need for the second index $n+1$ will become clear in a minute, for now it is just irrelevant.)

Now we have the values y's posted. In order from center C_n to center C_1, each center C_i does the following. For each $y_{i+1,j}$, C_i chooses a random value $r_{i,j}$ and broadcasts $y_{i,j'} = E_i(y_{i+1,j}, j)$. The new index j' is computed by taking a random permutation π_i of the integers $[1..n]$. That is $j' = \pi_i(j)$. C_i keeps the permutation secret.

At the end we have

$$y_{1,j} = E_1(E_2(\dots E_n(y_{n+1,j}, r_{n,j}) \dots r_{2,j}) r_{1,j})$$

At this point, the verification cycle begins. It consists of two rounds of decryption in the order $C_1 \longrightarrow C_2 \dots \longrightarrow C_n$.
The decrypted values are posted and the tally computed by taking the sums of the votes v_j's.

(1) and (2) are clearly satisfied. (3) is satisfied, as even if the votes are revealed, what is kept hidden is the connection between the vote and the voter who casted it. Indeed in order to reconstruct such link we need to know all the permutations π_i. (4) is not satisfied as voter V_1 can easily copy voter V_2, by for example casting the same encrypted string. (5) and (6) are satisfied using the random strings: during the first decryption rounds each center checks that his random strings appear in the decrypted values, making sure that all his ciphertexts are being counted. Also at the end of the second decryption round each voter looks for her string s_j to make sure her vote is being counted (choosing a large enough space for the random string should eliminate the risk of duplicates.) Notice that in order to verify the correctness of the election we need the cooperation of all the voters (a negative feature especially in large protocols.)

(7) requires a longer discussion. If we are concerned about the secrecy of the votes being lost because of parties going "bad", then the protocol is ideal. Indeed even if $n-1$ of the centers cooperate, they will not be able to learn who casted what vote. Indeed they need to know *all* the permutations π_i. However even if one of the government agencies fails, by for example crashing, the entire system falls apart. The whole election needs to be repeated.

(8) is not satisfied. Indeed the voter can be forced to reveal both v_j and s_j and she tries to lie about the vote she will be discovered since the declared values will not match the ciphertext $y_{n+1,j}$.

11.4.2 A fault-tolerant Election Protocol

In this section we describe a protocol which has the following features

- satisfies (4), meaning it will be impossible to copy other people vote (the protocol before did not)
- Does not require the cooperation of each voter to publicly verify the tally (better solution to (6) than the above)
- introduces fault-tolerance: we fix a threshold t and we assume that if there are less than t "bad" centers the protocol will correctly compute the tally and the secrecy of each vote will be preserved (better solution to (7) than the above.)

This protocol is still susceptible to coercion (requirement (8)). We will discuss this point at the end.

The ideas behind this approach are due to Josh Benaloh [28]. The protocol described in the following section is the most efficient one in the literature due to Cramer, Franklin, Schoemakers and Yung [60].

Homomorphic Commitments

Let B be a commitment scheme (a one-way function basically.)

We say that a commitment scheme B is $(+,\times)$-homomorphic if

$$B(X+Y) = B(X) \times B(Y)$$

One possible example of such commitment is the following (invented by Pedersen [155]):

Discrete-Log based Homomorphic Commitment: Let p be a prime of the form $p = kq+1$ and let g, h be two elements in the subgroup of order q. We assume nobody knows the discrete log in base g of h. To commit to a number m in $[1..q]$:

$$B_a(m) \;=\; g^a \; h^m \tag{11.2}$$

for a randomly chosen a modulo q. To open the commitment a and m must be revealed.

Notice that this is a $(+,\times)$-homomorphic commitment as:

$$B_{a_1}(m_1)B_{a_2}(m_2) = g^{a_1}h^{m_1}g^{a_2}h^{m_2} = g^{a_1+a_2}h^{m_1+m_2} = B_{a_1+a_2}(m_1+m_2)$$

For now on let E be an $(+,\times)$-homomorphic commitment scheme.

11.4.3 The protocol

For ease of presentation we will show the protocol in two version. First we assume that there is only one center. Then we show how to generalize the ideas to the case of many centers.

Vote Casting – 1 center

Assume for now that there is only one center C and let E be his encryption function.

Assuming the votes are either -1 or 1, each voter V_j encrypts his vote v_j by computing and posting $B_{a_j}(v_j)$ for a randomly chosen a_j. V_j also sends the values a_j and v_j to C encrypted.

The voter now must prove that the vote is correct (i.e. it's the encryption of a -1 or of a 1.) He does this by performing a zero-knowledge proof of validity.

For the discrete-log based homomorphic commitment scheme described above, here is a very efficient protocol. Let us drop the index j for simplicity.

For $v = 1$:

1. The voter V chooses at random a, r_1, d_1, w_2 modulo q. He posts $B_a(v) = g^a h$ and also posts $\alpha_1 = g^{r_1}(B_a(v)h)^{-d_1}$, $\alpha_2 = g^{w_2}$.
2. The center C sends a random challenge c modulo q
3. The voter V responds as follows: V computes $d_2 = c - d_1$ and $r_2 = w_2 + ad_2$ and posts d_1, d_2, r_1, r_2
4. The center C checks that
   - $d_1 + d_2 = c$
   - $g^{r_1} = \alpha_1(B_a(v)h)^{d_1}$
   - $g^{r_2} = \alpha_2(B_a(v)/h)^{d_2}$

For $v = -1$:

1. The voter V chooses at random a, r_2, d_2, w_1 modulo q. He posts $B_a(v) = g^a/h$ and also posts $\alpha_1 = g^{w_1}$, $\alpha_2 = g^{r_2}(B_a(v)/h)^{-d_2}$
2. The center C sends a random challenge c modulo q
3. The voter V responds as follows: V computes $d_1 = c - d_2$ and $r_1 = w_1 + ad_1$ and posts d_1, d_2, r_1, r_2
4. The center C checks that
   - $d_1 + d_2 = c$
   - $g^{r_1} = \alpha_1(B_a(v)h)^{d_1}$
   - $g^{r_2} = \alpha_2(B_a(v)/h)^{d_2}$

For now on we will refer to the above protocol as $\mathsf{Proof}(B_a(v))$.

Tally Computation – 1 center

At the end of the previous phase we were left with $B_{a_j}(v_j)$ for each voter V_j. The center reveals the tally $T = \sum_j v_j$ and also the value $A = \sum_j a_j$. Everybody can check that the tally is correct by performing the following operation:

$$B_A(T) = \prod_j B_{a_j}(v_j)$$

which should be true for the correct tally, because of the homomorphic property of B.

The 1-center version of the protocol however has the drawback that this center learns everybody's vote.

Vote Casting – n centers

Assume n centers $C_1, \ldots, C_n$ and let E_i be the encryption function of C_i.

In this case voter V_j encrypts the vote v_j in the following manner. First he commits to the vote by posting

$$B_j = B_{a_j}(v_j)$$

for a randomly chosen a_j modulo q. He also proves that this is a correct vote by performing $\mathsf{Proof}(B_{a_j}(v_j))$.

Then he shares the values a_j and v_j among the centers using Shamir's (t, n) threshold secret sharing. That is, he chooses random polynomials $H_j(X)$ and $A_j(X)$ of degree t such that $H_j(0) = v_j$ and $A_j(0) = a_j$. Let

$$R_j(X) = v_j + r_{1,j}X + \ldots + r_{t,j}X^t$$

$$S_j(X) = a_j + s_{1,j}X + \ldots + s_{t,j}X^t$$

The coefficients are all modulo q.

Now the voter sends the value $u_{i,j} = R_j(i)$ and $w_{i,j} = S_j(i)$ to the center C_i (encrypted with E_i.)

Finally he commits to the coefficients of the polynomial H_j by posting

$$B_{\ell,j} = B_{s_{\ell,j}}(r_{\ell,j})$$

The centers perform the following check

$$g^{w_{i,j}} h^{u_{i,j}} = B_j \prod_{\ell=1}^{t} (B_{\ell,j})^{i^\ell} \tag{11.3}$$

to make sure that the shares he received encrypted are correct.

Tally counting – n centers

Each center C_i posts the partial sums:

$$T_i = \sum_j u_{i,j}$$

this is the sum of the shares of the votes received by each player.

$$A_i = \sum_j w_{i,j}$$

this is the sum of the shares of the random string a_j used to commit to the vote by each player.

Anybody can check that the center is revealing the right stuff by using the homomorphic property of the commitment scheme B. Indeed it must hold that

$$g^{A_i} h^{T_i} = \prod_{j=1}^{m} \left(B_j \prod_{\ell=1}^{t} (B_{\ell,j})^{j^\ell} \right) \tag{11.4}$$

Notices that the correct T_i's are shares of the tally T in a (t,n) Shamir's secret sharing scheme. So it is enough to take $t+1$ of them to interpolate the tally.

Notice: Equations (11.3) and (11.4) are valid only under the assumption that nobody knows the discrete log in base g of h. Indeed who knows some value can open the commitment B in both ways and so reveal incorrect values that satisfies such equations.

Analysis: Let's go through the properties one by one. (1) and (2) are clearly satisfied. (3) is satisfied assuming that at most t centers can cooperate to learn the vote. If $t+1$ centers cooperate, then the privacy of the votes is lost. (4) is true for the following reason: assume that V_1 is trying to copy the action of V_2. When it comes to the point of proving the correctness of the vote (i.e. perform $\mathsf{Proof}(B)$), V_1 will probably receive a different challenge c than V_2. He will not be able to answer it and he will be eliminated from the election. (5) is true under the discrete-log assumption (see note above.) (6) is true as anybody can check on the the ZK proofs and Equations (11.3) and (11.4). (7) is true as we need only $t+1$ good centers to reconstruct the tally.

It is easy to see that because we need $t+1$ good centers and at most t centers can be bad, the maximum number of corrupted centers being tolerated by the protocol is $\frac{n}{2} - 1$.

(8) is *not* satisfied. This is because somebody could be coerced into revealing both a and v when posting the commitment $B_a(v)$.

11.4.4 Uncoercibility

The problem of coercion of voters is probably the most complicated one. What exactly does it mean? In how many ways can a coercer, try to force a voter to cast a given vote.

Let's try to simplify the problem. We will consider two possible kinds of coercer. One who contacts the voter *before* the election starts and one who contacts the voter *after* the election is concluded.

The "before" coercer has a greater power. He can tell the voter what vote to cast and also what randomness to use during the protocol. This basically would amount to **fix** the behavior of the voter during the protocol. If the voter does not obey, it will be easy for the coercer to detect such occurrence. There have been some solutions proposed to this problem that use some form of physical assumption. For example one could allow the voter to exchange a limited number of bits over a secure channel with the voting centers [27, 175]. This would hopefully prevent the coercer from noticing that the voter is not following his instructions. Or one could force the voter to use some tamper-proof device that encrypts messages for him, choosing the randomness. This would prevent the coercer from forcing the user to use some fixed coin tosses as the user has no control on what coins the tamper-proof device is going to generate.

The "after" coercer has a smaller power. He can only go to the voter and ask to see the vote v and the randomness ρ used by the voter during the protocol. Maybe there could be a way for the voter to construct different v' and ρ' that "match" his execution of the protocol. This is *not* possible in the protocol above (unless the voter solves the discrete log problem.) Recently however a protocol for this purpose has been proposed by Canetti and Gennaro [47]. They use a new tool called *deniable encryption* (invented by Canetti, Dwork, Naor and Ostrovsky [46]), which is a new form of public key probabilistic encryption E with the following property.

Let m be the message and r the coin tosses of the sender. The sender computes the ciphertext $c = E_r(m)$. After if somebody approaches him and asks for the value of m, the sender will be able to produce m' and r' such that $E_{r'}(m') = c$.

11.5 Digital Cash

The primary means of making monetary transactions on the Internet today is by sending credit card information or establishing an account with a vendor ahead of time.

The major opposition to credit card based Internet shopping is that it is not anonymous. Indeed it is susceptible to monitoring, since the identity of the customer is established every time he/she makes a purchase. In real life we have the alternative to use cash whenever we want to buy something without establishing our identity. The term *digital cash* describes cryptographic techniques and protocols that aim to recreate the concept of cash-based shopping over the Internet.

First we will describe a general approach to digital cash based on public-key cryptography. This approach was originally suggested by David Chaum [50]. Schemes based on such approach achieve the anonymity property.

11.5.1 Required properties for Digital Cash

The properties that one would like to have from Digital Cash schemes, are at least the following:

- forgery is hard
- duplication should be either prevented or detected
- preserve customers' anonymity
- minimize on-line operations on large database

11.5.2 A First-Try Protocol

A Digital Cash scheme consists usually of three protocols. The **withdrawal protocol** which allows a Userto obtain a digital coin from the Bank. A **payment protocol** during which the Userbuys goods from a Vendorin exchange of the digital coin. And finally a **deposit protocol** where the Vendorgives back the coin to the Bankto be credited on his/her account.

In the protocol below we assume that the Bankhas a secret key SK_B to sign messages and that the corresponding public key PK_B is known to everybody else. With the notation $\{M\}_{SK}$ we denote the message M together with its signature under key SK.

Let's look at this possible digital cash protocol.

Withdrawal Protocol:

1. Usertells Bankshe would like to withdraw \$100.

2. Bankreturns a \$100 bill which looks like this:

$$\{\text{I am a \$100 bill, \#4527)}\}_{SK_B}$$

and withdraws \$100 from Useraccount

3. Userchecks the signature and if it is valid accepts the bill

Payment Protocol:

1. The Userpays the Vendorwith the bill.
2. The Vendorchecks the signature and if it's valid accepts the bill.

Deposit Protocol:

1. The Vendorgives the bill to the Bank
2. The Bankchecks the signature and if it's valid, credits the Vendor's account.

Given some suitable assumption on the security of the signature scheme, it is clear that it is impossible to forge digital coins. However it is very easy to duplicate and double-spend the same digital coin several times. It is also cleat that anonymity is not preserved as the Bankcan link the name of the Userwith the serial number appearing on the bill and know where the Userspent the coin.

11.5.3 Blind signatures

Let's try to solve the anonymity problem first. This approach involves —em blind signatures. The user presents the bank with a bill inside a container. The bank signs the bill without seeing the contents of the bill. This way, the bank cannot determine the source of a bill when a merchant presents it for deposit.

A useful analogy: The user covers a check with a piece of carbon paper and then seals both of them inside an envelope. The user gives the envelope to the bank. The bank then signs the outside of the envelope with a ball-point pen and returns the envelope to the user (without opening it - acutally the bank is *unable* to open the envelope in the digital version). The user then removes the signed check from the envelope and can spend it. The bank has never seen what it signed, so it cannot associate it with the user when it is returned to be deposited, but it can verify the signature on the check and thus guarantee the validity of the check.

There is, *of course*, a problem with this: The bank can be fooled into signing phony bills. For example, a user could tell the bank he's making a \$1 withdrawal and then present a \$100 bill to be signed. The bank will, unknowingly, sign the \$100 bill and allow the user to cheat the bank out of \$99. We will deal with this problem later, for now let us show how to construct blind signatures.

11.5.4 RSA blind signatures

Recall the RSA signature scheme: if M is the message to be signed, then its signature is $s = M^{e^{-1}} \bmod n$ where n and e are publicly known values. The secret information that the bank possesses is the inverse of $e \bmod \phi(n)$, which we will denote by d. The signature can be verified by calculating $s^e \bmod n$ and verifying that it is equal to $M \bmod n$.

In the case of blind signatures, the Userwants the Bankto provide him with s, without revealing M to the bank. Here is a possible anonymous withdrawal protocol. Let M be a \$100 bill.

Withdrawal Protocol:

1. Userchooses some random number, $r \bmod n$.

2. Usercalculates $M' = M \cdot r^e \bmod n$.

3. Usergives the Bank$M$.

4. The Bankreturns a signature for M', say $s' = (M')^d \bmod n$. Note that

$$s' = (M')^d = M^d \cdot (r^e)^d = M^d \cdot r$$

5. The Bankdebits the Useraccount for \$100.

6. Since the Userknows r, he can divide s' by r to obtain

$$s = M^d$$

The payment and deposit protocol remain the same as above. This solves the problem of preserving the Useranonymity, as when the coin comes back to the Bankthere is no link between it and the Userit was issued to.

We still have two problems.

1. The bank can still be fooled into signing something that it shouldn't (like a \$100 bill that it thinks is a \$1 bill)

2. Coins can still be duplicated and double-spent

11.5.5 Fixing the dollar amount

One possible solution to the above problem is to have only one denomination (per public key, for example.) That is the Bankwould have several public keys $PK1_B, \ldots$ and the signature using PKi_B would be valid only on bills of i dollars.

Another possibility is to use a "cut and choose" procedure:

1. Usermakes up 100 \$20 bills

2. blinds them all

3. gives them to the Bank

4. The Bankpicks one to sign (at random), and requires that the Userunblind all of the rest (by revealing the r's). Before the signature is returned, the Bankensures that all of the bills that were unblinded were correct.

This way the Userhas only $\frac{1}{100}$ probability of cheating. Of course, one could set up the protocol to create an smaller cheating chance (by requiring that the user provided more blinded messages, for example).

So, now we have a protocol that satisfies the anonymity requirement and can provide sufficiently small possibilities for cheating. We still have to deal with the double-spending problem.

11.5.6 On-line digital cash

In the on-line version of digital cash schemes, one requires the Bankto record all of the bills it receives in a database. During the payment protocol the Vendorwould transmit the bill to the Bankand ask if the bill was already received. If this is the first time the bill is being used then the Vendoraccepts it, otherwise he will reject it.

Although this is a simple solution it incurs in a high communication overhead as now the payment protocol looks a lot like a credit card transaction, when the Vendorawaits for authorization to finish the trade. Also the size of the database to be managed by the Bankcould be problematic.

Notice that we are *preventing* double-spending this way. We are going to show now a way to *detect* double-spending which does not require on-line verification.

11.5.7 Off-line digital cash

The idea behind off-line digital cash is the following. During the payment protocol the Useris forced to write a "random identity string", or RIS, on the bill.

The RIS must have the following properties:

- must be different for every payment of the coin.
- only the Usercan create a valid RIS.
- two different RIS on the same coin should allow the Bankto retrieve the Username.

If the Bankreceives two identical bills with different RIS values, then the Userhas cheated and the bank can identify him. If the Bankreceives two identical bills with the same RIS values, then the Vendorhas cheated. The above idea appeared first in [52].

Here is a possible solution. Let H to be a one-way hash function.

Withdrawal Protocol:

1. The Userprepares 100 bills of \$20 which look like this:

$$M_i = \text{(I'm \$20 bill, \#4527i,} y_{i,1}, y'_{i,1}, y_{i,2}, y'_{i,2}, \ldots, y_{i,K}, y'_{i,K})$$

   where $y_{i,j} = H(x_{i,j})$, $y'_{i,j} = H(x'_{i,j})$, where $x_{i,j}$ and $x'_{i,j}$ are randomly chosen under the condition that

$$x_{i,j} \oplus x'_{i,j} = \text{ Username } \forall i, j$$

2. The Userblinds all the M_i to random messages M'_i (using the blinding protocol outlined above) and sends them to the Bank.
3. The Bankasks the Userto unblind 99 of the 100 blinded bills.
4. When the Userunblinds them, he also reveals the appropriate $x_{i,j}$ and $x'_{i,j}$.
5. The Bankchecks not only that the bills are indeed \$20 bills, but also that $y_{i,j} = H(x_{i,j})$, $y'_{i,j} = H(x'_{i,j})$ and $x_{i,j} \oplus x'_{i,j} =$ Username, for the unblinded bills.
6. The Bankreturns a signature on the only blind message (say M'_{17})
7. The Userretrieves the signature s_{17} on M_{17}.

From now on let us drop the index $i = 17$ for simplicity. The payment protocol is modified to force the Userto produce a RIS on the coin. The RIS is going to be one of x_j or x'_j for each $j = 1, \ldots, K$. Which one is going to be depends on a random challenge from the Vendor.

Payment Protocol:

1. The Usergives M, s to the Vendor.

2. The Vendorchecks the Banksignature on the bill and if it is valid, answers with a random bit string of length K, $b_1 \ldots b_K$.

3. If $b_j = 0$ Userreveals x_j, otherwise he reveals x'_j

4. The Vendorchecks that $y_j = H(x_j)$ or $y'_j = H(x'_j)$, whichever is the case. If the above equalities hold, he accepts the bill.

Notice that the above properties or RIS are satisfied. Indeed the probability that in a different payment the same RIS is produced is 2^{-K} since the Vendorchooses the "challenge" at random. Only the Usercan produce a valid RIS since the function H is one-way. Finally two different RIS on the same coin leak the name of the User, as if two RIS are different there must be an index j for which we have both x_j and x'_j.

Deposit Protocol:

1. The Vendorbrings the coin M, s, RIS back to the Bank.

2. The Bankverifies the signature and check if the coin M, s has already been returned to the Bank

3. If the coin is already in the database, the Bankcompares the RIS's of the two coins. If the RIS are different then the Userdouble-spent the coin, otherwise it is the Vendorwho is trying to deposit the coin twice.

Bibliography

[1] ISO/IEC 9796. Information technology security techniques – digital signature scheme giving message recovery, 1991. International Organization for Standards.

[2] L. M. Adleman. On breaking generalized knapsack public key cryptosystems. In *Proc. 15th ACM Symp. on Theory of Computing*, pages 402–412, Boston, 1983. ACM.

[3] L. M. Adleman. Factoring numbers using singular integers. Technical Report TR 90-20, U.S.C. Computer Science Department, September 1990.

[4] L. M. Adleman and M. A. Huang. Recognizing primes in random polynomial time. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 462–469, New York City, 1987. ACM.

[5] L. M. Adleman, C. Pomerance, and R. S. Rumely. On distinguishing prime numbers from composite numbers. *Ann. Math.*, 117:173–206, 1983.

[6] W. B. Alexi, B. Chor, O. Goldreich, and C. P. Schnorr. RSA/Rabin functions: certain parts are as hard as the whole. *SIAM J. Computing*, 17(2):194–209, April 1988.

[7] D. Angluin. Lecture notes on the complexity of some problems in number theory. Technical Report TR-243, Yale University Computer Science Department, August 1982.

[8] Eric Bach. How to generate factored random numbers. *SIAM J. Computing*, 17(2):179–193, April 1988.

[9] D. Balenson. *RFC 1423: Privacy Enhancement for Internet Electronic Mail: Part III – Algorithms, Modes, and Identifiers.* Internet Activities Board, February 1993.

[10] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 420–432. Springer, 1992. Lecture Notes in Computer Science No. 576.

[11] M. Bellare, R. Guérin, and P. Rogaway. XOR MACs: New methods for message authentication using finite pseudorandom functions. In Don Coppersmith, editor, *Proc. CRYPTO 95*, pages 15–28. Springer, 1995. Lecture Notes in Computer Science No. 963.

[12] M. Bellare, J. Kilian, and P. Rogaway. The security of cipher block chaining. In Yvo G. Desmedt, editor, *Proceedings of Crypto 94*, volume 839 of *Lecture Notes in Computer Science*, pages 341–358. Springer-Verlag, 1994. Full version to appear in *J. Computer and System Sciences*, available via `http://www-cse.ucsd.edu/users/mihir`.

[13] M. Bellare, J. Kilian, and P. Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, December 2000.

[14] M. Bellare and S. Micali. How to sign given any trapdoor permutation. *Journal of the ACM*, 39(1):214–233, January 1992.

[15] M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *First ACM Conference on Computer and Communications Security*, pages 62–73, Fairfax, 1993. ACM.

[16] M. Bellare and P. Rogaway. Entity authentication and key distribution. In Douglas R. Stinson, editor, *Proc. CRYPTO 93*, pages 232–249. Springer, 1994. Lecture Notes in Computer Science No. 773.

[17] M. Bellare and P. Rogaway. Provably secure session key distribution– the three party case. In *Proc. 27th ACM Symp. on Theory of Computing*, pages 57–66, Las Vegas, 1995. ACM.

[18] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In *Proceedings of Crypto 96*, volume 1109 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[19] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Pseudorandom functions revisited: The cascade construction and its concrete security. In *Proc. 37th IEEE Symp. on Foundations of Comp. Science*. IEEE, 1996.

[20] Mihir Bellare, Anand Desai, Eron Jokipii, and Phillip Rogaway. A concrete security treatment of symmetric encryption: Analysis of the DES modes of operation. In *Proc. 38th IEEE Symp. on Foundations of Comp. Science*. IEEE, 1997.

[21] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *Proceedings of Crypto 98*, volume 1462 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[22] Mihir Bellare and Phillip Rogaway. Distributing keys with perfect forward secrecy. Manuscript, 1994.

[23] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In *Proceedings of EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[24] Mihir Bellare and Phillip Rogaway. The exact security of digital signatures: How to sign with RSA and Rabin. In *Proceedings of EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[25] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, Chicago, 1988. ACM.

[26] J. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret sharing. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*. Springer, 1987. Lecture Notes in Computer Science No. 263.

[27] J. Benaloh and D. Tuinstra. Receipt-free secret ballot elections. In *26th ACM Symposium on Theory of Computing*, pages 544–553, 1994.

[28] Josh Benaloh. Verifiable secret ballot elections. Technical Report TR–561, Yale Department of Computer Science, September 1987.

[29] R. Berger, R. Peralta, and T. Tedrick. A provably secure oblivious transfer protocol. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proc. EUROCRYPT 84*, pages 379–386. Springer-Verlag, 1985. Lecture Notes in Computer Science No. 209.

[30] D. Bernstein. How to stretch random functions: the security of protected counter sums. *Journal of Cryptology*, 12(3):185–192, 1999.

[31] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic design of two-party authentication protocols. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 44–61. Springer, 1992. Lecture Notes in Computer Science No. 576.

[32] John Black, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. UMAC: fast and secure message authentication. In *Proceedings of CRYPTO'99*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[33] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.

[34] D. Bleichenbacher. A chosen ciphertext attack against protocols based on the RSA encryption standard pkcs #1. In *Proceedings of Crypto 98*, volume 1462 of *Lecture Notes in Computer Science*. Springer-Verlag, 1998.

[35] L. Blum, M. Blum, and M. Shub. A simple unpredictable pseudo-random number generator. *SIAM J. Computing*, 15(2):364–383, May 1986.

[36] M. Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, pages 133–137. IEEE, 1982.

[37] M. Blum. How to exchange (secret) keys. *Trans. Computer Systems*, 1:175–193, May 1983. (Previously published in ACM STOC '83 proceedings, pages 440–447.).

[38] M. Blum. Independent unbiased coin flips from a correlated biased source: A finite state Markov chain. In *Proc. 25th IEEE Symp. on Foundations of Comp. Science*, pages 425–433, Singer Island, 1984. IEEE.

[39] M. Blum and S. Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In G. R. Blakley and D. C. Chaum, editors, *Proc. CRYPTO 84*, pages 289–302. Springer, 1985. Lecture Notes in Computer Science No. 196.

[40] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Computing*, 13(4):850–863, November 1984.

[41] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero-knowledge. *SIAM J. Computing*, 20(6):1084–1118, December 1991.

[42] D. Boneh and R. Venkatesan. Hardness of computing the most significant bits of secret keys in Diffie-Hellman and related schemes. In *Proceedings of CRYPTO'96*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[43] Gilles Brassard and Claude Crépeau. Zero-knowledge simulation of boolean circuits. In A.M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 223–233. Springer-Verlag, 1987. Lecture Notes in Computer Science No. 263.

[44] E. F. Brickell. Solving low density knapsacks. In D. Chaum, editor, *Proc. CRYPTO 83*, pages 25–37, New York, 1984. Plenum Press.

[45] Michael Burrows, Martin Abadi, and Roger Needham. A logic of authentication. *ACM Transactions on Computer Systems*, 8(1):18–36, February 1990.

[46] Ran Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *Proc. CRYPTO 97*, Lecture Notes in Computer Science. Springer-Verlag, 1997.

[47] Ran Canetti and R. Gennaro. Incoercible multiparty computation. In *Proc. 37th IEEE Symp. on Foundations of Comp. Science*, 1996.

[48] E.R. Canfield, P. Erdös, and C. Pomerance. On a problem of Oppenheim concerning 'Factorisatio Numerorum'. *J. Number Theory*, 17:1–28, 1983.

[49] M. Cerecedo, T. Matsumoto, and H. Imai. Efficient and secure multiparty generation of digital signatures based on discrete logarithm. *IEICE Trans. on Fund. Electr. Comm. and Comp. Sci.*, E76–A(4):532–545, 1993.

[50] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24:84–88, February 1981.

[51] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 462–462. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[52] D. Chaum, A. Fiat, and M. Naor. Untraceable electronic cash. In S. Goldwasser, editor, *Proc. CRYPTO 88*, pages 319–327. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 403.

[53] D. L. Chaum. Verification by anonymous monitors. In Allen Gersho, editor, *Advances in Cryptology: A Report on CRYPTO 81*, pages 138–139. U.C. Santa Barbara Dept. of Elec. and Computer Eng., 1982. Tech Report 82-04.

[54] B. Chor and O. Goldreich. Unbiased bits from sources of weak randomness and probabilistic communication complexity. *SIAM J. Computing*, 17(2):230–261, April 1988.

[55] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th IEEE Symp. on Foundations of Comp. Science*, pages 383–395, Portland, 1985. IEEE.

[56] B. Chor and R. L. Rivest. A knapsack type public-key cryptosystem based on arithmetic in finite fields. *IEEE Trans. Inform. Theory*, 34(5):901–909, September 1988.

[57] D. Coppersmith. Evaluating logarithms in $GF(2^n)$. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 201–207, Washington, D.C., 1984. ACM.

[58] D. Coppersmith, M. K. Franklin, J. Patarin, and M. K. Reiter. Low-exponent RSA with related messages. In Ueli Maurer, editor, *Advances in Cryptology - EuroCrypt '96*, pages 1–9, Berlin, 1996. Springer-Verlag. Lecture Notes in Computer Science Volume 1070.

[59] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* MIT Press/McGraw-Hill, 1990.

[60] R. Cramer, M. Franklin, B. Schoenmakers, and M. Yung. Multi-authority secret-ballot elections with linear work. In *EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 72–83. Springer-Verlag, 1996.

[61] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. Theory of Cryptography Library Record 99-01, 1999.

[62] J. Daemen and V. Rijmen. Aes proposal: Rijndael, 2000.

[63] I. Damgård. A design principle for hash functions. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 416–427. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[64] D. Denning and G. Sacco. Timestamps in key distribution protocols. *Communications of the ACM*, 24(8):533–536, 1981.

[65] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 457–469. Springer, 1992. Lecture Notes in Computer Science No. 576.

[66] Yvo G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[67] W. Diffie and M. E. Hellman. Multiuser cryptographic techniques. In *Proc. AFIPS 1976 National Computer Conference*, pages 109–112, Montvale, N.J., 1976. AFIPS.

[68] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[69] Whitfield Diffie, Paul C. Van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges. *Designs, Codes, and Cryptography*, 2(2):107–125, June 1992.

[70] H. Dobbertin. MD5 is not collision-free. Manuscript, 1996.

[71] D. Dolev, C. Dwork, and M. Naor. Non-malleable cryptography. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 542–552. ACM, 1991.

[72] D. Dolev, S. Even, and R. M. Karp. On the security of ping-pong protocols. In R. L. Rivest, A. Sherman, and D. Chaum, editors, *Proc. CRYPTO 82*, pages 177–186, New York, 1983. Plenum Press.

[73] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proc. 22nd IEEE Symp. on Foundations of Comp. Science*, pages 350–357, Nashville, 1981. IEEE.

[74] C. Dwork and M. Naor. An efficient existentially unforgeable signature scheme and its applications. In Yvo G. Desmedt, editor, *Proc. CRYPTO 94*, pages 234–246. Springer, 1994. Lecture Notes in Computer Science No. 839.

[75] P. Elias. The efficient construction of an unbiased random sequence. *Ann. Math. Statist.*, 43(3):865–870, 1972.

[76] S. Even and O. Goldreich. On the security of multi-party ping-pong protocols. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 34–39, Tucson, 1983. IEEE.

[77] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28:637–647, 1985.

[78] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symp. on Foundations of Comp. Science*, pages 427–438, Los Angeles, 1987. IEEE.

[79] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 186–194. Springer, 1987. Lecture Notes in Computer Science No. 263.

[80] R. Fischlin and C. Schnorr. Stronger security proofs for RSA and Rabin bits. In *EUROCRYPT'97*, volume 1223 of *Lecture Notes in Computer Science*, pages 267–279. Springer-Verlag, 1997.

[81] National Institute for Standards and Technology. A proposed federal information processing standard for digital signature standard (DSS). Technical Report FIPS PUB XX, National Institute for Standards and Technology, August 1991. DRAFT.

[82] Y. Frankel, P. Gemmell, and M. Yung. Witness-based cryptographic program checking and robust function sharing. In *28th ACM Symposium on Theory of Computing*, 1996.

[83] A. M. Frieze, J. Hastad, R. Kannan, J. C. Lagarias, and A. Shamir. Reconstructing truncated integer variables satisfying linear congruences. *SIAM J. Computing*, 17(2):262–280, April 1988.

[84] E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is secure under the RSA assumption. In *CRYPTO'01*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[85] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[86] M. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[87] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *EUROCRYPT'99*, Lecture Notes in Computer Science. Springer-Verlag, 1999.

[88] R. Gennaro, S. Jarecki, Hugo Krawczyk, and T. Rabin. Robust and efficient sharing of rsa functions. In *CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[89] R. Gennaro, S. Jarecki, Hugo Krawczyk, and T. Rabin. Robust threshold dss signatures. In *EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 354–371. Springer-Verlag, 1996.

[90] O. Goldreich. Two remarks concerning the Goldwasser-Micali-Rivest signature scheme. Technical Report MIT/LCS/TM-315, MIT Laboratory for Computer Science, September 1986.

[91] O. Goldreich. A uniform complexity treatment of encryption and zero-knowledge. *Journal of Cryptology*, 6(1):21–53, 1993.

[92] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):792–807, October 1984.

[93] O. Goldreich, S. Goldwasser, and S. Micali. On the cryptographic applications of random functions. In G. R. Blakley and D. C. Chaum, editors, *Proc. CRYPTO 84*, pages 276–288. Springer, 1985. Lecture Notes in Computer Science No. 196.

[94] O. Goldreich and L. Levin. A hard-core predicate for all one-way functions. In *21st ACM Symposium on Theory of Computing*, 1989.

[95] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 174–187, Toronto, 1986. IEEE.

[96] S. Goldwasser and J. Kilian. Almost all primes can be quickly certified. In *Proc. 18th ACM Symp. on Theory of Computing*, pages 316–329, Berkeley, 1986. ACM.

[97] S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 365–377, San Francisco, 1982. ACM.

[98] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[99] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 291–304, Providence, 1985. ACM.

[100] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

[101] S. Goldwasser, S. Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[102] S. Goldwasser, S. Micali, and P. Tong. Why and how to establish a private code on a public network. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 134–144, Chicago, 1982. IEEE.

[103] S. Goldwasser, S. Micali, and A. Yao. Strong signature schemes. In *Proc. 15th ACM Symp. on Theory of Computing*, pages 431–439, Boston, 1983. ACM.

[104] S. W. Golomb. *Shift Register Sequences.* Aegean Park Press, Laguna Hills, 1982. Revised edition.

[105] L. Harn. Group–oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proc. Comput. Digit. Tech.*, 141(5):307–313, 1994.

[106] J. Hastad. Solving simultaneous modular equations of low degree. *SIAM J. Computing*, 17(2):336–341, April 1988.

[107] J. Håstad. Pseudo-random generators under uniform assumptions. In *22nd ACM Symposium on Theory of Computing*, 1990.

[108] J. Hastad, A.W. Schrift, and A. Shamir. The discrete logarithm modulo a composite hides $o(n)$ bits. *Journal of Computer and Systems Sciences*, 47:376–404, 1993.

[109] Johan Håstad, Russell Impagliazzo, Leonid Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[110] R. Impagliazzo and M. Luby. One-way functions are essential for complexity based cryptography. In *Proc. 30th IEEE Symp. on Foundations of Comp. Science*, 1989.

[111] Russell Impagliazzo, Leonid A. Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 12–24, Seattle, 1989. ACM.

[112] ISO. Data cryptographic techniques – data integrity mechanism using a cryptographic check function employing a block cipher algorithm. ISO/IEC 9797, 1989.

[113] D. Johnson, A. Lee, W. Martin, S. Matyas, and J. Wilkins. Hybrid key distribution scheme giving key record recovery. *IBM Technical Disclosure Bulletin*, 37(2A):5–16, February 1994. See also U.S. Patent 5,142,578.

[114] D. Johnson and M. Matyas. Asymmetric encryption: Evolution and enhancements. *RSA Labs Cryptobytes*, 2(1), Spring 1996.

[115] B. Kaliski and M. Robshaw. Message authentication with MD5. *CryptoBytes*, 1(1):5–8, Spring 1995.

[116] B. S. Kaliski, Jr. A pseudo-random bit generator based on elliptic logarithms. In A.M. Odlyzko, editor, *Proc. CRYPTO 86*, pages 84–103. Springer-Verlag, 1987. Lecture Notes in Computer Science No. 263.

[117] B. S. Kaliski, Jr. *Elliptic Curves and Cryptography: A Pseudorandom Bit Generator and Other Tools.* PhD thesis, MIT EECS Dept., January 1988. Published as MIT LCS Technical Report MIT/LCS/TR-411 (Jan. 1988).

[118] R. Kannan, A. Lenstra, and L. Lovász. Polynomial factorization and non-randomness of bits of algebraic and some transcendental numbers. In *Proc. 16th ACM Symp. on Theory of Computing*, pages 191–200, Washington, D.C., 1984. ACM.

[119] J. Kilian. Founding cryptography on oblivious transfer. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 20–31, Chicago, 1988. ACM.

[120] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, 1969. Second edition, 1981.

[121] Hugo Krawczyk. Skeme: A versatile secure key exchange mechanism for internet. In *Proceedings of the Symposium on Network and Distributed System Security*, 1996.

[122] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. Hmac: Keyed-hashing for message authentication, February 1997. Internet RFC 2104.

[123] J. C. Lagarias and A. M. Odlyzko. Solving low-density subset sum problems. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 1–10, Tucson, 1983. IEEE.

[124] X. Lai and J. Massey. A proposal for a new block encryption standard. In I.B. Damgård, editor, *Proc. EUROCRYPT 90*, pages 389–404. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 473.

[125] L. Lamport. Constructing digital signatures from a one-way function. Technical Report CSL-98, SRI International, October 1979.

[126] A. K. Lenstra and H. W. Lenstra, Jr. Algorithms in number theory. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Volume A: Algorithms and Complexity)*, chapter 12, pages 673–715. Elsevier and MIT Press, 1990.

[127] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Ann.*, 261:513–534, 1982.

[128] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 564–572, Baltimore, Maryland, 1990. ACM.

[129] H. W. Lenstra, Jr. Factoring integers with elliptic curves. *Annals of Mathematics*, 126:649–673, 1987.

[130] R. Lipton. How to cheat at mental poker. In *Proc. AMS Short Course on Cryptography*, 1981.

[131] D. L. Long and A. Wigderson. The discrete logarithm problem hides $O(\log n)$ bits. *SIAM J. Computing*, 17(2):363–372, April 1988.

[132] M. Luby, S. Micali, and C. Rackoff. How to simultaneously exchange a secret bit by flipping a symmetrically biased coin. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 11–22, Tucson, 1983. IEEE.

[133] M. Luby and C. Rackoff. How to construct pseudorandom permutations and pseudorandom functions. *SIAM J. Computing*, 17(2):373–386, April 1988.

[134] Maurice P. Luby and C. Rackoff. A study of password security. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 392–397. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[135] Ueli M. Maurer. Towards the equivalence of breaking the Diffie-Hellman protocol and computing discrete algorithms. In Yvo G. Desmedt, editor, *Proc. CRYPTO 94*, pages 271–281. Springer, 1994. Lecture Notes in Computer Science No. 839.

[136] R. J. McEliece. *A Public-Key System Based on Algebraic Coding Theory*, pages 114–116. Jet Propulsion Lab, 1978. DSN Progress Report 44.

[137] R. Merkle and M. Hellman. Hiding information and signatures in trapdoor knapsacks. *IEEE Trans. Inform. Theory*, IT-24:525–530, September 1978.

[138] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 218–238. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[139] Ralph Charles Merkle. Secrecy, authentication, and public key systems. Technical report, Stanford University, Jun 1979.

[140] M. Merritt. *Cryptographic Protocols*. PhD thesis, Georgia Institute of Technology, February 1983.

[141] S. Micali, C. Rackoff, and R. H. Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. Computing*, 17(2):412–426, April 1988.

[142] Gary L. Miller. Riemann's hypothesis and tests for primality. *JCSS*, 13(3):300–317, 1976.

[143] M. Naor and O. Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. In *Proc. 36th IEEE Symp. on Foundations of Comp. Science*. IEEE, 1995.

[144] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Proc. 38th IEEE Symp. on Foundations of Comp. Science*. IEEE, 1997.

[145] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 33–43, Seattle, 1989. ACM.

[146] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attack. In *Proc. of the Twenty-Second Annual ACM Symposium on Theory of Computing*, pages 427–437, Baltimore, Maryland, 1990. ACM.

[147] National Institute of Standards and Technology (NIST). *FIPS Publication 46: Announcing the Data Encryption Standard*, January 1977. Originally issued by National Bureau of Standards.

[148] R. M. Needham and M. D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21(12):993–999, December 1978.

[149] R. M. Needham and M. D. Schroeder. Authentication revisited. *Operating Systems Review*, 21(1):7, January 1987.

[150] I. Niven and H. S. Zuckerman. *An Introduction to the Theory of Numbers.* Wiley, 1972.

[151] A. M. Odlyzko. Cryptanalytic attacks on the multiplicative knapsack scheme and on Shamir's fast signature scheme. *IEEE Trans. Inform. Theory*, IT-30:594–601, July 1984.

[152] A. M. Odlyzko. Discrete logarithms in finite fields and their cryptographic significance. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Proc. EUROCRYPT 84*, pages 224–314, Paris, 1985. Springer. Lecture Notes in Computer Science No. 209.

[153] C. Park and K. Kurosawa. New elgamal type threshold signature scheme. *IEICE Trans. on Fund. Electr. Comm. and Comp. Sci.*, E79–A(1):86–93, 1996.

[154] T. Pedersen. Distributed provers with applications to undeniable signatures. In *EuroCrypt'91*, 1991.

[155] T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 129–140. Springer, 1992. Lecture Notes in Computer Science No. 576.

[156] E. Petrank and C. Rackoff. Cbc mac for real-time data sources. Manuscript, 1997.

[157] J. Plumstead. Inferring a sequence generated by a linear congruence. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 153–159, Chicago, 1982. IEEE.

[158] S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Trans. Inform. Theory*, IT-24:106–110, January 1978.

[159] D. Pointcheval and J. Stern. Security proofs for signatures. In *Proceedings of EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 387–398. Springer-Verlag, 1996.

[160] J. M. Pollard. Theorems on factorization and primality testing. *Proc. Cambridge Philosophical Society*, 76:521–528, 1974.

[161] V. Pratt. Every prime has a succinct certificate. *SIAM J. Comput.*, 4:214–220, 1975.

[162] B. Preneel and P.C. van Oorschot. On the security of two MAC algorithms. In *Proceedings of EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 19–32. Springer-Verlag, 1996.

[163] Bart Preneel and Paul C. van Oorschot. MDx-MAC and building fast MACs from hash functions. In Don Coppersmith, editor, *Proc. CRYPTO 94*, pages 1–14. Springer, 1995. Lecture Notes in Computer Science No. 963.

[164] M. Rabin. Digitalized signatures as intractable as factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, January 1979.

[165] M. Rabin. Probabilistic algorithms for testing primality. *J. Number Theory*, 12:128–138, 1980.

[166] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Harvard Aiken Computation Laboratory, 1981.

[167] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st ACM Symposium on Theory of Computing*, pages 73–85, 1989.

[168] R. L. Rivest and A. Shamir. How to expose an eavesdropper. *Communications of the ACM*, 27:393–395, April 1984.

[169] Ronald L. Rivest. The MD5 message-digest algorithm. Internet Request for Comments, April 1992. RFC 1321.

[170] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[171] John Rompel. One-way functions are necessary and sufficient for secure signatures. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 387–394, Baltimore, Maryland, 1990. ACM.

[172] J. Rosser and L. Schoenfield. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, 6:64–94, 1962.

[173] RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard*, June 1991. Version 1.4.

[174] RSA Data Security, Inc. *PKCS #7: Cryptographic Message Syntax Standard*, June 1991. Version 1.4.

[175] K. Sako and J. Kilian. Receipt-free mix-type voting schemes. a practical implementation of a voting booth. In *EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 393–403. Springer-Verlag, 1995.

[176] M. Santha and U. V. Vazirani. Generating quasi-random sequences from slightly-random sources. In *Proc. 25th IEEE Symp. on Foundations of Comp. Science*, pages 434–440, Singer Island, 1984. IEEE.

[177] Alredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 522–533, Montreal, Canada, 1994. ACM.

[178] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[179] R. J. Schoof. Elliptic curves over finite fields and the computation of square roots mod p. *Math. Comp.*, 44:483–494, 1985.

[180] R. Schroeppel and A. Shamir. A $TS^2 = O(2^n)$ time/space tradeoff for certain NP-complete problems. In *Proc. 20th IEEE Symp. on Foundations of Comp. Science*, pages 328–336, San Juan, Puerto Rico, 1979. IEEE.

[181] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

[182] A. Shamir. On the cryptocomplexity of knapsack schemes. In *Proc. 11th ACM Symp. on Theory of Computing*, pages 118–129, Atlanta, 1979. ACM.

[183] A. Shamir. On the generation of cryptographically strong pseudo-random sequences. In *Proc. ICALP*, pages 544–550. Springer, 1981.

[184] A. Shamir. A polynomial-time algorithm for breaking the basic Merkle-Hellman cryptosystem. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 145–152, Chicago, 1982. IEEE.

[185] A. Shamir, R. L. Rivest, and L. M. Adleman. Mental poker. In D. Klarner, editor, *The Mathematical Gardner*, pages 37–43. Wadsworth, Belmont, California, 1981.

[186] C. E. Shannon. A mathematical theory of communication. *Bell Sys. Tech. J.*, 27:623–656, 1948.

[187] C. E. Shannon. Communication theory of secrecy systems. *Bell Sys. Tech. J.*, 28:657–715, 1949.

[188] V. Shoup. Oaep reconsidered. In *CRYPTO'01*, Lecture Notes in Computer Science. Springer-Verlag, 2001.

[189] V. Shoup and A. Rubin. Session key distribution for smart cards. In U. Maurer, editor, *Proc. CRYPTO 96*. Springer-Verlag, 1996. Lecture Notes in Computer Science No. 1070.

[190] R.D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48:329–339, 1987.

[191] R. Solovay and V. Strassen. A fast Monte-Carlo test for primality. *SIAM J. Computing*, 6:84–85, 1977.

[192] William Stallings. *Network and Internetwork Security Principles and Practice.* Prentice Hall, 1995.

[193] J.G. Steiner, B.C. Neuman, and J.I. Schiller. Kerberos: an authentication service for open network systems. In *Usenix Conference Proceedings*, pages 191–202, Dallas, Texas, February 1988.

[194] Joseph D. Touch. Performance analysis of MD5. *Proceedings SIGCOMM*, 25(4):77–86, October 1995. Also at `ftp://ftp.isi.edu/pub/hpcc-papers/touch/sigcomm95.ps.Z`.

[195] Gene Tsudik. Message authentication with one-way hash functions. *ACM SIGCOMM, Computer Communication Review*, 22(5):29–38, October 1992.

[196] P. van Oorschot and M. Wiener. Parallel collision search with applications to hash functions and discrete logarithms. In *Proceedings of the 2nd ACM Conf. Computer and Communications Security*, November 1994.

[197] U. V. Vazirani. Towards a strong communication complexity theory, or generating quasi-random sequences from two communicating slightly-random sources. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 366–378, Providence, 1985. ACM.

[198] U. V. Vazirani and V. V. Vazirani. Trapdoor pseudo-random number generators, with applications to protocol design. In *Proc. 24th IEEE Symp. on Foundations of Comp. Science*, pages 23–30, Tucson, 1983. IEEE.

[199] Umesh V. Vazirani and Vijay V. Vazirani. RSA bits are $732 + \epsilon$ secure. In D. Chaum, editor, *Proc. CRYPTO 83*, pages 369–375, New York, 1984. Plenum Press.

[200] J. von Neumann. Various techniques for use in connection with random digits. In *von Neumann's Collected Works*, volume 5, pages 768–770. Pergamon, 1963.

[201] A. C. Yao. Theory and application of trapdoor functions. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 80–91, Chicago, 1982. IEEE.

[202] A.C. Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago, 1982. IEEE.

[203] A.C. Yao. How to generate and exchange secrets. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, pages 162–167, Toronto, 1986. IEEE.

Chapter A

Some probabilistic facts

A.1 The birthday problem

Some of our estimates in Chapters 6, 8 and 5 require precise bounds on the birthday probabilities, which for completeness we derive here, following [12].

The setting is that we have q balls. View them as numbered, $1, \ldots, q$. We also have N bins, where $N \geq q$. We throw the balls at random into the bins, one by one, beginning with ball 1. At random means that each ball is equally likely to land in any of the N bins, and the probabilities for all the balls are independent. A collision is said to occur if some bin ends up containing at least two balls. We are interested in $C(N, q)$, the probability of a collision.

The birthday phenomenon takes its name from the case when $N = 365$, whence we are asking what is the chance that, in a group of q people, there are two people with the same birthday, assuming birthdays are randomly and independently distributed over the 365 days of the year. It turns out that when q hits $\sqrt{365} \approx 19.1$ the chance of a collision is already quite high; for example at $q = 20$ the chance of a collision is at least 0.328.

The birthday phenomenon can seem surprising when first heard; that's why it is called a paradox. The reason it is true is that the collision probability $C(N, q)$ grows roughly proportional to q^2/N. This is the fact to remember. The following gives a more exact rendering, providing both upper and lower bounds on this probability.

Proposition A.1 Let $C(N, q)$ denote the probability of at least one collision when we throw $q \geq 1$ balls at random into $N \geq q$ buckets. Then

$$C(N, q) \quad \leq \quad \frac{q(q-1)}{2N} .$$

Also

$$C(N, q) \quad \geq \quad 1 - e^{-q(q-1)/2N} ,$$

and for $1 \leq q \leq \sqrt{2N}$

$$C(N, q) \quad \geq \quad 0.3 \cdot \frac{q(q-1)}{N} .$$

▮

In the proof we will find the following inequalities useful to make estimates.

Proposition A.2 For any real number $x \in [0,1]$—

$$\left(1 - \frac{1}{e}\right) \cdot x \;\leq\; 1 - e^{-x} \;\leq\; x \,.$$

▮

Proof of Proposition A.1: Let C_i be the event that the i-th ball collides with one of the previous ones. Then $\mathbf{P}\left[C_i\right]$ is at most $(i-1)/N$, since when the i-th ball is thrown in, there are at most $i-1$ different occupied slots and the i-th ball is equally likely to land in any of them. Now

$$\begin{aligned} C(N,q) &= \mathbf{P}\left[C_1 \vee C_2 \vee \cdots \vee C_q\right] \\ &\leq \mathbf{P}\left[C_1\right] + \mathbf{P}\left[C_2\right] + \cdots + \mathbf{P}\left[C_q\right] \\ &\leq \frac{0}{N} + \frac{1}{N} + \cdots + \frac{q-1}{N} \\ &= \frac{q(q-1)}{2N} \,. \end{aligned}$$

This proves the upper bound. For the lower bound we let D_i be the event that there is no collision after having thrown in the i-th ball. If there is no collision after throwing in i balls then they must all be occupying different slots, so the probability of no collision upon throwing in the $(i+1)$-st ball is exactly $(N-i)/N$. That is,

$$\mathbf{P}\left[D_{i+1} \mid D_i\right] \;=\; \frac{N-i}{N} \;=\; 1 - \frac{i}{N} \,.$$

Also note $\mathbf{P}\left[D_1\right] = 1$. The probability of no collision at the end of the game can now be computed via

$$\begin{aligned} 1 - C(N,q) &= \mathbf{P}\left[D_q\right] \\ &= \mathbf{P}\left[D_q \mid D_{q-1}\right] \cdot \mathbf{P}\left[D_{q-1}\right] \\ &\vdots \quad \vdots \\ &= \prod_{i=1}^{q-1} \mathbf{P}\left[D_{i+1} \mid D_i\right] \\ &= \prod_{i=1}^{q-1} \left(1 - \frac{i}{N}\right) \,. \end{aligned}$$

Note that $i/N \leq 1$. So we can use the inequality $1 - x \leq e^{-x}$ for each term of the above expression. This means the above is not more than

$$\prod_{i=1}^{q-1} e^{-i/N} \;=\; e^{-1/N - 2/N - \cdots - (q-1)/N} \;=\; e^{-q(q-1)/2N} \,.$$

Putting all this together we get

$$C(N,q) \;\geq\; 1 - e^{-q(q-1)/2N} \,,$$

which is the second inequality in Proposition A.1. To get the last one, we need to make some more estimates. We know $q(q-1)/2N \leq 1$ because $q \leq \sqrt{2N}$, so we can use the inequality $1 - e^{-x} \geq (1 - e^{-1})x$ to get

$$C(N,q) \;\geq\; \left(1 - \frac{1}{e}\right) \cdot \frac{q(q-1)}{2N} \,.$$

A computation of the constant here completes the proof. ▮

Chapter B

Some complexity theory background

As of today, we do not even know how to prove a linear lower bound on the time required to solve an NP-complete problem. Thus, in our development of a theory of cryptography in the presence of a computationally bounded adversary we must resort to making assumptions about the existence of hard problems. In fact, an important current research topic in cryptography (on which much progress has been made in recent years) is to find the minimal assumptions required to prove the existence of "secure" cryptosystems.

Our assumptions should enable us to quickly generate instances of problems which are hard to solve for anyone other than the person who generated the instance. For example, it should be easy for the sender of a message to generate a ciphertext which is hard to decrypt for any adversary (naturally, in this example, it should also be easy for the intended recipient of the message to decrypt the ciphertext). To formally describe our assumptions (the existence of one way functions and trapdoor function) we first need to recall some complexity theory definitions.

B.1 Complexity Classes and Standard Definitions

B.1.1 Complexity Class P

A language L is in P if and only if there exists a Turing machine $M(x)$ and a polynomial function $Q(y)$ such that on input string x

1. $x \in L$ iff M accepts x (denoted by $M(x)$).
2. M terminates after at most $Q(|x|)$ steps.

The class of languages P is classically considered to be those languages which are 'easily computable'. We will use this term to refer to these languages and the term 'efficient algorithm' to refer to a polynomial time Turing machine.

B.1.2 Complexity Class NP

A language L is in NP if and only if there exists a Turing machine $M(x, y)$ and polynomials p and l such that on input string x

1. $x \in L \Rightarrow \exists y$ with $|y| \leq l(|x|)$ such that $M(x,y)$ accepts and M terminates after at most $p(|x|)$ steps.
2. $x \notin L \Rightarrow \forall y$ with $|y| \leq l(|x|)$, $M(x,y)$ rejects.

Note that this is equivalent to the (perhaps more familiar) definition of $L \in$ NP if there exists a non-deterministic polynomial time Turing machine M which accepts x if and only if $x \in L$. The string y above corresponds to the *guess* of the non-determinstic Turing machine.

B.1.3 Complexity Class BPP

A language L is in BPP if and only if there exists a Turing machine $M(x,y)$ and polynomials p and l such that on input string x

1. $x \in L \Rightarrow \Pr_{|y|<l(|x|)}[M(x,y)\text{accepts}] \geq \frac{2}{3}$.
2. $x \notin L \Rightarrow \Pr_{|y|<l(|x|)}[M(x,y)\text{accepts}] \leq \frac{1}{3}$.
3. $M(x,y)$ always terminates after at most $p(|x|)$ steps.

As an exercise, you may try to show that if the constants $\frac{2}{3}$ and $\frac{1}{3}$ are replaced by $\frac{1}{2} + \frac{1}{p(|x|)}$ and $\frac{1}{2} - \frac{1}{p(|x|)}$ where p is any fixed polynomial then the class BPP remains the same.

Hint: Simply run the machine $M(x,y)$ on "many" y's and accept if and only if the majority of the runs accept. The magnitude of "many" depends on the polynomial p.

We know that P $\subseteq$ NP and P $\subseteq$ BPP. We do not know if these containments are strict although it is often conjectured to be the case. An example of a language known to be in BPP but not known to be in P is the language of all prime integers (that is, primality testing). It is not known whether BPP is a subset of NP.

B.2 Probabilistic Algorithms

The class BPP could be alternatively defined using probabilistic Turing machines (probabilistic algorithms). A *probabilistic polynomial time Turing machine* M is a Turing machine which can flip coins as an additional primitive step, and on input string x runs for at most a polynomial in $|x|$ steps. We could have defined BPP by saying that a language L is in BPP if there exists a probabilistic polynomial time Turing machine $M(x)$ such that when $x \in L$, the probability (over the coin tosses of the machine) that $M(x)$ accepts is greater than $\frac{2}{3}$ and when $x \notin L$ the probability (over the coin tosses of the machine) that $M(x)$ rejects is greater than $\frac{2}{3}$. The string y in the previous definition corresponds to the sequence of coin flips made by the machine M on input x.

From now on we will consider probabilistic polynomial time Turing machines as "efficient algorithms" (extending the term previously used for deterministic polynomial time Turing machines). We also call the class of languages in BPP "easily computable". Note the difference between a non-deterministic Turing machine and a probabilistic Turing machine. A non-deterministic machine is not something we could implement in practice (as there may be only one good guess y which will make us accept). A probabilistic machine is something we could implement in practice by flipping coins to yield the string y (assuming of course that there is a source of coin flips in nature). Some notation is useful when talking about probabilistic Turing machines.

B.2.1 Notation For Probabilistic Turing Machines

Let M denote a probabilistic Turing machine (PTM). $M(x)$ will denote a probability space of the outcome of M during its run on x. The statement $z \in M(x)$ indicates that z was output by M when running on input

x. $\Pr[M(x) = z]$ is the probability of z being the output of M on input x (where the probability is taken over the possible internal coin tosses made by M during its execution). $M(x, y)$ will denote the outcome of M on input x when internal coin tosses are y.

B.2.2 Different Types of Probabilistic Algorithms

Monte Carlo algorithms and Las Vegas algorithms are two different types of probabilistic algorithms. The difference between these two types is that a Monte Carlo algorithm always terminates within a polynomial number of steps but its output is only correct with high probability whereas a Las Vegas algorithm terminates within an expected polynomial number of steps and its output is always correct. Formally, we define these algorithms as follows.

Definition B.1 A Monte Carlo algorithm is a probabilistic algorithm M for which there exists a polynomial P such that for all x, M terminates within $P(|x|)$ steps on input x. Further,

$$\Pr[\mathrm{M}(x) \text{ is correct }] > \frac{2}{3}$$

(where the probability is taken over the coin tosses of M).

A Las Vegas algorithm is a probabilistic algorithm M for which there exists a polynomial p such that for all x, E(running time) $= \sum_{t=1}^{\infty} t \cdot \Pr[\mathrm{M}(x) \text{ takes exactly } t \text{ steps}] < p(|x|)$. Further, the output of M(x) is always correct. ▮

All Las Vegas algorithms can be converted to Monte Carlo algorithms but it is unknown whether all Monte Carlo algorithms can be converted to Las Vegas algorithms. Some examples of Monte Carlo algorithms are primality tests such as Solovay Strassen (see [191]) or Miller-Rabin (see [165]) and testing the equivalence of multivariate polynomials and some examples of Las Vegas algorithms are computing square roots modulo a prime p, computing square roots modulo a composite n (if the factors of n are known) and primality tests based on elliptic curves (see [4] or [96]).

B.2.3 Non-Uniform Polynomial Time

An important concept is that of polynomial time algorithms which can behave differently for inputs of different size, and may even be polynomial in the size of the input (rather than constant as in the traditional definition of a Turing machine).

Definition B.2 A *non-uniform algorithm* A is an infinite sequence of algorithms $\{M_i\}$ (one for each input size i) such that on input x, $M_{|x|}(x)$ is run. We say that $A(x)$ accepts if and only if $M_{|x|}(x)$ accepts. We say that A is a *polynomial time* non-uniform algorithm if there exist polynomials P and Q such that $M_{|x|}(x)$ terminates within $P(|x|)$ steps and the size of the description of M_i (according to some standard encoding of all algorithms) is bounded by $Q(|i|)$. ▮

Definition B.3 We say that a language L is in P/poly if $\exists$ a polynomial time non-uniform algorithm $A = \{M_i\}$ such that $x \in L$ iff $M_{|x|}(x)$ accepts. ▮

There are several relationships known about P/poly. It is clear that $P \subset$ P/poly and it has been shown by Adleman that BPP $\subset$ P/poly.

We will use the term 'efficient non-uniform algorithm' to refer to a non-uniform polynomial time algorithm and the term 'efficiently non-uniform computable' to refer to languages in the class P/poly.

B.3 Adversaries

We will model the computational power of the adversary in two ways. The first is the (uniform) adversary (we will usually drop the "uniform" when referring to it). A *uniform adversary* is any polynomial time probabilistic algorithm. A *non-uniform adversary* is any non-uniform polynomial time algorithm. Thus, the adversary can use different algorithms for different sized inputs. Clearly, the non-uniform adversary is stronger than the uniform one. Thus to prove that "something" is "secure" even in presence of a non-uniform adversary is a better result than only proving it is secure in presence of a uniform adversary.

B.3.1 Assumptions To Be Made

The weakest assumption that must be made for cryptography in the presence of a uniform adversary is that $\mathrm{P} \neq \mathrm{NP}$. Namely, $\exists L \in \mathrm{NP}$ such that $L \notin \mathrm{P}$. Unfortunately, this is not enough as we assumed that our adversaries can use probabilistic polynomial time algorithms. So we further assume that $\mathrm{BPP} \neq \mathrm{NP}$. Is that sufficient? Well, we actually need that it would be hard for an adversary to crack our systems most of the time. It is not sufficient that our adversary can not crack the system once in a while. Assuming that $\mathrm{BPP} \neq \mathrm{NP}$ only means that there exists a language in $L \in \mathrm{NP}$ such that every uniform adversary makes (with high probability) the wrong decision about infinitely many inputs x when deciding whether $x \in L$. These wrong decisions, although infinite in number, may occur very infrequently (such as once for each input size).

We thus need yet a stronger assumption which will guarantee the following. There exists a language $L \in \mathrm{NP}$ such that for every sufficiently large input size n, every uniform adversary makes (with high probability) the wrong decision on many inputs x of length n when deciding whether x is in L. Moreover, we want it to be possible, for every input size n, to generate input x of length n such that with high probability every uniform adversary will make the wrong decision on x.

The assumption that will guarantee the above is the existence of (uniform) one way functions. The assumption that would guarantee the above in the presence of non-uniform adversary is the existence non-uniform one way functions. For definitions, properties, and possible examples of one-way functions see Chapter 2.

B.4 Some Inequalities From Probability Theory

Proposition B.4 [Markov's Inequality] If Z is a random variable that takes only non-negative values, then for any value $a > 0$, $\Pr[Z \geq a] \leq \frac{E[Z]}{a}$.

▮

Proposition B.5 [Weak Law of Large Numbers] Let $z_1, \ldots, z_n$ be independent 0-1 random variables (Bernoulli random variables) with mean μ. Then $\Pr[|\frac{\sum_{i=1}^n z_i}{n} - \mu| < \epsilon] > 1 - \delta$ provided that $n > \frac{1}{4\epsilon^2\delta}$. ▮

CHAPTER C

Some number theory background

Many important constructions of cryptographic primitives are based on problems from number theory which seem to be computationally intractable. The most well-known of these problems is that of factoring composite integers. In order to work with these problems, we need to develop some basic materiel on number theory and number theoretic algorithms. Accordingly, we provide here a mini-course on this subject. The materiel here will be used later when we discuss candidate example one-way and trapdoor functions.

There are many sources for information on number theory in the literatures. For example try Angluin's notes [7] and Chapter 33 of Cormen, Leiserson and Rivest [59].

C.1 Groups: Basics

A *group* is a set G together with some operation, which we denote $*$. It takes pairs of elements to another element, namely $a * b$ is the result of $*$ applied to a, b. A group has the following properties:

(1) If $a, b \in G$ so is $a * b$

(2) The operation is associative: $(a * b) * c = a * (b * c)$

(3) There is an *identity element* I such that $I * a = a * I = a$ for all $a \in G$

(4) Every $a \in G$ has an inverse, denoted a^{-1}, such that $a * a^{-1} = a^{-1} * a = I$.

We will encounter this kind of structure a lot. First recall Z, N and $\mathbf{R}$. Now, for example:

- Integers under addition: $I = 0$; $a^{-1} = -a$.
- Real numbers under multiplication: $I = 1$; $a^{-1} = 1/a$.
- What about N under addition? Not a group!
- What about Z under multiplication? Not a group!

Notation: a^m is a multiplied by itself m times. Etc. Namely, notation is what you expect. What is a^{-m}? It is $(a^{-1})^m$. Note that it "works" like it should.

These groups are all infinite. We are usually interested in finite ones. In such a case:

Def: We call $|G|$ the order of G.

Fact C.1 Let $m = |G|$. Then $a^m = I$ for any $a \in G$. ▮

We will use this later.

$a \equiv b \pmod{n}$ means that if we divide a by n then the remainder is b. (In C, this is $a\%n = b$).

An important set is the set of integers modulo an integer n. This is $Z_n = \{0, \ldots, n-1\}$. We will see it is a group under addition modulo n. Another related important set is $Z_n^* = \{m : 1 \leq m \leq n \text{ and } \gcd(m,n) = 1\}$, the set of integers less than n which are relatively prime to n. We will see this is a group under multiplication modulo n. We let $\phi(n) = |Z_n^*|$. This is the Euler totient function.

A subset $S \subseteq G$ is called a sub-group if it is a group in its own right, under the operation making G a group. In particular if $x, y \in S$ so is xy and x^{-1}, and $1 \in S$.

Fact C.2 Suppose S is a subgroup of G. Then $|S|$ divides $|G|$. ▮

C.2 Arithmatic of numbers: +, *, GCD

Complexity of algorithms operating of a number a is measured in terms of the size (length) of a, which is $|a| \approx \lg(a)$. How long do basic operations take? In terms of the number of bits k in the number:

- Addition is linear time. Ie. two k-bit numbers can be added in $O(k)$ time.
- Multiplication of a and b takes $O(|a| \cdot |b|)$ bit operations. Namely it is an $O(k^2)$ algorithm.
- Division of a by b (integer division: we get back the quotient and remainder) takes time $O((1 + |q|)|b|)$ where q is the quotient obtained. Thus this too is a quadratic time algorithm.

Euclid's algorithm can be used to compute GCDs in polynomial time. The way it works is to repeatedly use the identity $\gcd(a, b) = \gcd(b, a \bmod b)$. For examples, see page 10 of [7].

What is the running time? Each division stage takes quadratic time and we have k stages, which would say it is a $O(k^3)$ algorithm. But see Problem 33-2, page 850, of [59]. This yields the following:

Theorem C.3 Euclid's algorithm can be implemented to use only $O(|a| \cdot |b|)$ bit operations to compute $\gcd(a, b)$. That is, for k-bit numbers we get a $O(k^2)$ algorithm. ▮

Fact C.4 $\gcd(a, b) = 1$ if and only if there exist integers u, v such that $1 = au + bv$. ▮

The Extended Euclid Algorithm is given a, b and it returns not only $d = \gcd(a, b)$ but integers u, v such that $d = au + bv$. It is very similar to Euclid's algorithm. We can keep track of some extra information at each step. See page 11 of [7].

C.3 Modular operations and groups

C.3.1 Simple operations

Now we go to modular operations, which are the main ones we are interested in

- Addition is now the following: Given a, b, n with $a, b \in Z_n$ compute $a + b \bmod n$. This is still linear time. Ie. two k-bit numbers can be added in $O(k)$ time. Why? You can't go much over N. If you do, just subtract n. That too is linear time.
- Taking $a \bmod n$ means divide a by n and take remainder. Thus, it takes quadratic time.

- Multiplication of a and b modulo n: First multiply them, which takes $O(|a| \cdot |b|)$ bit operations. Then divide by n and take the remainder. We saw latter too was quadratic. So the whole thing is quadratic.

Z_n is a group under addition modulo N. This means you can add two elements and get back an element of the set, and also subtraction is possible. Under addition, things work like you expect.

We now move to Z_n^*. We are interested in the multiplication operation here. We want to see that it is a group, in the sense that you can multiply and divide. We already saw how to multiply.

Theorem C.5 There is a $O(k^2)$ algorithm which given a, n with $a \in Z_n^*$ outputs $b \in Z_n^*$ satisfying $ab \equiv 1 \pmod{n}$, where $k = |n|$. ▮

See page 12 of [7]. The algorithms uses the extended Euclid. We know that $1 = \gcd(a, n)$. Hence it can find integers u, v such that $1 = au + nv$. Take this modulo n and we get $au \equiv 1 \pmod{n}$. So can set $b = u \bmod n$. Why is this an element of Z_n^*? Claim that $\gcd(u, n) = 1$. Why? By Fact C.4, which says that $1 = au + nv$ means $\gcd(u, n) = 1$.

The b found in the theorem can be shown to be unique. Hence:

Notation: The b found in the theorem is denoted a^{-1}.

C.3.2 The main groups: Z_n and Z_n^*

Theorem C.6 For any positive integer n, Z_n^* forms a group under multiplication modulo n. ▮

This means that $a, b \in Z_n^*$ implies $ab \bmod n$ is in Z_n^*, something one can verify without too much difficulty. It also means we can multiply and divide. We have an identity (namely 1) and a cancellation law.

Notation: We typically stop writing mod$n$ everywere real quick. It should just be understood.

The way to think about Z_n^* is like the real numbers. You can manipulate things like you are used to. The following is a corollary of Fact C.1.

Theorem C.7 For any $a \in Z_n^*$ it is the case that $a^{\phi(n)} = 1$. ▮

Corollary C.8 (Fermat's little theorem) If p is prime then $a^{p-1} \equiv 1 \pmod{p}$ for any $a \in \{1, \ldots, p-1\}$. ▮

Why? Because $\phi(p) = p - 1$.

C.3.3 Exponentiation

This is the most basic operation for public key cryptography. The operation is just that given a, n, m where $a \in Z_n$ and m is an integer, computes $a^m \bmod n$.

Example C.9 Compute $2^{21} \bmod 22$. Naive way: use 21 multiplications. What's the problem with this? It is an *exponential* time algorithm. Because we want time poly(k) where $k = |n|$. So we do it by repeated squaring:

$$\begin{aligned} 2^1 &\equiv 2 \\ 2^2 &\equiv 4 \\ 2^4 &\equiv 16 \\ 2^8 &\equiv 14 \\ 2^{16} &\equiv 20 \end{aligned}$$

Now $2^{21} = 2^{16+4+1} = 2^{16} * 2^4 * 2^1 = 20 * 16 * 2 \equiv 10 \bmod 21$. ∎

This is the repeated squaring algorithm for exponentiation. It takes cubic time.

Theorem C.10 There is an algorithm which given a, n, m with $a, m \in Z_n$ outputs $a^m \bmod n$ in time $O(k^3)$ where $k = |n|$. More precisely, the algorithm uses at most $2k$ modular multiplications of k-bit numbers. ∎

Algorithm looks at binary expansion of m. In example above we have $21 = 10101$. What we did is collect all the powers of two corresponding to the ones and multiply them.

4	3	2	1	0
a^{16}	a^8	a^4	a^2	a^1
1	0	1	0	1

Exponentiate(a, n, m)

Let $b_{k-1}...b_1b_0$ be the binary representation of m
Let $x_0 = a$
Let $y = 1$
For $i = 0, \ldots, k-1$ do
  If $b_i = 1$ let $y = y * x_i \bmod n$
  Let $x_{i+1} = x_i^2 \bmod n$
Output y

C.4 Chinese remainders

Let $m = m_1 m_2$. Suppose $y \in Z_m$. Consider the numbers

$$\begin{aligned} a_1 &= y \bmod m_1 \quad \in Z_{m_1} \\ a_2 &= y \bmod m_2 \quad \in Z_{m_2} \end{aligned}$$

The chinese remainder theorem considers the question of recombining a_1, a_2 back to get y. It says there is a *unique* way to do this under some conditions, and under these conditions says how.

Example C.11 $m = 6 = 3 * 2$

$$\begin{aligned} 0 &\to (0,0) \\ 1 &\to (1,1) \\ 2 &\to (2,0) \\ 3 &\to (0,1) \\ 4 &\to (1,0) \\ 5 &\to (2,1) \end{aligned}$$

∎

Example C.12 $m = 4 = 2 * 2$

$$\begin{aligned} 0 &\to (0,0) \\ 1 &\to (1,1) \\ 2 &\to (0,0) \\ 3 &\to (1,1) \end{aligned}$$

∎

The difference here is that in the first example, the association is unique, in the second it is not. It turns out uniqueness happens when the m_1, m_2 are relatively prime. Here is a simple form of the theorem.

Theorem C.13 [Chinese Remainder Theorem] Let $m_1, m_2, \ldots, m_k$ be pairwise relatively prime integers. That is, $\gcd(m_i, m_j) = 1$ for $1 \le i < j \le k$. Let $a_i \in \mathbf{Z}_{m_i}$ for $1 \le i \le k$ and set $m = m_1 m_2 \cdots m_k$. Then there exists a unique $y \in \mathbf{Z}_m$ such that $y \equiv a_i \bmod m_i$ for $i = 1 \ldots k$. Furthermore there is an $O(k^2)$ time algorithm to compute y given a_1, a_2, m_1, m_2, where $k = \max(|m_1|, |m_2|)$.

Proof: For each i, let $n_i = \left(\frac{m}{m_i}\right) \in \mathbf{Z}$. By hypothesis, $\gcd(m_i, n_i) = 1$ and hence $\exists b_i \in \mathbf{Z}_{m_i}$ such that $n_i b_i \equiv 1 \bmod m_i$. Let $c_i = b_i n_i$. Then $c_i = \begin{cases} 1 \bmod m_i \\ 0 \bmod m_j \text{ for } j \neq i \end{cases}$.

Set $y = \sum_{i=1}^{k} c_i a_i \bmod m$. Then $y \equiv a_i \bmod m_i$ for each i.

Further, if $y' \equiv a_i \bmod m_i$ for each i then $y' \equiv y \bmod m_i$ for each i and since the m_i's are pairwise relatively prime it follows that $y \equiv y' \bmod m$, proving uniqueness. ∎ ∎

Remark C.14 The integers c_i appearing in the above proof will be referred to as the Chinese Remainder Theorem coefficients. Note that the proof yields a polynomial time algorithm for finding y because the elements $b_i \in \mathbf{Z}_{m_i}$ can be determined by using the Euclidean algorithm and the only other operations involved are division, multiplication, and addition. ∎

A more general form of the Chinese Remainder Theorem is the following result.

Theorem C.15 Let $a_i \in \mathbf{Z}_{m_i}$ for $1 \le i \le k$. A necessary and sufficient condition that the system of congruences $x \equiv a_i \bmod m_i$ for $1 \le i \le k$ be solvable is that $\gcd(m_i, m_j) | (a_i - a_j)$ for $1 \le i < j \le k$. If a solution exists then it is unique modulo $\operatorname{lcm}(m_1, m_2, \ldots, m_k)$. ∎

Solution Of The Quadratic Congruence $a \equiv x^2 \bmod n$ When $a \in \mathbf{Z}_n$.

First observe that for p an odd prime and $a \in \mathbf{Z}_p^{*2}$ so that $a \equiv x^2 \bmod p$ for some $x \in \mathbf{Z}_p^*$ there are exactly two solutions to $a \equiv x^2 \bmod p$ because x and $-x$ are two distinct solutions modulo p and if $y^2 \equiv a \equiv x^2 \bmod p$ then $p|[(x-y)(x+y)] \implies p|(x-y)$ or $p|(x+y)$ so that $y \equiv \pm x \bmod p$. (Note that $x \not\equiv -x \bmod p$ for otherwise, $2x \equiv 0 \bmod p \Rightarrow p|x$ as p is odd.) Thus, for $a \in \mathbf{Z}_p^*$, $a \equiv x^2 \bmod p$ has either 0 or 2 solutions.

Next consider the congruence $a \equiv x^2 \bmod p_1 p_2$ where p_1 and p_2 are distinct odd primes. This has a solution if and only if both $a \equiv x^2 \bmod p_1$ and $a \equiv x^2 \bmod p_2$ have solutions. Note that for each pair (x_1, x_2) such that $a \equiv x_1^2 \bmod p_1$ and $a \equiv x_2^2 \bmod p_2$ we can combine x_1 and x_2 by using the Chinese Remainder Theorem to produce a solution y to $a \equiv x^2 \bmod p_1 p_2$ such that $y \equiv x_1 \bmod p_1$ and $y \equiv x_2 \bmod p_2$. Hence, the congruence $a \equiv x^2 \bmod p_1 p_2$ has either 0 or 4 solutions.

More generally, if $p_1, p_2, \ldots, p_k$ are distinct odd primes then the congruence $a \equiv x^2 \bmod p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k}$ has either 0 or 2^k solutions. Again, these solutions can be found by applying the Chinese Remainder Theorem to solutions of $a \equiv x^2 \bmod p_i^{\alpha_i}$. Furthermore, for a prime p a solution to the congruence $a \equiv x^2 \bmod p^k$ can be found by first finding a solution x_0 to $a \equiv x^2 \bmod p$ by using algorithm A of Lemma 2.39 and viewing it as an approximation of the desired square root. Then the approximation is improved by the iteration $x_j \equiv \frac{1}{2}(x_{j-1} + \frac{a}{x_{j-1}}) \bmod p^{2^j}$ for $j \ge 1$.

Claim C.16 For each integer $j \ge 0$, $a \equiv x_j^2 \bmod p^{2^j}$. **Proof:** The claim is certainly true for $j = 0$. Suppose that for $j > 0$, $a \equiv x_j^2 \bmod p^{2^j}$.

Then $x_j - a x_j^{-1} \equiv 0 \bmod p^{2^j} \Rightarrow (x_j - a x_j^{-1})^2 \equiv 0 \bmod p^{2^{j+1}}$.

Expanding and adding $4a$ to both sides gives $x_j^2+2a+a^2x_j^{-2} \equiv 4a \bmod p^{2^{j+1}}$ and therefore, $\left(\frac{1}{2}(x_j + \frac{a}{x_j})\right)^2 \equiv a \bmod p^{2^{j+1}}$ or $x_{j+1}^2 \equiv a \bmod p^{2^{j+1}}$.

Hence, the claim follows by induction. ∎ ∎

From the claim, it follows that after $\lceil \log k \rceil$ iterations we will obtain a solution to $a \equiv x^2 \bmod p^k$.

C.5 Primitive elements and Z_p^*

C.5.1 Definitions

Let G be a group. Let $a \in G$. Look at the powers of a, namely $a^0, a^1, a^2, \ldots,$. We let

$$\langle a \rangle \quad = \quad \{ \, a^i \, : \, i \geq 0 \, \} \, .$$

Let $m = |G|$ be the order of G. We know that a^0 is the identity, call it 1, and $a^m = 1$ also. So the sequence repeats after m steps, ie. $a^{m+1} = a$, etc. But it could repeat before too. Let's look at an example.

Example C.17 $Z_9^* = \{1, 2, 4, 5, 7, 8\}$. Size $\phi(9) = 6$. Then:

$$\begin{aligned} \langle 1 \rangle &= \{1\} \\ \langle 2 \rangle &= \{1, 2, 4, 8, 7, 5\} \\ \langle 4 \rangle &= \{1, 4, 7\} \\ \langle 5 \rangle &= \{1, 5, 7, 8, 4, 2\} \end{aligned}$$

What we see is that sometimes we get everything, sometimes we don't. It might wrap around early. ∎

Fact C.18 $\langle a \rangle$ is a subgroup of G, called the subgroup generated by a. ∎

Let $t = |\langle a \rangle|$. Then we know that t divides m. And we know that in fact $\langle a \rangle = \{a^0, a^1, \ldots, a^{t-1}\}$. That is, these are the distinct elements. All others are repeats.

Definition C.19 The order of an element a is the least positive integer t such that $a^t = 1$. That is, $\mathrm{order}(a) = |\langle a \rangle|$. ∎

Computation in the indices can be done modulo t. That is, $a^i = a^{i \bmod t}$. This is because $a^t = a^0 = 1$.

What's the inverse of a^i? Think what it "should" be: a^{-i}. Does this make sense? Well, think of it as $(a^{-1})^i$. This is correct. On the other hand, what is it as member of the subgroup? It must be a^j for some j. Well $j = t - i$. In particular, inverses are pretty easy to compute if you are given the index.

Similarly, like for real numbers, multiplication in the base corresponds to addition in the exponent. Namely $a^{i+j} = a^i \cdot a^j$. Etc.

Definition C.20 An element $g \in G$ is said to be a primitive element, or *generator*, of G if the powers of g generate G. That is, $\langle g \rangle = G$ is the *whole* group G. A group G is called *cyclic* if it has a primitive element. ∎

Note this means that for any $y \in G$ there is a *unique* $i \in \{0, \ldots, m-1\}$ such that $g^i = y$, where $m = |G|$.

Notation: This unique i is denoted $\log_g(y)$ and called the *discrete logarithm of x to base g*.

Consider the following problem. Given g, y, figure out $\log_g(y)$. How could we do it? One way is to go through all $i = 0, \ldots, m-1$ and for each i compute g^i and check whether $g^i = y$. But this process takes exponential time.

It turns out that computing discrete logarithms is hard for many groups. Namely, there is no known polynomial time algorithm. In particular it is true for Z_p^* where p is a prime.

C.5.2 The group Z_p^*

Fact C.21 [7, Section 9] The group Z_p^* is cyclic. ▮

Remark C.22 What is the order of Z_p^*? It is $\phi(p)$, the number of positive integers below p which are relatively prime to p. Since p is prime this is $p-1$. Note the order is *not* prime! In particular, it is even (for $p \geq 3$). ▮

A one-way function: Let p be prime and let $g \in Z_p^*$ be a generator. Then the function $f_{p,g} \colon Z_p \to Z_p^*$ defined by

$$x \mapsto g^x$$

is conjectured to be one-way as long as some technical conditions hold on p. That is, there is no efficient algorithm to invert it, for large enough values of the parameters. See Chapter 2.

Homomorphic properties: A useful property of the function $f_{p,g}$ is that $g^{a+b} = g^a \cdot g^b$.

Now, how can we use this function? Well, first we have to set it up. This requires two things. First that we can find primes; second that we can find generators.

C.5.3 Finding generators

We begin with the second. We have to look inside Z_p^* and find a generator. How? Even if we have a candidate, how do we test it? The condition is that $\langle g \rangle = G$ which would take $|G|$ steps to check.

In fact, finding a generator given p is in general a hard problem. In fact even checking that g is a generator given p, g is a hard problem. But what we can exploit is that $p = 2q + 1$ with q prime. Note that the order of the group Z_p^* is $p - 1 = 2q$.

Fact C.23 Say $p = 2q + 1$ is prime where q is prime. Then $g \in Z_p^*$ is a generator of Z_p^* iff $g^q \neq 1$ and $g^2 \neq 1$. ▮

In other words, testing whether g is a generator is easy given q. Now, given $p = 2q + 1$, how do we find a generator?

Fact C.24 If g is a generator and i is not divisible by q or 2 then g^i is a generator.

Proof: $g^{iq} = g^{q+(i-1)q} = g^q \cdot (g^{2q})^{(i-1)/2} = g^q \cdot 1 = g^q$ which is not 1 because g is not a generator. Similarly let $i = r + jq$ and we have $g^{2i} = g^{2r} \cdot g^{2jq} = g^{2r}$. But $2r < 2q$ since $r < q$ so $g^{2r} \neq 1$. ▮ ▮

So how many generators are there in Z_p^*? All things of form g^i with i not divisible by 2 or q and $i = 1, \ldots, 2q$. Namely all i in Z_{2q}^*. So there are $\phi(2q) = q - 1$ of them.

So how do we find a generator? Pick $g \in Z_p^*$ at random, and check that $g^q \neq 1$ and $g^2 \neq 1$. If it fails, try again, up to some number of times. What's the probability of failure? In one try it is $(q+1)/2q$ so in l tries it is

$$\left(\frac{q+1}{2q}\right)^l$$

which is rougly 2^{-l} because q is very large.

C.6 Quadratic residues

An element $x \in Z_N^*$ is a square, or quadratic residue, if it has a square root, namely there is a $y \in Z_N^*$ such that $y^2 \equiv x \bmod N$. If not, it is a non-square or non-quadratic-residue. Note a number may have lots of square roots.

It is easy to compute square roots modulo a prime. It is also easy modulo a composite whose prime factorization you know, via Chinese remainders. (In both cases, you can compute all the roots.) But it is hard modulo a composite of unknown factorization. In fact computing square roots is equivalent to factoring.

Also, it is hard to decide quadratic residuosity modulo a composite.

Fact C.25 If N is the product of two primes, every square $w \in Z_N^*$ has exactly four square roots, $x, -x$ and $y, -y$ for some $x, y \in Z_N^*$. If you have two square roots x, y such that $x \neq \pm y$, then you can easily factor N. ∎

The first fact is basic number theory. The second is seen like this. Say $x > y$ are the roots. Consider $x^2 - y^2 = (x-y)(x+y) \equiv 0 \bmod N$. Let $a = x - y$ and $b = x + y \bmod N$. So N divides ab. So p divides ab. Since p is prime, this means either p divides a or p divides b. Since $1 \leq a, b < N$ this means either $\gcd(a, N) = p$ or $\gcd(b, N) = p$. We can compute the gcds and check whether we get a divisor of N.

C.7 Jacobi Symbol

We previously defined the Legendre symbol to indicate the quadratic character of $a \in \mathbf{Z}_p^*$ where p is a prime. Specifically, for a prime p and $a \in \mathbf{Z}_p$

$$\mathbf{J}_p(a) = \begin{cases} 1 & \text{if } a \text{ is a square in } \mathbf{Z}_p^* \\ 0 & \text{if } a = 0 \\ -1 & \text{if } a \text{ is not a square in } \mathbf{Z}_p^* \end{cases}$$

For composite integers, this definition is extended, as follows, giving the *Jacobi Symbol*. Let $n = \prod_{i=1}^{k} {p_i}^{\alpha_i}$ be the prime factorization of n. For $a \in \mathbf{Z}_n$ define

$$\mathbf{J}_n(a) = \prod_{i=1}^{k} \mathbf{J}_{p_i}(a)^{\alpha_i}.$$

However, the Jacobi Symbol does not generalize the Legendre Symbol in the respect of indicating the quadratic character of $a \in \mathbf{Z}_n^*$ when n is composite. For example, $\mathbf{J}_9(2) = \mathbf{J}_3(2)\mathbf{J}_3(2) = 1$, although the equation $x^2 \equiv 2 \bmod 9$ has no solution.

The Jacobi Symbol also satisfies identities similar to those satisfied by the Legendre Symbol. We list these here. For proofs of these refer to [150].

1. If $a \equiv b \bmod n$ then $\mathbf{J}_n(a) = \mathbf{J}_n(b)$.
2. $\mathbf{J}_n(1) = 1$.
3. $\mathbf{J}_n(-1) = (-1)^{\frac{n-1}{2}}$.
4. $\mathbf{J}_n(ab) = \mathbf{J}_n(a)\mathbf{J}_n(b)$.
5. $\mathbf{J}_n(2) = (-1)^{\frac{n^2-1}{8}}$.
6. If m and n are relatively prime odd integers then $\mathbf{J}_n(m) = (-1)^{\frac{n-1}{2}\frac{m-1}{2}}\mathbf{J}_m(n)$.

Using these identities, the Jacobi Symbol $\mathbf{J}_n(a)$ where $a \in \mathbf{Z}_n$ can be calculated in polynomial time even without knowing the factorization of n. Recall that to calculate the Legendre Symbol in polynomial time we can call upon Euler's Theorem; namely, for $a \in \mathbf{Z}_p^*$, where p is prime, we have $\mathbf{J}_p(a) \equiv a^{\frac{p-1}{2}} \bmod p$. However, for a composite integer n it is not necessarily true that $\mathbf{J}_n(a) \equiv a^{\frac{n-1}{2}} \bmod n$ for $a \in \mathbf{Z}_n^*$. In fact, this statement is true for at most half of the elements in $\mathbf{Z}_n^*$. From this result, we can derive a Monte Carlo primality test as we shall see later.

C.8 RSA

Here we have a composite modulus $N = pq$ product of two distinct primes p and q of roughly equal length. Let $k = |N|$; this is about 1024, say. It is generally believed that such a number is hard to factor.

Recall that $\phi(N) = |Z_N^*|$ is the Euler Phi function. Note that $\phi(N) = (p-1)(q-1)$. (To be relatively prime to N, a number must be divisible neither by p nor by q. Eliminating multiples of either yields this. Note we use here that $p \neq q$.)

Now let e be such that $\gcd(e, \phi(N)) = 1$. That is, $e \in Z_{\phi(N)}^*$. The RSA function is defined by

$$\begin{array}{rccc} f\colon & Z_N^* & \to & Z_N^* \\ & x & \mapsto & x^e \bmod N \,. \end{array}$$

We know that $Z_{\phi(N)}^*$ is a group. So e has an inverse $d \in Z_{\phi(N)}^*$. Since d is an inverse of e it satisfies

$$ed \equiv 1 \pmod{\phi(N)}$$

Now let $x \in Z_N^*$ be arbitrary and look at the following computation:

$$(x^e)^d \bmod N = x^{ed \bmod \phi(N)} \bmod N = x^1 \bmod N = x \,.$$

In other words, the function $y \mapsto y^d$ is an inverse of f. That is, $f^{-1}(y) = y^d \bmod N$.

Can we find d? Easy: computing inverses can be done in quadratic time, as we already say, using the extended GCD algorithm! But note a crucial thing. We are working modulo $\phi(N)$. So finding d this way requires that we know $\phi(N)$. But the latter involves knowing p, q.

It seems to be the case that given only N and e it is hard to find d. Certainly we agreed it is hard to find p, q; but even more, it seems hard to find d. This yields the conjecture that RSA defines a trapdoor one-way permutation. Namely given N, e defining f, $x \mapsto f(x)$ is easy; $y \mapsto f^{-1}(y)$ is hard; but f^{-1} is easy given p, q (or d). Note that this trapdoorness is a property the discrete logarithm problem did not have.

Computation of f is called encryption, and computation of f^{-1} is called decryption.

Both encryption and decryption are exponentiations, which a priori are cubic in k time operations. However, one often chooses e to be small, so encryption is faster. In hardware, RSA is about 1000 times slower than DES; in software it is about 100 times slower, this with small encryption exponent.

Formally, RSA defines a *family of trapdoor permutations.* The family is indexed by a security parameter k which is the size of the modulus. The RSA generator is an algorithm G which on input 1^k picks two distinct, random $(k/2)$-bit primes p, q, multiplies them to produce $N = pq$, and also computes e, d. It outputs N, e as the description of f and N, d as the description of f^{-1}. See Chapter 2.

RSA provides the ability to do public key cryptography.

C.9 Primality Testing

For many cryptographic purposes, We need to find primes. There is no known polynomial time algorithm to test primality of a given integer n. What we use are probabilistic, polynomial time (PPT) algorithms.

We will first show that the problem of deciding whether an integer is prime is in NP. Then we will discuss the Solovay-Strassen and Miller-Rabin probabilistic primality tests which efficiently find proofs of compositeness. Finally, we will give a primality test due to Goldwasser and Kilian which uses elliptic curves and which efficiently finds a proof of primality.

C.9.1 PRIMES $\in$ NP

We will first present two algorithms for testing primality, both of which are inefficient because they require factoring as a subroutine. However, either algorithm can be used to show that the problem of deciding whether an integer is prime is in NP. In fact, the second algorithm that is presented further demonstrates that deciding primality is in UP$\cap$coUP. Here UP denotes the class of languages L accepted by a polynomial time nondeterministic Turing machine having a unique accepting path for each $x \in L$.

Definition C.26 Let PRIMES $= \{p : p \text{ is a prime integer}\}$. ∎

C.9.2 Pratt's Primality Test

Pratt's primality testing algorithm is based on the following result.

Proposition C.27 For an integer $n > 1$, the following statements are equivalent.

1. $|\mathbf{Z}_n^*| = n - 1$.
2. The integer n is prime.
3. There is an element $g \in \mathbf{Z}_n^*$ such that $g^{n-1} \equiv 1 \bmod n$ and for every prime divisor q of $n-1$, $g^{\frac{n-1}{q}} \not\equiv 1 \bmod n$.

∎

Pratt's algorithm runs as follows on input a prime p and outputs a proof (or certificate) that p is indeed prime.

1. Find an element $g \in \mathbf{Z}_p^*$ whose order is $p-1$.
2. Determine the prime factorization $\prod_{i=1}^{k} q_i^{\alpha_i}$ of $p-1$.
3. Prove that p is prime by proving that g is a generator of $\mathbf{Z}_p^*$. Specifically, check that $g^{p-1} \equiv 1 \bmod p$ and for each prime q_i check that $g^{\frac{p-1}{q_i}} \not\equiv 1 \bmod p$.
4. Recursively show that q_i is prime for $1 \le i \le k$.

Note that if p is a prime, then $\mathbf{Z}_p^*$ has $\varphi(p-1) = \Omega(\frac{p}{\log\log p})$ generators (see [172]). Thus, in order to find a generator g by simply choosing elements of $\mathbf{Z}_p^*$ at random, we expect to have to choose $O(\log\log p)$ candidates for g. If we find a generator g of $\mathbf{Z}_p^*$ and if we can factor $p-1$ and recursively prove that the prime factors of $p-1$ are indeed primes then we have obtained a proof of the primality of p. Unfortunately, it is not known how to efficiently factor $p-1$ for general p. Pratt's primality testing algorithm does demonstrate, however, that PRIMES $\in$ NP because both the generator g in step 1 and the required factorization in step 2 can be guessed. Moreover, the fact that the factorization is correct can be verified in polynomial time and the primality of each q_i can be verified recursively by the algorithm. Note also, as Pratt showed in [161] by a simple inductive argument, that the total number of primes involved is $O(\log p)$. Thus, verifying a Pratt certificate requires $O(\log^2 p)$ modular multiplications with moduli at most p.

C.9.3 Probabilistic Primality Tests

C.9.4 Solovay-Strassen Primality Test

We can derive a Monte Carlo primality test. This algorithm, which we state next, is due to Solovay and Strassen (see [191]).

The Solovay-Strassen primality test runs as follows on input an odd integer n and an integer k, indicating the desired reliability.

1. Test if $n = b^e$ for integers $b, e > 1$; if so, output composite and terminate.
2. Randomly choose $a_1, a_2, \ldots, a_k \in \{1, 2, \ldots, n-1\}$.
3. If $\gcd(a_i, n) \neq 1$ for any $1 \le i \le k$ then output composite and terminate.
4. Calculate $\alpha_i = a_i^{\frac{n-1}{2}} \bmod n$ and $\beta_i = \mathbf{J}_n(a_i)$.
5. If for any $1 \le i \le k$, $\alpha_i \neq \beta_i \bmod n$ then output composite. If for all $1 \le i \le k$, $\alpha_i = \beta_i \bmod n$ then output probably prime.

Since the calculations involved in the Solovay-Strassen primality test are all polynomial time computable (verify that this statement is indeed true for step 1), it is clear that the algorithm runs in time polynomial in $\log n$ and k. The following result guarantees that if n is composite then in step 5 of the algorithm, $\Pr[\alpha_i = \beta_i \bmod n] \le \frac{1}{2}$ and thus, $\Pr[\alpha_i = \beta_i \bmod n \text{ for } 1 \le i \le k] \le \left(\frac{1}{2}\right)^k$.

Proposition C.28 Let n be an odd composite integer which is not a perfect square and let $G = \{a \in \mathbf{Z}_n^* \text{ such that } \mathbf{J}_n(a) \equiv a^{\frac{n-1}{2}} \bmod n\}$. Then $|G| \le \frac{1}{2}|\mathbf{Z}_n^*|$. ▮

Proof: Since G is a subgroup of $\mathbf{Z}_n^*$ it suffices to show that $G \neq \mathbf{Z}_n^*$.

Since n is composite and not a perfect square, it has a nontrivial factorization $n = rp^\alpha$ where p is prime, α is odd, and $\gcd(r, p) = 1$.

Suppose that $a^{\frac{n-1}{2}} \equiv \mathbf{J}_n(a) \bmod n$ for all $a \in \mathbf{Z}_n^*$. Then

$$a^{\frac{n-1}{2}} \equiv \pm 1 \bmod n \text{ for all } a \in \mathbf{Z}_n^*. \tag{C.1}$$

We first show that in fact $a^{\frac{n-1}{2}} \equiv 1 \bmod n$ for all such a. If not, then there is an $a \in \mathbf{Z}_n^*$ with $a^{\frac{n-1}{2}} \equiv -1 \bmod n$. By the Chinese Remainder Theorem there is a unique element $b \in \mathbf{Z}_n$ such that $b \equiv 1 \bmod r$ and $b \equiv a \bmod p^\alpha$. Then $b \in \mathbf{Z}_n^*$ and $b^{\frac{n-1}{2}} \equiv 1 \bmod r$ and $b^{\frac{n-1}{2}} \equiv -1 \bmod p^\alpha$ contradicting equation $(C.1)$. Therefore, $\mathbf{J}_n(a) = 1$ for all $a \in \mathbf{Z}_n^*$.

However, by the Chinese Remainder Theorem, there is a unique element $a \in \mathbf{Z}_{rp}$ such that $a \equiv 1 \bmod r$ and $a \equiv z \bmod p$ where z is one of the $\frac{p-1}{2}$ quadratic nonresidues modulo p. Then $a \in \mathbf{Z}_n^*$ and thus, $\mathbf{J}_n(a) = \mathbf{J}_r(1)\mathbf{J}_p(z)^\alpha = -1$ because α is odd. This is a contradiction. ▮

Note that if we reach step 5 of the Solovay-Strassen algorithm then n is not a perfect square and each $a_i \in \mathbf{Z}_n^*$ because the algorithm checks for perfect powers in step 1 and computes $\gcd(a_i, n)$ in step 3. Thus, the hypotheses of Proposition C.28 are satisfied and for each $1 \le i \le k$, $\Pr[\alpha_i = \beta_i \bmod n] \le \frac{1}{2}$.

Remark The assertion in Proposition C.28 is in fact true even if n is a perfect square. The proof of the more general statement is very similar to the proof of Proposition C.28. For details refer to [7].

Finally, it follows from Proposition C.28 that the Solovay-Strassen algorithm runs correctly with high probability. Specifically,

$$\Pr[\text{Solovay-Strassen outputs probably prime} \mid n \text{ is composite}] \leq (\frac{1}{2})^k$$

and

$$\Pr[\text{Solovay-Strassen outputs probably prime} \mid n \text{ is prime}] = 1.$$

C.9.5 Miller-Rabin Primality Test

Fermat's Little Theorem states that for a prime p and $a \in \mathbf{Z}_p^*$, $a^{p-1} \equiv 1 \bmod p$. This suggests that perhaps a possible way to test if n is prime might be to check, for instance, if $2^{n-1} \equiv 1 \bmod n$. Unfortunately, there are composite integers n (called base-2 pseudoprimes) for which $2^{n-1} \equiv 1 \bmod n$. For example, $2^{340} \equiv 1 \bmod 341$ and yet $341 = 11 \cdot 31$. In fact, replacing 2 in the above exponentiation by a random $a \in \mathbf{Z}_n^*$ would not help for some values of n because there are composite integers n (called Carmichael numbers) for which $a^{n-1} \equiv 1 \bmod n$ for all $a \in \mathbf{Z}_n^*$. 561, 1105, and 1729 are the first three Carmichael numbers.

The Miller-Rabin primality test overcomes the problems of the simple suggestions just mentioned by choosing several random $a \in \mathbf{Z}_n^*$ for which $a^{n-1} \bmod n$ will be calculated by repeated squaring. While computing each modular exponentiation, it checks whether some power of a is a nontrivial square root of 1 modulo n (that is, a root of 1 not congruent to ± 1 modulo n). If so, the algorithm has determined n to be composite. The quality of this test relies on Proposition C.30 which Rabin proved in [165]. For a simpler proof which only yields $|\{b : \mathrm{W}_n(b) \text{ holds }\}| \geq \frac{1}{2}(n-1)$ (but is nevertheless sufficient for the purposes of the Miller-Rabin algorithm) see Chapter 33, pages 842-843 of [59].

Definition C.29 Let n be an odd positive integer. Denote the following condition on an integer b by $\mathrm{W}_n(b)$:

1. $1 \leq b < n$ and
2. (i) $b^{n-1} \not\equiv 1 \bmod n$ or

   (ii) there is an integer i such that $2^i \mid (n-1)$ and $b^{(n-1)/2^i} \not\equiv \pm 1 \bmod n$ but $\left(b^{(n-1)/2^i}\right)^2 \equiv 1 \bmod n$.

An integer b for which $\mathrm{W}_n(b)$ holds will be called a witness to the compositeness of n. ∎

Remark Rabin originally defined the condition $\mathrm{W}_n(b)$ to hold if $1 \leq b < n$ and either $b^{n-1} \not\equiv 1 \bmod n$ or for some integer i such that $2^i \mid (n-1)$, $1 < \gcd(b^{(n-1)/2^i} - 1, n) < n$. In [165] Rabin proves that the two definitions for $\mathrm{W}_n(b)$ are equivalent. This condition was in fact first considered by Miller (see [142]), who used it to give a nonprobabilistic test for primality assuming the correctness of the extended Riemann hypothesis. Rabin's results, however, do not require any unproven assumptions.

Proposition C.30 If n is an odd composite integer then $|\{b : \mathrm{W}_n(b) \text{ holds}\}| \geq \frac{3}{4}(n-1)$. ∎

The Miller-Rabin algorithm runs as follows on input an odd integer n and an integer k, indicating the desired reliability.

1. Randomly choose $b_1, b_2, \ldots, b_k \in \{1, 2, \ldots, n-1\}$.
2. Let $n - 1 = 2^l m$ where m is odd.
3. For $1 \leq i \leq k$ compute $b_i^m \bmod n$ by repeated squaring.
4. Compute $b_i^{2^j m} \bmod n$ for $j = 1, 2, \ldots, l$. If for some j, $b_i^{2^{j-1} m} \not\equiv \pm 1 \bmod n$ but $b_i^{2^j m} \equiv 1 \bmod n$ then $\mathrm{W}_n(b_i)$ holds.

5. If $b_i^{n-1} \not\equiv 1 \bmod n$ then $\mathrm{W}_n(b_i)$ holds.
6. If for any $1 \le i \le k$, $\mathrm{W}_n(b_i)$ holds then output composite. If for all $1 \le i \le k$, $\mathrm{W}_n(b_i)$ does not hold then output probably prime.

Proposition C.30 shows that the Miller-Rabin algorithm runs correctly with high probability. Specifically,

$$\Pr[\text{Miller-Rabin outputs probably prime} \mid n \text{ is composite}] \le (\frac{1}{4})^k$$

and

$$\Pr[\text{Miller-Rabin outputs probably prime} \mid n \text{ is prime}] = 1.$$

Furthermore, Miller-Rabin runs in time polynomial in $\log n$ and k as all the computations involved can be performed in polynomial time.

C.9.6 Polynomial Time Proofs Of Primality

Each of the two algorithms discussed in the previous section suffers from the deficiency that whenever the algorithm indicates that the input n is prime, then it is prime with high probability, but no certainty is provided. (In other words, the algorithms are Monte Carlo algorithms for testing primality.) However, when either algorithm outputs that the input n is composite then it has determined that n is indeed composite. Thus, the Solovay-Strassen and Miller-Rabin algorithms can be viewed as compositeness provers. In this section we will discuss a primality test which yields in expected polynomial time a short (verifiable in deterministic polynomial time) proof that a prime input is indeed prime. Therefore, for a general integral input we can run such a primality prover in parallel with a compositeness prover and one of the two will eventually terminate either yielding a proof that the input is prime or a proof that the input is composite. This will provide us with a Las Vegas algorithm for determining whether an integer is prime or composite.

C.9.7 An Algorithm Which Works For Some Primes

Suppose that we could find a prime divisor q of $p-1$ such that $q > \sqrt{p}$. Then the following algorithm can be used to prove the primality of p.

1. Determine a prime divisor q of $p-1$ for which $q > \sqrt{p}$.
2. Randomly choose $a \in \mathbf{Z}_p^* - \{1\}$.
3. If $1 < \gcd(a-1, p) < p$ then output that p is composite.
4. Check that $a^q \equiv 1 \bmod p$.
5. Recursively prove that q is prime.

The correctness of this algorithm follows from the next result.

Claim C.31 If $q > \sqrt{p}$ is a prime and for some $a \in \mathbf{Z}_p^*$ $\gcd(a-1, p) = 1$ and $a^q \equiv 1 \bmod p$ then p is a prime. ∎

Proof: Suppose p is not prime. Then there is a prime $d \le \sqrt{p}$ such that $d \mid p$ and therefore, by the hypothesis, $a \not\equiv 1 \bmod d$ and $a^q \equiv 1 \bmod d$. Thus, in $\mathbf{Z}_d^*$, $\mathrm{ord}(a) \mid q$. But q is prime and a does not have order 1. Hence, $q = \mathrm{ord}(a) \le |\mathbf{Z}_d^*| = d - 1 < \sqrt{p}$ and this contradicts the assumption that $q > \sqrt{p}$. ∎

Note that if p is prime then in step 4, the condition $a^q \equiv 1 \bmod p$ will be verified with probability at least $\frac{q-1}{p-2} > \frac{1}{\sqrt{p}}$ (since $q > \sqrt{p}$). However, in order for the algorithm to succeed, there must exist a prime divisor

q of $p-1$ such that $q > \sqrt{p}$, and this must occur at every level of the recursion. Namely, there must be a sequence of primes $q = q_0, q_1, \ldots, q_k$, where q_k is small enough to identify as a known prime, such that $q_i \mid (q_{i-1} - 1)$ and $q_i > \sqrt{q_{i-1}}$ for $i = 1, \ldots, k$ and this is very unlikely.

This obstacle can be overcome by working with elliptic curves modulo primes p instead of $\mathbf{Z}_p^*$. This will allow us to randomly generate for any prime modulus, elliptic groups of varying orders. In the following sections, we will exploit this additional freedom in a manner similar to Lenstra's elliptic curve factoring algorithm.

C.9.8 Goldwasser-Kilian Primality Test

The Goldwasser-Kilian primality test is based on properties of elliptic curves. The idea of the algorithm is similar to the primality test presented in Section C.9.7, except that we work with elliptic curves $E_{a,b}(\mathbf{Z}_p)$ instead of $\mathbf{Z}_p^*$. By varying a and b we will be able to find an elliptic curve which exhibits certain desired properties.

The Goldwasser-Kilian algorithm runs as follows on input a prime integer $p \neq 2, 3$ of length l and outputs a proof that p is prime.

1. Randomly choose $a, b \in \mathbf{Z}_p$, rejecting choices for which $\gcd(4a^3 + 27b^2, p) \neq 1$.
2. Compute $|E_{a,b}(\mathbf{Z}_p)|$ using the polynomial time algorithm due to Schoof (see [179]).
3. Use a probabilistic pseudo-primality test (such as Solovay-Strassen or Miller-Rabin) to determine if $|E_{a,b}(\mathbf{Z}_p)|$ is of the form cq where $1 < c \leq O(\log^2 p)$ and q is a probable prime. If $|E_{a,b}(\mathbf{Z}_p)|$ is not of this form then repeat from step 1.
4. Select a point $M = (x, y)$ on $E_{a,b}(\mathbf{Z}_p)$ by choosing $x \in \mathbf{Z}_p$ at random and taking y to be the square root of $x^3 + ax + b$, if one exists. If $x^3 + ax + b$ is a quadratic nonresidue modulo p then repeat the selection process.
5. Compute $q \cdot M$.
   (i) If $q \cdot M = O$ output (a, b, q, M). Then, if $q > 2^{l^{(\frac{1}{\log \log l})}}$, [5]), recursively prove that q is prime (specifically, repeat from step 1 with p replaced by q). Otherwise, use the deterministic test due to Adleman, Pomerance, and Rumely (see [5]) to show that q is prime and terminate.
   (ii) If $q \cdot M \neq O$ then repeat from step 4.

Remark The test mentioned in step 5i is currently the best deterministic algorithm for deciding whether an input is prime or composite. It terminates within $(\log n)^{O(\log \log \log n)}$ steps on input n.

C.9.9 Correctness Of The Goldwasser-Kilian Algorithm

Note first that as we saw in Section C.9.3, the probability of making a mistake at step 3 can be made exponentially small. The correctness of the Goldwasser-Kilian algorithm follows from Theorem C.32. This result is analogous to Claim C.31.

Theorem C.32 Let $n > 1$ be an integer with $\gcd(n, 6) = 1$. Let $E_{a,b}(\mathbf{Z}_n)$ be an elliptic curve modulo n and let $M \neq O$ be a point on $E_{a,b}(\mathbf{Z}_n)$. If there is a prime integer q such that $q > (n^{1/4} + 1)^2$ and $q \cdot M = O$ then n is prime. ■

Proof: Suppose that n were composite. Then there is a prime divisor p of n such that $p < \sqrt{n}$.

Let $\mathrm{ord}_E(M)$ denote the order of the point M on the elliptic curve E. If $q \cdot M = O$ then $q \cdot M_p = O_p$. Thus, $\mathrm{ord}_{E_p}(M_p) \mid q$.

$$\begin{aligned}\text{However, } \mathrm{ord}_{E_p}(M_p) \leq |E_{a,b}(\mathbf{Z}_p)| &\leq p + 1 + 2\sqrt{p} \qquad \text{(by Hasse's Inequality)}\\ &< n^{1/2} + 1 + 2n^{1/4}\\ &< q\end{aligned}$$

and since q is prime, we have that $\mathrm{ord}_{E_p}(M_p) = 1$. Therefore, $M_p = O_p$ which implies that $M = O$, a contradiction. ▮

Theorem C.33 By using the sequence of quadruples output by the Goldwasser-Kilian algorithm, we can verify in time $O(\log^4 p)$ that p is indeed prime. ▮

Proof: Let $p_0 = p$. The sequence of quadruples output by the algorithm will be of the form $(a_1, b_1, p_1, M_1), (a_2, b_2, p_2, M_2), \ldots, (a_k, b_k, p_k, M_k)$ where $\gcd(4a_i^3 + 27b_i^2, p_{i-1}) \neq 1$, $M_i \neq O$ is a point on $E_{a_i,b_i}(\mathbf{Z}_{p_{i-1}})$, $p_i > p_{i-1}^{1/2} + 1 + 2p_{i-1}^{1/4}$, and $p_i \cdot M_i = O$ for $1 \leq i \leq k$. These facts can all be verified in $O(\log^3 p)$ time for each value of i. By Theorem C.32 it follows that p_i prime $\Rightarrow p_{i-1}$ prime for $1 \leq i \leq k$. Further, note that in step 3 of the algorithm, $c \geq 2$ and hence, $p_i \leq \frac{p_{i-1}+2\sqrt{p_{i-1}}}{2}$. Therefore, the size of k will be $O(\log p)$ giving a total of $O(\log^4 p)$ steps. Finally, p_k can be verified to be prime in $O(\log p)$ time due to its small size. ▮

C.9.10 Expected Running Time Of Goldwasser-Kilian

The algorithm due to Schoof computes $|E_{a,b}(\mathbf{Z}_p)|$ in $O(\log^9 p)$ time. Then to check that $|E_{a,b}(\mathbf{Z}_p)| = cq$ where $1 < c \leq O(\log^2 p)$ and q is prime requires a total of $O(\log^6 p)$ steps if we use Solovay-Strassen or Miller-Rabin with enough iterations to make the probability of making a mistake exponentially small (the algorithm may have to be run for each possible value of c and each run of the algorithm requires $O(\log^4 p)$ steps).

Next, selecting the point $M = (x, y)$ requires choosing an expected number of at most $\frac{2p}{|E_{a,b}(\mathbf{Z}_p)|-1} \approx 2$ values for x before finding one for which $x^3 + ax + b$ is a quadratic residue modulo p. Note that the computation of square roots modulo a prime p (to find y) can be done in $O(\log^4 p)$ expected time. Since $|E_{a,b}(\mathbf{Z}_p)| = cq$ where q is prime, $E_{a,b}(\mathbf{Z}_p)$ is isomorphic to $\mathbf{Z}_{c_1 q} \times \mathbf{Z}_{c_2}$ where $c = c_1 c_2$ and $c_2 | c_1$. Therefore, $E_{a,b}(\mathbf{Z}_p)$ has at least $q - 1$ points of order q and hence with probability at least $\frac{q-1}{cq} \approx \frac{1}{c}$, the point M selected in step 4 will have order q. Thus, the expected number of points that must be examined before finding a point M of order q will be $c = O(\log^2 p)$. Further, the computation of $q \cdot M$ requires $O(\log p)$ additions, using repeated doubling and so can be done in $O(\log^3 p)$ time. Therefore, dealing with steps 4 and 5 requires $O(\log^5 p)$ expected time.

As remarked previously, the recursion depth is $O(\log p)$. Therefore, the only remaining consideration is to determine how often an elliptic curve $E_{a,b}(\mathbf{Z}_p)$ has to be selected before $|E_{a,b}(\mathbf{Z}_p)| = cq$ where $c = O(\log^2 p)$ and q is prime. By the result of Lenstra concerning the distribution of $|E_{a,b}(\mathbf{Z}_p)|$ in $(p+1-\sqrt{p}, p+1+\sqrt{p})$ (see [126]) this is $O(\frac{\sqrt{p}\log p}{|S|-2})$ where S is the set of integers in $(p+1-\sqrt{p}, p+1+\sqrt{p})$ of the desired form cq. Note that $|S| \geq \pi(\frac{p+1+\sqrt{p}}{2}) - \pi(\frac{p+1-\sqrt{p}}{2})$ because S contains those integers in $(p+1-\sqrt{p}, p+1+\sqrt{p})$ which are twice a prime. Therefore, if one assumes that the asymptotic distribution of primes holds in small intervals, then the expected number of elliptic curves that must be considered is $O(\log^2 p)$. However, there is only evidence to assume the following conjecture concerning the number of primes in small intervals.

Conjecture C.34 There is a positive constant s such that for all $x \in \mathbf{R}_{\geq 2}$, the number of primes between x and $x + \sqrt{2x}$ is $\Omega\left(\frac{\sqrt{x}}{\log^s x}\right)$. ▮

Under this assumption, the Goldwasser-Kilian algorithm proves the primality of p in $O((\log p)^{11+s})$ expected time.

C.9.11 Expected Running Time On Nearly All Primes

Although the analysis presented in Section C.9.10 relies on the unproven result stated in Conjecture C.34, a theorem due to Heath-Brown concerning the density of primes in small intervals can be used to show that the fraction of primes of length l for which the Goldwasser-Kilian algorithm runs in expected time polynomial in l is at least $1 - O(2^{-l^{\frac{1}{\log \log l}}})$. The Heath-Brown result is the following.

Theorem C.35 Let $\#_p[a, b]$ denote the number of primes x satisfying $a \leq x \leq b$.
Let $i(a,b) = \begin{cases} 1 \text{ if } \#_p[a,b] \leq \frac{b-a}{2\lfloor \log a \rfloor} \\ 0 \text{ otherwise} \end{cases}$. Then there exists a positive constant α such that
$\sum_{x \leq a \leq 2x} i(a, a + \sqrt{a}) \leq x^{\frac{5}{6}} \log^{\alpha} x$. ▮

Using this theorem, Goldwasser and Kilian were able to prove in [96] that their algorithm terminates in expected time $O(l^{12})$ on at least a $1 - O(2^{l^{\frac{1}{\log \log l}}})$ fraction of those primes of length l. In [4] Adleman and Huang showed, by a more careful analysis of the Goldwasser-Kilian algorithm, that in fact the fraction of primes of length l for which Goldwasser-Kilian may not terminate in expected polynomial time is strictly exponentially vanishing. Further, they proposed a new algorithm for proving primality based on hyperelliptic curves which they showed will terminate in exponential polynomial time on all prime inputs. Thus, the goal of obtaining a Las Vegas algorithm has been finally achieved.

C.10 Factoring Algorithms

In this lecture we discuss some general properties of elliptic curves and present Lenstra's elliptic curve factoring algorithm which uses elliptic curves over $\mathbf{Z}_n$ to factor integers.

Pollard's $p - 1$ Method

We begin by introducing a predecessor of the elliptic curve factoring algorithm which uses ideas analogous to those used in the elliptic curve factoring algorithm. This algorithm, known as Pollard's $p - 1$ method, appears in [160]. Let n be the composite number that we wish to split. Pollard's algorithm uses the idea that if we can find integers e and a such that $a^e \equiv 1 \bmod p$ and $a^e \not\equiv 1 \bmod q$ for some prime factors p and q of n then, since $p \mid (a^e - 1)$ and $q \not\mid (a^e - 1)$, $\gcd(a^e - 1, n)$ will be a nontrivial factor of n divisible by p but not by q.

The algorithm proceeds as follows on input n.

1. Choose an integer e that is a multiple of all integers less than some bound B. For example, e might be the least common multiple of all integers $\leq B$. To simplify this, we might even let $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_1, p_2, \ldots, p_{\pi(B)}$ are the primes $\leq B$ and α_i is chosen minimally so that $p_i^{\alpha_i} \geq \sqrt{n} > \min_{p|n}\{p-1\}$.

2. Choose a random integer a between 2 and $n - 2$.

3. Compute $a^e \bmod n$ by repeated squaring.

4. Compute $d = \gcd(a^e - 1, n)$ by the Euclidean algorithm. If $1 < d < n$ output the nontrivial factor d. Otherwise, repeat from step 2 with a new choice for a.

To explain when this algorithm works, assume that the integer e is divisible by every integer $\leq B$ and that p is a prime divisor of n such that $p-1$ is the product of prime powers $\leq B$. Then $e = m(p-1)$ for some integer m and hence $a^e = \left(a^{(p-1)}\right)^m \equiv 1^m = 1 \bmod p$. Therefore, $p \mid \gcd(a^e - 1, n)$ and the only way that we could fail to obtain a nontrivial factor of n in step 4 is if $a^e \equiv 1 \bmod n$. In other words, we could only fail here if for every prime factor q of n the order of $a \bmod q$ divides e and this is unlikely.

Unfortunately, it is not true that for general n there is a prime divisor p of n for which $p-1$ is divisible by no prime power larger than B for a bound B of small size. If $p-1$ has a large prime power divisor for each prime divisor p of n, then Pollard's $p-1$ method will work only for a large choice of the bound B and so will be inefficient because the algorithm runs in essentially $O(B)$ time. For example, if n is the product of two different primes p and q where $|p| \approx |q|$ are primes and $p-1$ and $q-1$ are $O(\sqrt{n})$-smooth then the method will likely require a bound B of size $O(\sqrt{n})$.

Reiterating, the problem is that given input $n = \prod {p_i}^{\alpha_i}$ where the p_i's are the distinct prime factors of n, we are restricted by the possibility that none of the integers $p_i - 1$ are sufficiently smooth. However, we can ameliorate this restriction by working with the group of points defined over elliptic curves. For each prime p we will obtain a large collection of groups whose orders essentially vary "uniformly" over the interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$. By varying the groups involved we can hope to always find one whose order is smooth. We will then show how to take advantage of such a collection of groups to obtain a factorization of n.

C.11 Elliptic Curves

Definition C.36 An elliptic curve over a field F is the set of points (x, y) with $x,\ y \in F$ satisfying the Weierstrass equation $y^2 = x^3 + ax + b$ where $a, b \in F$ and $4a^3 + 27b^2 \neq 0$ together with a special point O called the point at infinity. We shall denote this set of points by $E_{a,b}(F)$. ▮

Remark The condition $4a^3 + 27b^2 \neq 0$ ensures that the curve is nonsingular. That is, when the field F is $\mathbf{R}$, the tangent at every point on the curve is uniquely defined.

Let P, Q be two points on an elliptic curve $E_{a,b}(F)$. We can define the negative of P and the sum $P+Q$ on the elliptic curve $E_{a,b}(F)$ according to the following rules.

1. If P is the point at infinity O then we define $-P$ to be O.
   Otherwise, if $P = (x, y)$ then $-P = (x, -y)$.

2. $O + P = P + O = P$.

3. Let $P, Q \neq O$ and suppose that $P = (x_1, y_1)$, $Q = (x_2, y_2)$.

   (i) If $P = -Q$ (that is, $x_1 = x_2$ and $y_1 = -y_2$) then we define $P + Q = O$

   (ii) Otherwise, let
   $$\alpha = \frac{y_1 - y_2}{x_1 - x_2} \text{ if } P \neq Q \tag{C.2}$$
   or
   $$\alpha = \frac{3x_1^2 + a}{y_1 + y_2} \text{ if } P = Q \tag{C.3}$$

That is, if the field $F = \mathbf{R}$, α is the slope of the line defined by P and Q, if $P \neq Q$, or the slope of the tangent at P, if $P = Q$.
Then $P + Q = R$ where $R = (x_3, y_3)$ with $x_3 = \alpha^2 - (x_1 + x_2)$ and $y_3 = \alpha(x_1 - x_3) - y_1$.

It can be shown that this definition of addition on the elliptic curve is associative and always defined, and thus it imposes an additive abelian group structure on the set $E_{a,b}(F)$ with O serving as the additive identity of the group. We will be interested in $F = \mathbf{Z}_p$ where $p \neq 2, 3$ is a prime. In this case, addition in $E_{a,b}(\mathbf{Z}_p)$ can be computed in time polynomial in $|p|$ as equations $(C.2)$ and $(C.3)$ involve only additions, subtractions, and divisions modulo p. Note that to compute $z^{-1} \bmod p$ where $z \in \mathbf{Z}_p^*$ we can use the extended Euclidean algorithm to compute an integer t such that $tz \equiv 1 \bmod p$ and then set $z^{-1} = t \bmod p$.

To illustrate negation and addition, consider the elliptic curve $y^2 = x^3 - x$ over $\mathbf{R}$ as shown in Figure C.1.

Figure C.1: Addition on the elliptic curve $y^2 = x^3 - x$.

The graph is symmetric about the x-axis so that the point P is on the curve if and only if $-P$ is on the curve. Also, if the line l through two points $P, Q \neq O$ on the curve $E(\mathbf{R})$ is not vertical then there is exactly one more point where this line intersects the curve. To see this, let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and let $y = \alpha x + \beta$ be the equation of the line through P and Q where $\alpha = \frac{y_1 - y_2}{x_1 - x_2}$ if $P \neq Q$ or $\alpha = \frac{3x_1^2 + a}{y_1 + y_2}$ if $P = Q$ and $\beta = y_1 - \alpha x_1$. Note that in the case where $P = Q$, we take l to be the tangent at P in accordance with the rules of addition on the curve $E(\mathbf{R})$. A point $(x, \alpha x + \beta)$ on the line l lies on the elliptic curve if and only if $(\alpha x + \beta)^2 = x^3 + ax + b$. Thus, there is one intersection point for each root of the cubic equation $x^3 - (\alpha x + \beta)^2 + ax + b = 0$. The numbers x_1 and x_2 are roots of this equation because $(x_1, \alpha x_1 + \beta)$ and $(x_2, \alpha x_2 + \beta)$ are, respectively, the points P and Q on the curve. Hence, the equation must have a third root x_3 where $x_1 + x_2 + x_3 = \alpha^2$. This leads to the expression for x_3 mentioned in the rules of addition for the curve $E(\mathbf{R})$. Thus, geometrically, the operation of addition on $E(\mathbf{R})$ corresponds to drawing the line through P and Q, letting the third intercept of the line with the curve be $-R = (x, y)$ and taking $R = (x, -y)$ to be the sum $P + Q$.

C.11.1 Elliptic Curves Over Z_n

Lenstra's elliptic curve factoring algorithm works with elliptic curves $E_{a,b}(\mathbf{Z}_n)$ defined over the ring $\mathbf{Z}_n$ where n is an odd, composite integer. The nonsingularity condition $4a^3 + 27b^2 \neq 0$ is replaced by $\gcd(4a^3 + 27b^2, n) = 1$. The negation and addition rules are given as was done for elliptic curves over fields. However, the addition of two points involves a division (refer to equations $(C.2)$ and $(C.3)$ in the rules for arithmetic on elliptic curves given at the beginning of this section) which is not always defined over the ring $\mathbf{Z}_n$. For addition to be defined the denominators in these equations must be prime to n. Consequently, $E_{a,b}(\mathbf{Z}_n)$ is not necessarily a group. Nevertheless, we may define a method for computing multiples $e \cdot P$ of a point $P \in E_{a,b}(\mathbf{Z}_n)$ as follows.

1. Let $a_0 + a_1 2 + \cdots + a_{m-1} 2^{m-1}$ be the binary expansion of e. Let $j = 0$, $S = 0$.

2. If $a_j = 1$ then $S \leftarrow S + 2^j P$. If this sum is not defined (namely, the division in equation $(C.2)$ or equation $(C.3)$ has failed) then output undefined and terminate.

3. $j \leftarrow j + 1$. If $j = m$ then output S as the defined value for $e \cdot P$ and terminate.

4. Calculate $2^j P := 2^{j-1} P + 2^{j-1} P$. If this sum is not defined then output that $e \cdot P$ cannot be calculated and terminate. Otherwise, repeat from step 2.

The elliptic curve algorithm will use[1] this method which will be referred to as repeated doubling. Note that if the repeated doubling method is unable to calculate a given multiple $e \cdot P$ and outputs undefined then we have encountered points $Q_1 = (x_1, y_1)$ and $Q_2 = (x_2, y_2)$ on $E_{a,b}(\mathbf{Z}_n)$ such that $Q_1 + Q_2$ is not defined modulo n (the division in equation $(C.2)$ or equation $(C.3)$ has failed) and hence, either $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of n.

Next, we state some facts concerning the relationship between elliptic curves defined over $\mathbf{Z}_n$ and elliptic curves defined over $\mathbf{Z}_p$ when p is a prime divisor of n. Let $a, b \in \mathbf{Z}_n$ be such that $\gcd(4a^3 + 27b^2, n) = 1$. Let p be a prime divisor of n and let $a_p = a \bmod p$, $b_p = b \bmod p$.

Fact C.37 $E_{a_p,b_p}(\mathbf{Z}_p)$ is an additive abelian group. ▮

Further, given $P = (x, y) \in E_{a,b}(\mathbf{Z}_n)$, define $P_p = (x \bmod p, y \bmod p)$. P_p is a point on the elliptic curve $E_{a_p,b_p}(\mathbf{Z}_p)$.

Fact C.38 Let P and Q be two points on $E_{a,b}(\mathbf{Z}_n)$ and let p be a prime divisor of n. If $P + Q$ is defined modulo n then $P_p + Q_p$ is defined on $E_{a,b}(\mathbf{Z}_p)$ and $P_p + Q_p = (P + Q)_p$. Moreover, if $P \neq -Q$ then the sum $P + Q$ is undefined modulo n if and only if there is some prime divisor q of n such that the points P_q and Q_q add up to the point at infinity O on $E_{a_q,b_q}(\mathbf{Z}_q)$ (equivalently, $P_q = -Q_q$ on $E_{a_q,b_q}(\mathbf{Z}_q)$). ▮

C.11.2 Factoring Using Elliptic Curves

The main idea in Lenstra's elliptic curve factoring algorithm is to find points P and Q in $E_{a,b}(\mathbf{Z}_n)$ such that $P + Q$ is <u>not defined</u> in $E_{a,b}(\mathbf{Z}_n)$. We will assume throughout that n has no factors of 2 or 3 because these can be divided out before the algorithm commences.

The algorithm runs as follows on input n.

1. Generate an elliptic curve $E_{a,b}(\mathbf{Z}_n)$ and a point $P = (x, y)$ on $E_{a,b}(\mathbf{Z}_n)$ by randomly selecting x, y and a in $\mathbf{Z}_n$ and setting $b = y^2 - x^3 - ax \bmod n$.

2. Compute $\gcd(4a^3 + 27b^2, n)$. If $1 < \gcd(4a^3 + 27b^2, n) < n$ then we have found a divisor of n and we stop. If $\gcd(4a^3 + 27b^2, n) = 1$ then $4a^3 + 27b^2 \not\equiv 0 \bmod p$ for every prime divisor p of n and hence $E_{a,b}$ is an elliptic curve over $\mathbf{Z}_p$ for each prime divisor p of n and we may proceed. But if $\gcd(4a^3 + 27b^2, n) = n$ then we must generate another elliptic curve $E_{a,b}$.

3. Set $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_1, p_2, \ldots, p_{\pi(B)}$ are the primes $\leq B$ and α_i is chosen maximally so that $p_i^{\alpha_i} \leq C$. B and C are bounds that will be determined later so as to optimize the running time and ensure that the algorithm will most likely succeed.

4. Compute $e \cdot P$ in $E_{a,b}(\mathbf{Z}_n)$ by repeated doubling. Every time before adding two intermediate points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ check if $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of n. If so, output the factor and stop. Otherwise, repeat from step 1.

An elliptic curve $E_{a,b}(\mathbf{Z}_n)$ will lead to a nontrivial factorization of n if for some prime factors p and q of n, $e \cdot P_p = O$ on $E_{a_p,b_p}(\mathbf{Z}_p)$ but P_q does not have order dividing e on $E_{a_q,b_q}(\mathbf{Z}_q)$. Notice the analogy here between Lenstra's elliptic curve algorithm and Pollard's $p - 1$ algorithm. In Pollard's algorithm we seek prime divisors p and q of n such that e is a multiple of the order of $a \in \mathbf{Z}_p^*$ but not a multiple of the order of $a \in \mathbf{Z}_q^*$. Similarly, in Lenstra's algorithm we seek prime divisors p and q of n such that e is a multiple of the order of $P_p \in E_{a_p,b_p}(\mathbf{Z}_p)$ but not a multiple of the order of $P_q \in E_{a_q,b_q}(\mathbf{Z}_q)$. However, there is a key

[1] This statement is incorrect and will soon be corrected in these notes. For the method used in the algorithm, see section 2.3 of [129]

difference in versatility between the two algorithms. In Pollard's algorithm, the groups $\mathbf{Z}_p^*$ where p ranges over the prime divisors of n are fixed so that if none of these groups have order dividing e then the method fails. In Lenstra's elliptic curve algorithm the groups $E_{a_p,b_p}(\mathbf{Z}_p)$ can be varied by varying a and b. Hence, if for every prime divisor p of n, $|E_{a_p,b_p}(\mathbf{Z}_p)| \not| e$, then we may still proceed by simply working over another elliptic curve; that is, choosing new values for a and b.

C.11.3 Correctness of Lenstra's Algorithm

Suppose that there are prime divisors p and q of n such that e is a multiple of $|E_{a_p,b_p}(\mathbf{Z}_p)|$ but in $E_{a_q,b_q}(\mathbf{Z}_q)$, P_q does not have order dividing e. Then $e \cdot P_p = O$ in $E_{a_p,b_p}(\mathbf{Z}_p)$ but $e \cdot P_q \neq O$ in $E_{a_q,b_q}(\mathbf{Z}_q)$ and thus there exists an intermediate addition of two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$ in the calculation of $e \cdot P$ such that

$$x_1 \equiv x_2 \bmod p \text{ but } x_1 \not\equiv x_2 \bmod q \text{ if } P_1 \neq P_2$$

or

$$y_1 \equiv -y_2 \bmod p \text{ but } y_1 \not\equiv -y_2 \bmod q \text{ if } P_1 = P_2.$$

Hence, either $\gcd(x_1 - x_2, n)$ or $\gcd(y_1 + y_2, n)$ is a nontrivial factor of n. The points P_1, and P_2 will be encountered when $(P_1)_p + (P_2)_p = O$ in $E_{a_p,b_p}(\mathbf{Z}_p)$ but $(P_1)_q + (P_2)_q \neq O$ in $E_{a_q,b_q}(\mathbf{Z}_q)$.

C.11.4 Running Time Analysis

The time needed to perform a single addition on an elliptic curve can be taken to be $M(n) = O(\log^2 n)$ if one uses the Euclidean algorithm. Consequently, since the computation of $e \cdot P$ uses repeated doubling, the time required to process a given elliptic curve is $O((\log e)(M(n))$. Recall that $e = \prod_{i=1}^{\pi(B)} p_i^{\alpha_i}$ where $p_i^{\alpha_i} \leq C$. Then $e \leq C^{\pi(B)}$ and therefore, $\log e \leq \pi(B) \log C$. Now, let p be the smallest prime divisor of n and consider the choice $B = L^\beta(p) = \exp[\beta(\log p)^{1/2}(\log\log p)^{1/2}]$ where β will be optimized. Then

$$\log B = \beta(\log p)^{1/2}(\log\log p)^{1/2} = e^{[\log\beta + \frac{1}{2}(\log\log p) + \frac{1}{2}(\log\log\log p)]}$$

and thus

$$\pi(B) \approx \frac{B}{\log B} \approx O\left(e^{[\beta(\log p)^{1/2}(\log\log p)^{1/2} - \frac{1}{2}(\log\log p)]}\right) \approx O(L^\beta(p)).$$

Hence, the time required for each iteration of the algorithm is $O(L^\beta(p)M(n)(\log C))$. In choosing C, note that we would like e to be a multiple of $|E_{a_p,b_p}(\mathbf{Z}_p)|$ for some prime divisor p of n and thus it is sufficient to take $C = |E_{a_p,b_p}(\mathbf{Z}_p)|$ where p is some prime divisor of n, provided that $|E_{a_p,b_p}(\mathbf{Z}_p)|$ is B-smooth. The value of p is unknown, but if p is the smallest prime divisor of n then $p < \sqrt{n}$. We also know by Hasse's Inequality (see, for example, page 131 of [190]) that $p + 1 - 2\sqrt{p} < |E_{a_p,b_p}(\mathbf{Z}_p)| < p + 1 + 2\sqrt{p}$ and thus $|E_{a_p,b_p}(\mathbf{Z}_p)| < \sqrt{n} + 1 + 2\sqrt[4]{n}$. Hence, it is safe to take $C = \sqrt{n} + 1 + 2\sqrt[4]{n}$.

The only remaining consideration is to determine the expected number of elliptic curves that must be examined before we obtain a factorization of n and this part of the analysis relies on the following result due to Lenstra which appears as Proposition 2.7 in [129].

Proposition C.39 Let $S = \{s \in \mathbf{Z} : |s - (p+1)| < \sqrt{p} \text{ and } s \text{ is } L(p)^\beta\text{-smooth}\}$. Let n be a composite integer that has at least two distincet prime divisors exceeding 3. Then

$$\Pr[\text{Lenstra's algorithm factors } n] \geq \Omega\left(\frac{|S| - 2}{2\sqrt{p}}\right)\left(\frac{1}{\log p}\right)$$

(where the probability is taken over x, y, and a in $\mathbf{Z}_n$). ∎

In other words, the proposition asserts that the probability that a random triple (x, y, a) leads to a factorization of n is essentially the probability that a random integer in the interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$ is $L(p)^\beta$-smooth; the latter probability being $\frac{|S|}{2\lfloor\sqrt{p}\rfloor+1}$.

Recall that we dealt earlier with the smoothness of integers less than some bound and saw that a theorem due to Canfield, Erdös and Pomerance (see [48]) implies that

$$\Pr[m \le x \text{ is } L(x)^\alpha\text{-smooth}] = L^{-\frac{1}{2\alpha}}(x).$$

However, as we have just seen, we require here the unproven conjecture that the same result is valid if m is a random integer in the small interval $(p+1-\sqrt{p}, p+1+\sqrt{p})$; specifically, that

$$\Pr[m \in (p+1-\sqrt{p}, p+1+\sqrt{p}) \text{ is } L(p)^\beta\text{-smooth}] = L^{-\frac{1}{2\beta}}(p).$$

Consequently, the lower bound on the probability of success in Proposition C.39 can be made explicit. Hence, we have

$$\Pr[\text{Lenstra's algorithm factors } n] \ge \Omega\left(L^{-\frac{1}{2\beta}}(p)\right)\frac{1}{\log p}.$$

Therefore, we expect to have to try $L^{\frac{1}{2\beta}}(p)(\log p)$ elliptic curves before encountering one of $L^\beta(p)$-smooth order. Thus, the total running time required is expected to be $O(L^{\frac{1}{2\beta}}(p)(\log p)L^\beta(p)M(n)(\log C)) = O(L^{\beta+\frac{1}{2\beta}}(p)(\log^4 n))$. This achieves a minimum of $O(L^{\sqrt{2}}(p)(\log^4 n))$ when $\beta = \frac{1}{\sqrt{2}}$.

Remark In step 3 of Lenstra's algorithm a minor practical problem arises with the choice of $B = L^\beta(p)$ because the smallest prime divisor p of n is not known before the algorithm begins. This problem can be resolved by taking $B = L^\beta(v)$ and performing the algorithm for a gradually increasing sequence of values for v while factorization continues to be unsuccessful and declaring failure if v eventually exceeds $\sqrt{n}$ because the smallest prime divisor of n is less than $\sqrt{n}$.

Chapter D

About PGP

PGP is a free software package that performs cryptographic tasks in association with email systems. In this short appendix we will review some of its features. For a complete description of its functioning readers are referred to Chapter 9 in [192].

D.1 Authentication

PGP performs authentication of messages using a hash-and-sign paradigm. That is given a message M, the process is as following:

- The message is timestamped, i.e. date and time are appended to it;
- it is then hashed using MD5 (see [169]);
- the resulting 128-bit digest is signed with the sender private key using RSA [170];
- The signature is prepended to the message.

D.2 Privacy

PGP uses a hybrid system to ensure privacy. That is each message is encrypted using a fast symmetric encryption scheme under a one-time key. Such key is encrypted with the receiver public-key and sent together with the encrypted message.

In detail, assume A wants to send an encrypted message to B.

- A compresses the message using the ZIP compression package; let M be the resulting compressed message.
- A generates a 128-bit random key k;
- The message M is encrypted under k using the symmetric encryption scheme IDEA (see [124] or Chapter 7 of [192]); let C be the corresponding ciphertext;
- k is encrypted under B's public key using RSA; let c be the corresponding ciphertext.

- The pair (c, C) is sent to B.

If both authentication and privacy are required, the message is first signed, then compressed and then encrypted.

D.3 Key Size

PGP allows for three key sizes for RSA

- *Casual* 384 bits
- *Commercial* 512 bits
- *Military* 1024 bits

D.4 E-mail compatibility

Since e-mail systems allow only the transmission of ASCII characters, PGP needs to recovert eventual encrypted parts of the message (a signature or the whole ciphertext) back to ASCII.

In order to do that PGP applies the radix-64 conversion to bring back a binary stream into the ASCII character set. This conversion expands the message by 33%. However because of the original ZIP compression, the resulting ciphertext is still one-third smaller than the original message.

In case the resulting ciphertext is still longer than the limit on some e-mail systems, PGP breaks into pieces and send the messages separately.

D.5 One-time IDEA keys generation

Notice that PGP does not have session keys, indeed each message is encrypted under a key k generated ad hoc for that message.

The generation of such key is done using a pseudo-random number generator that uses IDEA as a building block. The seed is derived from the keystrokes of the user. That is, form the actual keys being typed and the time intervals between them.

D.6 Public-Key Management

Suppose you think that PK is the public key of user B, while instead it is C who knows the corresponding secret key SK.

This can create two major problems:

1. C can read encrypted messages that A thinks she is sending to B
2. C can have A accept messages as coming from B.

The problem of establishing trust in the connection between a public-key and its owner is at the heart of public-key systems, no just of PGP.

There are various ways of solving this problem:

- *Physical exchange* B could give the key to A in person, stored in a floppy disk.
- *Verification* A could call B on the phone and verify the key with him
- *Certification Authorithy* There could be a trusted center $AUTH$ that signs public keys for the users, establishing the connection between the key and the ID of the user (such a signature is usually referred to as a *certificate*.)

Only the last one seems reasonable and it appears to be the way people are actually implementing public key systems in real life.

PGP does not use any of the above systems, but it rather uses a *decentralized trust* system. Users reciprocally certify each other's keys and one trusts a key to the extent that he/she trusts the user who certify it for it. Details can be found in [192]

CHAPTER E

Problems

This chapter contains some problems for you to look at.

E.1 Secret Key Encryption

E.1.1 DES

Let $\bar{m}$ be the bitwise complement of the string m. Let $\mathrm{DES}_K(m)$ denote the encryption of m under DES using key K. It is not hard to see that if

$$c = \mathrm{DES}_K(m)$$

then

$$\bar{c} = \mathrm{DES}_{\bar{K}}(\bar{m})$$

We know that a brute–force attack on DES requires searching a space of 2^{56} keys. This means that we have to perform that many DES encryptions in order to find the key, in the worst case.

1. Under *known plaintext attack* (i.e., you are given a single pair (m, c) where $c = \mathrm{DES}_K(m)$) do the equations above change the number of DES encryption you perform in a brute–force attack to recover K?

2. What is the answer to the above question in the case of *chosen plaintext attack* (i.e., when you are allowed to choose many m's for which you get the pair (m, c) with $c = \mathrm{DES}_K(m)$)?

E.1.2 Error Correction in DES ciphertexts

Suppose that n plaintext blocks $x_1, \ldots, x_n$ are encrypted using DES producing ciphertexts $y_1, \ldots, y_n$. Suppose that one ciphertext block, say y_i, is transmitted incorrectly (i.e. some 1's are changed into 0's and viceversa.) How many plaintext blocks will be decrypted incorrectly if the ECB mode was used for encryption? What if CBC is used?

E.1.3 Brute force search in CBC mode

A brute-force key search for a known-plaintext attack for DES in the ECB mode is straightforward: given the 64-bit plaintext and the 64 bit ciphertext, try all of the possible 2^{56} keys until one is found that generates the known ciphertext from the known plaintext. The situation is more complex for the CBC mode, which includes the use of a 64-bit IV. This seems to introduce an additional 64 bits of uncerntainty.

1. Suggest strategies for known-plaintext attack on the CBC mode that are of the same order of magnitude of effort as the ECB attack.
2. Now consider a ciphertext only attack. For ECB mode the strategy is to try to decrypt the given ciphertext with all possible 2^{56} keys and test each result to see if it appears to be a syntactically correct plaintext. Will this strategy work for the CBC mode? If so, explain. If not, describe an attack strategy for the CBC mode and estimate its level of effort.

E.1.4 E-mail

Electronic mail systems differ in the way in which multiple recipients are handled. In some systems the originating mail handler makes all the necessary copies, and these are sent out independently. An alterantive approach is to determine the route for each destination first. Then a single message is sent out on a common portion of the route and copies are made when the routes diverge (this system is known as *mail-bagging*.)

1. Leaving aside security considerations, discuss the relative advantages and disadvantages of the two methods.
2. Discuss the security requirements and implications of the two methods

E.2 Passwords

The framework of (a simplified version of) the Unix password scheme is this. We fix some function $h\colon \{0,1\}^k \to \{0,1\}^L$. The user chooses a k-bit password, and the system stores the value $y = h(K)$ in the password file. When the user logs in he must supply K. The system then computes $h(K)$ and declares you authentic if this value equals y.

We assume the attacker has access to the password file and hence to y. The intuition is that it is computationally infeasible to recover K from y. Thus h must be chosen to make this true.

The specific choice of h made by Unix is $h(K) = \mathrm{DES}_K(0)$ where "0" represents the 64 bit string of all zeros. Thus $k = 56$ and $L = 64$.

In this problem you will analyze the generic scheme and the particular DES based instantiation. The goal is to see how, given a scheme like this, to use the models we have developed in class, in particular to think of DES as a pseudorandom function family.

To model the scheme, let $F\colon \{0,1\}^k \times \{0,1\}^l \to \{0,1\}^L$ be a pseudorandom function family, having some given insecurity function $\mathbf{Adv}_F^{\mathrm{prf}}(\cdot,\cdot)$, and with $L > k$. We let T_F denote the time to compute F. (Namely the time, given K, x, to compute $F_K(x)$.) See below for the definition of a one-way function, which we will refer to now.

(a) Define $h\colon \{0,1\}^k \to \{0,1\}^L$ by $h(K) = F_K(0)$, where "0" represents the l-bit string of all zeros. Prove that h is a one-way function with

$$\mathbf{Adv}_{(}^{\mathrm{owf}} h, t) \;\leq\; 2 \cdot \mathbf{Adv}_F^{\mathrm{prf}}(t', 1)\ ,$$

where $t' = t + O(l + L + k + T_F)$.

Hints: Assume you are given an inverter I for h, and construct a distinguisher D such that

$$\mathbf{Adv}_F^{\mathrm{prf}}(D) \;\geq\; \frac{1}{2} \cdot \mathbf{Adv}_{h,I}^{\mathrm{owf}}\ .$$

Use this to derive the claimed result.

(b) Can you think of possible threats or weaknesses that might arise in a real world usage of such a scheme, but are not covered by our model? Can you think of how to protect against them? Do you think this is a good password scheme "in practice"?

We now provide the definition of security for a one-way function to be used above.

Let $h\colon \{0,1\}^k \to \{0,1\}^L$ be a function. It is one-way, if, intuitively speaking, it is hard, given y, to compute a point x' such that $h(x') = y$, when y was chosen by drawing x at random from $\{0,1\}^k$ and setting $y = h(x)$.

In formalizing this, we say an *inverter* for h is an algorithm I that given a point $y \in \{0,1\}^L$ tries to compute this x'. We let

$$\mathbf{Adv}^{\mathrm{owf}}_{h,I} \;=\; \mathbf{P}\left[\, h(x') = y \;:\; x \stackrel{R}{\leftarrow} \{0,1\}^k \;;\; y \leftarrow h(x) \;;\; x' \leftarrow I(y) \,\right]$$

be the probability that the inverter is successful, taken over a random choice of x and any coins the inverter might toss. We let

$$\mathbf{Adv}^{\mathrm{owf}}_{h}(t') \;=\; \max_I\{\mathbf{Adv}^{\mathrm{owf}}_{h,I}\}\;,$$

where the maximum is over all inverters I that run in time at most t'.

E.3 Number Theory

E.3.1 Number Theory Facts

Prove the following facts:

1. If k is the number of distinct prime factors of n then the equation $x^2 = 1 \bmod n$ has 2^k distinct solutions in Z_n^*. *Hint: use Chinese Remainder Theorem*
2. If p is prime and $x \in Z_p^*$ then $(\frac{x}{p}) = x^{\frac{p-1}{2}}$
3. g is a generator of Z_p^* for a prime p, iff $g^{p-1} = 1 \bmod p$ and $g^q \neq 1 \bmod p$ for all q prime divisors of $p-1$

E.3.2 Relationship between problems

Let n be the product of two primes $n = pq$. Describe reducibilities between the following problems (e.g. if we can factor we can invert RSA.) Don't prove anything formally, just state the result.

- computing $\phi(n)$
- factoring n
- computing $QR_n(a)$ for some $a \in Z_n^*$
- computing square roots modulo n
- computing k-th roots modulo n, where $gcd(k, \phi(n)) = 1$

E.3.3 Probabilistic Primality Test

Let $SQRT(p,a)$ denote an expected polynomial time algorithm that on input p, a outputs x such that $x^2 = a \bmod p$ if a is a quadratic residue modulo p. Consider the following probabilistic primality test, which takes as an input an odd integer $p > 1$ and outputs "composite" or "prime".

1. Test if there exist $b, c > 1$ such that $p = b^c$. If so output "composite"

2. Choose $i \in Z_p^*$ at random and set $y = i^2$
3. Compute $x = SQRT(p, y)$
4. If $x = i \bmod p$ or $x = -i \bmod p$ output "prime", otherwise output "composite"

(A) Does the above primality test always terminate in expected polynomial time? Prove your answer.

(B) What is the probability that the above algorithm makes an error if p is prime?

(C) What is the probability that the above algorithm makes an error if p is composite?

E.4 Public Key Encryption

E.4.1 Simple RSA question

Suppose that we have a set of block encoded with the RSA algorithm and we don't have the private key. Assume $n = pq, e$ is the public key. Suppose also someone tells us they know one of the plaintext blocks has a common factor with n. Does this help us in any way?

E.4.2 Another simple RSA question

In the RSA public-key encyption scheme each user has a public key n, e and a private key d. Suppose Bob leaks his private key. Rather than generating a new modulus, he decides to generate a new pair e', d'. Is this a good idea?

E.4.3 Protocol Failure involving RSA

Remember that an RSA public–key is a pair (n, e) where n is the product of two primes.

$$RSA_{(n,e)}(m) = m^e \bmod n$$

Assume that three users in a network Alice, Bob and Carl use RSA public–keys $(n_A, 3)$, $(n_B, 3)$ and $(n_C, 3)$ respectively. Suppose David wants to send the *same* message m to the three of them. So David computes

$$y_A = m^3 \bmod n_A \ , \ y_B = m^3 \bmod n_B \ , \ y_C = m^3 \bmod n_C$$

and sends the ciphertext to the relative user.

Show how an eavesdropper Eve can now compute the message m even without knowing any of the secret keys of Alice, Bob and Carl.

E.4.4 RSA for paranoids

The best factoring algorithm known to date (the *number field sieve*) runs in

$$e^{O(\log^{1/3} n \log \log^{2/3} n)}$$

That is, the running time does not depend on the size of the smallest factor, but rather in the size of the whole composite number.

The above observation seem to suggest that in order to preserve the security of RSA, it may not be necessary to increase the size of both prime factors, but only of one of them.

Shamir suggested the follwong version of RSA that he called *unbalanced RSA* (also known as RSA for paranoids). Choose the RSA modulus n to be 5,000 bits long, the product of a 500-bits prime p and a 4,500-bit prime q. Since usually RSA is usually used just to exchange DES keys we can assume that the messages being encrypted are smaller than p.

(A) How would you choose the public exponent e? Is 3 a good choice?

Once the public exponent e is chosen, one computes $d = e^{-1} \bmod \phi(n)$ and keep it secret. The problem with such a big modulus n, is that decrypting a ciphertext $c = m^e \bmod n$ may take a long time (since one has to compute $c^d \bmod n$.) But since we know that $m < p$ we can just use the Chinese Remainder Theorem and compute $m_1 = c^d \bmod p = m$. Shamir claimed that this variant of RSA achieves better security against the advances of factoring, without losing in efficiency.

(B) Show how with a single chosen message attack (i.e. obtaining the decryption of a message of your choice) you can completely break the unbalanced RSA scheme, by factoring n.

E.4.5 Hardness of Diffie-Hellman

Recall the Diffie-Hellman key exchange protocol. p is a prime and g a generator of Z_p^*. Alice's secret key is a random $a < p$ and her public key is $g^a \bmod p$. Similarly Bob's secret key is a random $b < p$ and his public key is $g^b \bmod p$. Their common key is g^{ab}.

In this problem we will prove that if the Diffie-Hellman key exchange protocol is secure for a small fraction of the values (a, b), then it is secure for almost all values (a, b).

Assume that there is a *ppt* algorithm $\mathcal{A}$ that

$$Prob[\mathcal{A}(g^a, g^b) = g^{ab}] > \frac{1}{2} + \epsilon$$

(where the probability is taken over the choices of (a, b) and the internal coin tosses of $\mathcal{A}$)

Your task is to prove that for any $\delta < 1$ there exists a *ppt* algorithm $\mathcal{B}$ such that for all (a, b)

$$Prob[\mathcal{B}(g^a, g^b) = g^{ab}] > 1 - \delta$$

(where the probability is now taken only over the coin tosses of $\mathcal{B}$)

E.4.6 Bit commitment

Consider the following "real life" situation. Alice and Bob are playing "Guess the bit I am thinking". Alice thinks a bit $b = 0, 1$ and Bob tries to guess it. Bob declares his guess and Alice tells him if the guess is right or not.

However Bob is losing all the time so he suspects that Alice is cheating. She hears Bob's guess and she declares she was thinking the opposite bit. So Bob requires Alice to write down the bit in a piece of paper, seal it in an envelope and place the envelope on the table. At this point Alice is committed to the bit. However Bob has no information about what the bit is.

Our goal is to achieve this bit commitment without envelopes. Consider the following method. Alice and Bob together choose a prime p and a generator g of Z_p^*. When Alice wants to commit to a bit b she choose a random $x \in Z_p^*$ such that $lsb(x) = b$ and she publishes $y = g^x \bmod p$. Is this a good bit commitment? Do you have a better suggestion?

E.4.7 Perfect Forward Secrecy

Suppose two parties, Alice and Bob, want to communicate privately. They both hold public keys in the traditional Diffie-Hellman model.

An eavesdropper Eve stores all the encrypted messages between them and one day she manages to break into Alice and Bob's computer and find their secret keys, correspondent to their public keys.

Show how using only public–key cryptography we can achieve *perfect forward secrecy*, i.e., Eve will not be able to gain any knowledge about the messages Alice and Bob exchanged before the disclosure of the secret keys.

E.4.8 Plaintext-awareness and non-malleability

We say that an encryption scheme is *plaintext–aware* if it is impossible to produce a valid ciphertext without knowing the corresponding plaintext.

Usually plaintext-aware encryption schemes are implemented by adding some redundancy to the plaintext. Decryption of a ciphertext results either in a valid message or in a flag indicating non–validity (if the redundancy is not of the correct form.) Correct decryption convinces the receiver that the sender *knows* the plaintext that was encrypted.

The concept of plaintext–awareness is related to the concept of malleability. We say that an encryption scheme E is *non–malleable* if it given a ciphertext $c = E(m)$ it is impossible to produce a valid ciphertext c' of a related message m'.

Compare the two definitions and tell us if one implies the other.

E.4.9 Probabilistic Encryption

Assume that you have a message m that you want to encrypt in a probabilistic way. For each of the following methods, tell us if you think it is a good or a bad method.

1. Fix p a large prime and let g be a generator. For each bit b_i in m, choose at random $x_i \in Z_{p-1}$ such that $lsb(x_i) = b_i$ ($lsb(x) =$ least significant bit of x.) The ciphertext is the concatenation of the $y_i = g^{x_i} \bmod p$. What about if you use x such that $msb(x_i) = b_i$?
2. Choose an RSA public key n, e such that $|n| > 2|m|$. Pad m with random bits to get it to the same length of n. Let m' be the padded plaintext. Encrypt $c = m^e \bmod n$.
3. Choose an RSA public key n, e. Assume that $|m|$ is smaller than $\log\log n$ (you can always break the message in blocks of that size.) Pad m with random bits to get it to the same length of n. Let m' be the padded plaintext. Encrypt $c = m^e \bmod n$.
4. Choose two large primes $p, q = 3 \bmod 4$. Let $n = pq$. For each bit b_i in m, choose at random $x_i \in Z_n^*$ and set $y_i = x_i^2 \bmod n$ if $b_i = 0$ or $y_i = -x_i^2 \bmod n$ if $b_i = 1$. The ciphertext is the concatenation of the y_i's.

E.5 Secret Key Systems

E.5.1 Simultaneous encryption and authentication

Let $(\mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme (cf. Chapter 6and MAC a message authentication code (cf. Chapter 8). Suppose Alice and Bob share two keys K_1 and K_2 for privacy and authentication respectively. They want to exchange messages M in a private and authenticated way. Consider sending each of the following as a means to this end:

1. $M, MAC_{K_2}(\mathcal{E}_{K_1}(M))$
2. $\mathcal{E}_{K_1}(M, MAC_{K_2}(M))$

3. $\mathcal{E}_{K_1}(M), MAC_{K_2}(M)$
4. $\mathcal{E}_{K_1}(M), \mathcal{E}_{K_1}(MAC_{K_2}(M))$
5. $\mathcal{E}_{K_1}(M), MAC_{K_2}(\mathcal{E}_{K_1}(M))$
6. $\mathcal{E}_{K_1}(M, A)$ where A encodes the identity of Alice. Bob decrypts the ciphertext and checks that the second half of the plaintext is A

For each say if it secure or not and briefly justify your answer.

E.6 Hash Functions

E.6.1 Birthday Paradox

Let H be a hash function that outputs m-bit values. Assume that H behaves as a *random oracle*, i.e. for each string s, $H(s)$ is uniformly and independently distributed between 0 and $2^m - 1$.

Consider the following brute–force search for a collision: try all possible $s_1, s_2, \ldots$ until a collision is found. (That is, keep hashing until some string yields the same hash value as a previously hashed string.)

Prove that the expected number of hashing performed is approximately $2^{\frac{m}{2}}$.

E.6.2 Hash functions from DES

In this problem we will consider two proposals to construct hash functions from symmetric block encryption schemes as DES.

Let E denote a symmetric block encryption scheme. Let $E_k(M)$ denote the encryption of the 1–block message M under key k. Let $M = M_0 \circ M_1 \circ M_2 \circ \ldots \circ M_n$ denote a message of $n+1$ blocks.

The first proposed hash function h_1 works as follows: let $H_0 = M_0$ and then define

$$H_i = E_{M_i}(H_{i-1}) \oplus H_{i-1} \text{ for } i = 1, \ldots, n.$$

The value of the hash function is defined as

$$h_1(M) = H_n$$

The second proposed hash function h_2 is similar. Again $H_0 = M_0$ and then

$$H_i = E_{H_{i-1}}(M_i) \oplus M_i \text{ for } i = 1, \ldots, n.$$

The value of the hash function is defined as

$$h_2(M) = H_n$$

For both proposals, show how to find collisions if the encryption scheme E is chosen to be DES.

E.6.3 Hash functions from RSA

Consider the followng hash function H. Fix an RSA key n, e and denote with $RSA_{n,e}(m) = m^e \bmod n$. Let the message to be hashed be $m = m_1 \ldots m_k$. Denote with $h_1 = m_1$ and for $i > 1$,

$$h_i = RSA_{n,e}(h_{i-1}) \oplus m_i$$

Then $H(m) = h_n$. Show how to find a collision.

E.7 Pseudo-randomness

E.7.1 Extending PRGs

Suppose you are given a PRG G which stretches a k bit seed into a $2k$ bit pseudorandom sequence. We would like to construct a PRG G' which stretches a k bit seed into a $3k$ bit pseudorandom sequence.

Let $G_1(s)$ denote the first k bits of the string $G(s)$ and let $G_2(s)$ the last k bits (that is $G(s) = G_1(s).G_2(s)$ where $a.b$ denotes the concatenation of strings a and b.)

Consider the two constructions

1. $G'(s) = G_1(s).G(G_1(s))$
2. $G''(s) = G_1(s).G(G_2(s))$

For each construction say whether it works or not and justify your answer. That is, if the answer is no provide a simple statistical test that distinguishes the output of, say, G' from a random $3k$ string. If the answer is yes prove it.

E.7.2 From PRG to PRF

Let us recall the construction of PRFs from PRGs we saw in class. Let G be a length-doubling PRG, from seed of length k to sequences of length $2k$.

Let $G_0(x)$ denote the first k bits of $G(x)$ and $G_1(x)$ the last k bits. In other words $G_0(x) \circ G_1(x) = G(x)$ and $|G_0(x)| = |G_1(x)|$.

For any bit string z recursively define $G_{0\circ z}(x) \circ G_{1\circ z}(x) = G(G_z(x))$ with $|G_{0\circ z}(x)| = |G_{1\circ z}|$.

The PRF family we constructed in class was defined as $\mathcal{F} = \{f_i\}$. $f_i(x) = G_x(i)$. Suppose instead that we defined $f_i(x) = G_i(x)$. Would that be a PRF family?

E.8 Digital Signatures

E.8.1 Table of Forgery

For both RSA and ElGamal say if the scheme is

1. universally forgeable
2. selectively forgeable
3. existentially forgeable

and if it is under which kind of attack.

E.8.2 ElGamal

Suppose Bob is using the ElGamal signature scheme. Bob signs two messages m_1 and m_2 with signatures (r, s_1) and (r, s_2) (the same value of r occurs in both signatures.) Suppose also that $gcd(s_1 - s_2, p - 1) = 1$.

1. Show how k can be computed efficiently given this information
2. Show how the signature scheme can subsequently be broken

E.8.3 Suggested signature scheme

Consider the following discrete log based signature scheme. Let p be a large prime and g a generator. The private key is $x < p$. The public key is $y = g^x \bmod p$.

To sign a message M, calculate the hash $h = H(M)$. If $gcd(h, p-1)$ is different than 1 then append h to M and hash again. Repeat this until $gcd(h, p-1) = 1$. Then solve for Z in

$$Zh = X \bmod (p-1)$$

The signature of the message is $s = g^Z \bmod p$. To verify the signature, a user checks that $s^h = Y \bmod p$.

1. Show that valid signatures are always accepted
2. Is the scheme secure?

E.8.4 Ong-Schnorr-Shamir

Ong, Schnorr and Shamir suggested the following signature scheme.

Let n be a large integer (it is not necessary to know the factorization of n.) Then choose $k \in Z_n^*$. Let

$$h = -k^{-2} \bmod n = -(k^{-1})^2 \bmod n$$

The public key is (n, h), the secret key is k.

To sign a message M, generate a random number r, such that r and n are relatively prime. Then calculate

$$S_1 = \frac{M/r + r}{2} \bmod n$$

$$S_2 = \frac{k}{2}(M/r - r)$$

The pair (S_1, S_2) is the signature.

To verify the signature, check that

$$M = S_1^2 + hS_2^2 \bmod$$

1. Prove that reconstructing the private key, from the public key is equivalent to factor n.
2. Is that enough to say that the scheme is secure?

E.9 Protocols

E.9.1 Unconditionally Secure Secret Sharing

Consider a generic Secret Sharing scheme. A dealer D wants to share a secret s between n trustees so that no t of them have *any* information about s, but $t+1$ can reconstruct the secret. Let s_i be the share of trustee T_i. Let v denote the number of possible values that s might have, and let w denote the number of different possible share values that a given trustee might receive, as s is varied. (Let's assume that w is the same for each trustee.)

Argue that $w \geq v$ for any Secret Sharing Scheme. (It then follows that the number of bits needed to represent a share can not be smaller than the number of bits needed to represent the secret itself.)

Hint: Use the fact that t players have NO *information about the secret—no matter what t values they have received, any value of s is possible.*

E.9.2 Secret Sharing with cheaters

Dishonest trustees can prevent the reconstruction of the secret by contributing *bad* shares $\hat{s_i} \neq s_i$. Using the cryptographic tools you have seen so far in the class show how to prevent this denial of service attack.

E.9.3 Zero–Knowledge proof for discrete logarithms

Let p be a prime and g a generator modulo p. Given $y = g^x$ Alice claims she knows the discrete logarithm x of y. She wants to convince Bob of this fact but she does not want to reveal x to him. How can she do that? (Give a zero-knowledge protocol for this problem.)

E.9.4 Oblivious Transfer

An oblivious transfer protocol is a communication protocol between Alice and Bob. Alice runs it on input a value s. At the end of the protocol either Bob learns s or he has no information about it. Alice has no idea which event occurred.

An 1-2 oblivious transfer protocol is a communication protocol between Alice and Bob. Alice runs it on inout two values s_0 and s_1. Bob runs it on input a bit b. At the end of the protocol, Bob learns s_b but has no information about s_{1-b}. Alice has no information about b.

Show that given an oblivious transfer protocol as a black box, one can design a 1-2 oblivious transfer protocol.

E.9.5 Electronic Cash

Real-life cash has two main properties:

- It is *anonymous*: meaning when you use cash to buy something your identity is not revealed, compare with credit cards where your identity and spending habits are disclosed
- It is *transferable*: that is the vendor who receives cash from you can in turn use it to buy something else. He would not have this possibility if you had payed with a non-transferable check.

The electronic cash proposals we saw in class are all "non–transferable". that is the user gets a coin from the bank, spends it, and the vendor must return the coin to the bank in order to get credit. As such they really behave as anonymous non-transferable checks. In this problem we are going to modify such proposals in order to achieve transferability.

The proposal we saw in class can be abstracted as follows: we have three agents: the Bank, the Userand the Vendor.

The Bankhas a pair of keys (S, P). A signature with S is a *coin* worth a fixed amount (say \$1.). It is possible to make blind signatures, meaning the Usergets a signature $S(m)$ on a message m, but the Bankgets no information about m.

Withdrawal protocol

1. The Userchooses a message m
2. The Bankblindly signs m and withdraws \$1 from User's account.
3. The Userrecovers $S(m)$. The coin is the pair $(m, S(m))$.

Payment Protocol

1. The Usergives the coin $(m, S(m))$ to the Vendor.

2. The Vendorverifies the Banksignature and sends a random challenge c to the User.
3. The Userreplies with an answer r
4. the Vendorverifies that the answer is correct.

The challenge–response protocol is needed in order to detect double–spending. Indeed the system is constructed in such a way that if the Useranswers two different challenges on the same coin (meaning he's trying to spend the coin twice) his identity will be revealed to the Bankwhen the two coins return to the bank. This is why the whole history of the payment protocol must be presented to the Bankwhen the Vendordeposits the coin.

Deposit protocol

1. The Vendorsends $m, S(m), c, r$ to the Bank
2. The Bankverifies it and add \$1 to the Vendor's account.
3. The Banksearches its database to see if the coin was deposited already and if it was reconstruct the identity of the double–spender User.

In order to make the whole scheme transferrable we give the bank a different pair of keys $(\mathcal{S}, \mathcal{P})$. It is still possible to make blind signatures with $\mathcal{S}$. However messages signed with $\mathcal{S}$ have no value. We will call them *pseudo-coins*. When people open an account with the Bank, they get a lot of these anonymous pseudo–coins by running the withdrawal protocol with $\mathcal{S}$ as the signature key.

Suppose now the Vendorreceived a payed coin $m, S(m), c, r$ and instead of depositing it wants to use it to buy something from OtherVendor. What she could do is the following:

Transfer protocol

1. The Vendorsends $m, S(m), c, r$ and a pseudo–coin $m', \mathcal{S}(m')$ to OtherVendor
2. OtherVendorverifies all signatures and the pair (c, r). Then sends a random challenge c' for the pseudo–coin.
3. Vendorreplies with r'
4. OtherVendorchecks the answer.

Notice however that Vendorcan still double–spend the coin $m, S(m), c, r$ if she uses two different pseudo–coins to transfer it to two differnt people. Indeed since she will never answer two different challenges on the same pseudo–coin, her identity will never be revealed. The problem is that there is no link between the real coin and the pseudo-coin used during the transfer protocol. If we could force Vendorto use only one pseudo–coin for each real coin she wants to transfer then the problem would be solved.

Show how to achieve the above goal. You will need to modify both the payment and the transfer protocol.

Hint: If Vendor*wants to transfer the true coin she is receiving during the payment protocol, she must be forced* then *to create a link between the true coin and the pseudo–coin she will use for the transfer later. Notice that* Vendor*chooses c at random, maybe c can be chosen in some different way?*

E.9.6 Atomicity of withdrawal protocol

Recall the protocol that allows a Userto withdraw a coin of \$1 from the Bank. Let $(n, 3)$ be the RSA public key of the Bank.

1. The Userprepares 100 messages $m_1, \ldots, m_{100}$ which are all \$1 coins. The User*blinds* them, that is she chooses at random $r_1, \ldots, r_{100}$ and computes $w_i = r_i^3 m_i$. The Usersends $w_1, \ldots, w_{100}$ to the Bank.

2. The Bankchooses at random 99 of the blindings and asks the Userto open them. That the Bankchooses $i_1, \ldots, i_{99}$ and sends it to the User.

3. The Useropens the required blindings by revealing $r_{i_1}, \ldots, r_{i_{99}}$.

4. The Bankchecks that the blindings are constructed correctly and then finally signs the unopened blinding. W.l.o.g. assume this to be the first one. So the Banksigns w_1 by sending to the User$w_1^{\frac{1}{3}} = r_1 m_1^{\frac{1}{3}}$

5. The Userdivides this signature by r_1 and gets a signature on m_1 which is a valid coin.

Notice that the Userhas a probability of $1/100$ to succesfully cheat.

Suppose now that the protocol is not atomic. That is the communication line may go down at the end of each step between the Bankand the User. What protocol should be followed for each step if the line goes down at the end of that step in order to prevent abuse or fraud by either party?

E.9.7 Blinding with ElGamal/DSS

In class we saw a way to blind messages for signatures using RSA. In this problem we ask you to construct blind signatures for a variation of the ElGamal signature scheme.

The ElGamal-like signature we will consider is as follows. Let p be a large prime, q a large prime dividing $p-1$, g an element of order q in Z_p^*, x the secret key of the Bankand $y = g^x$ the corresponding public key. Let H be a collision-free hash function.

When the Bankwants to sign a message m she computes

$$a = g^k \bmod p$$

for a random k and

$$c = H(m, a)$$

and finally

$$b = kc + xa \bmod q$$

The signature of the message m is $sig(m) = (a, b)$. Given the triple (m, a, b) the verification is performed by computing $c = H(m, a)$ and checking that

$$g^b = a^c y^a$$

So the withdrawal protocol could be as following:

1. The Usertells the bank she wants a \$1 coin.

2. The Bankreplies with 100 values $a_i = g^{k_i}$ for random k_i.

3. The Usersends back $c_i = H(m_i, a_i)$ where m_i are all \$1 coins.

4. The Bankasks the user to open 99 of those.

5. The Userreveals 99 of the m_i's.

6. The Bankreplies with $b_i = k_i c_i + x a_i \bmod (p-1)$ for the unopened index i

However this is not anonymous since the Bankcan recognize the Userwhen the coin comes back. In order to make the protocol really anonymous, the Userhas to change the value of "challenge" c_i computed at step 3. This modification will allow him to compute a different signature on m_i on her own which will not be recognizable to the Bankwhen the coin comes back. During the protocol the Bankwill check as usual that this modification has been performed correctly by asking the Userto open 99 random blindings.