CS 282A/MATH 209A: Foundations of Cryptography Prof. Rafail Ostrosky

Lecture 1

Lecture date: 1/10-1/12, 2005 *Scribe: Vishwa Goudar, Bo Huang, Kirill Minkovich*

OUTLINE

1. Survey of Cryptography Topics
2. Complexity Classes and Reducing Error Probability of Algorithms
3. Comparison of Uniform and Non-Uniform Complexity Classes

1 Survey of Cryptography Topics

This section comprises of an introduction to cryptography, the various problems and cryptographic primitives.

1.1 Brief History

Cryptography is a science with a long history. Although it has recently been transformed to conform to the rigorousness of a science, its applications can be traced back to the times of the Romans when ciphers such as the Caesar Cipher were used to ensure the secrecy of messages. These types of ciphers are in fact easy to break. The concepts of cryptography have only recently been formalized:

- Claude Shannon (1940s)
  - Formalized "Perfect Security" while working on the ENIGMA project
  - Defined the private key communication model
- Whitfield Diffie and Martin Hellman (1970s)
  - Laid foundations for cryptographic techniques based on complexity theoretical assumptions
  - Defined Public-Key Cryptography
- Shafi Goldwasser and Silvio Micali (1980s)
  - Provided formal and first satisfactory definition of encryption

1.2 Problem Space

Secrecy

The problem of secrecy is defined as follows: Alice wants to send Bob a message and Eve could wiretap the line and listen to the message. But Alice wants to make sure that Eve won't understand the message (see Figure 1). Note that Eve could cut the wire and disrupt the communication but this is not the problem that secrecy considers.

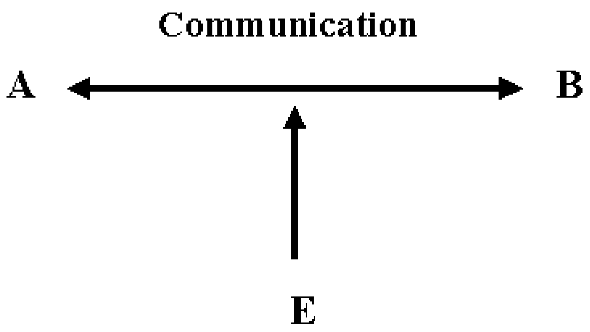

Figure 1: A communicates with B while E tries to intercept and understand the communication

A solution to this problem includes private key communication (studied by Shannon) where Alice and Bob get together beforehand and share a secret key. Then, when Alice and Bob communicate over the wire, Alice transforms the message (using the key) before sending it. Bob then reverses the transformation to get the message. Note that Eve does not know this private key and ,hopefully, will not be able to understand the message.

Public Key Encryption

Another solution is the public key communication (studied by Diffie Helman). This system uses two keys: a public key known to everyone and a private key known only to the recipient of the message. When Alice wants to send a secure message to Bob, she uses Bob's public key (which can be found in a public directory) to encrypt the message. Bob, upon receiving the message, then uses his private key to decrypt it.

Additional requirements imposed by today's systems

Today's systems impose an additional requirement on secrecy protocols. In addition to not being able to decipher the transformed message, Eve should not be able to replace the transformed message with another message whose meaning is somewhat related to the original message.

Authentication/Data Integrity

In this problem Alice would like to send Bob a message. Eve can replace the message (see Figure 2). In this scenario, Bob can unknowingly receive a message that is different from the one Alice originally sent him.

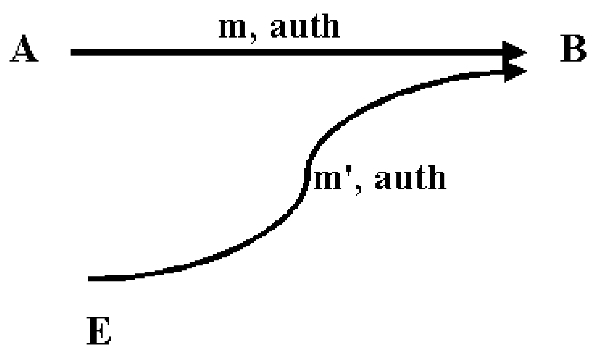

Figure 2: B tries to check if the message is coming from A or E

Therefore, Bob must figure out whether the message he received was the same as the one originally sent from Alice or if it was sent by someone else. If Eve changes Alice's original message, Bob must detect it, otherwise, Bob must confirm that the message he received is really the same one that Alice sent him.

One solution to this problem is the private key protocol where Alice and Bob get together beforehand and share a secret key. Then, when Alice and Bob communicate over the wire, Alice sends the message to Bob and appends some form of "authentication" using the secret key they both agreed on. Bob can verify if the message is indeed from Alice by checking the "authentication" since he knows the secret key as well. Note that Eve does not know this private key and hopefully will not be able to come up with the authentication of any other message M'.

Another solution is to use public key cryptography that works in the following way: each participant in the communication owns a secret private key that corresponds to a public key. Every participant's public key is listed in a directory and hence everyone including the adversary (such as Eve) and the participants knows the participants' public keys. In order to ensure data integrity of her message, Alice would sign her message using her private key before sending it. This involves Alice using her private key and appending this "signature" to her plain text message. The "signature" is, of course, a function of the original plain text message. Bob then uses Alice's public key (that he would have retrieved from the directory) to verify the signature and check whether the plain text message was indeed signed by Alice.

1.3 Cryptographic Primitives

Cryptographic primitives are used as building blocks to build more advanced security protocols and crypto-systems.

Zero Knowledge Proofs

The task at hand here is the following: Prover P would like to convince verifier V that she(P) knows a secret without actually giving away the secret to V. For the time being, let us look at this somewhat informal example: if P says that she has the magical ability to look at a big bushy tree and count the exact number of leaves. Since V cannot directly verify this, she could do the following. V could have P count the number of leaves on a big bushy tree, ask P to turn around and blind fold P. Then V could secretly flip a coin to determine whether or not to pluck a leaf from the tree. V could then have P recount the leaves. Given that P does not know of V's secret actions, if P does not, in fact, have the magical ability, she has a 1/2 probability of producing the right number. This probability could be further reduced to a tiny number by repeating the process multiple times. If P gets it right every time, she indeed has the magical ability with a high probability. Note that throughout the procedure, P does not reveal her "magical" method of counting the leaves to V.

Coin Flipping Protocol

A coin flipping protocol is one where two participants A and B, who don't trust each other, agree on a fair coin flip over the phone such that neither of them can cheat.

Commitment Protocol

A commitment protocol is a communication protocol between A and B where A first passes on a message that is sealed in a safe to B. Since the message is sealed, B cannot read the message, however A can no longer change her mind. At a later time A gives the combination to the safe to B, that B then uses to unlock the message (see Figure 3).

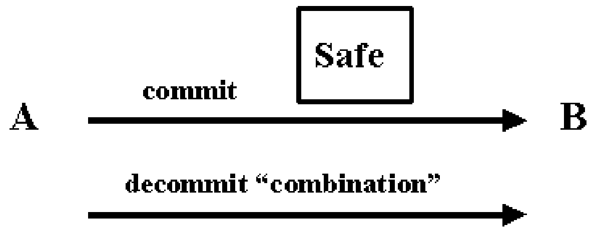

Figure 3: A first sends the safe then the combination

Secure two-party Computation

The goal here is for two participants to agree on a decision without revealing their respective inputs to the other participant. For example, consider two millionaires A and B who want to figure out who is richer (example due to Yao's). To be discrete, they would like to do this without revealing their net worth to each other. In fact we will show that they can do it later in this course. In general, if A and B want to compute the output of a function f with two inputs x and y (where x is a secret input for A and y is a secret input for B), and a binary output such that input x comes from A and input y comes from B. We will see that this is possible even though A does not reveal x to B and B does not reveal y to A for any polynomial-time computable function f.

2 Complexity Classes and Reducing Error Probability of Algorithms

2.1 Cryptography and Complexity Classes

Definitions

- A **One-Way Function** is a function that is "easy" to compute on any given input, but whose output is "hard" to invert in order to retrieve the input.
- A **One-Way Permutation** is a One-Way Function that is a bijection or one-to-one and onto.
- A **One-Way Trapdoor Permutation** is a One-Way Permutation that is hard to invert but given a "trapdoor" becomes easy to invert.

NOTE: An "easy" computation is a poly-time computation. A "hard" computation is a super poly-time computation.

The aim of cryptography is to make it computationally impossible for an adversary to break a protocol (unless, of course, the adversary is incredibly lucky and guesses correctly all the secret randomness used or has exponential memory that can be used to pre-compute all the possibilities). This assumes computationally hard problems exist. Since we do not know the status of P vs. NP, we do not know if hard problems really exist. Thus we take an axiomatic approach to prove computational infeasibility of the algorithm. That is, as long as there are some "hard" problems, we want to build cryptographic protocols on any such problem. For example, factoring a number is believed to be computationally infeasible if the number, N, is a product of two large primes. There is no known poly-time algorithm that can factor such a number. This is used to make further inferences such as the existence of cryptographic primitives.

A sequence of proofs that we will see as an example to show this: we assume there is no poly-time algorithm to factor N

$\Rightarrow \exists$ A One-way Function

$\Rightarrow \exists$ A pseudo-random generator

$\Rightarrow \exists$ A secure commitment scheme

$\Rightarrow \exists$ A coin-flipping protocol

Therefore, given that we do not know anything about the equality relationship between P and NP, we cannot say that there exists a truly secure coin flipping protocol but what we can say is the following: Assuming there is no poly-time algorithm to factor N, there exists a sequence of implications that imply that a secure coin flipping protocol does exists.

Hence given a secure coin-flipping protocol, if we can come up with an algorithm that can cheat in such a protocol, it can be used as a subroutine in an algorithms to cheat a secure commitment scheme and so on until we are able to factorize N, thus we are able to reach a contradiction to our assumption that factoring is hard.

As a final note, we never advocate security by obscurity. For any protocol, we assume that the adversary is aware of the protocol being used and we make no assumptions about the adversary.

2.2 Algorithms versus Languages

A language is a subset of a universal set of alphabet, that adheres to some rules. E.g. the set of palindromes constitutes a languages. So does the set of integers that are products of two large primes.

An algorithm, on the other hand, is a procedure that decides if a given input is in the language. Since we do not allow security by obscurity (i.e. all algorithms should be known to the adversary) there is no way to keep a secret from the adversary if everything is deterministic. The trick is to make the machine M flip coins (see Figure 4).

A **Probabilistic Turing Machine** can be modelled as a Turing Machine which has an additional input tape that is a special read-once tape that contains all random coin flips that the machine will flip. (see Figure 5) That is, if the machine wants to remember previous inputs from the random tape it must store these in its memory.

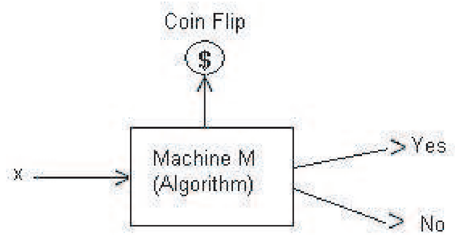

Figure 4: Decides if x is in the language

Now given such a Probabilistic Turing Machine that runs a probabilistic algorithm to decide if an input string is in the language, there will be a probability for error in the decision process.

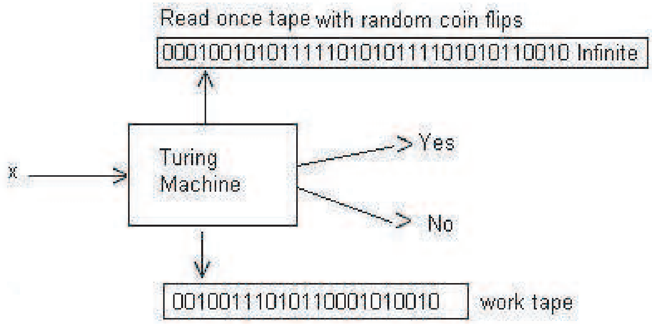

Figure 5: Example of a Turing Machine with a read-once tape

2.3 Randomized Poly-time Algorithms (RP)

Randomized Poly-time Algorithms are probabilistic algorithms that run in poly-time and incur a 1-sided error. That is, given a machine M with coin flips W running such an algorithm to detect whether an input string x is in the language L,

$$Pr_w[M(x) = yes|x \in L] \geq p \tag{1}$$
$$Pr_w[M(x) = yes|x \notin L] = 0 \tag{2}$$

where p is some probability, e.g. 1/2. Intuitively we would like p to be large. (see Figure 6 and Figure 7 for a graphical representation)

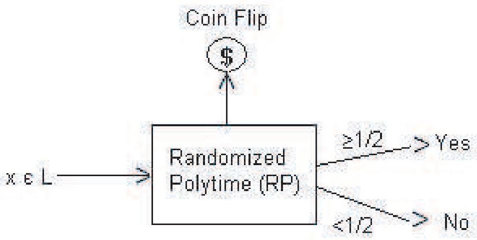

Figure 6: When x is in the set L

The set of Randomized Poly-time Algorithms make up the complexity class RP. We can also define the set co-RP where the 1-sided error is on false positive instances. That is, it consists of the set of algorithms wherein given a machine M running such an algorithm to detect whether an input string x is in the language L,

$$Pr_w[M(x) = yes|x \in L] = 1 \tag{3}$$
$$Pr_w[M(x) = yes|x \notin L] \leq p \tag{4}$$

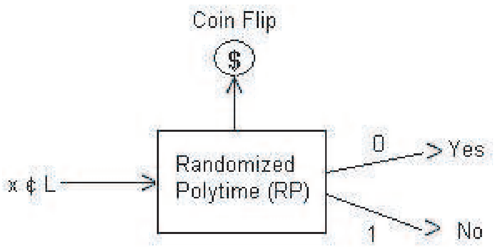

Figure 7: When x is not in the set L

where p is some probability, e.g. 1/2.

Note that since RP and co-RP algorithms are poly-time, their running time is polynomial in the length of the input string x. Therefore, the number of coin-flips is also also polynomial in the length of the input string x.

Monte Carlo algorithms are 1-sided estimation algorithms that run in poly-time. The monte-carlo algorithms can be in the set RP or co-RP.

2.4 Bounded Probabilistic Poly-time Algorithms (BPP)

Bounded Probabilistic Poly-time Algorithms are probabilistic algorithms that are also poly-time but incur 2-sided error. This means that given a machine M running such an algorithm to detect whether an input string x is in the language L,

$$Pr_w[M(x) = yes | x \in L] \geq \frac{1}{2} + \epsilon \tag{5}$$

$$Pr_w[M(x) = yes | x \notin L] \leq \frac{1}{2} - \epsilon \tag{6}$$

Where ϵ is some constant, for our examples, let $\epsilon = \frac{1}{6}$.

The set of Bounded Probabilistic Poly-time Algorithms make up the BPP complexity class. The Atlantic city algorithm is BPP (i.e. a two-sided error estimation algorithm) (see Figure 8 and Figure 9).

2.5 Expected Poly-time Algorithms

Expected Poly-time algorithms have an expected running time that is polynomial in the size of the input. They are also called Las Vegas algorithms. Notice that such algorithms

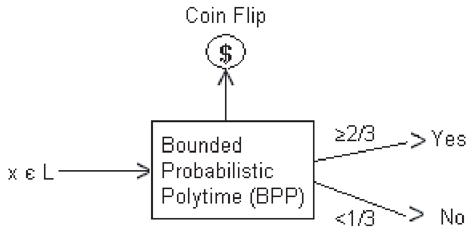

Figure 8: BPP (Atlantic city) algorithm where x is in the set L

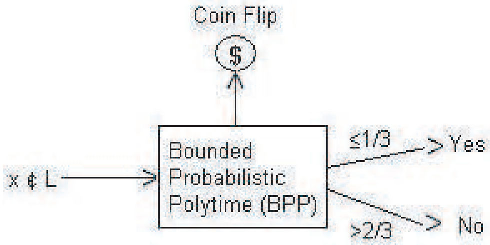

Figure 9: BPP (Atlantic city) algorithm where x is not in the set L

can run with small probability for exponential time. For example, they can run for 2^k steps with probability of $1/2^k$.

2.6 Reducing Error Probability of Algorithms

Given the definitions of RP and BPP algorithms, we would like to reduce the probability of error in both cases.

RP Algorithms

For RP algorithms, the probability of error is reduced by simply running the algorithm on the same input multiple times using fresh coin flips each time. If it says "yes" even once, then $x \epsilon L$. The new machine M' created by k runs of machine M now is a machine running an RP algorithm, but whose probability of error is $(1\text{-}p)^k$. For example, if p is 1/2, then the probability of M' rejecting an input string x that is truly in the language is $1/2^k$. A similar approach applies to reducing the error probability for co-RP algorithms but instead it checks if the machine says "no" at least once.

BPP Algorithms

Intuitively, it makes sense to run a machine M running a BPP algorithm multiple times to reduce the error probability. But how do we decide what to output? We do it by running machine M many times with fresh coin flips each time and takeing a majority vote as a new output (i.e. if we have more yes than no, then yes is our new output). But we would like to know how the reduced probability of this new maching M' relates to the number of times, k, it runs machine M. This relationship is quantified using the Chernoff bound.

The **Chernoff Bound** states that given n independent random variables X_1, X_2 .... X_n with identical probability distributions, if X = $\sum_{i=1}^{n}$ X_i

$$Pr[X \geq (1+\beta)E(x)] < e^{-\beta^2 E(X)/2} \tag{7}$$

Consider the case where BPP machine M works with 2/3, 1/3 error probability. The new machine M' runs M k times. Let X_i=1 if M' makes a mistake on the i^{th} run of machine M. Since the probability of M making an error is independent of previous outcomes of M,

$$Pr[X_i = 1] = (1 - 2/3) = 1/3 \tag{8}$$

$$\tag{9}$$

If we run the machine k time, then

$$E(X) = k/3 \tag{10}$$

M' rejects an input string x if more than half of the outcomes of M are rejects, and accepts it otherwise. Thus M' falsely rejects input string x if $\sum_{i=1}^{n}$ $X_i \geq$ k/2. Set β to 1/2. Plugging these numbers into the Chernoff Bound, we see that

$$Pr[X \geq \frac{3}{2} \cdot \frac{k}{2}] < e^{-k/24} \tag{11}$$

Notice that the probability of M' making a mistake reduces exponentially as a function of the number of runs of M, in our case k.

3 Comparison of Uniform and Non-Uniform Complexity Classes

3.1 Uniform Complexity Classes

Uniform complexity algorithms are those that use Turing Machines to make their decision. An example of this class is the set of BPP algorithms.

3.2 Non-Uniform Complexity Classes

Non-uniform complexity algorithms are defined using families of circuits to make their decision. That is, for each input length, we define a different circuit of polynomial size that makes decisions about language membership. One way to think about this is Turing Machines running algorithms in this class magically get as an advice, a circuit that makes poly-time decisions about inclusion of inputs of that length to the language and then simulates the circuit with the given input. This class is the set of **P/poly** algorithms.

3.3 P/Poly $\subseteq$ NP?

The following is a bogus proof for the claim that P/Poly $\subseteq$ NP.

We can construct an NP machine that guesses if there exists an advice to a P/Poly machine that would make the P/Poly machine accept input x. If so the NP machine would accept x, otherwise it would reject. Therefore, we can claim that P/Poly $\subseteq$ NP. The problem with this proof however is that the NP machine uses the advice to make a decision but never attempts to check the validity of the advice.

To give an idea of the power of P/poly we will in fact show that it can recognize undecidable languages. Consider some standard enumeration of the Turing machines $M_1, M_2, ...$ Now consider the following language L: x$\in$L iff machine number $|x|$ halts. (i.e. the length of x indicates the number of the machine in the standard enumeration above). Since we are allowed magic advice for each input length, the magic advice could tell us which machines halt and which do not in the above enumeration. But this is undecidable... so there are very hard languages which can be recognized in P/poly.

3.4 BPP $\subseteq$ P/Poly

P/poly circuits are so powerful that they do not require coin flips to make decisions. In fact, Leonard Adleman proved that BPP $\subseteq$ P/Poly.

Theorem 1 *BPP $\subseteq$ P/Poly. [Adleman]*

Proof Given a BPP machine M with 2/3 probability of falsely rejecting an input and 1/3 probability of falsely accepting an input, we can easily create another BPP machine M' that runs M many times on the input to reduce the probability of error to a very small value [see previous subsection]. Let this new machine M' be characterized in the following way:

$$Pr_w[M'(x) = yes | x \in L] > (1 - 2^{-(n+1)}) \quad (12)$$

$$Pr_w[M'(x) = yes | x \notin L] < 2^{-(n+1)} \quad (13)$$

where **n** is the size of the input x. Let **r** be the number of coin flips w performed by the machine.

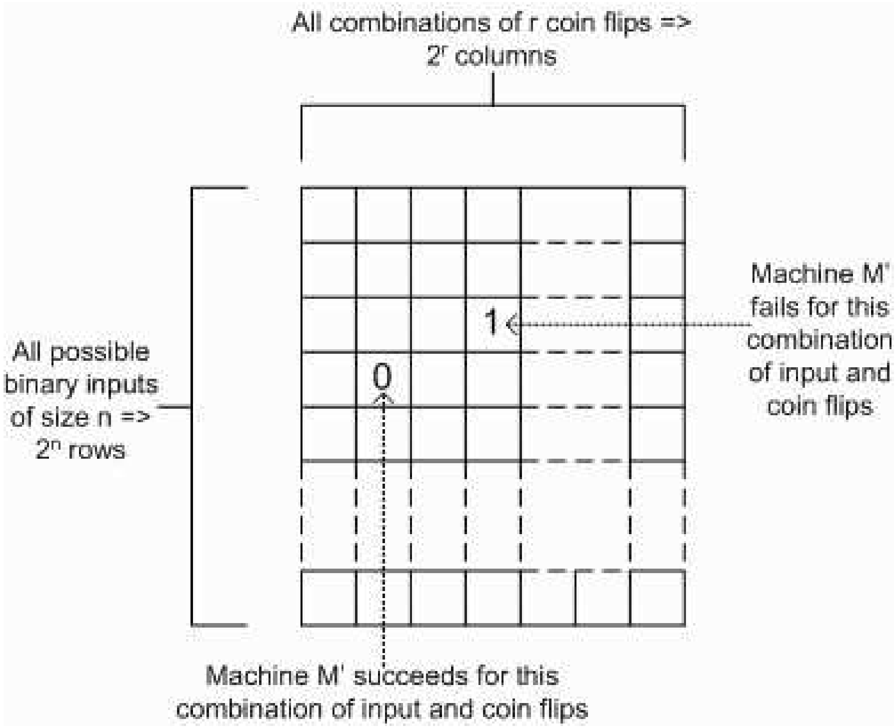

Figure 10: Input space for a BPP machine M' with r coin flips and input string of size n

Now let us construct a matrix with 2^n rows and 2^r columns, where the rows are all possible inputs and columns are all possible coin-flips sequences. This matrix is the space of all possible inputs (including coin flips) to the Machine M'. Put in a cell of the matrix a value 1 when there is an error corresponding to the input and the coin flips of the cell's position, and a value 0 when there is no error (see Figure 10). The total number of errors is given by the total number of ones in the matrix:

Total Number of Errors = (Total Number of columns * Pr that a column is 1) * (Number of rows)
Total Number of Errors = (Total Number of 1's per row) * (Number of rows)
Total Number of Errors = $(2^r \cdot (2^{-(n+1)})) \cdot 2^n = 2^{r-1}$

Since the total number of 1's is 2^{r-1} and there are 2^r columns, by the pigeonhole principle, at least 1/2 of the columns will not produce an error. Therefore, we can hardwire the P/poly circuit with some column (sequence of coin flips) where the machine does not make any mistakes. ■