Efficient Cryptographic Protocols Preventing "Man-in-the-Middle" Attacks

Jonathan Katz

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2002

ABSTRACT

Efficient Cryptographic Protocols Preventing "Man-in-the-Middle" Attacks

Jonathan Katz

In the analysis of many cryptographic protocols, it is useful to distinguish two classes of attacks: *passive attacks* in which an adversary eavesdrops on messages sent between honest users and *active attacks* (i.e., "man-in-the-middle" attacks) in which — in addition to eavesdropping — the adversary inserts, deletes, or arbitrarily modifies messages sent from one user to another. Passive attacks are well characterized (the adversary's choices are inherently limited) and techniques for achieving security against passive attacks are relatively well understood. Indeed, cryptographers have long focused on methods for countering passive eavesdropping attacks, and much work in the 1970's and 1980's has dealt with formalizing notions of security and providing provably-secure solutions for this setting. On the other hand, active attacks are not well characterized and precise modeling has been difficult. Few techniques exist for dealing with active attacks, and designing practical protocols secure against such attacks remains a challenge.

This dissertation considers active attacks in a variety of settings and provides new, provably-secure protocols preventing such attacks. Proofs of security are in the standard cryptographic model and rely on well-known cryptographic assumptions. The protocols presented here are *efficient* and *practical*, and may find application in real-world systems.

Contents

List of Figures

Acknowledgments

It has been a pleasure and an honor to work with Moti Yung and Rafail Ostrovsky these past few years. Moti was kind enough to take a chance when he agreed to supervise me three-and-a-half years ago, and I will be forever in his debt for this favor. As an advisor, he patiently and skillfully guided me, always recommending the relevant papers to read and the appropriate problems on which to focus. In addition to teaching me the proper way to conduct research and making me think (hard!) about the best ways to present my results, he has shared with me his vast experience and broad perspective of the field and has taught me the importance of avoiding politics. I am very appreciative for all he has done for me. Rafi has tremendously contributed to my development as a scientist in more ways than can be enumerated here. As a mentor, he has provided guidance, insight, and direction, has suggested challenging and fruitful research directions to pursue, and has given generous encouragement (and new ideas) whenever I got stuck. I thank him for his intellectual guidance and his contribution to my personal and intellectual growth.

I am grateful to my advisor Zvi Galil for his patient advice, unwavering support, and continual help throughout my time in the PhD program. I also appreciate very much the fact that he took me seriously when I emailed him four years ago asking about the possibility of my transferring to the department of computer science. His kind words at that time gave me the necessary encouragement to proceed.

This dissertation would not have been possible without the financial support I received from a number of sources. First and foremost, I thank the Department of Defense for a DoD NDSEG Fellowship and for graciously allowing me to continue my fellowship when I joined the department of computer science. I am also very grateful for the financial support given to me by Luca Trevisan and by the department of computer science at Columbia University. I thank Mary Van Starrex and James Cunha for their help in orchestrating the latter.

I would not have been able, from a financial standpoint, to remain in graduate school and complete my PhD had I not begun working full-time at Telcordia Technologies two years ago. My time at Telcordia was a fantastic learning experience and, indeed, much of the work included in this dissertation was done while working there. I am indebted to Jon Kettenring, Rich Graveman, and Rafi Ostrovsky for giving me the opportunity to work at Telcordia and for making my experience there so positive. I am also grateful to Jason Baron for first putting me in touch with Rich Graveman. It is my pleasure to acknowledge Giovanni Di Crescenzo, Steven Myers, "Raj" Rajagopalan, and Adam Smith with whom (in addition to Rafi) I collaborated while at Telcordia.

Thanks to Jonathan Gross and Andrew Kosoresow for serving on my thesis committee and

for helping me make this dissertation (somewhat) more accessible to a wider audience. I am also grateful to Jonathan Gross for his help during my transition to the department of computer science. I thank Phil MacKenzie and Adam Smith for careful readings of Chapters 3 and 4, respectively; their comments on content and style are much appreciated.

My time in graduate school was made much more enjoyable by those who shared the experience with me at Columbia and elsewhere. I would like to specifically mention Yael Gertner who always inspired me to press forward with my research yet kept me from taking my work too seriously, and Ted Diament with whom I had many enjoyable and intellectually stimulating discussions. Dario Catalano and Dalia Yablon also made my time at Columbia much more pleasant.

Finally, I would like to thank my wife Jill for her support, for putting up with my less-than-sunny disposition these past few months, and for continually reminding me about those things in life that are more important than this thesis.

Chapter 1

Introduction

Demonstrating the security of a cryptographic protocol is a delicate task. For one thing, the term "secure" is meaningless (in the context of cryptography) except in reference to a specific, well-defined adversarial model. This model must specify, among other things, what constitutes a breach of security, what classes of adversarial behavior are being protected against, and what initial conditions (setup assumptions) are assumed. Furthermore, the security of a protocol must be understood in the context of the assumptions under which a proof of security is given. The canonical goal of cryptography is to reduce the security of a protocol to a plausible and well-studied assumption (such as: "factoring is hard"); such a proof implies that a given protocol is secure (with respect to a given adversarial model) as long as the underlying assumption is true. Rigorous proofs of this type are clearly preferable to "heuristic" arguments in favor of the security of a protocol or proofs of security which require idealized assumptions known to be untrue.

The security of certain primitives against "passive" adversaries has become, by now, well-studied and well-understood. A salient example is encryption, for which definitions and secure constructions have been given by, among others, Goldwasser and Micali [69] and Blum and Goldwasser [21] (see [62] for more details). The security desired from an encryption scheme against a passive eavesdropper is intuitive, even if developing a formal definition is more difficult: to prevent the adversary from learning any information about the message transmitted between honest users. Furthermore, many techniques are known [69, 21] for achieving such privacy against an eavesdropper who is limited to passively monitoring the communication network. Unfortunately, adversaries (who don't often play by the rules) may not restrict themselves to being passive! In fact, it is quite reasonable to consider [99, 105, 44] an *active* adversary — a "man-in-the-middle" — who modifies the data as it is transmitted from sender to receiver. Attacks of this type raise a host of new questions; for example: What kind of security does one (optimally) want in such a setting? What does security mean in a setting in which all communication is controlled by the adversary? Finally, how can one hope to

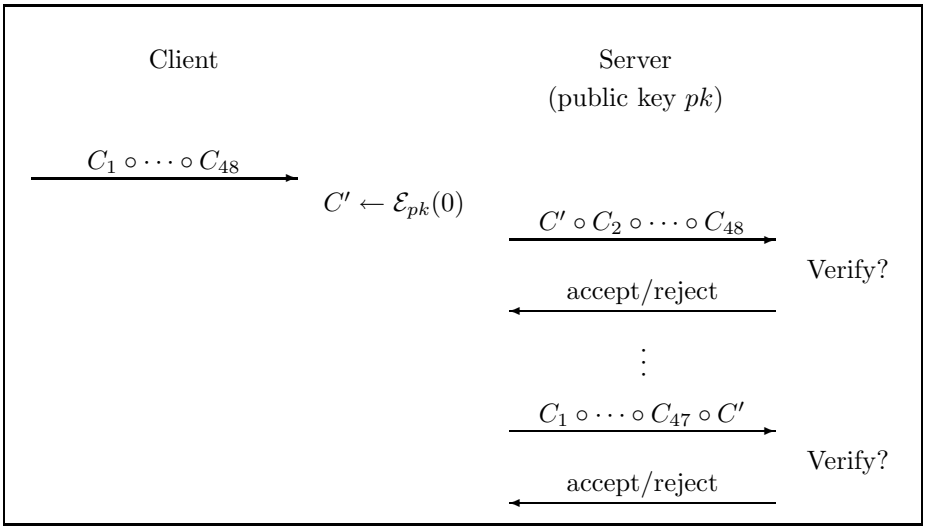

Figure 1.1: Man-in-the-middle attack on a public-key encryption scheme.

attain the desired level of security against such a powerful adversary? It is these types of questions that are addressed by this dissertation.

We provide a concrete example to motivate this line of research. Consider a scenario in which a client transmits a 48-bit credit-card number $b_1 \circ \cdots \circ b_{48}$ to a server, perhaps along with a PIN and whatever additional information is necessary to authorize a transaction. To ensure the client's privacy, all information sent from the client to the server is encrypted using the public key of the server. Assume encryption is done bit-wise; i.e., a string is encrypted by encrypting each bit of the string and then concatenating the resulting ciphertexts.

If the encryption algorithm $\mathcal{E}$ used to encrypt each bit of the string is secure against passive eavesdroppers, the encryption of any polynomial-length string (as above) is also secure against a passive eavesdropper. Unfortunately, the scheme is completely vulnerable against an active adversary as shown by the following attack (cf. Figure 1.1). The adversary, upon observing transmission of $C_1 \circ \cdots \circ C_{48}$ (where C_i represents encryption of b_i), generates ciphertext C' by encrypting 0 using the public-key of the server (note that the adversary has access to the server's public key since it is public information). The adversary then sends $C' \circ C_2 \circ \cdots \circ C_{48}$ to the server and waits to see whether the server accepts (i.e., authorizes the transaction) or not. If the server accepts, the adversary concludes that the first bit of the client's credit-card number is 0; if the server rejects, the first bit of the credit-card number must be 1. Repeating this attack for each bit of the message, the adversary obtains the client's credit-card number after sending only 48 messages to the server! This may be compared with randomly guessing the credit-card number, where the adversary needs to send (on average) 2^{47} messages to the server before learning the correct value. Clearly, man-in-

the-middle attacks such as this represent a real threat and need to be taken into account explicitly when designing real-world systems.

Encryption is typically a simple, one-round protocol in which a single message is sent between two parties. Dealing with man-in-the-middle attacks is even more challenging [44] when considering more complex protocols consisting of many rounds of interaction, possibly among more than two parties. Even the correct formalization of security against man-in-the-middle attacks in a way which prevents undesired adversarial behavior while making realistic assumptions about the adversary's capabilities is non-trivial; in fact, entirely satisfactory definitions have not yet been given for some cryptographic tasks. Protocol design has been even more difficult, with few efficient protocols known that prevent man-in-the-middle attacks in a provably-secure way.

In practice, man-in-the-middle attacks are often dealt with by designing protocols that protect against a list of known attacks; such an approach, however, leaves the protocol vulnerable to new attacks as they are developed. Furthermore, many widely deployed protocols have only heuristic arguments in favor of their security. Such an approach does not engender much confidence, and, unfortunately, many of these protocols have been broken soon after their introduction. A step in the right direction is to "validate" a protocol by proving the protocol secure in an idealized model such as the random oracle model[1] [12]. Clearly, however, proofs of security requiring no unrealistic assumptions are preferable.

1.1 Summary of Contributions

In this dissertation, we consider the security of a wide range of cryptographic primitives against man-in-the-middle attacks.[2] We present formal definitions of security against these attacks (in some cases, these are the first such definitions to appear) and give constructions of efficient protocols which are proven secure using standard cryptographic assumptions. Our contributions include the following (terms below are defined in Chapter 2 and the relevant chapter in which the term appears):

- In Chapter 3, we consider the problem of password-only authenticated key exchange over a network completely controlled by an adversary. Here, the password shared by two parties is explicitly modeled as a weak, human-memorizable secret which may be easily guessed by the adversary [11, 24]. Due to the inherent weakness of the password, a secure protocol must ensure (among other things) that an adversary can not determine the password — and hence

[1] In the random oracle model, all participants are given oracle access to a truly random function. In practice, the random oracle is instantiated with a cryptographic hash function. However, there are protocols which are secure in the random oracle model but are known to be insecure when instantiated with *any* concrete function [29].

[2] "Man-in-the-middle attack" is a broad term for any attack in which communication between honest parties may be corrupted by an adversary. In this work we include precise definitions of security against man-in-the-middle attacks for all primitives we consider.

break the protocol — using an off-line dictionary attack. We work in a setting in which a common string is known to all parties [20] and users otherwise share *only* a weak password (in particular, a public-key infrastructure is not required), and give the first efficient and provably-secure protocol for password-only authenticated key exchange in this setting. A preliminary version of this work has appeared previously [83].

- In Chapter 4, we consider the important cryptographic primitive of commitment [96]. There, we describe the first efficient and non-malleable protocols for non-interactive, perfect commitment. The security of our schemes may be based on either the RSA assumption or the discrete logarithm assumption. We also propose and analyze a construction of a non-interactive, non-malleable, standard commitment scheme which has near-optimal commitment length. A preliminary version of this work has appeared previously [41].

- We next consider the case of *interactive* proofs of knowledge. Extending previous definitions in the non-interactive setting [44, 110, 38], we formally define the notion of a non-malleable, interactive proof of plaintext knowledge (PPK). We then give efficient constructions of non-malleable PPKs (for a number of standard cryptosystems) which remain secure even when executed in a concurrent fashion. Finally, we show applications of these protocols to (1) chosen-ciphertext-secure public-key encryption [99, 105], (2) password-based authenticated key exchange in the public-key model [76, 22], (3) deniable authentication [44, 49], and (4) identification [56]. In many cases, our work provides the first efficient solutions to these problems based on factoring or other number-theoretic assumptions. These results are described in Chapter 5.

Chapter 2 contains a description of the basic adversarial model considered in this dissertation along with an overview of some previous definitional approaches to security against man-in-the-middle attacks. There, we also summarize related work in this area and provide relevant cryptographic background.

Chapter 2

The Model and Definitions

This chapter contains a description of the adversarial model[1] considered in this work, beginning with a high-level overview and followed by a more detailed and formal definition. We also review prior approaches to security against man-in-the-middle attacks and discuss relevant previous work in this area. We conclude by introducing some notation and by defining cryptographic terms used throughout the remainder of this dissertation.

2.1 The Model: Overview

Our model includes some number of honest users (also called participants or parties) who may, if and when they choose, take part in an execution of the protocol. The entire protocol is viewed as taking place in two, distinct phases. In the *initialization phase*, the set of users is established; furthermore, during this phase certain information may be generated and distributed among these users. Following this, the *protocol execution phase* begins and the desired protocol is run. It is important to note that the protocol may be executed an arbitrary (polynomial) number of times during the protocol execution phase following only a single execution of the initialization phase.

During the protocol execution phase, each participant executes a local algorithm to generate outgoing messages as specified by the protocol; these messages may depend on information received in the initialization phase and on the set of messages sent and received by the participant thus far. Delivery of these messages is, however, not guaranteed. In fact, as we will see below, all communication between participants is completely controlled by an adversary. Details follow.

[1] Although all our results may be cast in this model, we present a simplified view of the model when considering a specific cryptographic task. Furthermore, we do not state a generic definition of security in this chapter; instead, we give formal security definitions for each task we consider in the relevant chapter.

2.1.1 Initialization Phase

During the initialization phase, information is distributed among the parties using some (unspecified) mechanism which is independent of the protocol itself. Examples of information which may be distributed during this phase include:

- *Secret information* shared between two (or more) parties. Because the information is assumed to be shared secretly during the initialization phase, this data is not available to the adversary.

  We distinguish two types of secret information. A *key* always refers to a cryptographically-strong key; i.e., a string with sufficiently-large entropy so as to be resistant to guessing attacks by the adversary. On the other hand, a *password* refers to a short string which may be easily memorized by a human user (cryptographic keys, including public/secret keys, are too long to be easily memorized!). Nothing about the entropy of a password is assumed. In particular, passwords may be easily guessed by an adversary; hence, proofs of security (when passwords are used) must explicitly take this fact into account.

- *Public keys* established as part of a *public-key infrastructure* (PKI). In this case, each participant may generate a public key/secret key pair; the public key is distributed to all other parties while the secret key remains known only to the party that generated it. Since the public keys are freely distributed (with no effort made to limit their dissemination), they are also available to the adversary; of course, the corresponding secret keys are unavailable to the adversary.

- *Public information* known to all parties [20]. A secure PKI has been notoriously difficult to implement and PKIs have well-known problems associated with registration, revocation, etc. Much simpler than establishing a PKI is to fix public parameters (generated by a known, probabilistic algorithm) and distribute this information to all participants. An example of this is the *common random string* model in which all participants possess identical copies of a uniformly-distributed string. Since this information is publicly distributed, it is also assumed to be available to the adversary.

Of course, the initialization phase may consist of some combination of the above; it is also possible to consider a null initialization phase in which no information is shared in advance of protocol execution.

It is assumed that the initialization phase specified for a particular protocol is executed correctly before protocol execution begins. Such an approach allows separation of the analysis of the initialization phase from the analysis of the protocol itself. On the other hand, it is important to

recognize that this (in some sense) introduces a new assumption in the proofs of security. For this reason, protocols using weaker (i.e., easier to achieve) initialization phases are preferable to protocols requiring stronger set-up assumptions.

2.1.2 The Adversary

We consider a very powerful adversary who controls all communication between the honest participants. More precisely, we view transmission of a message msg from A to B as a two-step process in which A first sends msg to the adversary and the adversary then sends msg to B. However, the adversary may choose not to deliver messages (without notifying any participants), may deliver messages out of order, may insert his own messages, and may arbitrarily modify messages in transit. Furthermore, the adversary may read all messages sent between users. This type of adversary, though strong, is realistic in settings such as wireless communication networks in which messages may be easily intercepted and potentially changed.

A protocol defines a probabilistic algorithm for each honest participant. The adversary is aware of the algorithms defining the protocol, and can cause participants to execute the protocol by interacting with the participants in the appropriate way. A formal description of the adversary appears in Section 2.2.

This adversarial model considered here is distinct from (and incomparable with) other models which have been proposed for security in a multi-party setting (e.g., [66, 95, 61, 26]). In particular, other models typically assume authenticated channels (so that the identity of the sender of a message is unambiguous) and guaranteed delivery of messages; neither condition is assumed here. Additionally, other models typically allow some fraction of the participants to be corrupted by the adversary in which case the adversary learns all their local information including long-term secrets as well as state information used during protocol execution. Here, we generally assume that participants themselves may *not* be corrupted (although in Chapter 3 a limited form of corruption is considered).[2]

Recently, a model capturing aspects of both the previous models and the model outlined here has been proposed by Canetti [27]. Canetti has shown [27] that protocols secure in this model enjoy very strong composability properties; in particular, protocols secure in this model are secure against many types of man-in-the-middle attacks. The model is sufficiently general to allow the formulation of many cryptographic tasks; on the other hand, the resulting definitions are often complex and protocols proven secure in this model are extremely inefficient. For this reason, it is still often

[2]In other models, the corrupted players may deviate from a correct execution of the protocol. Here — since authenticated communication is not assumed — the adversary may send whatever messages he likes (claiming they were sent by one of the participants) and in this way achieve the same effect.

desirable to introduce alternate, simpler definitions for specific functionalities. We also remark that Canetti's model does not necessarily deal with all possible types of man-in-the-middle attacks (e.g., those considered in [81, 32, 44, 82]).

2.2 The Model: Details

The number of participants n is fixed during the initialization phase and is polynomial in the security parameter. We model the participants and the adversary as interactive Turing machines (our definition follows [70, 62]):

Definition 2.1 *A probabilistic, multi-tape Turing machine M is an* interactive Turing machine *(ITM) if it satisfies the following:*

- *M's tapes consist of: (1) a read-only* input tape*, (2) a read-only* private-auxiliary-input tape*, (3) a read-only* public-auxiliary-input tape*, (4) a write-only* output tape*, (5) a read-and-write* work tape*, (6) a read-only* random tape*, (7) a read-only* communication-in tape*, and (8) some number of write-only* communication-out tapes*.*
- *M is* message-driven*; in other words, M is* activated *when a new message is written on its communication-in tape and* deactivated *when it writes a special symbol $\perp$ on its communication-out tape. A period from when M is activated to when it is next deactivated is called a* period of activation*. We may analogously define a* period of deactivation*.*
- *The content written on the communication-out tape during a particular period of activation (not including the symbol $\perp$) is called the* message sent by *M during that period; the content written on the communication-in tape during a particular period of deactivation is called the* message received by *M during that period.*

To fully define our model, we describe how the participants jointly execute a protocol in the presence of an adversary. We stress that joint execution among n participants does not mean that they must all take part in every invocation of the protocol; it simply means that each participant has the option of executing the protocol with some other (subset of the) participants of its choosing.

Definition 2.2 *Let $M_1, \ldots, M_n$, and $\mathcal{A}$ be interactive Turing machines. We say that $M_1, \ldots, M_n$ are* linked via *$\mathcal{A}$ if:*

- *Each M_i has a single communication-out tape.*
- *$\mathcal{A}$ has n communication-out tapes labeled $1, \ldots, n$.*

- *The public-auxiliary-input tapes of $M_1, \ldots, M_n$, and $\mathcal{A}$ coincide.*
- *The i^{th} communication-out tape of $\mathcal{A}$ coincides with the communication-in tape of M_i. The communication-out tapes of each M_i coincide with the communication-in tape of $\mathcal{A}$.*
- *All other tapes are distinct.*

$M_1, \ldots, M_n$ are called the participants *and $\mathcal{A}$ is called the* adversary.

The system is initialized as follows: the random tapes of each ITM are chosen independently at random. The work tape, input tape, output tape, communication-in tape, and communication-out tape(s) of all ITMs are initially empty. The public-auxiliary-input tape contains 1^k so that the security parameter is well-defined for all ITMs. The public- and private-auxiliary-input tapes may contain additional information generated during the initialization phase[3] (as described in Section 2.1.1). Typically, this auxiliary information will be the output of some probabilistic initialization protocol. As an illustrative example, an ℓ-bit secret may be shared between users i and j by choosing w at random from $\{0,1\}^\ell$ and writing w on the private auxiliary-input tapes of M_i and M_j (and on no other tapes). Public-key generation by player i is achieved by running a key generation algorithm to obtain (SK, PK), having (i, PK) written on the public auxiliary-input tape, and having SK written on the private auxiliary-input tape of M_i. Finally, public information may be distributed by running some algorithm to yield σ and then writing σ on the public auxiliary-input tape. Note that $\mathcal{A}$ has access to the information written on the public auxiliary-input tape; this explicitly models the public nature of this information.

During the protocol execution phase, $\mathcal{A}$ is the first to be activated. Computation proceeds as follows: $\mathcal{A}$ begins by writing a (possibly empty) message on its i^{th} communication-out tape, for some i. When $\mathcal{A}$ is done, M_i is activated and may read the message written on its communication-in tape; M_i also writes a (possibly empty) message on its communication-out tape. We assume the intended recipient is clear from a description of the protocol; alternately, one may modify any protocol so that M_i sends message (i, j, msg) whenever it would otherwise send msg intended for player j. When M_i is done, the adversary is re-activated. Even though the adversary has only a single communication-in tape, $\mathcal{A}$ can determine the sender of any message because $\mathcal{A}$ decides the order of activation. $\mathcal{A}$ may write any message of his choice on the communication-in tape of any recipient; we stress that the adversary is not required to deliver those messages output by the honest parties. Computation proceeds in this manner until the adversary halts.

[3] In the non-uniform model, additional private auxiliary input may be given to the adversary. In this case, it is often crucial for the security of the protocol that this auxiliary information be fixed *before* the initialization phase. Alternately, this auxiliary information may arise from the *composition* of two (or more) protocols.

All participants are assumed to run in polynomial time (in the security parameter k). In particular, there exists some polynomial $p(\cdot)$ such that each M_i halts within at most $p(k)$ steps regardless of the behavior of the adversary. $\mathcal{A}$ is also assumed to run in polynomial time.

2.3 Security Against Man-in-the-Middle Attacks

The distinguishing feature of the above model is that *there is no direct communication between any of the honest parties*. Instead, all communication is "routed" through $\mathcal{A}$. Yet, not all hope is lost in the face of such a strong adversary. Consider the problem of security against passive eavesdropping.[4] If a PKI is established during the initialization phase, the desired level of security may be achieved by having A encrypt all messages (using the public key of B) before sending them to B.

It is instructive to consider other types of attacks, beyond mere eavesdropping, so as to recognize what is *not* achieved in this example. First of all, there is no guarantee that messages from A will ever be received by B. Worse, however, is the fact that B is not assured that messages he receives are actually from A, since the adversary (who has access to PK_B) can also encrypt messages and send them to B. In general, these types of attacks cannot be prevented (although there may be ways to ensure that these attacks are detected) due to the strong adversarial model we consider; for example, we do not assume authenticated channels between parties.

This discussion indicates that certain things cannot be achieved in our model. More precisely, we can not hope to prevent the adversary from:

- Preventing communication between parties of his choice.
- Attempting to impersonate one of the honest participants.
- Faithfully forwarding messages, thereby "copying" all messages sent by a particular user (we will return to this point below).

Our goal, then, is to prevent more devious attacks whose feasibility is not an immediate consequence of the model. As an example of such an attack, assume A is sending a "yes/no" response (represented by a "1/0") to B. The adversary might be able to "flip" the contents of A's message by somehow modifying the ciphertext C sent by A. Note that the adversary may (in theory) achieve this *without ever determining A's original message*, and thus the privacy of the encryption scheme is not violated. Preventing this type of attack is among the problems considered in this work.

The above discussion suggests that defining security against man-in-the-middle attacks is not simple. Indeed, many cryptographic primitives still have no universally agreed-upon definitions for

[4] In the context of our model, we may define a passive eavesdropper as an adversary who always reliably forwards messages, without any modifications, to the intended recipient.

security against man-in-the-middle attacks. Yet, it will be useful to review some definitions which have been suggested previously (in a variety of different settings) for eventual comparison with the definitions under which we prove security of our constructions.

2.3.1 Definitional Approaches

Ping-pong protocols and early approaches. Man-in-the-middle attacks on cryptographic protocols have long been recognized as a fundamental problem. Early attempts to deal with such attacks focused on the security of *ping-pong protocols* (in which the output of a party is a simple function of the current input) against adversarial man-in-the-middle behavior [46, 45, 119, 52]. Although a formal approach is taken, certain limitations of this approach are apparent. First is that the class of ping-pong protocols is very limited; in particular, it does not include protocols which maintain state during a multi-round execution. Thus, the approach does not address some very natural scenarios; e.g., a party who executes a protocol only once and then refuses to execute the protocol again. Second, the analysis of ping-pong protocols assumes that cryptographic primitives such as encryption are ideal; for example, the analysis assumes that given a ciphertext C which is an encryption of some message m, it is infeasible to come up with a different ciphertext C' which is also an encryption of m. Real-life encryption protocols, however, are *not* ideal; the resulting ciphertexts are simply bit strings which may be manipulated in a variety of ways (cf. the man-in-the-middle attack in Figure 1.1). Finally, secure ping-pong protocols have typically been designed assuming identities have been established for all participants. In a large network, however, this may not be the case.

The work of Rivest and Shamir [106] represents another early attempt to prevent man-in-the-middle attacks (in this case, for a key-exchange protocol). Although their approach is novel, they neither define nor prove security of their protocol. Analysis of their protocol shows that users need to (effectively) share a cryptographic key in advance of protocol execution. Furthermore, although the protocol achieves a "basic" level of security as outlined by the authors, the protocol is easily seen to be susceptible to more complex attacks.

Non-malleability and simulatability. The limitations of the approaches mentioned above illustrate the need for formal approaches that do not idealize cryptographic primitives and that allow for the analysis of more complex, "real-world" protocols. Dolev, Dwork, and Naor [44] were the first to present an approach that may be applied to man-in-the-middle attacks in many different contexts. Specifically, they introduce the notion of *non-malleability* and formally define non-malleability of commitment, encryption, and (interactive) zero-knowledge proofs. Their approach may be viewed as follows. Assume participants A and B, jointly executing some protocol, are linked via adversary $\mathcal{A}$. The protocol is said to be non-malleable if (informally) B's "view" in the real world — in which

B interacts with man-in-the-middle $\mathcal{A}$ — can be simulated by an efficient algorithm *that does not interact with* A. As an informal example, consider the case of public-key encryption. Assume that A encrypts a message m (chosen from some specified distribution $\mathcal{D}$) to yield a ciphertext C. Let C' be a second ciphertext generated by $\mathcal{A}$ after receiving C. Note that B can decrypt C' to obtain some message m', and this real-world experiment therefore defines some distribution $\mathcal{D}'$ over (m, m'). Very loosely speaking, a non-malleable encryption scheme has the property that there exists a simulator (who is not given any ciphertext) that can output a ciphertext C'' (with decryption m'') such that the distribution $\mathcal{D}''$ over (m, m'') (where m is chosen from $\mathcal{D}$) is indistinguishable from distribution $\mathcal{D}'$. In other words, ciphertext C is of no help to the adversary in constructing C'.

There is one additional point, however, which needs to be taken into account. $\mathcal{A}$ can always exactly copy A's messages and forward them to B; in the encryption example, the adversary can output simply $C' = C$. On the other hand, a simulator cannot do the same. Thus, the formal definition of non-malleability must rule out such behavior.

A related approach is to require that a real execution of the protocol be *simulatable* (in a way made more precise below) by a simulator who is given access to an idealized functionality performing the same task. Depending on the precise definition, simulatability may guarantee (some form of) non-malleability. As an example, this approach has been used to define the security of key-exchange protocols [4, 114, 24, 30] against man-in-the-middle attacks. Using this methodology, an ideal model is defined in which the desired task is carried out. For the case of key exchange, this idealized model might include a special, trusted party who generates session keys and delivers these keys securely to any pair of users upon request. A simulator's interaction with this ideal model is limited to a specific set of actions; for example, in the case of key exchange the simulator might be allowed to request that session keys be established between any two parties of its choosing. During a real execution of the protocol, of course, messages must be exchanged (via the adversary) between parties desiring to establish a session key. A protocol is said to be secure if these messages (that is, a real execution of the protocol) can be efficiently simulated — for an arbitrary adversary — by a simulator running in the ideal model. This implies that anything that can be done by the adversary attacking the real protocol can be done equally well by a simulator attacking the ideal protocol. Since the ideal protocol is secure against man-in-the-middle attacks by definition (assuming the ideal functionality is defined properly), this implies that the real-world protocol is secure.

Recent work by Canetti [27] presents a unified framework in which to analyze protocols, roughly along the lines sketched above (although we have omitted many important details in our discussion). One of the contributions of this framework is that protocols simulatable under the given definition are automatically secure against (certain classes of) man-in-the-middle attacks. Yet, it is difficult

(and in certain cases, impossible [28]) to design secure protocols in this model without additional assumptions. Furthermore, for specific tasks (i.e., key exchange) it is often beneficial to design a model with that task specifically in mind.

Oracle-based models. While the previously-mentioned approaches are appealing, they are often very cumbersome to work with. Furthermore, they may in certain cases be too restrictive when full simulatability is not required. This motivates other definitions which are easier to work with, and — perhaps more importantly — often yield simpler and more efficient protocols. One such approach is to allow the adversary to interact with *oracles* specified in a manner appropriate for the task at hand. For example, in the case of encryption, the adversary may be given access to a *decryption oracle* which takes as input any ciphertext C and returns the underlying plaintext. We can then define a new type of secure encryption as follows [99, 105, 6] (see also Definition 2.5): First, the adversary receives ciphertext C. Then, the adversary may interact with the decryption oracle, obtaining the plaintext corresponding to any ciphertext(s) C' of the adversary's choosing.[5] The encryption scheme is said to be "chosen-ciphertext secure" if the contents of C remain hidden from the adversary *even after interaction with the decryption oracle*. Oracle-based models have also been proposed for analyzing key exchange and mutual authentication protocols [13, 15, 11].

In some cases, an oracle-based definition of security has been proven equivalent to a simulation-based definition [114] or to non-malleability [6, 16]. In these cases, it is often significantly easier to prove security of a protocol under the oracle-based definition; definitional equivalence then implies that the protocol inherits the security properties guaranteed by the (seemingly) stronger definition.

Other approaches. Other approaches to man-in-the-middle attacks are also possible, especially when the security desired is different from the security guarantees outlined above. For example, when user identities and a PKI are assumed, we may imagine a situation in which a zero-knowledge proof should be "meaningful" only to a specific receiver [81, 32]. Here, even if the adversary copies a proof to a third party, the third party should not be convinced by the proof; thus, the definition of security must deal explicitly with copying instead of simply disallowing it. One may also consider the complementary notion in which proofs must remain uniquely identified with a particular prover [19, 44, 82]; in this case, even when an adversary copies a proof it should remain clear which party actually generated it.

[5] As before, we need to explicitly rule out copying. Thus, the adversary is not allowed to submit $C' = C$ to the decryption oracle.

2.4 Previous Work

Here, we survey some of the most important work dealing with non-malleability. More detailed discussion of previous work related to a particular application appears in the relevant chapters of this thesis.

As mentioned above, definitions for non-malleable encryption, commitment, and (interactive) zero-knowledge proofs have previously appeared [44]. Constructions of these primitives which are provably non-malleable are also known [44]. Subsequently, definitions for non-malleable, non-interactive zero-knowledge proofs (and non-interactive proofs of knowledge) appeared [110, 38], along with constructions achieving these definitions. These were also used to construct improved non-malleable encryption protocols, following the paradigm established in [99]. A revised definition of non-malleable commitment (appropriate for the case of perfect commitment) appears in [40, 58]. The first construction of a non-interactive, non-malleable commitment scheme is given in [40].

The above constructions are all based on general assumptions and are therefore highly impractical. Only one efficient and provably-secure construction of a non-malleable encryption scheme is known [36].[6] An efficient (interactive) non-malleable commitment scheme has been given [58]. In the random oracle model, many efficient chosen-ciphertext-secure encryption schemes are known; e.g., [12, 14].

Relations among definitions of security for public-key encryption (especially non-malleability vs. chosen-ciphertext security) are considered in [6, 16]. Similar relations have been established for private-key encryption [84], where the adversary's interaction with an *encryption oracle* must also be taken into account. Other areas for which a formal approach to man-in-the-middle attacks has been given include: key exchange and mutual authentication [18, 13, 15, 4, 76, 114, 22, 11, 24, 23, 92, 30, 65], deniable authentication [49, 50, 48], identification [7], and designated-verifier proofs [81, 32].

2.5 Notation and Preliminaries

2.5.1 Notation

We adopt the now-standard notation of Goldwasser, Micali, and Rackoff [70]. The set of n-bit strings is denoted by $\{0,1\}^n$, the set of all binary strings of length at most n is denoted by $\{0,1\}^{\leq n}$, and the set of all finite, binary strings is denoted by $\{0,1\}^*$. Concatenation of two strings x_1, x_2 is denoted by $x_1 \circ x_2$ or $x_1 \mid x_2$. The output y of a deterministic function f on input $x_1, \ldots, x_n$ is denoted by $y := f(x_1, \ldots, x_n)$. Similarly, if random tape r is fixed, the (deterministic) output y of probabilistic algorithm A on input $x_1, \ldots, x_n$ and random tape r is denoted by $y := A(x_1, \ldots, x_n; r)$. We let

[6] Very recently, other efficient constructions of non-malleable encryption schemes have been given [37].

$y \leftarrow A(x_1, \ldots, x_n)$ refer to the (randomized) experiment in which r is chosen uniformly at random and y is set to the output of $A(x_1, \ldots, x_n; r)$. For a finite set S, the notation $x \leftarrow S$ means that x is chosen uniformly at random from S. If $p(x_1, x_2, \ldots)$ is a predicate, the notation

$$\Pr\left[x_1 \leftarrow S; x_2 \leftarrow A(x_1, \ldots); \cdots : p(x_1, x_2, \ldots)\right]$$

denotes the probability that $p(x_1, x_2, \ldots)$ is true after ordered execution of the listed experiments. An important convention is that all appearances of a given variable in a probabilistic statement refer to the same random variable. If f is a deterministic function, the predicate $f(x_1, \ldots, x_n) = y$ is true exactly when the output of $f(x_1, \ldots, x_n)$ is equal to y; for a randomized algorithm f, we write (slightly abusing notation) $f(x_1, \ldots, x_n) = y$ to refer to the predicate $f(x_1, \ldots, x_n; r) = y$ for randomly chosen r.

A probabilistic, polynomial-time (PPT) ITM M is one for which there exists a polynomial $p(\cdot)$ such that, for all inputs $x_1, \ldots, x_n$, all random tapes r, and arbitrary behavior of other machines with which M is interacting, $M(x_1, \ldots, x_n; r)$ runs in time bounded by $p(|x_1 \circ \cdots \circ x_n|)$. An expected-polynomial-time ITM M is one for which there exists a polynomial $p(\cdot)$ such that, for all inputs $x_1, \ldots, x_n$ and regardless of the behavior of other machines with which M is interacting, the expected running time of $M(x_1, \ldots, x_n; r)$ (over choice of r) is bounded by $p(|x_1 \circ \cdots \circ x_n|)$. All algorithms we consider are (at least implicitly) given the security parameter k (in unary) as input; the lengths of other inputs are always polynomially related to k and therefore, in most cases, running times are measured as a function of k. All algorithms (including adversarial algorithms) are modeled as uniform Turing machines (although our results extend to the non-uniform case by modifying the computational assumptions appropriately).

A function $\varepsilon : \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for all $c > 0$ there exists an n_c such that, for $n > n_c$ we have $\varepsilon(n) < 1/n^c$. In other words, $\varepsilon(\cdot)$ is asymptotically bounded from above by any inverse polynomial. A function $f : \mathbb{N} \to \mathbb{R}^+$ is *non-negligible* (or *noticeable*) if there exists a $c > 0$ and an n_c such that, for $n > n_c$ we have $f(n) > 1/n^c$. Note that it is possible for a function to be neither negligible nor non-negligible. Two distributions X_k, X'_k (indexed by parameter $k \in \mathbb{N}$) are *computationally indistinguishable* if, for all PPT distinguishing algorithms D, the following is negligible (in k):

$$\left|\Pr[x \leftarrow X_k : D(x) = 1] - \Pr[x \leftarrow X'_k : D(x) = 1]\right|.$$

Two distributions X_k, X'_k are *statistically indistinguishable* if the following is negligible (in k):

$$\sum_{\alpha} \left|\Pr[x \leftarrow X_k : x = \alpha] - \Pr[x \leftarrow X'_k : x = \alpha]\right|.$$

In this case, we may also say that X_k, X'_k have *negligible statistical difference*. Note that statistical indistinguishability implies computational indistinguishability.

2.5.2 Cryptographic Assumptions

Hardness of factoring. This assumption, one of the most widely known, states that it is infeasible for any algorithm to find the factors of a random product of two primes. More formally, let $\text{2-FACTOR}_k \stackrel{\text{def}}{=} \{pq \mid p \text{ and } q \text{ are prime; } |p| = |q| = k\}$. Then, the factoring assumption states that for every PPT algorithm A, the following probability is negligible (in k):

$$\Pr[N \leftarrow \text{2-FACTOR}_k; (p, q) \leftarrow A(N) : p \cdot q = N].$$

If we restrict ourselves to N of a special form the factoring assumption needs to be appropriately modified. For example, N is a *Blum integer* if $N = pq$ with p, q prime and $p, q = 3 \bmod 4$; it is widely believed that factoring Blum integers is intractable.

A result due to Rabin [104] shows that the hardness of inverting the squaring function, defined by $f(x) = x^2 \bmod N$, is equivalent to the hardness of factoring. A similar result holds for the function $f_i(x) = x^{2^i} \bmod N$ (for fixed i). Note this implies that both of these functions can be efficiently inverted if the factorization of N is known.

The RSA assumption [107]. Informally, for a modulus $N = pq$ which is the product of two primes, a fixed e which is relatively prime to $\varphi(N)$, and a random $r \in \mathbb{Z}_N^*$, the RSA assumption states that it is infeasible to compute $r^{1/e} \bmod N$. More formally, the RSA assumption states that for all PPT algorithms A, the following probability is negligible (in k):

$$\Pr\left[N \leftarrow \text{2-FACTOR}_k; r \leftarrow \mathbb{Z}_N^*; x \leftarrow A(N, e, r) : x^e = r \bmod N\right],$$

where e is any number relatively prime to $\varphi(N)$. If the factorization of N is known, then for all e relatively prime to $\varphi(N)$ and for all $r \in \mathbb{Z}_N^*$, the value $r^{1/e}$ can be computed efficiently. Thus, the RSA assumption is at least as strong as the assumption that factoring is hard.

Discrete-logarithm-based assumptions [42]. Assume a finite, cyclic group $\mathbb{G}$ such that the order of $\mathbb{G}$ is prime (this condition is not essential for the assumptions below, yet all $\mathbb{G}$ used in this work have this property). For any elements $g, h \in \mathbb{G}$ with $g \neq 1$, the value $\log_g h$ is well-defined as the unique $a \in \mathbb{Z}_{|\mathbb{G}|}$ for which $g^a = h$. For convenience, we use groups $\mathbb{G}$ defined as follows: let $p = 2q + 1$, with p, q prime and let $\mathbb{G}$ be the (unique) subgroup of $\mathbb{Z}_p^*$ with order q.

The *discrete logarithm assumption* states that, given randomly-chosen elements g, h, computing $\log_g h$ is infeasible. More formally, let $\mathcal{G}$ be an efficient algorithm which, on input 1^k, generates p, q prime with $p = 2q + 1$ and $|q| = k$ (thereby defining group $\mathbb{G}$). The discrete logarithm assumption states that for all PPT algorithms A, the following is negligible (in k):

$$\Pr\left[(p, q) \leftarrow \mathcal{G}(1^k); g, h \leftarrow \mathbb{G}; a \leftarrow A(p, q, g, h) : g^a = h\right].$$

Related (but possibly stronger) assumptions include the *computational Diffie-Hellman (CDH)* and *decisional Diffie-Hellman (DDH)* assumptions. For $\mathbb{G}$ defined as above, the CDH assumption states that for any PPT algorithm A, the following is negligible (in k):

$$\Pr\left[(p,q) \leftarrow \mathcal{G}(1^k); g \leftarrow \mathbb{G}; x, y \leftarrow \mathbb{Z}_q : A(p,q,g,g^x,g^y) = g^{xy}\right].$$

The DDH assumption states that the following distributions are computationally indistinguishable, where the first distribution is over *random tuples* and the second is over *Diffie-Hellman tuples*:

$$\{(p,q) \leftarrow \mathcal{G}(1^k); g \leftarrow \mathbb{G}; x, y, z \leftarrow \mathbb{Z}_q : (p,q,g,g^x,g^y,g^z)\}$$
$$\{(p,q) \leftarrow \mathcal{G}(1^k); g \leftarrow \mathbb{G}; x, y \leftarrow \mathbb{Z}_q : (p,q,g,g^x,g^y,g^{xy})\}.$$

Clearly, the DDH assumption implies the CDH assumption which in turn implies the discrete logarithm assumption. It is not known whether the converses hold.

Expected-polynomial-time algorithms. Occasionally, we will need to assume that the above-mentioned problems are hard even with respect to algorithms which are permitted to run in expected polynomial time (the assumptions above are stated with respect to PPT algorithms). Although these represent stronger assumptions, they are still widely believed to hold for the problems listed above.

2.5.3 Cryptographic Tools

Cryptographic hash functions. Two distinct notions of cryptographic hash functions are primarily used.[7] The first notion is that of (families of) *collision-resistant hash functions* (CRFs). These are functions h for which it is infeasible to find distinct pre-images x, x' such that $h(x) = h(x')$ (typically, h compresses its input). Formally, let $\mathcal{H} = \{H_k\}_{k \in \mathbb{N}}$, where each H_k is a finite family of efficiently-evaluable functions such that, for each $h \in H_k$, we have $h : \{0,1\}^* \rightarrow \{0,1\}^{\leq p(k)}$ for some polynomial $p(\cdot)$. Such a collection is called *collision-resistant* if for any PPT algorithm A, the following is negligible (in k):

$$\Pr[h \leftarrow H_k; (x, x') \leftarrow A(1^k, h) : x \neq x' \wedge h(x) = h(x')].$$

Collision-resistant hash families are known to exist based on the hardness of factoring or the discrete logarithm assumption; see [109] for minimal assumptions on which CRFs may be based.

In practice, very efficient hash functions mapping $\{0,1\}^*$ to a fixed output length are used; examples include SHA-1 and MD5. Although the security of these hash functions is at best heuristic, these functions are generally considered to be collision-resistant for all practical purposes.

[7] A third approach is to model specific cryptographic hash functions as random oracles; see Chapter 1, footnote 1. As discussed there, this approach represents an idealized view of hash functions which cannot be realized in practice. Since we do not use this approach in this thesis, we omit further discussion.

The second notion is that of (families of) *universal one-way hash functions*. Here, an adversary first outputs a value x before receiving a hash function h chosen at random from the family (recall that for collision-resistant hash functions the adversary may choose *both* x and x' after receiving h). It is then infeasible for the adversary to find an $x' \neq x$ such that $h(x) = h(x')$. More formally, let $\mathcal{H} = \{H_k\}_{k \in \mathbb{N}}$, where each H_k is a finite family of efficiently-evaluable hash functions such that, for each $h \in H_k$, we have $h : \{0,1\}^* \to \{0,1\}^{\leq p(k)}$ for some polynomial $p(\cdot)$. Such a collection is called *universal one-way* if for any PPT algorithm $A = (A_1, A_2)$, the following is negligible (in k):

$$\Pr[(x, s) \leftarrow A_1(1^k); h \leftarrow H_k; x' \leftarrow A_2(1^k, s, h) : x \neq x' \wedge h(x) = h(x')].$$

Universal one-way hash functions were introduced by Naor and Yung [98], who provide a construction based on any one-way permutation. Subsequently, it was shown that one-way functions are sufficient for the construction of universal one-way hash functions [108]. Note that any collision-resistant hash family is also universal one-way; however, there is evidence that the existence of collision-resistant hash functions is a strictly stronger assumption [85].

Public- and private-key encryption. An encryption scheme allows one party to send a message to another such that the contents of the message remain hidden from anyone intercepting the communication. Though the intuition is simple, formalizing this intuition correctly requires care. The generally-accepted notion of security for encryption is *semantic security*, introduced by Goldwasser and Micali [69]; this definition states (informally) that anything which can be efficiently computed about a plaintext message when given access to the encryption of that message can be efficiently computed without access to the encryption of the message (in particular, this implies that the message itself cannot be determined without the decryption key). A second definition of security is that of *indistinguishability* [69]; here, an adversary outputs two messages x_0, x_1 and is then given an encryption of one of them (chosen at random). The adversary succeeds if he can determine which message was encrypted. An encryption scheme is indistinguishable if the success probability of any PPT adversary is negligibly close to 1/2 (the adversary can always succeed half the time by guessing randomly).

The basic definition of indistinguishability given above is equivalent to that of semantic security [69]. Under the basic definition, however, the adversary is given only the ciphertext. Subsequent work has considered stronger attacks in the public- [99, 105, 6] and symmetric-key [5, 84] settings.

We begin with a generic definition of an encryption scheme:

Definition 2.3 *An* encryption scheme Π *is a triple of algorithms* $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ *such that, for some polynomial* $p(\cdot)$*:*

- *The* key generation algorithm $\mathcal{K}$ *is a* PPT *algorithm that takes as input a security parameter* k *(in unary) and returns keys* sk *and* pk*.*
- *The* encryption algorithm $\mathcal{E}$ *is a* PPT *algorithm that takes as input* 1^k*, key* pk*, and a message* $x \in \{0,1\}^{\leq p(k)}$ *and returns a ciphertext* C *(we denote this by* $C \leftarrow \mathcal{E}_{pk}(x)$*).*
- *The* decryption algorithm $\mathcal{D}$ *is a deterministic, poly-time algorithm that takes as input* 1^k*, key* sk*, and a ciphertext* C *and returns either a message* $x \in \{0,1\}^{\leq p(k)}$ *or a special symbol* $\perp$ *to indicate that the ciphertext* C *is invalid (we denote this by* $x := \mathcal{D}_{sk}(C)$*).*

We require that for all k*, for all* (sk, pk) *which can be output by* $\mathcal{K}(1^k)$*, for all* $x \in \{0,1\}^{\leq p(k)}$*, and for all* C *which can be output by* $\mathcal{E}_{pk}(x)$*, we have* $\mathcal{D}_{sk}(C) = x$*.*

We may define a private-key (also known as symmetric-key) encryption scheme as one in which, for all (sk, pk) output by $\mathcal{K}(1^k)$, we have $pk = sk$. A public-key encryption scheme will have $pk \neq sk$; note that this is not implied by the definition above, yet will be implied by the definition of security given below.

We first present the basic notion of indistinguishability for public-key encryption [69]. Note that the adversary is explicitly given the public key at all times.

Definition 2.4 $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is a* public-key encryption scheme secure in the sense of indistinguishability *(equivalently [69],* semantically secure*) if for any* PPT *adversary* $A = (A_1, A_2)$*, the following is negligible (in* k*):*

$$\Big|\Pr[(sk, pk) \leftarrow \mathcal{K}(1^k); (m_0, m_1, s) \leftarrow A_1(1^k, pk); b \leftarrow \{0,1\};$$
$$C \leftarrow \mathcal{E}_{pk}(m_b); b' \leftarrow A_2(1^k, pk, C, s) : b' = b] - 1/2\Big|.$$

One may also consider stronger classes of adversaries which are given access to certain oracles. For an adversary attacking a public-key encryption scheme, the only oracle of interest is a decryption oracle $\mathcal{D}_{sk}(\cdot)$ which returns the decryption of any ciphertext C given to it by the adversary. Encryption schemes which remain secure even against this class of adversaries are termed *(adaptive*[8]*) chosen-ciphertext secure* [105].

Definition 2.5 $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is a* chosen-ciphertext-secure (CCA2) public-key encryption scheme *if for any* PPT *adversary* $A = (A_1, A_2)$*, the following is negligible (in* k*):*

$$\Big|\Pr[(sk, pk) \leftarrow \mathcal{K}(1^k); (m_0, m_1, s) \leftarrow A_1^{\mathcal{D}_{sk}(\cdot)}(1^k, pk); b \leftarrow \{0,1\};$$
$$C \leftarrow \mathcal{E}_{pk}(m_b); b' \leftarrow A_2^{\mathcal{D}_{sk}(\cdot)}(1^k, pk, C, s) : b' = b] - 1/2\Big|,$$

[8] A definition of *non-adaptive* chosen-ciphertext security, in which A_1 has access to the oracle but A_2 does not, is also possible. In fact, this was the first such definition presented [99]. However, since we do not explicitly use this weaker definition in this work, we omit further discussion.

where we require that A_2 not submit C to its decryption oracle.

Semantically-secure public-key encryption schemes may be based on any (family of) trapdoor functions with polynomial preimage-size [9]; in particular, public-key cryptosystems exist based on the hardness of factoring [104] and RSA [107] assumptions. Security of the El-Gamal encryption scheme [51] is based on the DDH assumption. CCA2 public-key encryption schemes may be based on trapdoor permutations [44]; an efficient construction based on the DDH assumption is also known [36].

Semantic security for private-key encryption is defined in an analogous manner. Here, one may also consider adversaries with access to an *encryption* oracle [84]; however, we will not need such a notion for the present work. For completeness, we provide the definition for (adaptive) chosen-ciphertext security here (we omit pk since, for private-key encryption, $pk = sk$).

Definition 2.6 $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *is a* chosen-ciphertext-secure private-key encryption scheme *if for any* PPT *adversary $A = (A_1, A_2)$, the following is negligible (in k):*

$$\Big|\Pr[sk \leftarrow \mathcal{K}(1^k); (m_0, m_1, s) \leftarrow A_1^{\mathcal{D}_{sk}(\cdot)}(1^k); b \leftarrow \{0,1\}; \\ C \leftarrow \mathcal{E}_{pk}(m_b); b' \leftarrow A_2^{\mathcal{D}_{sk}(\cdot)}(1^k, C, s) : b' = b] - 1/2\Big|,$$

where we require that A_2 not submit C to its decryption oracle.

One-way functions are necessary and sufficient for constructing semantically-secure [78] and chosen-ciphertext-secure [44] private-key encryption schemes.

Message authentication codes. A message authentication code (MAC) allows two parties, who have shared a secret key in advance, to authenticate their subsequent communication. More formally, a MAC is a key-based algorithm which associates a tag with every valid message. The tag for a particular message may be efficiently verified by the party sharing the key. Furthermore, an adversary who sees many message/tag pairs is unable to forge a tag on a new message. We begin with a definition of a MAC algorithm and then define an appropriate notion of security.

Definition 2.7 *A* message authentication code Π *is a triple of algorithms* $(\mathcal{K}, \text{MAC}, \mathsf{Vrfy})$ *such that, for some polynomial $p(\cdot)$:*

- *The* key generation algorithm $\mathcal{K}$ *is a* PPT *algorithm that takes as input a security parameter k (in unary) and returns key sk.*
- *The* tagging algorithm MAC *is a* PPT *algorithm that takes as input 1^k, a key sk, and a message $m \in \{0,1\}^{\leq p(k)}$ and returns a tag T (we denote this by $T \leftarrow \text{MAC}_{sk}(m)$).*

- *The* verification algorithm Vrfy *is a deterministic algorithm that takes as input* 1^k*, a key* sk*, a message* $m \in \{0,1\}^{\leq p(k)}$*, and a tag* T *and returns a single bit (we denote this by* $b := \mathsf{Vrfy}_{sk}(m,T)$*).*

We require that for all k*, all* sk *output by* $\mathcal{K}(1^k)$*, all* $m \in \{0,1\}^{\leq p(k)}$*, and all* T *output by* $\text{MAC}_{sk}(m)$*, we have* $\mathsf{Vrfy}_{sk}(m,T) = 1$*.*

A MAC is secure if an adversary is unable to forge a valid message/tag pair. Yet we need to specify the class of adversary we consider (i.e., the type of attack allowed). Following [10], we consider the strongest type of attack: the adversary may interact — adaptively and polynomially-many times — with an oracle $\text{MAC}_{sk}(\cdot)$ that returns the correct tag for any message submitted by the adversary.

Definition 2.8 *A message authentication code* $\Pi = (\mathcal{K}, \text{MAC}, \mathsf{Vrfy})$ *is* secure under adaptive chosen message attack *if for all* PPT *forging algorithms* $\mathcal{F}$*, the following probability is negligible (in* k*):*

$$\Pr[sk \leftarrow \mathcal{K}(1^k); (m,T) \leftarrow \mathcal{F}^{\text{MAC}_{sk}(\cdot)}(1^k) : \mathsf{Vrfy}_{sk}(m,T) = 1],$$

where we require that T *not be a tag previously output by* $\text{MAC}_{sk}(\cdot)$ *on input* m*.*

Since this definition is now standard, the term "secure MAC" in this work refers to a MAC that is secure under adaptive chosen message attack. A variety of MAC constructions are known, and a secure MAC may be based on any one-way function [63, 77].

Signatures. A formal definition of security for signature schemes was first given by Goldwasser, Micali, and Rivest [71]. Here, a signer publishes a (public) verification key VK and keeps secret a signing key SK. A signing algorithm, which takes as additional input a signing key SK, associates a signature with every valid message; this signature may be validated by anyone who knows the corresponding verification key. As with a secure MAC, an adversary should be unable to forge a valid signature on a previously-unsigned message. The formal definitions of a signature scheme and its security under adaptive chosen-message attack exactly parallel those given above for MACs. For completeness, we give the full definitions here (following [71]).

Definition 2.9 *A* signature scheme Π *is a triple of algorithms* $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ *such that, for some polynomial* $p(\cdot)$*:*

- *The* key generation algorithm $\mathcal{K}$ *is a* PPT *algorithm that takes as input a security parameter* k *(in unary) and returns verification key* VK *and signing key* SK*.*
- *The* signing algorithm Sign *is a* PPT *algorithm that takes as input* 1^k*, a key* SK*, and a message* $m \in \{0,1\}^{\leq p(k)}$ *and returns a signature* s *(we denote this by* $s \leftarrow \mathsf{Sign}_{SK}(m)$*).*

- *The* verification algorithm Vrfy *is a deterministic algorithm that takes as input* 1^k*, a key* VK*, a message* $m \in \{0,1\}^{\leq p(k)}$*, and a signature* s *and returns a single bit (we denote this by* $b = \mathsf{Vrfy}_{VK}(m,s)$*).*

We require that for all k*, all* (VK, SK) *output by* $\mathcal{K}(1^k)$*, all* $m \in \{0,1\}^{\leq p(k)}$*, and all* s *output by* $\mathsf{Sign}_{SK}(m)$*, we have* $\mathsf{Vrfy}_{VK}(m,s) = 1$*.*

Definition 2.10 *A signature scheme* $\Pi = (\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ *is* secure under adaptive chosen message attack *if for all* PPT *forging algorithms* $\mathcal{F}$*, the following probability is negligible (in* k*):*

$$\Pr[(VK, SK) \leftarrow \mathcal{K}(1^k); (m,s) \leftarrow \mathcal{F}^{\mathsf{Sign}_{SK}(\cdot)}(1^k, VK) : \mathsf{Vrfy}_{VK}(m,s) = 1],$$

where we require that s *not be a signature previously output by* $\mathsf{Sign}_{SK}(\cdot)$ *on input* m*.*

As with MACs, the above definition is standard, so that the term "secure signature scheme" in this work refers to security in the sense of the definition above. Secure signature schemes may be constructed from any one-way function [98, 108].

A weaker notion is that of a *one-time signature scheme* [87, 94], in which the adversary is allowed to request only one signature from the Sign oracle before attempting a forgery. Although secure signature schemes and one-time signature schemes may both be constructed from one-way functions, known constructions of one-time signature schemes are more efficient [53, 112].

Chapter 3

Password-Authenticated Key Exchange

3.1 Introduction

Protocols for mutual authentication of two parties and generation of a cryptographically-strong shared key between them (*authenticated key exchange*) underly most interactions taking place on the Internet. Indeed, it would be near-impossible to achieve any level of security over an unauthenticated network without mutual authentication and key-exchange protocols. The former are necessary because you always need to know "with whom you are communicating"; the latter are required because cryptographic techniques (such as encryption, message authentication, etc.) are useless without a shared cryptographically-strong key which must be periodically refreshed (e.g., for each new session). Furthermore, high-level protocols are frequently developed and analyzed using the assumption of "authenticated channels" (see [4] for discussion); yet, this assumption cannot be realized without a secure mechanism for implementing such channels using previously-shared information.

We focus here on *password-only* protocols in which the information previously-shared between two parties consists only of a short, easily-memorized password.[1] We additionally assume some public information known to all participants including the adversary attacking the protocol; this public information may be established by some trusted party or may be generated in some alternate secure way (e.g., flipping coins publicly). In the password-only setting, it is important to design protocols which explicitly prevent *off-line dictionary attacks* in which an adversary enumerates all possible passwords, one-by-one, in an attempt to determine the correct password based on previously-recorded transcripts. Consideration of such attacks is crucial if security is to be guaranteed even when users of the protocol choose passwords "poorly" (say, from a dictionary of English words).

[1] See Chapter 2 for the distinction between "password" and "key".

One might argue that users should be forced to choose high-entropy passwords which cannot be easily guessed; indeed, the security of an authentication protocol can only improve as the entropy of the shared secret increases. This recommendation misses the point. Although good password selection should be encouraged, it remains true that, in practice, user-selected passwords are weak [118]. This is to be expected since high-entropy passwords are difficult (if not impossible) to remember. It is preferable to recognize this and design protocols which remain secure despite this limitation.

Previous work on *password-based* protocols [76, 22] (where, in addition to a shared password, a public-key infrastructure (PKI) is assumed), are a step in the right direction since they recognize the weakness of passwords selected in practice. However, password-only protocols (even when public information is assumed) have many practical advantages over password-based protocols. For one, the password-only model eliminates the need for *secure* implementation of a PKI, thereby avoiding the need to deal with issues like user registration, key management, or key revocation (although these concerns are somewhat mitigated when only servers need certified public keys). Furthermore, avoiding the use of a PKI means that an on-line, trusted certification authority (CA) is not needed throughout the lifetime of the protocol; note that the need to access a CA in the public-key setting is often a performance bottleneck as the number of users becomes large. In the password-only model, once public information is established new users may join the network at any time and do not need to inform anyone else of their presence. Finally, in the password-only model no participants need to know any "secret key" associated with the public parameters. This eliminates the risk that compromise of a participant will compromise the security of the entire system.

3.1.1 Previous Work

The importance of key exchange as a cryptographic primitive has been recognized since the influential paper of Diffie and Hellman [42]. Soon after, the importance of *authenticated* key exchange (and mutual authentication) became apparent. Many protocols for these tasks were proposed (see [23] for an exhaustive bibliography), followed by increased realization that precise definitions and formalizations were necessary. The first formal treatments [18, 43, 13, 15, 86, 4, 114] were in a model in which participants had established cryptographically-strong information in advance of protocol execution: either a shared key [18, 13, 15, 4, 114] which is used for authentication of messages, or a public key [4, 114] which is used for encryption or digital signatures. Under these strong setup assumptions, secure protocols for the two-party [18, 13, 4, 114] and two-party assisted [15] case were designed and proven secure.

The setting arising most often in practice — in which human users generate and share only weak passwords — has only recently received formal treatment. It is important to note that the known

protocols which guarantee security when users share keys are demonstrably insecure when users share passwords. For example, a challenge-response protocol in which the client sends a nonce r and the server replies with $x = f_K(r)$ (where $\{f_s\}_{s\in\{0,1\}^k}$ is a family of pseudorandom functions and K is a shared key) prevents a passive eavesdropper from determining K *only when the entropy of* K *is sufficiently large*. When K has low entropy, an eavesdropper who monitors a single conversation (r, x) can determine (with high probability) the value of K, off-line, by trying all possibilities until a value K' is found such that $f_{K'}(r) = x$. This example clearly indicates the need for new protocols in the password-only setting.

In the public-key setting (where, as mentioned above, in addition to sharing a password the client requires the public key of the server), Lomas et. al [88] were the first to present password-based authentication protocols resistant to off-line dictionary attacks; these protocols were subsequently improved [72]. However, formal definitions and proofs of security are not given. Formal definitions and provably-secure protocols for the public-key setting were given by Halevi and Krawczyk [76], and extensions of these definitions and protocols to the multi-user setting have also appeared [22].

A protocol for password-only (i.e., where no PKI is assumed) authentication and key exchange was first introduced by Bellovin and Merritt [17], and many additional protocols have subsequently been proposed [73, 115, 79, 80, 89, 117]. These protocols have only informal arguments for their security; in fact, some of these protocols were later broken [102] indicating the need for proofs of security in a well-defined model. Formal models of security for the password-only setting were given independently by Bellare, Pointcheval, and Rogaway [11] (building on [13, 15, 89]) and Boyko, MacKenzie, Patel, and Swaminathan [92, 24, 23] (building on [4, 114]); these works also give protocols for password-only key exchange which are provably-secure in the ideal cipher and random oracle models, respectively. A different model of security for the password-only setting was introduced by Goldreich and Lindell [65]; they also present a provably-secure protocol under standard assumptions (i.e., without random oracles or ideal ciphers). Subsequent to the work described in this chapter, other protocols with provable security in the random oracle model have been demonstrated [90, 91].

3.1.2 Our Contribution

Proofs of security in idealized models (random oracle/ideal cipher) do not necessarily translate to real-world security [29]. In fact, protocols are known which may be proven secure in an idealized model yet are demonstrably insecure when given any concrete implementation in the standard model [29]. This illustrates the importance of proofs of security in the standard model, using well-studied cryptographic assumptions.

The existence of a secure protocol for password-only key exchange in the standard model [65]

is remarkable since it was not *a priori* clear whether a solution was achievable. In contrast to the present work, the protocol of Goldreich and Lindell [65] does *not* require public parameters. On the other hand (unlike the protocol presented here) their solution does not allow concurrent executions of the protocol between parties using the same password. Most importantly, their protocol is not at all efficient. The proposed scheme requires techniques from generic multi-party computation (making it computationally inefficient) and concurrent zero-knowledge (making the round-complexity prohibitive); thus, their protocol may be viewed as a plausibility result that does not settle the important question of whether a practical solution is possible. We note that efficiency is especially important in the password-only setting since security concerns are motivated by practical considerations (i.e., human users' inability to remember long keys).

Here, we present a protocol which is provably secure in the standard model under the decisional Diffie-Hellman assumption, a well-studied cryptographic assumption [42] used in constructing previous password-only schemes [11, 24]. The construction is secure under both the notion of "basic security" and the stronger notion of "forward security" (see Section 3.2.1 for definitions). The protocol is remarkably efficient even when compared to the original key-exchange protocol of Diffie and Hellman [42] which provides no authentication at all. Only three rounds of communication are needed, and the protocol requires computation only (roughly) 4 times greater than the aforementioned schemes [42, 11, 24].

Although our solution relies on public-key techniques (in fact, this is necessary [76]), our protocol does *not* use the public-key model. In particular, we do not require *any* participant to have a public key but instead rely on one set of common parameters shared by everyone in the system. From a practical point of view, the requirement of public parameters is not a severe limitation. Previous password-only protocols [11, 24] require public parameters and the existence of such parameters seems to be implicitly assumed in most previous work.[2] Furthermore, the public parameters can be hard-coded into any implementation of the protocol. We note, however, that it would be preferable to rely on public parameters into which no "secret information" can be embedded (e.g., a single generator g). This makes the problem of generating the public information much easier when no trusted parties are assumed.

3.2 Definitions and Preliminaries

We begin with an informal description of the adversarial model, followed by a more formal treatment in Section 3.2.1. Two parties within a larger network who share a weak (low-entropy) password wish

[2]For example, Diffie and Hellman [42] implicitly assume that both parties know a group $\mathbb{G}$ and generator g to use in the protocol. Although this can be avoided (these may be included in the first message), the public nature of these parameters has generally been assumed in subsequent work.

to authenticate each other and generate a strong session key to secure their future communication. An adversary controls all communication in the network. The adversary may view, tamper with, deliver out-of-order, or refuse to deliver messages sent by the honest parties. The adversary may also initiate concurrent (arbitrarily-interleaved) executions of the protocol between the honest parties; during these executions, the adversary may attempt to impersonate one (or both) of the parties or may simply eavesdrop on an honest execution of the protocol. Finally, the adversary may corrupt the honest parties (in a way we describe below) to expose previous session keys or even the long-term shared password. The adversary succeeds (informally) if he can distinguish an actual session key generated by an honest party from a randomly-chosen session key.

A notion of security in this setting must be carefully defined. Indeed, since passwords are chosen from a small space, an adversary can always try each possibility one at a time in an impersonation (on-line) attack. We say a password-only protocol is secure (informally) if on-line guessing is the best an adversary can do. On-line attacks are the hardest to mount, and they are also the easiest to detect. Furthermore, on-line attacks may be limited by, for example, shutting down a user's account after three failed authentication attempts. It is therefore very realistic to assume that the number of on-line attacks an adversary is allowed is severely limited, while other attacks (eavesdropping, off-line password guessing) are not.

3.2.1 The Model

Our model is essentially identical to that proposed by Bellare, Pointcheval, and Rogaway [11] with a few small differences. A formal description of the model follows.

Participants, passwords, and initialization. We have a fixed set of protocol participants (also called principals or users) each of which is either a client $C \in \mathsf{Client}$ or a server $S \in \mathsf{Server}$, where Client and Server are disjoint. We let $\mathsf{User} \stackrel{\text{def}}{=} \mathsf{Client} \cup \mathsf{Server}$. Each $U \in \mathsf{User}$ may be viewed as a string (of length polynomial in the security parameter) identifying that user.

Each $C \in \mathsf{Client}$ has a password pw_C. Each $S \in \mathsf{Server}$ has a vector $PW_S = \langle pw_{S,C} \rangle_{C \in \mathsf{Client}}$ which contains the passwords of each of the clients (we assume that all clients share passwords with all servers). Recall that pw_C is what client C remembers for future authentication; therefore, it is assumed to be chosen from a relatively small space of possible passwords.

Before the protocol is run, an initialization phase occurs during which public parameters are established and passwords pw_C are chosen for each client. We assume that passwords for each client are chosen independently and uniformly[3] at random from the set $\{1, \ldots, N\}$, where N is a constant which is fixed independently of the security parameter. The correct passwords are stored at each

[3]Our analysis extends easily to handle arbitrary distributions, including users with inter-dependent passwords.

server so that $pw_{S,C} = pw_C$ for all $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$.

It is possible for additional information to be generated during this initialization phase. For example, in the public-key model [76, 22] public/secret key pairs are generated for each server, with the secret key given as private input to the appropriate server and the public key provided as public input for all participants. Here, we use the weaker requirement of a set of parameters provided as public input to all participants. See Chapter 1 for further discussion of these different models.

Execution of the protocol. In the real world, a protocol determines how principals behave in response to signals (input) from their environment. In the model, these signals are sent by the adversary. Each principal is able to execute the protocol multiple times with different partners; this is modeled by allowing each principal to have an unlimited number of *instances* with which to execute the protocol [15]. We denote instance i of user U as Π_U^i. A given instance may be used only once. Each instance Π_U^i has associated with it various variables:

- state_U^i denotes the state of the instance including any information necessary for execution of the protocol. During the initialization stage, this variable is set to NULL for all instances.
- term_U^i is a boolean variable denoting whether a given instance has terminated (i.e., is done sending and receiving messages); during the initialization phase, this variable is set to FALSE for all instances.
- acc_U^i is a boolean variable denoting whether a given instance has accepted, where acceptance is defined by the protocol specification. When an instance accepts, the values of sid_U^i, pid_U^i, and sk_U^i (see below) are non-null. As an example, acceptance might indicate that instance Π_U^i is convinced of the identity of the user with whom it was interacting. During the initialization phase, this variable is set to FALSE for all instances.
- used_U^i is a boolean variable denoting whether a given instance has begun executing the protocol; this variable is used to ensure that a given instance is used only once. During the initialization phase, this variable is set to FALSE for all instances.
- sid_U^i, pid_U^i, and sk_U^i are variables containing the *session id*, *partner id*, and *session key* for an instance, respectively. Computation of the session key is the goal of the protocol, and this key will be used to secure future communication between the interacting parties. The precise function of sid_U^i and pid_U^i will be explained in more detail below. During the initialization phase, these variables are set to NULL for all instances.

The adversary is assumed to have complete control over all communication in the network. The adversary's interaction with the principals (more specifically, with the various instances) is modeled

via access to *oracles* which we describe in detail below. Local state (in particular, values for the variables described above) is maintained for each instance with which the adversary interacts; this state is not directly visible to the adversary. The state for an instance may be updated during an oracle call, and the oracle's output may depend upon this state. The oracle types are:

- $\mathsf{Send}(U, i, M)$ — This sends message M to instance Π_U^i. The oracle runs this instance according to the protocol specification, maintaining state as appropriate. The output of Π_U^i (i.e., the message sent by the instance) is given to the adversary, and in addition the adversary receives the updated values of sid_U^i, pid_U^i, acc_U^i, and term_U^i.

- $\mathsf{Execute}(C, i, S, j)$ — If Π_C^i and Π_S^j have not yet been used (where $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$), this oracle executes the protocol between these instances and outputs the transcript of this execution. This oracle call represents occasions when the adversary passively eavesdrops on a protocol execution. In addition to the transcript, the adversary receives the values of sid, pid, acc, and term, for both instances, at each step of protocol execution.

- $\mathsf{Reveal}(U, i)$ — This outputs the current value of session key sk_U^i. This oracle call models possible leakage of session keys due to, for example, improper erasure of session keys after use, compromise of a host computer, or cryptanalysis.

- $\mathsf{Corrupt}(C, (pw, S))$ — This oracle call outputs pw_C (where $C \in \mathsf{Client}$) and, if $pw \neq \perp$, sets $pw_{S,C} = pw$. This represents possible password exposures and also gives the adversary the ability to install bogus passwords of his choice on the various servers.

- $\mathsf{Test}(U, i)$ — This query is allowed only once, at any time during the adversary's execution. A random bit b is generated; if $b = 1$ the adversary is given sk_U^i, and if $b = 0$ the adversary is given a random session key. This oracle call does not correspond to any real-world event, but will allow us to define a notion of security.

Specification of the Corrupt oracle, above, corresponds to the "weak-corruption" case [11]; in the "strong-corruption" case the adversary also obtains the state information for all instances associated with C.

Correctness. Any key-exchange protocol must satisfy the following notion of correctness: Let $C \in \mathsf{Client}$ and $S \in \mathsf{Server}$. If two instances Π_C^i and Π_S^j satisfy $\mathsf{sid}_C^i = \mathsf{sid}_S^j$ and $\mathsf{acc}_C^i = \mathsf{acc}_S^j = \text{TRUE}$, and furthermore it was the case that $pw_C = pw_{S,C}$ throughout the time these instances were active, then it must be the case that $\mathsf{sk}_C^i = \mathsf{sk}_S^j$.

Partnering. We say that two instances Π_U^i and $\Pi_{U'}^j$ are *partnered* if: (1) $U \in \mathsf{Client}$ and $U' \in \mathsf{Server}$, or $U \in \mathsf{Server}$ and $U' \in \mathsf{Client}$; (2) $\mathsf{sid}_U^i = \mathsf{sid}_{U'}^j \neq \text{NULL}$; (3) $\mathsf{pid}_U^i = U'$ and $\mathsf{pid}_{U'}^j = U$; and (4) $\mathsf{sk}_U^i = \mathsf{sk}_{U'}^j$. The notion of partnering will be fundamental in defining the notion of security.

Advantage of the adversary. Informally, the adversary succeeds if it can guess the bit b used by the Test oracle. Before formally defining the adversary's success, we define a notion of *freshness*. The adversary can succeed only if the Test query was made for an instance which is fresh at the end of the adversary's execution. This is necessary for any reasonable definition of security; if the adversary's behavior were unrestricted the adversary could always succeed by, for example, submitting a Test query for an instance for which it had already submitted a Reveal query.

Two notions of freshness may be defined, one for the "basic" case and one for the "forward-secure" case. An instance Π_U^i is *fresh* (in the basic case) unless one of the following is true:

- At some point, the adversary queried $\mathsf{Reveal}(U, i)$.
- At some point, the adversary queried $\mathsf{Reveal}(U', j)$ where $\Pi_{U'}^j$ and Π_U^i are partnered.
- At some point, the adversary queried the $\mathsf{Corrupt}$ oracle.

For the case of forward security, an instance Π_U^i is *fs-fresh* unless one of the following is true:

- At some point, the adversary queried $\mathsf{Reveal}(U, i)$.
- At some point, the adversary queried $\mathsf{Reveal}(U', j)$ where $\Pi_{U'}^j$ and Π_U^i are partnered.
- The adversary makes the Test query after a $\mathsf{Corrupt}$ query and at some point the adversary queried $\mathsf{Send}(U, i, M)$ for some M.

In the basic case, adversary $\mathcal{A}$ *succeeds* if it makes a single query $\mathsf{Test}(U, i)$ to the Test oracle, where acc_U^i and term_U^i are true[4] at the time of this query and Π_U^i is fresh, outputs a single bit b', and $b' = b$ (where b is the bit chosen by the Test oracle). We denote this event by Succ. For the forward-secure case, $\mathcal{A}$ succeeds if it makes a single query $\mathsf{Test}(U, i)$ to the Test oracle, where acc_U^i was true at the time of this query and Π_U^i is fs-fresh, outputs a single bit b', and $b' = b$. We denote this event by fsSucc. The advantage of adversary $\mathcal{A}$ in attacking protocol P in the basic sense is defined by:

$$\mathsf{Adv}_{\mathcal{A},P}(k) \stackrel{\text{def}}{=} 2 \cdot \Pr[\mathsf{Succ}] - 1.$$

A similar definition may be given for $\mathsf{fsAdv}_{\mathcal{A},P}(k)$ in the forward secure case. Note that probabilities of success are functions of the security parameter k, and are taken over the random coins used by the

[4]For the protocol presented here, $\mathsf{acc}_U^i = \text{TRUE}$ automatically implies that $\mathsf{term}_U^i = \text{TRUE}$.

adversary, random coins used during the initialization phase, and random coins used by the various oracles during the actual experiment.

We have not yet defined what we mean by a secure protocol. Note that a PPT adversary can always succeed by trying all passwords one-by-one in an on-line impersonation attack (recall that the number of possible passwords is constant). Informally, we say a protocol is secure if this is the best an adversary can do. More formally, an instance Π_U^i represents an *on-line attack*[5] if both the following are true:

- At some point, the adversary queried $\mathsf{Send}(U, i, M)$ for some M.
- At some point, the adversary queried $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$.

In particular, instances with which the adversary interacts via Execute calls are not counted as on-line attacks. The number of on-line attacks represents a bound on the number of passwords the adversary could have tested in an on-line fashion. This motivates the following definition (with a similar definition for the case of forward security):

Definition 3.1 *Protocol P is a* secure password-only key-exchange protocol *(in the basic sense) if, for all N and for all* PPT *adversaries $\mathcal{A}$ making at most $Q(k)$ on-line attacks, there exists a negligible function $\varepsilon(\cdot)$ such that:*

$$\mathsf{Adv}_{\mathcal{A},P}(k) \leq Q(k)/N + \varepsilon(k).$$

In particular, this indicates that the adversary can (essentially) do no better than guess a single password during each on-line attempt. Calls to the Execute oracle, which are not included in the count $Q(k)$, are of no help to the adversary in breaking the security of the protocol; this means that passive eavesdropping and off-line dictionary attacks are of (essentially) no use.

Some previous definitions of security for password-only key-exchange protocols [11, 65] consider protocols secure as long as the adversary can do no better than guess a *constant* number of passwords in each on-line attempt. We believe the strengthening given by Definition 3.1 (in which the adversary can guess only a *single* password per on-line attempt) is an important one. The space of possible passwords is small to begin with, so any degradation in security should be avoided if possible. This is not to say that protocols which do not meet this definition of security should never be used; however, before using such a protocol, one should be aware of the constant implicit in the proof of security.

An examination of the security proofs for some protocols [11, 24, 92] shows that these protocols achieve the stronger level of security given by Definition 3.1. However, security proofs for other

[5]The definition of an on-line attack given here is valid only for key-exchange protocols without explicit authentication.

protocols [65, 91] are inconclusive, and leave open the possibility that more than one password can be guessed by the adversary in a single on-line attack. In at least one case [117], an explicit attack is known which allows an adversary to guess two passwords per on-line attack.

3.2.2 Protocol Components

We discuss some cryptographic tools used to construct our protocol.

Basic primitives. The security of our protocol relies on the DDH assumption [42], introduced in Section 2.5.2. We also use universal one-way hash families [98] and one-time digital signature schemes, as discussed in Section 2.5.3. It should be noted, however, that these may both be constructed from any one-way function [108]; in particular, the DDH assumption (which implies that group exponentiation is one-way) implies the existence of universal one-way hash families and one-time signature schemes.

Non-malleable commitment. Our protocol requires a particular non-malleable commitment scheme that we construct based on the Cramer-Shoup [36] cryptosystem, whose security against chosen-ciphertext attacks (cf. Definition 2.5) relies on the DDH assumption. It is interesting that chosen-ciphertext-secure public-key encryption has been used previously to construct authentication and key exchange protocols [4, 76, 22, 114]. We stress that our protocol differs from these works in that we use the scheme for *commitment* only and not for encryption. No party publishes their own public key and no one need hold any secret key; in fact, "decryption" is never performed during execution of the protocol.

As we point out in Chapter 4, any CCA2 public-key encryption scheme immediately yields a non-malleable commitment scheme when public parameters are available to all parties. We may therefore describe our modified scheme as a public-key encryption scheme since this form is simplest for the proof of security; we emphasize that it is used only as a commitment scheme in the actual key-exchange protocol. Key generation proceeds by running $\mathcal{G}(1^k)$ to yield primes p, q defining group $\mathbb{G}$ in which the DDH assumption is assumed to hold. Random values $g_1, g_2 \in \mathbb{G}$ are selected, along with random $z, x_1, x_2, y_1, y_2 \in \mathbb{Z}_q$. Additionally, sets Client and Server of polynomial size (in k) are fixed; these sets contain strings which will be necessary for the key-exchange protocol but whose exact structure is unimportant here. Finally, a random hash function H is chosen from a family of universal hash functions H_k. The public key is $pk = (\mathbb{G}, g_1, g_2, h = g_1^z, c = g_1^{x_1} g_2^{x_2}, d = g_1^{y_1} g_2^{y_2}, H, \mathsf{Client}, \mathsf{Server})$.

Ciphertexts are of the form $\langle A|B|C|D|E|F\rangle$, where $C, D, E, F \in \mathbb{G}$ and the purpose of A, B will be described below. The actions of the decryption oracle $\mathcal{D}_{sk}(\cdot)$ are as in the original scheme [36]:

first, $\alpha = H(A, B, C, D, E)$ is computed, and the following condition is checked:

$$C^{x_1+y_1\alpha} D^{x_2+y_2\alpha} \stackrel{?}{=} F.$$

If this fails, the output is $\perp$. (We also output $\perp$ if any of C, D, E, F are not in $\mathbb{G}$; note that this may be efficiently verified.) Otherwise, output the plaintext E/C^z.

The central modification we introduce is in the definition of encryption, and more precisely in the definition of the encryption oracle to which the adversary will have access. Besides sending messages $m_0, m_1 \in \mathbb{G}$ to the encryption oracle, the adversary also includes a bit t specifying an *encryption-type* which is either *client-encryption* or *server-encryption*. Depending on the encryption-type selected, the adversary also submits some additional information which is used in the encryption. For a client-encryption, the adversary includes a value $Client \in \mathsf{Client}$; for a server-encryption, the adversary includes values $Server \in \mathsf{Server}$ and $\alpha \in \mathbb{Z}_q$. The encryption oracle chooses a random bit b and encrypts message m_b according to the requested encryption-type. The encryption oracle outputs the resulting ciphertext along with some additional information. Formally, the encryption oracle is defined as follows:

$\mathcal{O}_{1^k,pk,b}(m_0, m_1, t, \mathsf{input})$
  if $t = 0$ and $\mathsf{input} \in \mathsf{Client}$ then
    Client-encryption$(1^k, pk, m_b, \mathsf{input})$
  if $t = 1$ and $\mathsf{input} \in \mathsf{Server} \times \mathbb{Z}_q$ then
    Server-encryption$(1^k, pk, m_b, \mathsf{input})$

where Client-encryption and Server-encryption are defined by:

Client-encryption$(1^k, pk, m, Client)$	Server-encryption$(1^k, pk, m, (Server, \alpha))$
$(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathcal{K}(1^k)$	$x, y, z, w, r \leftarrow \mathbb{Z}_q$
$A := Client;\ B := \mathsf{VK}$	$A := Server;\ B := g_1^x g_2^y h^z (cd^\alpha)^w$
$r \leftarrow \mathbb{Z}_q$	$C := g_1^r;\ D := g_2^r;\ E = h^r m$
$C := g_1^r;\ D := g_2^r;\ E := h^r m$	$\beta := H(A\|B\|C\|D\|E)$
$\alpha := H(A\|B\|C\|D\|E)$	$F := (cd^\beta)^r$
$F := (cd^\alpha)^r$	$\mathsf{return}(\langle A\|B\|C\|D\|E\|F\rangle, (x, y, z, w))$
$\mathsf{return}(\langle A\|B\|C\|D\|E\|F\rangle, \mathsf{SK})$	

(here, $\mathcal{K}$ is a key-generation algorithm for a secure one-time signature scheme).

Let Gen denote the key-generation algorithm for this scheme. For any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, define the adversary's advantage $\mathsf{Adv}^{\mathsf{nm}}_{\mathcal{A}}(k)$ in guessing the bit b used by the encryption oracle as

$$\mathsf{Adv}^{\mathsf{nm}}_{\mathcal{A}}(k) \stackrel{\text{def}}{=} \Big|\Pr[pk \leftarrow \mathsf{Gen}(1^k); b \leftarrow \{0,1\}; (m_0, m_1, t, \mathsf{input}, s) \leftarrow \mathcal{A}_1^{\mathcal{D}_{sk}(\cdot)}(pk); (C, \mathsf{info}) \leftarrow \mathcal{O}_{1^k,pk,b}(m_0, m_1, t, \mathsf{input}) : \mathcal{A}_2^{\mathcal{D}_{sk}(\cdot)}(C, \mathsf{info}, s) = b] - 1/2\Big|,$$

where $\mathcal{A}_2$ may not submit C to the decryption oracle.

Lemma 3.1 *Under the DDH assumption,* $\mathsf{Adv}^{\mathsf{nm}}_{\mathcal{A}}(k)$ *is negligible for any* PPT *adversary* $\mathcal{A}$.

Sketch of Proof The proof of security exactly follows that given by Cramer and Shoup [36], and it can be verified easily that the additional information info given to the adversary does not improve her advantage. One point requiring careful consideration is the adversary's probability of finding a collision for the hash function H included in the public key. If H is chosen from a collision-resistant hash family (a stronger assumption than being universal one-way), there is nothing left to prove. If H is universal one-way, the value t and an appropriate value of *Client* or *Server* can be guessed by the encryption oracle in advance of the adversary's query; an appropriate value for B, based on the guess for t, can also be generated in advance (in one case by running the key-generation algorithm $\mathcal{K}(1^k)$ for the one-time signature scheme and in the other case by choosing random $x \in \mathbb{Z}_q$ and setting $B = g_1^x$). In particular, these values may all be determined before the encryption oracle is given the function H. Although the encryption oracle cannot guess the value α in advance, this will not present a problem since the encryption oracle can provide a random representation of B with respect to $(g_1, g_2, h, (cd^\alpha))$ for any value α output by the adversary as long as the encryption oracle knows the discrete logarithms of g_2, h, c, d with respect to g_1. This can be ensured during key generation.

The encryption oracle finds a collision for H if (1) the oracle correctly guesses the values t and *Client*/*Server* (depending on t); and (2) the adversary finds a collision for H. The probability of guessing the adversary's choices correctly is at least $\frac{1}{2 \cdot \max\{|\mathsf{Client}|, |\mathsf{Server}|\}}$; since $|\mathsf{Client}|$ and $|\mathsf{Server}|$ are polynomial in k, this probability is an inverse polynomial in k. Thus, if the adversary's advantage in finding a collision in the experiment is non-negligible, this implies a non-negligible advantage in finding a collision in H. ■

Using a standard hybrid argument, Lemma 3.1 implies that $\mathsf{Adv}^{\mathsf{nm}}_{\mathcal{A}}(k)$ is negligible for any PPT adversary $\mathcal{A}$ that queries the encryption oracle *polynomially-many* times (with arbitrary values of $m_0, m_1, t, \mathsf{input}$), as long as the adversary may not submit any of the returned ciphertexts to the decryption oracle.

Since we use the scheme for commitment within our protocol, we refer to *Client-commitment* and *Server-commitment* of a message m. The mechanism for this is exactly as sketched above (and decommitment is not needed in the present context).

3.3 Protocol Details

A high-level description of the protocol is given in Figure 3.1, and a more detailed description follows here. A formal specification of the protocol appears in Section 3.4.

During the initialization phase, public information is established. Given a security parameter k, primes p, q are chosen such that $|q| = k$ and $p = 2q + 1$ using algorithm $\mathcal{G}$; these values define a

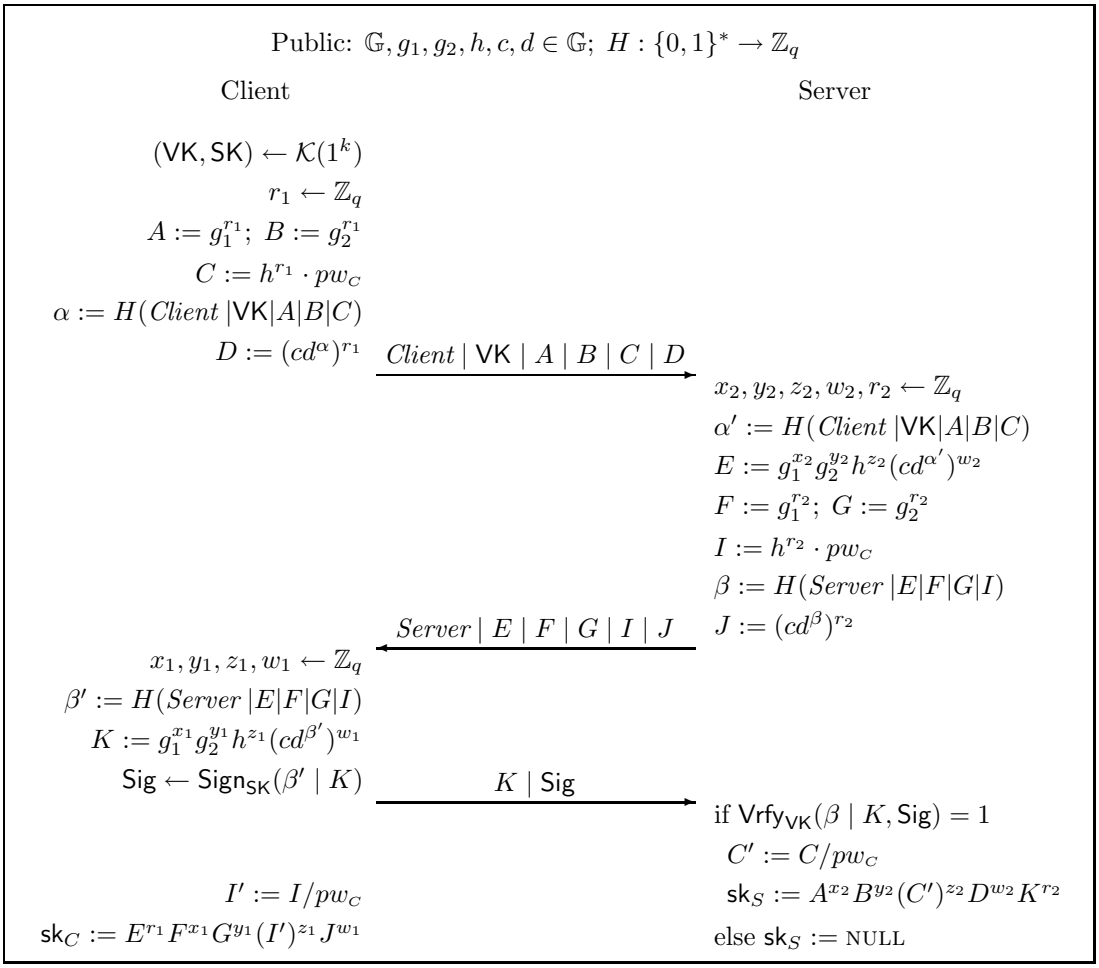

Figure 3.1: A password-only key-exchange protocol.

group $\mathbb{G}$ as discussed previously. Values $g_1, g_2, h, c, d \in \mathbb{G}$ are selected at random, and additionally a function $H : \{0,1\}^* \to \mathbb{Z}_q$ is chosen at random from a universal one-way hash family. The public information consists of a description of $\mathbb{G}$ (typically, p and q), the values g_1, g_2, h, c, d, and (a description of) the hash function H. We do not require that any party know any secret information associated with these values.

As part of the initialization phase, passwords are chosen randomly for each client and stored with each server (see Section 3.2.1). We assume that all passwords lie in (or can be mapped in a one-to-one fashion to) $\mathbb{G}$. As an example, if passwords lie in the range $\{1, \ldots, N\}$ (as we assume here), password pw can be mapped to $g_1^{pw} \in \mathbb{G}$; this will be a one-to-one mapping for reasonable values of N and $|q|$ whenever g_1 is a generator (which occurs with all but negligible probability).

When a client $Client \in \mathsf{Client}$ wants to connect to a server $Server \in \mathsf{Server}$, the client computes a random Client-commitment (see Section 3.2.2) of pw_C, where $pw_C \in \mathbb{G}$ is the client's password.

In more detail, the client begins by running the key-generation algorithm for the one-time signature scheme, giving VK and SK. The client chooses random $r_1 \in \mathbb{Z}_q$ and computes $A = g_1^{r_1}$, $B = g_2^{r_1}$, and $C = h^{r_1} \cdot pw_C$. The client then computes $\alpha = H(\mathit{Client}|\mathsf{VK}|A|B|C)$ and sets $D = (cd^{\alpha})^{r_1}$. The resulting commitment (which includes the client's name) is sent to the server as the first message of the protocol.

Upon receiving the first message, the server computes a Server-commitment to $pw_{S,C}$, where $pw_{S,C}$ is the password stored at the server corresponding to the client named in the incoming message. In more detail, let the received message be $\langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$. The server first chooses random $x_2, y_2, z_2, w_2 \in \mathbb{Z}_q$, computes $\alpha' = H(\mathit{Client}|\mathsf{VK}|A|B|C)$, and sets $E = g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha'})^{w_2}$. Additionally, a random $r_2 \in \mathbb{Z}_q$ is chosen and the server computes $F = g_1^{r_2}$, $G = g_2^{r_2}$, and $I = h^{r_2} \cdot pw_C$. The server then computes $\beta = H(\mathit{Server}|E|F|G|I)$ and sets $J = (cd^{\beta})^{r_2}$. The resulting commitment is sent to the client as the second message of the protocol.

Upon receiving the second message $\langle \mathit{Server}|E|F|G|I|J\rangle$, the client chooses random $x_1, y_1, z_1, w_1 \in \mathbb{Z}_q$, computes $\beta' = H(\mathit{Server}|E|F|G|I)$, and sets $K = g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta'})^{w_1}$. The client then signs $\beta'|K$ using SK. The value K and the resulting signature are sent as the final message of the protocol. At this point, the client accepts and determines the session key by first computing $I' = I/pw_C$ and then setting $\mathsf{sk}_C = E^{r_1} F^{x_1} G^{y_1} (I')^{z_1} J^{w_1}$.

Upon receiving the final message $K|\mathsf{Sig}$, the server checks that Sig is a valid signature of $\beta|K$ under VK (where β is the value previously used by the server). If so, the server accepts and determines the session key by first computing $C' = C/pw_{S,C}$ and then setting $\mathsf{sk}_S = A^{x_2} B^{y_2} (C')^{z_2} D^{w_2} K^{r_2}$. If the received signature is not valid, the server does not accept and the session key is set to NULL.

Although omitted in the above description, we assume that the client and server always check that incoming messages are well-formed. In particular, when the server receives the first message, it verifies that $\mathit{Client} \in \mathsf{Client}$, that VK is a valid public key for the one-time signature scheme being used, and that $A, B, C, D \in \mathbb{G}$ (note that membership in $\mathbb{G}$ can be efficiently verified). When the client receives the second message, it verifies that $\mathit{Server} \in \mathsf{Server}$, that *Server* is indeed the name of the server to whom the client desired to connect, and that $E, F, G, I, J \in \mathbb{G}$. Finally, when the server receives the last message, it verifies correctness of the signature (as discussed above) and that $K \in \mathbb{G}$. If an ill-formed message is ever received, the receiving party terminates immediately without accepting and the session key remains NULL.

Correctness. In an honest execution of the protocol (when all messages are received correctly), the client and the server calculate identical session keys. To see this, first note that $pw_C = pw_{S,C}$, $\alpha = \alpha'$, and $\beta = \beta'$ in an honest execution. We thus have

$$E^{r_1} \quad = \quad (g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2})^{r_1}$$

$$\begin{aligned} &= (g_1^{r_1})^{x_2}(g_2^{r_1})^{y_2}(h^{r_1})^{z_2}((cd^{\alpha})^{r_1})^{w_2} \\ &= A^{x_2}B^{y_2}(C')^{z_2}D^{w_2} \end{aligned}$$

and

$$\begin{aligned} K^{r_2} &= (g_1^{x_1}g_2^{y_1}h^{z_1}(cd^{\beta})^{w_1})^{r_2} \\ &= (g_1^{r_2})^{x_1}(g_2^{r_2})^{y_1}(h^{r_2})^{z_1}((cd^{\beta})^{r_2})^{w_1} \\ &= F^{x_1}G^{y_1}(I')^{z_1}J^{w_1}; \end{aligned}$$

therefore:

$$E^{r_1}(F^{x_1}G^{y_1}(I')^{z_1}J^{w_1}) = (A^{x_2}B^{y_2}(C')^{z_2}D^{w_2})K^{r_2}$$

and the session keys are equal.

Efficiency considerations. In practice, a collision resistant hash function like SHA-1 may be used instead of a universal one-way hash function. Even from a theoretical point of view, the DDH assumption implies the existence of collision resistant hash functions [109], so a stronger assumption is not necessarily required. One-time signatures are much more efficient than known signature schemes secure against adaptive (polynomially-many) chosen message attacks [53], and may be based on any presumed one-way function like SHA-1 or a block cipher. Efficiency can be improved by using an on-line/off-line signature scheme [53, 112], where the off-line computation is done while the client is waiting for the server to respond. Clearly, the values K and β which are signed could be hashed before signature computation.

If the key-generation phase of the signature scheme is computationally prohibitive (for example, when the client algorithm is run on a smart card), key generation may be done once per client at the outset of the protocol and the same values of VK and SK may be used throughout the lifetime of the client. In fact, key generation need not be done by the client/smart card itself; instead, these keys may be generated on and downloaded from a "master" computer. In this case, the signature scheme must be secure even after polynomially-many messages have been signed; this tradeoff might be acceptable for certain applications.

When passwords pw typed by users lie between 1 and N, they must be converted (as discussed above) to a group element $pw_C = g_1^{pw}$ via exponentiation. The server, however, may store $pw_{S,C} = pw_C$ directly and avoid performing the exponentiation each time. Furthermore, although the number of exponentiations in Figure 3.1 seems high, algorithms for simultaneous multiple exponentiation [93, Chapter 14] may be used to vastly speed up the computation. If this is done, the computation for each user is (roughly) equivalent to 7 exponentiations in $\mathbb{G}$; this may be compared to the 2 exponentiations required (per user) in previous key-exchange protocols [42, 24, 11].

Mutual authentication. The protocol of Figure 3.1 does not achieve mutual authentication; however, this may be accomplished with one additional message using standard techniques (see, e.g., [13, 86, 4, 76, 11]). In particular, let $\{f_s\}_{s\in\{0,1\}^k}$ be a pseudorandom function family with output length k. After successful completion of the key-exchange protocol, mutual authentication can be achieved by having the client send $V_0 = f_{\mathsf{sk}_C}(0)$ to the server (this may be included with the final message) and having the server (after verifying V_0) reply with $V_1 = f_{\mathsf{sk}_S}(1)$. Parties do not accept until receiving (and verifying) the appropriate message. The session key used for the parties' future communication is then $\mathsf{sk}'_C = \mathsf{sk}'_S = f_{\mathsf{sk}_C}(2)$.

3.4 Proofs of Security

We first provide a formal specification of the initialization process and the oracles to which the adversary has access. During the initialization phase (cf. Figure 3.2), algorithm Initialize is run to generate public parameters $\mathbb{G}, g_1, g_2, h, c, d, H$ as discussed previously. Furthermore, the sets Client and Server, whose sizes are polynomial k, are determined using some (arbitrary) algorithm UserGen.[6] Passwords for each client are chosen at random from the set $\{1, \ldots, N\}$ (for some constant N) and are then mapped to $\mathbb{G}$ as discussed earlier; the passwords for each client are stored at each server. We require $N < q - 1$; note that for typical values of q this is not a severe limitation. We stress that the assumption of a uniform and independent distribution on the passwords is not necessary; the proof of security given below can be easily modified to accommodate arbitrary distributions.

A formal specification of the Execute, Reveal, and Test oracles appears in Figure 3.3. The description of the Execute oracle matches the high-level protocol description of Figure 3.1, but additional details (for example, the updating of the global state information) are included. We let status^i_U denote the vector of values $\langle \mathsf{sid}^i_U, \mathsf{pid}^i_U, \mathsf{acc}^i_U, \mathsf{term}^i_U \rangle$ associated with instance Π^i_U. A formal specification of the Send oracle appears in Figure 3.4. Although the model technically has only one type of Send oracle (see Section 3.2.1), the adversary's queries to this oracle can easily be replaced with queries to four different oracles $\mathsf{Send}_0, \ldots, \mathsf{Send}_3$ representing the four different types of messages which may be sent as part of the protocol (this includes the three message types shown in Figure 3.1 as well as an "initiate" message). The third argument to each oracle is denoted by *msg-in*. In the figure, all variables are local to each oracle query and are erased following the oracle query unless explicitly stored in a global variable. Following a Send oracle query, the adversary receives both an outgoing message as well as the current status of the instance (i.e., the values of sid, pid, acc, and term).

[6] One could augment the model to allow the adversary to dynamically add new users during the course of protocol execution via queries to an additional oracle; the protocol would remain secure in this case, and a proof of security is substantially similar to that given below.

$\mathsf{Initialize}(1^k)$ —
  $(p, g) \leftarrow \mathcal{G}(1^k)$
  $g_1, g_2, h, c, d \leftarrow \mathbb{G}$
  $H \leftarrow \mathsf{UOWH}(1^k)$
  $(\mathsf{Client}, \mathsf{Server}) \leftarrow \mathsf{UserGen}(1^k)$
  for each $C \in \mathsf{Client}$
    $pw'_C \leftarrow \{1, \ldots, N\}$
    $pw_C := g_1^{pw'_C}$
    for each $S \in \mathsf{Server}$
      $pw_{S,C} := pw_C$
  return $\mathsf{Client}, \mathsf{Server}, \mathbb{G}, g_1, g_2, h, c, d, H$

Figure 3.2: Specification of protocol initialization.

$\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$ —
  if $\mathit{Client} \notin \mathsf{Client}$ or $\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^i_{\mathit{Client}}$ or $\mathsf{used}^j_{\mathit{Server}}$
    return $\perp$
  $\mathsf{used}^i_{\mathit{Client}} := \text{TRUE}$; $\mathsf{used}^j_{\mathit{Server}} := \text{TRUE}$
  $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathcal{K}(1^k)$
  $x_1, x_2, y_1, y_2, z_1, z_2, w_1, w_2, r_1, r_2 \leftarrow \mathbb{Z}_q$
  $A := g_1^{r_1}$; $B := g_2^{r_1}$; $C := h^{r_1} \cdot pw_{\mathit{Client}}$; $\alpha := H(\mathit{Client}|\mathsf{VK}|A|B|C)$
  $D := (cd^\alpha)^{r_1}$; $\mathit{msg\text{-}out}_1 := \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$
  $\mathsf{status}^i_{\mathit{Client},1} := \langle \text{NULL}, \mathit{Server}, \text{FALSE}, \text{FALSE}\rangle$

  $F := g_1^{r_2}$; $G := g_2^{r_2}$; $I := h^{r_2} \cdot pw_{\mathit{Server},\mathit{Client}}$; $E := g_1^{x_2} g_2^{y_2} h^{z_2} (cd^\alpha)^{w_2}$
  $\beta := H(\mathit{Server}|E|F|G|I)$; $J := (cd^\beta)^{r_2}$
  $\mathit{msg\text{-}out}_2 := \langle \mathit{Server}|E|F|G|I|J\rangle$
  $\mathsf{status}^j_{\mathit{Server},1} := \langle \text{NULL}, \mathit{Client}, \text{FALSE}, \text{FALSE}\rangle$

  $K := g_1^{x_1} g_2^{y_1} h^{z_1} (cd^\beta)^{w_1}$; $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\beta|K)$; $\mathit{msg\text{-}out}_3 := \langle K|\mathsf{Sig}\rangle$

  $\mathsf{sid}^i_{\mathit{Client}} := \mathsf{sid}^j_{\mathit{Server}} := \langle \mathit{msg\text{-}out}_1|\mathit{msg\text{-}out}_2|\mathit{msg\text{-}out}_3\rangle$
  $\mathsf{pid}^i_{\mathit{Client}} := \mathit{Server}$; $\mathsf{pid}^j_{\mathit{Server}} := \mathit{Client}$
  $\mathsf{acc}^i_{\mathit{Client}} := \mathsf{term}^i_{\mathit{Client}} := \mathsf{acc}^j_{\mathit{Server}} := \mathsf{term}^j_{\mathit{Server}} := \text{TRUE}$
  $\mathsf{sk}^i_{\mathit{Client}} := E^{r_1} F^{x_1} G^{y_1} (I/pw_{\mathit{Client}})^{z_1} J^{w_1}$
  $\mathsf{sk}^j_{\mathit{Server}} := A^{x_2} B^{y_2} (C/pw_{\mathit{Server},\mathit{Client}})^{z_2} D^{w_2} K^{r_2}$
  return $\mathit{msg\text{-}out}_1, \mathit{msg\text{-}out}_2, \mathit{msg\text{-}out}_3,$
    $\mathsf{status}^i_{\mathit{Client},1}, \mathsf{status}^j_{\mathit{Server},1}, \mathsf{status}^i_{\mathit{Client}}, \mathsf{status}^j_{\mathit{Server}}$

$\mathsf{Reveal}(U, i)$ —
  return sk^i_U

$\mathsf{Test}(U, i)$ —
  $b \leftarrow \{0, 1\}$; $\mathsf{sk}' \leftarrow \mathbb{G}$
  if $b = 0$ return sk' else return sk^i_U

Figure 3.3: Specification of the Execute, Reveal, and Test oracles.

$\mathsf{Send}_0(\mathit{Client}, i, \mathit{Server})$ —
  if $\mathit{Client} \notin \mathsf{Client}$ or $\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^i_{\mathit{Client}}$
    return $\perp$
  $\mathsf{used}^i_{\mathit{Client}} := \text{TRUE}$; $(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathcal{K}(1^k)$; $r \leftarrow \mathbb{Z}_q$
  $A := g_1^r$; $B := g_2^r$; $C := h^r \cdot pw_{\mathit{Client}}$; $\alpha := H(\mathit{Client}|\mathsf{VK}|A|B|C)$
  $D := (cd^\alpha)^r$; $\mathit{msg\text{-}out} := \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$
  $\mathsf{status}^i_{\mathit{Client}} := \langle \text{NULL}, \mathit{Server}, \text{FALSE}, \text{FALSE}\rangle$
  $\mathsf{state}^i_{\mathit{Client}} := \langle \mathsf{SK}, r, \mathit{msg\text{-}out}\rangle$
  return $\mathit{msg\text{-}out}, \mathsf{status}^i_{\mathit{Client}}$

$\mathsf{Send}_1(\mathit{Server}, j, \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle)$ —
  if $\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^j_{\mathit{Server}}$
    return $\perp$
  $\mathsf{used}^j_{\mathit{Server}} := \text{TRUE}$
  if $A, B, C, D \notin \mathbb{G}$ or $\mathit{Client} \notin \mathsf{Client}$
    $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$; return $\mathsf{status}^j_{\mathit{Server}}$
  $x, y, z, w, r \leftarrow \mathbb{Z}_q$; $\alpha := H(\mathit{Client}|\mathsf{VK}|A|B|C)$
  $F := g_1^r$; $G := g_2^r$; $I := h^r \cdot pw_{\mathit{Server},\mathit{Client}}$; $E := g_1^x g_2^y h^z (cd^\alpha)^w$
  $\beta := H(\mathit{Server}|E|F|G|I)$; $J := (cd^\beta)^r$; $\mathit{msg\text{-}out} := \langle \mathit{Server}|E|F|G|I|J\rangle$
  $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \mathit{Client}, \text{FALSE}, \text{FALSE}\rangle$
  $\mathsf{state}^j_{\mathit{Server}} := \langle \mathit{msg\text{-}in}, x, y, z, w, r, \beta, \mathit{msg\text{-}out}, pw_{\mathit{Server},\mathit{Client}}\rangle$
  return $\mathit{msg\text{-}out}, \mathsf{status}^j_{\mathit{Server}}$

$\mathsf{Send}_2(\mathit{Client}, i, \langle \mathit{Server}|E|F|G|I|J\rangle)$ —
  if $\mathit{Client} \notin \mathsf{Client}$ or not $\mathsf{used}^i_{\mathit{Client}}$ or $\mathsf{term}^i_{\mathit{Client}}$
    return $\perp$
  if $\mathit{Server} \neq \mathsf{pid}^i_{\mathit{Client}}$ or $E, F, G, I, J \notin \mathbb{G}$
    $\mathsf{status}^i_{\mathit{Client}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$; return $\mathsf{status}^i_{\mathit{Client}}$
  $\langle \mathsf{SK}, r, \mathit{first\text{-}msg\text{-}out}\rangle := \mathsf{state}^i_{\mathit{Client}}$
  $x, y, z, w \leftarrow \mathbb{Z}_q$; $\beta := H(\mathit{Server}|E|F|G|I)$
  $K := g_1^x g_2^y h^z (cd^\beta)^w$; $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\beta|K)$; $\mathit{msg\text{-}out} := \langle K|\mathsf{Sig}\rangle$
  $\mathsf{sid}^i_{\mathit{Client}} := \langle \mathit{first\text{-}msg\text{-}out}|\mathit{msg\text{-}in}|\mathit{msg\text{-}out}\rangle$
  $\mathsf{acc}^i_{\mathit{Client}} := \mathsf{term}^i_{\mathit{Client}} := \text{TRUE}$
  $\mathsf{sk}^i_{\mathit{Client}} := E^r F^x G^y (I/pw_{\mathit{Client}})^z J^w$
  return $\mathit{msg\text{-}out}, \mathsf{status}^i_{\mathit{Client}}$

$\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$ —
  if $\mathit{Server} \notin \mathsf{Server}$ or not $\mathsf{used}^j_{\mathit{Server}}$ or $\mathsf{term}^j_{\mathit{Server}}$
    return $\perp$
  $\langle \mathit{first\text{-}msg\text{-}in}, x, y, z, w, r, \beta, \mathit{first\text{-}msg\text{-}out}, pw\rangle := \mathsf{state}^j_{\mathit{Server}}$
  $\langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle := \mathit{first\text{-}msg\text{-}in}$
  if $K \notin \mathbb{G}$ or $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) \neq 1$
    $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$; return $\mathsf{status}^j_{\mathit{Server}}$
  $\mathsf{sid}^j_{\mathit{Server}} := \langle \mathit{first\text{-}msg\text{-}in}|\mathit{first\text{-}msg\text{-}out}|\mathit{msg\text{-}in}\rangle$
  $\mathsf{acc}^j_{\mathit{Server}} := \mathsf{term}^j_{\mathit{Server}} := \text{TRUE}$
  $\mathsf{sk}^j_{\mathit{Server}} := A^x B^y (C/pw)^z D^w K^r$
  return $\mathsf{status}^j_{\mathit{Server}}$

Figure 3.4: Specification of the Send oracle.

Theorem 3.1 *Assuming (1) the hardness of the DDH problem for groups $\mathbb{G}$ defined by the output of $\mathcal{G}$; (2) the security of $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ as a one-time signature scheme; and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 3.1 is a secure, password-only, key-exchange protocol.*

Proof We refer to the first assumption of the theorem as "the DDH assumption". A formal specification of the protocol appears in Figures 3.2–3.4. Note that in the basic case dealt with by this Theorem (where no Corrupt queries are allowed), we always have $pw_{Server,Client} = pw_{Client}$ for all $Client \in \mathsf{Client}$ and $Server \in \mathsf{Server}$; furthermore, during execution of the Send_3 oracle it is always the case that $pw = pw_{Client}$. Given an adversary A, we imagine a simulator that runs the protocol for A. More precisely, the simulator begins by running algorithm $\mathsf{Initialize}(1^k)$ (which includes choosing passwords for clients) and giving the public output of the algorithm to A. When A queries an oracle, the simulator responds by executing the appropriate algorithm as in Figures 3.3 and 3.4; we may further assume that the simulator records all state information defined during the course of the experiment. In particular, when the adversary queries the Test oracle, the simulator chooses (and records) the random bit b. When the adversary completes its execution and outputs a bit b', the simulator can tell whether the adversary succeeds by checking whether (1) a single Test query was made on instance Π_U^i; (2) acc_U^i was true at the time of the Test query; (3) instance Π_U^i is fresh; and (4) $b' = b$. Success of the adversary is denoted by event Succ.

For any experiment P we define $\mathsf{Adv}_{A,P}(k) \stackrel{\text{def}}{=} 2 \cdot \Pr_{A,P}[\mathsf{Succ}] - 1$, where $\Pr_{A,P}[\cdot]$ denotes the probability of an event when the simulator interacts with the adversary in accordance with experiment P. We refer to the real execution of the experiment, as described above, as P_0. We will introduce a sequence of transformations to the original experiment and bound the effect of each transformation on the adversary's advantage. We then bound the adversary's advantage in the final experiment; this immediately yields a bound on the adversary's advantage in the original experiment.

In experiment P_0', the simulator interacts with the adversary as before except that the adversary does *not* succeed when any of the following occur:

1. Any of g_1, g_2, h, c, d are not generators of $\mathbb{G}$ (i.e., they are equal to 1).
2. At any point during the experiment, a verification key VK used by the simulator in responding to a Send_0 query is repeated.
3. At any point during the experiment, the adversary forges a new, valid message/signature pair for any verification key used by the simulator in responding to a Send_0 query.
4. At any point during the experiment, a value β used by the simulator in responding to Send_1 queries is repeated.

5. At any point during the experiment, a value β used by the simulator in responding to a Send_1 query (with *msg-out* $= \langle Server|E|F|G|I|J\rangle$) is equal to a value β used by the simulator in responding to a Send_2 query (with *msg-in* $= \langle Server'|E'|F'|G'|I'|J'\rangle$) and furthermore $\langle Server|E|F|G|I\rangle \neq \langle Server'|E'|F'|G'|I'\rangle$.

Since the adversary, by definition, cannot succeed once any of these events occur, we assume that the simulator immediately halts execution when any of these events occur.

Claim 3.1 *Assuming the security of* $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme and the security of* UOWH *as a universal one-way hash family,* $\mathsf{Adv}_{A,P_0}(k) \leq \mathsf{Adv}_{A,P_0'}(k) + \varepsilon(k)$ *for some negligible function* $\varepsilon(\cdot)$.

Let Bad denote the event that case 1 or 2 occurs, let Forge denote the event that case 3 occurs, and let $\mathsf{Collision}$ denote the event that case 4 or 5 occurs. Then

$$\begin{aligned}
\Pr_{A,P_0}[\mathsf{Succ}] &\leq \Pr_{A,P_0}[\mathsf{Succ} \wedge \overline{\mathsf{Bad}} \wedge \overline{\mathsf{Forge}} \wedge \overline{\mathsf{Collision}}] + \Pr_{A,P_0}[\mathsf{Bad}] + \Pr_{A,P_0}[\mathsf{Forge}] \\
&\quad + \Pr_{A,P_0}[\mathsf{Collision}] \\
&\leq \Pr_{A,P_0'}[\mathsf{Succ}] + \Pr_{A,P_0}[\mathsf{Bad}] + \Pr_{A,P_0}[\mathsf{Forge}] + \Pr_{A,P_0}[\mathsf{Collision}].
\end{aligned}$$

It is clear that $\Pr_{A,P_0}[\mathsf{Bad}]$ is negligible. Also, $\Pr_{A,P_0}[\mathsf{Forge}]$ is negligible assuming the security of the one-time signature scheme (details omitted). Finally, $\Pr_{A,P_0}[\mathsf{Collision}]$ is negligible assuming the security of the universal one-way hash family. To see this, first note that the probability that β repeats because the values E, F, G, I repeat during two different queries to the Send_1 oracle is negligible. So, assume that β repeats because $H(Server|E|F|G|I) = H(Server'|E'|F'|G'|I')$ but $\langle Server|E|F|G|I\rangle \neq \langle Server'|E'|F'|G'|I'\rangle$, where $\langle Server|E|F|G|I|J\rangle$ (for some J) represents a *msg-out* for a Send_1 query. In other words, a collision has been found in H. The simulator can "guess" which query to the Send_1 oracle will result in a collision, "guess" appropriate values for *Server* and *Client*, choose $r \in \mathbb{Z}_q$ to use for this oracle query, and choose $E \in \mathbb{G}$ to use for this oracle query; furthermore, this may all be done at the outset of the experiment, before the simulator has chosen hash function H. If the simulator knows the discrete logarithms of g_2, h, c, d with respect to g_1 (this can be ensured during the initialization phase), the simulator can provide a random representation x, y, z, w of E with respect to $g_1, g_2, h, (cd^{\alpha})$ for any value α defined by the adversary's query to the Send_1 oracle. Hence, the adversary's view during this experiment is exactly the same as in P_0. With inverse polynomial probability, the simulator correctly guesses both the query at which a collision occurs and the values of *Server* and *Client* used by the adversary; this follows since A makes only polynomially-many queries to the Send_1 oracle and since $|\mathsf{Server}|$ and $|\mathsf{Client}|$ are polynomial in k. Therefore, if $\Pr_{A,P_0}[\mathsf{Collision}]$ were not negligible, the simulator's probability of finding a collision in H would not be negligible, contradicting the security of the universal one-way hash family. □

In experiment P_1, the simulator interacts with the adversary as in P_0' except that the adversary's queries to the Execute oracle are handled differently. In particular, for each Execute query the values C and I are chosen independently at random from $\mathbb{G}$. Furthermore, the session keys are computed as

$$\mathsf{sk}^i_{Client} := \mathsf{sk}^j_{Server} := A^{x_2} B^{y_2} (C/pw_{Client})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{Client})^{z_1} J^{w_1}. \tag{3.1}$$

The following bounds the effect this transformation can have on the adversary's advantage.

Claim 3.2 *Under the DDH assumption,* $|\mathsf{Adv}_{A,P_0'}(k) - \mathsf{Adv}_{A,P_1}(k)| \leq \varepsilon(k)$ *for some negligible function* $\varepsilon(\cdot)$.

We show how the simulator can use A as a subroutine in order to distinguish Diffie-Hellman (DH) tuples from random tuples. The simulator is given primes p, q (which define a group $\mathbb{G}$) generated using $\mathcal{G}(1^k)$ and is additionally given a tuple (g, h, s, t) chosen at random from either from the set of DH tuples or from the set of random tuples. The simulator begins by running the following modified initialization protocol:

$\mathsf{Initialize}'(1^k, \mathbb{G}, g, h, s, t)$ —
  $\gamma, \delta, \zeta \leftarrow \mathbb{Z}_q$
  $g_1 := g$; $g_2 := g^\gamma$; $c := g^\delta$; $d := g^\zeta$
  $H \leftarrow \mathsf{UOWH}(1^k)$
  $(\mathsf{Client}, \mathsf{Server}) \leftarrow \mathsf{UserGen}(1^k)$
  for each $C \in \mathsf{Client}$
    $pw'_C \leftarrow \{1, \ldots, N\}$
    $pw_C := g_1^{pw'_C}$
    for each $S \in \mathsf{Server}$
      $pw_{S,C} := pw_C$
  return $\mathsf{Client}, \mathsf{Server}, \mathbb{G}, g_1, g_2, h, c, d, H$

The simulator responds to Execute queries as shown in Figure 3.5. The simulator responds to Send, Reveal, and Test queries as in experiments P_0', P_1. Recall that pw_{Client} is always equal to $pw_{Server,Client}$ since no Corrupt oracle queries are allowed in the basic case. This also implies that sk^i_{Client} and sk^j_{Server} are always equal in P_0'. When A terminates, the simulator outputs 1 if and only if A succeeds (using the definition of success for experiments P_0' and P_1).

The remainder of the proof relies on a random self-reducibility property of the DDH problem that has been observed and used previously [114, 3]. When the tuple (g, h, s, t) is a DH tuple, the distribution on the view of A throughout this experiment is equivalent to the distribution on the view of A during experiment P_0'. The public output of the modified initialization protocol is identically distributed to the public output of the initialization protocol for experiment P_0'. As for queries to the Execute oracle, if (g, h, s, t) is a DH tuple we may write $h = g^a$, $s = g^b$ and $t = g^{ab}$ for some $a, b \in \mathbb{Z}_q$. In a particular query to the Execute oracle, one can check that $A = g_1^{br_1 + r'_1}$, $B = g_2^{br_1 + r'_1}$,

$\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$ —
if $\mathit{Client} \notin \mathsf{Client}$ or $\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^i_{\mathit{Client}}$ or $\mathsf{used}^j_{\mathit{Server}}$
return $\perp$
$\mathsf{used}^i_{\mathit{Client}} := \text{TRUE}$; $\mathsf{used}^j_{\mathit{Server}} := \text{TRUE}$
$(\mathsf{VK}, \mathsf{SK}) \leftarrow \mathcal{K}(1^k)$
$x_1, x_2, y_1, y_2, z_1, z_2, w_1, w_2, r_1, r'_1, r_2, r'_2 \leftarrow \mathbb{Z}_q$
$A := s^{r_1} g_1^{r'_1}$; $B := A^{\gamma}$; $C := t^{r_1} h^{r'_1} \cdot pw_{\mathit{Client}}$; $\alpha := H(\mathit{Client}|\mathsf{VK}|A|B|C)$
$D := A^{\delta+\alpha\zeta}$; $\mathit{msg\text{-}out}_1 := \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$
$\mathsf{status}^i_{\mathit{Client},1} := \langle \text{NULL}, \mathit{Server}, \text{FALSE}, \text{FALSE}\rangle$

$F := s^{r_2} g_1^{r'_2}$; $G := F^{\gamma}$; $I := t^{r_2} h^{r'_2} \cdot pw_{\mathit{Client}}$; $E := g_1^{x_2} g_2^{y_2} h^{z_2} (cd^{\alpha})^{w_2}$
$\beta := H(\mathit{Server}|E|F|G|I)$; $J := F^{\delta+\beta\zeta}$
$\mathit{msg\text{-}out}_2 := \langle \mathit{Server}|E|F|G|I|J\rangle$
$\mathsf{status}^j_{\mathit{Server},1} := \langle \text{NULL}, \mathit{Client}, \text{FALSE}, \text{FALSE}\rangle$

$K := g_1^{x_1} g_2^{y_1} h^{z_1} (cd^{\beta})^{w_1}$; $\mathsf{Sig} \leftarrow \mathsf{Sign}_{\mathsf{SK}}(\beta|K)$; $\mathit{msg\text{-}out}_3 := \langle K|\mathsf{Sig}\rangle$

$\mathsf{sid}^i_{\mathit{Client}} := \mathsf{sid}^j_{\mathit{Server}} := \langle \mathit{msg\text{-}out}_1|\mathit{msg\text{-}out}_2|\mathit{msg\text{-}out}_3\rangle$
$\mathsf{pid}^i_{\mathit{Client}} := \mathit{Server}$; $\mathsf{pid}^j_{\mathit{Server}} := \mathit{Client}$
$\mathsf{acc}^i_{\mathit{Client}} := \mathsf{term}^i_{\mathit{Client}} := \mathsf{acc}^j_{\mathit{Server}} := \mathsf{term}^j_{\mathit{Server}} := \text{TRUE}$
$\mathsf{sk}^i_{\mathit{Client}} := \mathsf{sk}^j_{\mathit{Server}} := A^{x_2} B^{y_2} (C/pw_{\mathit{Client}})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{\mathit{Client}})^{z_1} J^{w_1}$
return $\mathit{msg\text{-}out}_1, \mathit{msg\text{-}out}_2, \mathit{msg\text{-}out}_3,$
$\mathsf{status}^i_{\mathit{Client},1}, \mathsf{status}^j_{\mathit{Server},1}, \mathsf{status}^i_{\mathit{Client}}, \mathsf{status}^j_{\mathit{Server}}$

Figure 3.5: The modified Execute oracle for the proof of Claim 3.2.

$C = h^{br_1+r'_1} pw_{\mathit{Client}}$, and $D = (cd^{\alpha})^{br_1+r'_1}$ where $br_1 + r'_1$ is uniformly distributed in $\mathbb{Z}_q$ independent of the remainder of the experiment. Similarly, $F = g_1^{br_2+r'_2}$, $G = g_2^{br_2+r'_2}$, $I = h^{br_2+r'_2} pw_{\mathit{Client}}$, and $J = (cd^{\beta})^{br_2+r'_2}$ where $br_2 + r'_2$ is uniformly distributed in $\mathbb{Z}_q$ independent of the remainder of the experiment. One can also check that the values of the session keys are as they would be in P'_0.

On the other hand, when the tuple (g, h, s, t) is a random tuple, the distribution on the view of A throughout the experiment has negligible statistical difference from the distribution on the view of A during experiment P_1. The public output of the modified initialization protocol is identically distributed to the public output of the initialization protocol for experiment P_1. As for queries to the Execute oracle, if (g, h, s, t) is a random tuple we may write $s = g^a$ and $t = h^b$ for uniformly distributed $a, b \in \mathbb{Z}_q$, where $a \neq b$ except with negligible probability $1/q$. In a particular query to the Execute oracle, we have $A = g_1^{ar_1+r'_1}$, $B = g_2^{ar_1+r'_1}$, $C = h^{br_1+r'_1} pw_{\mathit{Client}}$, and $D = (cd^{\alpha})^{ar_1+r'_1}$, where r_1, r'_1 are independently and uniformly distributed in $\mathbb{Z}_q$. Thus, when $a \neq b$ the joint distribution on $(ar_1 + r'_1, br_1 + r'_1)$ is uniform over $\mathbb{Z}_q^2$, independent of the remainder of the experiment; this follows from the linear independence of $ar_1 + r'_1$ and $br_1 + r'_1$ as non-zero equations in r_1, r'_1. In other words (assuming h is a generator), C is uniformly distributed in $\mathbb{G}$ independent of the rest of

the experiment. One can analogously verify that I is uniformly distributed in $\mathbb{G}$ independent of the rest of the experiment. Furthermore, the values of the session keys are as they would be in P_1.

The simulator's advantage in solving the DDH problem is therefore

$$\left|\Pr_{A,P_0'}[\mathsf{Succ}] - \tfrac{q-1}{q} \cdot \Pr_{A,P_1}[\mathsf{Succ}] - \tfrac{1}{q} \cdot \Pr_{A,P_0'}[\mathsf{Succ}]\right|$$

(if the random tuple satisfies $\log_g s = \log_h t$, the distribution on the view of A in the above experiment is equivalent to the distribution on the view of A in experiment P_0'; for a random tuple, this occurs with probability $1/q$). The claim follows from the observation that this advantage is negligible under the DDH assumption and from the fact that $1/q$ is negligible. □

In experiment P_2, the simulator interacts with the adversary as in P_1 except that during queries $\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$ the session key $\mathsf{sk}^i_{\mathit{Client}}$ is chosen uniformly at random from $\mathbb{G}$; session key $\mathsf{sk}^j_{\mathit{Server}}$ is set equal to $\mathsf{sk}^i_{\mathit{Client}}$.

Claim 3.3 $|\mathsf{Adv}_{A,P_1}(k) - \mathsf{Adv}_{A,P_2}(k)| \leq \varepsilon(k)$ *for some negligible function* $\varepsilon(\cdot)$.

The claim follows from the negligible statistical difference between the distributions on the adversary's view in the two experiments. In P_1, elements C and I are chosen at random and the session keys are computed as in (3.1). Assuming that h is a generator, we may write $C = h^{r_1'} pw_{\mathit{Client}}$ and $I = h^{r_2'} pw_{\mathit{Client}}$ for some $r_1', r_2' \in \mathbb{Z}_q$. With all but negligible probability $1/q^2$, we have either $r_1' \neq r_1$ or $r_2' \neq r_2$. Assume the former. For any $\mu, \nu \in \mathbb{G}$ and fixing the random choices for the remainder of experiment P_1, the probability over choice of x_2, y_2, z_2, w_2 that $E = \mu$ and $\mathsf{sk}^i_{\mathit{Client}} = \nu$ is exactly the probability that

$$\log_{g_1} \mu \quad = \quad x_2 + y_2 \cdot \log_{g_1} g_2 + z_2 \cdot \log_{g_1} h + w_2 \cdot \log_{g_1}(cd^\alpha) \tag{3.2}$$

and

$$\begin{aligned} &\log_{g_1} \nu - \log_{g_1}(F^{x_1} G^{y_1} (I/pw_{\mathit{Client}})^{z_1} J^{w_1}) = \\ &\qquad x_2 \cdot r_1 + y_2 \cdot r_1 \log_{g_1} g_2 + z_2 \cdot r_1' \log_{g_1} h + w_2 \cdot r_1 \log_{g_1}(cd^\alpha) \end{aligned} \tag{3.3}$$

where we assume g_1 is a generator (if this is not the case the experiment is aborted). Viewing (3.2) and (3.3) as equations in x_2, y_2, z_2, w_2 we see that they are linearly independent and not identically zero whenever $r_1' \neq r_1$ (here, we use the fact that h is a generator and therefore $\log_{g_1} h \neq 0$), the desired probability is $1/q^2$. In other words, when $r_1' \neq r_1$ the value of $\mathsf{sk}^i_{\mathit{Client}}$ is independent of the value of E and hence independent of the remainder of experiment P_1. A similar argument shows that when $r_2' \neq r_2$, the value of $\mathsf{sk}^i_{\mathit{Client}}$ is independent of K and hence independent of the rest of experiment P_1.

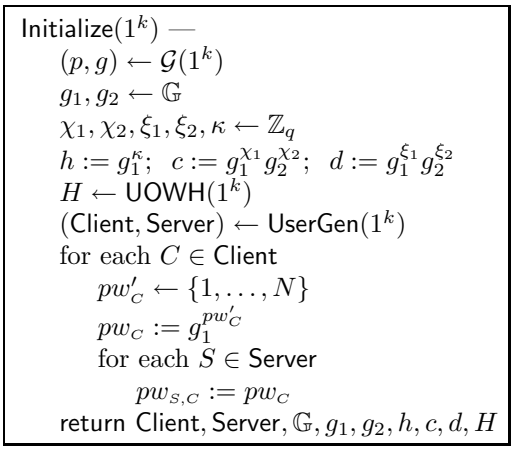

Figure 3.6: Modified initialization procedure.

Thus, the adversary's view in P_1 is distributed identically to the adversary's view in P_2 assuming that, for all Execute queries, either $r'_1 \neq r_1$ or $r'_2 \neq r_2$. For a particular Execute query, this condition holds except with negligible probability $1/q^2$. Since the adversary is permitted to query the Execute oracle only polynomially-many times, the claim follows. □

Before continuing, we introduce some notation. For a query $\mathsf{Send}_1(\mathit{Server}, j, \mathit{msg\text{-}in})$, where $\mathit{msg\text{-}in} = \langle \mathit{Client} | \mathsf{VK} | A | B | C | D \rangle$, we say that *msg-in* is *previously-used* if it was ever previously output by a Send_0 oracle. Similarly, when the adversary queries $\mathsf{Send}_2(\mathit{Client}, i, \mathit{msg\text{-}in})$, where $\mathit{msg\text{-}in} = \langle \mathit{Server} | E | F | G | I | J \rangle$, we say that *msg-in* is *previously-used* if it was ever previously output by a Send_1 oracle. A *msg-in* for either a Send_1 or Send_2 oracle query which is not previously-used is called *new*.

In experiment P_3, the simulator runs the modified initialization procedure shown in Figure 3.6, where the values $\chi_1, \chi_2, \xi_1, \xi_2, \kappa$ are stored for future use. Furthermore, queries to the Send_2 oracle are handled differently. Upon receiving query $\mathsf{Send}_2(\mathit{Client}, i, \langle \mathit{Server} | E | F | G | I | J \rangle)$, the simulator examines *msg-in*. If *msg-in* is previously-used, the query is answered as in experiment P_2. If *msg-in* is new, the simulator checks whether $F^{\chi_1 + \beta\xi_1} G^{\chi_2 + \beta\xi_2} \stackrel{?}{=} J$ and $I / pw_{Client} \stackrel{?}{=} F^\kappa$. If not, the query is said to *appear invalid* and is answered as in experiment P_2. Otherwise, *msg-in* is said to *appear valid*; the query is answered as in experiment P_2 except that if sk^i_{Client} is to be assigned a value, it is assigned the special value ∇.

Queries to the Send_3 oracle are also handled differently. Upon query $\mathsf{Send}_3(\mathit{Server}, j, \mathit{msg\text{-}in})$, the simulator examines $\mathit{first\text{-}msg\text{-}in} = \langle \mathit{Client} | \mathsf{VK} | A | B | C | D \rangle$, the message sent to the Send_1 oracle for the same instance; note that if *first-msg-in* is not defined, the query to Send_3 simply returns $\perp$ as in experiment P_2. If *first-msg-in* is previously-used, the query is answered as in experiment

P_2. If *first-msg-in* is new, the simulator computes $\alpha = H(Client|\mathsf{VK}|A|B|C)$ and checks whether $A^{\chi_1+\alpha\xi_1}B^{\chi_2+\alpha\xi_2} \stackrel{?}{=} D$ and $C/pw_{Client} \stackrel{?}{=} A^{\kappa}$. If not, *first-msg-in* is said to *appear invalid* and the query is answered as in experiment P_2. Otherwise, *first-msg-in* is said to *appear valid* and the query is answered as in experiment P_2 except that if sk^j_{Server} is to be assigned a value, it is assigned the special value ∇.

Finally, the definition of the adversary's success is changed. If the adversary ever queries $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$ where $\mathsf{sk}^i_U = \nabla$, the simulator halts execution and the adversary immediately succeeds. Otherwise, the adversary's success is determined as in experiment P_2.

Claim 3.4 $\mathsf{Adv}_{A,P_2}(k) \leq \mathsf{Adv}_{A,P_3}(k)$.

The probability that g_1 and g_2 are both generators is the same in experiments P_2 and P_3. Conditioned on the event that g_1 and g_2 are generators (if not, the experiment is aborted), the distributions on the adversary's views in experiments P_2 and P_3 are identical until the adversary queries $\mathsf{Reveal}(U, i)$ or $\mathsf{Test}(U, i)$ where $\mathsf{sk}^i_U = \nabla$; if such a query is never made, the distributions on the views are identical. The claim follows immediately since there are more ways for the adversary to succeed in experiment P_3. □

In experiment P_4, queries to the Send_3 oracle are handled differently. First, whenever the simulator responds to a Send_2 query, the simulator stores the values (K, β, x, y, z, w), where $K = g_1^x g_2^y h^z (cd^\beta)^w$. Upon receiving query $\mathsf{Send}_3(Server, j, \langle K|\mathsf{Sig}\rangle)$, the simulator checks the value of *first-msg-in* $= \langle Client|\mathsf{VK}|A|B|C|D\rangle$ (if *first-msg-in* is not defined the query to Send_3 simply returns $\perp$ as in experiment P_3). If *first-msg-in* is new, the query is answered as in experiment P_3. If *first-msg-in* is previously-used and $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) \neq 1$, the query is answered as in experiment P_3 and the session key is not assigned a value. If *first-msg-in* is previously-used, $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) = 1$, and the experiment is not aborted, the simulator first checks whether there exists an i such that $\mathsf{sid}^i_{Client} = \mathsf{sid}^j_{Server}$ (if such an i exists it must be unique since the experiment is aborted if a verification key VK repeats during the experiment). If so, sk^j_{Server} (if it is assigned a value at all) is assigned the value sk^i_{Client}. Otherwise, let *first-msg-out* $= \langle Server|E|F|G|I|J\rangle$. The simulator must have stored values x', y', z', w' such that $K = g_1^{x'} g_2^{y'} h^{z'} (cd^\beta)^{w'}$ (this is true since the experiment is aborted if Sig is a valid signature on $\beta|K$ that was not output by the simulator following a Send_2 query). The session key (assuming it is assigned a value at all) is then assigned the value:

$$\mathsf{sk}^j_{Server} := A^x B^y (C/pw_{Client})^z D^w F^{x'} G^{y'} (I/pw_{Client})^{z'} J^{w'}.$$

Claim 3.5 $\mathsf{Adv}_{A,P_4}(k) = \mathsf{Adv}_{A,P_3}(k)$.

The distribution on the adversary's view is identical in experiments P_3 and P_4. Indeed, when *first-msg-in* is previously used, $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) = 1$, and there exists an i as described above, it is always

the case that $\mathsf{sk}^j_{Server} = \mathsf{sk}^i_{Client}$. When *first-msg-in* is previously used, $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) = 1$, and an i such that $\mathsf{sid}^j_{Server} = \mathsf{sid}^i_{Client}$ does not exist, then as long as sk^j_{Server} is to be assigned a value it is the case that $K^r = F^{x'} G^{y'} (I/pw_{Client})^{z'} J^{w'}$, where x', y', z', w' are as above. Therefore, the session key computed in P_4 matches the session key that would have been computed in P_3. □

In experiment P_5, queries to the Send_3 oracle are handled differently. Upon receiving query $\mathsf{Send}_3(Server, j, \langle K|\mathsf{Sig}\rangle)$, the simulator checks the value of *first-msg-in* (if *first-msg-in* is not defined the simulator returns $\perp$ as in experiment P_4). If *first-msg-in* is new and appears invalid and the session key is to be assigned a value, the session key is assigned a value randomly chosen in $\mathbb{G}$. Otherwise, the query is answered as in experiment P_4.

Claim 3.6 $\mathsf{Adv}_{A,P_5}(k) = \mathsf{Adv}_{A,P_4}(k)$.

The claim will follow from the equivalence of the distributions on the adversary's view in the two experiments. For a given query $\mathsf{Send}_3(Server, j, msg\text{-}in)$ where *first-msg-in* $= \langle Client|\mathsf{VK}|A|B|C|D\rangle$ is new and appears invalid, let *first-msg-out* $= \langle Server|E|F|G|I|J\rangle$ and $\alpha = H(Client|\mathsf{VK}|A|B|C)$. Since *first-msg-in* appears invalid, it must be the case that either $A^{\chi_1+\alpha\xi_1} B^{\chi_2+\alpha\xi_2} \neq D$ or else $C/pw_{Client} \neq A^{\kappa}$ (or possibly both). For any $\mu, \nu \in \mathbb{G}$ and fixing the randomness used in the rest of experiment P_4, the probability over choice of x, y, z, w that $E = \mu$ and $\mathsf{sk}^j_{Server} = \nu$ is exactly the probability that

$$\log_{g_1} \mu = x + y \cdot \log_{g_1} g_2 + z \cdot \log_{g_1} h + w \cdot \log_{g_1}(cd^{\alpha}) \tag{3.4}$$

and

$$\log_{g_1} \nu - r \log_{g_1} K = x \cdot \log_{g_1} A + y \cdot \log_{g_1} B + z \cdot \log_{g_1}(C/pw_{Client}) + w \cdot \log_{g_1} D, \tag{3.5}$$

where we use the fact that g_1 is a generator (if not, the experiment is aborted). If $\log_{g_1} A = 0$, it can be verified immediately that (3.4) and (3.5) are linearly independent and not identically zero (this last fact follows by noting that $\log_{g_1} A = 0$ implies $\log_{g_1}(C/pw_{Client}) \neq 0$). If $\log_{g_1} A \neq 0$, it can be similarly verified that (3.4) and (3.5) are linearly independent and not identically zero. In either case, then, the desired probability is $1/q^2$. In other words, the value of sk^j_{Server} is independent of the value of E and hence independent of the remainder of the experiment. □

In experiment P_6, queries to the Send_1 oracle are handled differently. Now, I is computed as $h^r g_1^{N+1}$, where the dictionary of legal passwords is $\{1, \ldots, N\}$; note that g_1^{N+1} represents an invalid password since $N < q - 1$.

Claim 3.7 *Under the DDH assumption,* $|\mathsf{Adv}_{A,P_5}(k) - \mathsf{Adv}_{A,P_6}(k)| \leq \varepsilon(k)$, *for some negligible function* $\varepsilon(\cdot)$.

The claim follows from the non-malleability of the commitment scheme used (i.e., the Cramer-Shoup encryption scheme). We show that the simulator can use A as a subroutine in order to distinguish encryptions of the correct client password(s) from encryptions of g_1^{N+1}. The simulator is given a public key $pk = \langle \mathbb{G}, g_1, g_2, h, c, d, H \rangle$ for an instance of the Cramer-Shoup encryption scheme and may repeatedly query an encryption oracle $\mathcal{O}_{1^k, pk, \widetilde{b}}(\cdot, \cdot, \cdot, \cdot)$ where $\widetilde{b}$ is a randomly-chosen bit. The simulator may also query a decryption oracle $\mathcal{D}_{sk}(\cdot)$ using any ciphertext *except* those received from its encryption oracle. The advantage of the simulator is half the absolute value of the difference between the probability the simulator outputs 1 when $\widetilde{b} = 1$ and the probability the simulator outputs 1 when $\widetilde{b} = 0$.

The simulator begins by running the following modified initialization protocol:

$\mathsf{Initialize}'(1^k, \mathbb{G}, g_1, g_2, h, c, d, H)$ —
  $(\mathsf{Client}, \mathsf{Server}) \leftarrow \mathsf{UserGen}(1^k)$
  for each $C \in \mathsf{Client}$
    $pw'_C \leftarrow \{1, \ldots, N\}$
    $pw_C := g_1^{pw'_C}$
    for each $S \in \mathsf{Server}$
      $pw_{S,C} := pw_C$
  return $\mathsf{Client}, \mathsf{Server}, \mathbb{G}, g_1, g_2, h, c, d, H$

The simulator responds to $\mathsf{Send}_0, \mathsf{Send}_2$, Execute, Reveal, and Test oracle queries as in experiments P_5, P_6. The simulator responds to Send_1 and Send_3 queries as shown in Figure 3.7 (the simulator's response to Send_3 queries is the same as in experiments P_5, P_6, but is included in Figure 3.7 for convenience). In particular, when a response to a Send_1 query is needed, the simulator queries the encryption oracle, requesting a server-encryption of either the correct password or the value g_1^{N+1}. To respond to a Send_3 query, the simulator checks whether *first-msg-in* is previously-used or new. If *first-msg-in* is new, the server determines whether it appears valid or appears invalid by submitting *first-msg-in* to the decryption oracle. Note that the simulator never need submit to the decryption oracle a ciphertext that it received from the encryption oracle. The simulator outputs 1 if and only if A succeeds.

Examination of Figure 3.7 shows that when $\widetilde{b} = 0$ the distribution on the view of A throughout the experiment is equivalent to the distribution on the view of A in experiment P_5. On the other hand, when $\widetilde{b} = 1$ the distribution on the view of A throughout the experiment is equivalent to the distribution on the view of A in experiment P_6. The simulator's advantage is therefore

$$\frac{1}{2} \cdot |\Pr_{A,P_5}[\mathsf{Succ}] - \Pr_{A,P_6}[\mathsf{Succ}]|.$$

The claim follows from the observation that this advantage is negligible under the DDH assumption (cf. Lemma 3.1). □

$\mathsf{Send}_1(\mathit{Server}, j, \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle)$ —
  if $\mathit{Server} \notin \mathsf{Server}$ or $\mathsf{used}^j_{\mathit{Server}}$
    return $\perp$
  $\mathsf{used}^j_{\mathit{Server}} := \text{TRUE}$
  if $A, B, C, D \notin \mathbb{G}$ or $\mathit{Client} \notin \mathsf{Client}$
    $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$; return $\mathsf{status}^j_{\mathit{Server}}$
  $\alpha := H(\mathit{Client}|\mathsf{VK}|A|B|C)$
  $(\langle \mathit{Server}|E|F|G|I|J\rangle, (x, y, z, w)) \leftarrow \mathcal{O}_{1^k, pw, \tilde{b}}(pw_{\mathit{Client}}, g_1^{N+1}, 1, (\mathit{Server}, \alpha))$
  $\beta := H(\mathit{Server}|E|F|G|I)$; $\mathit{msg\text{-}out} := \langle \mathit{Server}|E|F|G|I|J\rangle$
  $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \mathit{Client}, \text{FALSE}, \text{FALSE}\rangle$
  $\mathsf{state}^j_{\mathit{Server}} := \langle \mathit{msg\text{-}in}, x, y, z, w, \beta, \mathit{msg\text{-}out}, pw_{\mathit{Server},\mathit{Client}}\rangle$
  return $\mathit{msg\text{-}out}, \mathsf{status}^j_{\mathit{Server}}$

$\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$ —
  if $\mathit{Server} \notin \mathsf{Server}$ or not $\mathsf{used}^j_{\mathit{Server}}$ or $\mathsf{term}^j_{\mathit{Server}}$
    return $\perp$
  $\langle \mathit{first\text{-}msg\text{-}in}, x, y, z, w, \beta, \mathit{first\text{-}msg\text{-}out}, pw\rangle := \mathsf{state}^j_{\mathit{Server}}$
  $\langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle := \mathit{first\text{-}msg\text{-}in}$
  if $K \notin \mathbb{G}$ or $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) \neq 1$
    $\mathsf{status}^j_{\mathit{Server}} := \langle \text{NULL}, \text{NULL}, \text{FALSE}, \text{TRUE}\rangle$; return $\mathsf{status}^j_{\mathit{Server}}$
  $\mathsf{sid}^j_{\mathit{Server}} := \langle \mathit{first\text{-}msg\text{-}in}|\mathit{first\text{-}msg\text{-}out}|\mathit{msg\text{-}in}\rangle$
  $\mathsf{acc}^j_{\mathit{Server}} := \mathsf{term}^j_{\mathit{Server}} := \text{TRUE}$
  if *first-msg-in* is previously-used
    if there exists an i such that $\mathsf{sid}^i_{\mathit{Client}} = \mathsf{sid}^j_{\mathit{Server}}$
      $\mathsf{sk}^j_{\mathit{Server}} := \mathsf{sk}^i_{\mathit{Client}}$
    else
      retrieve x', y', z', w' such that $K = g_1^{x'} g_2^{y'} h^{z'} (cd^\beta)^{w'}$
      (if such values are not stored, abort the experiment)
      $\mathsf{sk}^j_{\mathit{Server}} := A^x B^y (C/pw)^z D^w F^{x'} G^{y'} (I/pw)^{z'} J^{w'}$
  else
    $pw' := \mathcal{D}_{sk}(\mathit{first\text{-}msg\text{-}in})$
    if $pw' = pw_{\mathit{Client}}$ then $\mathsf{sk}^j_{\mathit{Server}} := \nabla$
    if $pw' \neq pw_{\mathit{Client}}$ then $\mathsf{sk}^j_{\mathit{Server}} \leftarrow \mathbb{G}$
  return $\mathsf{status}^j_{\mathit{Server}}$

Figure 3.7: The modified Send_1 and Send_3 oracles for the proof of Claim 3.7.

In experiment P_7, queries to the Send_2 oracle are handled differently. Whenever *msg-in* is new and appears invalid or is previously-used, the session key (if it is assigned a value at all) is assigned a value chosen randomly from $\mathbb{G}$. Queries to the Send_3 oracle are also handled differently. If *first-msg-in* is previously-used, $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K,\mathsf{Sig}) = 1$, and there does not exist an i such that $\mathsf{sid}^i_{Client} = \mathsf{sid}^j_{Server}$, then the session key is assigned a value chosen randomly from $\mathbb{G}$.

Claim 3.8 $\mathsf{Adv}_{A,P_7}(k) = \mathsf{Adv}_{A,P_6}(k)$.

The claim follows from the equivalence of the distributions on the adversary's views in the two experiments. First consider a particular query $\mathsf{Send}_2(\mathit{Client}, i, \mathit{msg\text{-}in})$ where $\mathit{msg\text{-}in} = \langle \mathit{Server}|E|F|G|I|J\rangle$ is previously-used. Let $\beta = H(\mathit{Server}|E|F|G|I)$ and $r = \log_{g_1} E$. Since *msg-in* is previously-used, it was output in response to some query to the Send_1 oracle; therefore, regardless of *Client* we have $F = g_1^r$, $F = g_2^r$, $J = (cd^\beta)^r$, and $I = h^{r'} \cdot pw_{Client}$ for some $r' \neq r$. In particular, $I/pw_{Client} \neq F^\kappa$. A proof similar to that of Claim 3.6 indicates that sk^i_{Client} is uniformly distributed independent of the rest of the experiment.

If the simulator responds to a query $\mathsf{Send}_2(\mathit{Client}, i, \mathit{msg\text{-}in})$ and $\mathit{msg\text{-}in} = \langle \mathit{Server}|E|F|G|I|J\rangle$ is new and appears invalid, let $\langle K|\mathsf{Sig}\rangle$ be the message output by the simulator in responding to this query. There are two cases to consider. In the first case, the values x, y, z, w used by instance Π^i_{Client} are used to compute only K and sk^i_{Client} (and, in particular, are never used to compute a session key sk^j_{Server} during a Send_3 query). In this case, an argument exactly as in the proof of Claim 3.6 shows that the value of sk^i_{Client} is independent of the value K and hence independent of the remainder of the experiment. In the second case, the values x, y, z, w are used at some point to compute a session key sk^j_{Server} when the simulator responds to a Send_3 query. Note that these values can be used at most once to compute a session key during a Send_3 query, since the session key is assigned a value only if $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K,\mathsf{Sig}) = 1$ and the experiment is aborted if a value β used by the simulator in responding to a Send_1 query is used twice. We show that in this case the joint distribution on $(K, \mathsf{sk}^i_{Client}, \mathsf{sk}^j_{Server})$ is uniform, independent of the rest of the experiment.

Let

$$\mathsf{sid}^i_{Client} = \langle \mathit{Client}|\mathsf{VK}|A|B|C|D|\mathit{Server}|E|F|G|I|J|K|\mathsf{Sig}\rangle$$

and

$$\mathsf{sid}^j_{Server} = \langle \mathit{Client}'|\mathsf{VK}'|A'|B'|C'|D'|\mathit{Server}'|E'|F'|G'|I'|J'|K|\mathsf{Sig}\rangle,$$

where the same randomly-chosen values x, y, z, w are used during computation of K, sk^i_{Client}, and sk^j_{Server}. Since verification keys used by the simulator in responding to Send_0 queries do not repeat and $\langle \mathit{Client}'|\mathsf{VK}'|A'|B'|C'|D'\rangle$ is previously-used (if not, x, y, z, w are not used to compute sk^j_{Server}), it must be the case that $\langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle = \langle \mathit{Client}'|\mathsf{VK}'|A'|B'|C'|D'\rangle$. Furthermore, we must

have $\langle Server|E|F|G|I\rangle = \langle Server'|E'|F'|G'|I'\rangle$ (otherwise a collision in H has been found and the experiment is aborted) and $J \neq J'$ (otherwise $\mathsf{sid}^j_{Server} = \mathsf{sid}^i_{Client}$ and x, y, z, w are not used to compute sk^j_{Server}). Denote pw_{Client} by pw_C and let $\log(\cdot)$ denote $\log_{g_1}(\cdot)$ (recall that g_1 is a generator). For any $\mu, \nu_1, \nu_2 \in \mathbb{G}$ and fixing the randomness used in the rest of experiment P_6, the probability over choice of x, y, z, w that $K = \mu$, $\mathsf{sk}^i_{Client} = \nu_1$, and $\mathsf{sk}^j_{Server} = \nu_2$ is exactly the probability that

$$\begin{aligned} \log \mu &= x + y \cdot \log g_2 + z \cdot \log h + w \cdot \log(cd^\beta) \\ \log \nu_1 - r \log E &= x \cdot \log F + y \cdot \log G + z \cdot \log(I/pw_C) + w \cdot \log J \end{aligned}$$

and

$$\log \nu_2 - \log(A^{x'} B^{y'} (C/pw_C)^{z'} D^{w'}) = x \cdot \log F + y \cdot \log G + z \cdot \log(I/pw_C) + w \cdot \log J'.$$

Letting $R \stackrel{\text{def}}{=} \log F$ and γ denote $A^{x'} B^{y'} (C/pw_C)^{z'} D^{w'}$, and using the fact that $\langle Server|E|F|G|I|J\rangle$ was output by the simulator in response to a Send_1 query, we may re-write these equations as

$$\log \mu = x + y \cdot \log g_2 + z \cdot \log h + w \cdot \log(cd^\beta) \tag{3.6}$$

$$\log \nu_1 - r \log E = x \cdot R + y \cdot R \log g_2 + z \cdot R' \log h + w \cdot R \log(cd^\beta) \tag{3.7}$$

$$\log \nu_2 - \log \gamma = x \cdot R + y \cdot R \log g_2 + z \cdot R' \log h + w \cdot R'' \log(cd^\beta), \tag{3.8}$$

for some values $R', R'' \in \mathbb{Z}_q$ such that $R \neq R'$ and $R \neq R''$. If $R = 0$ then $R', R'' \neq 0$ and it is easy to verify that (3.6)–(3.8) are linearly independent and not identically zero. If $R \neq 0$ one can similarly verify that (3.6)–(3.8) are linearly independent and not identically zero. In either case, the joint distribution of $(K, \mathsf{sk}^i_{Client}, \mathsf{sk}^j_{Server})$ is uniform independent of the remainder of the experiment. □

In experiment P_8, queries to the Send_0 oracle are handled differently. In particular, C is computed as $h^r g_1^{N+1}$, where $\{1, \ldots, N\}$ is the dictionary of legal passwords.

Claim 3.9 *Under the DDH assumption,* $|\mathsf{Adv}_{A,P_8}(k) - \mathsf{Adv}_{A,P_7}(k)| \leq \varepsilon(k)$ *for some negligible function* $\varepsilon(\cdot)$.

The proof exactly follows that of Claim 3.7. In particular, in responding to Send_2 queries, the simulator never requires the value r to compute a session key: if *msg-in* is previously-used, the session key (if it is assigned a value) is assigned a randomly-chosen value; if *msg-in* is new and appears invalid (which can be verified using the decryption oracle), the session key (if it is assigned a value) is assigned a random value; finally, if *msg-in* is new and appears valid (which can be verified using the decryption oracle), the session key (if it is assigned a value) is assigned ∇. □

The adversary's view in experiment P_8 is independent of the passwords chosen by the simulator, until one of the following occurs:

- The adversary queries $\mathsf{Reveal}(\mathit{Client}, i)$ or $\mathsf{Test}(\mathit{Client}, i)$, where the adversary had previously queried $\mathsf{Send}_2(\mathit{Client}, i, \mathit{msg\text{-}in})$ and *msg-in* was new and appeared valid.
- The adversary queries $\mathsf{Reveal}(\mathit{Server}, j)$ or $\mathsf{Test}(\mathit{Server}, j)$, where the adversary had previously queried $\mathsf{Send}_3(\mathit{Server}, j, \mathit{msg\text{-}in})$ and *first-msg-in* was new and appeared valid.

The probability that one of these events occurs is therefore at most $Q(k)/N$, where $Q(k)$ is the number of on-line attacks made by A. The adversary succeeds when one of the above events occurs or else by guessing the value of b. Assuming neither of the above events occur and the adversary queries $\mathsf{Test}(U, i)$ where Π_U^i is fresh and $\mathsf{acc}_U^i = \text{TRUE}$, then sk_U^i is randomly-distributed in $\mathbb{G}$ independent of the rest of the experiment. Thus, the adversary's probability of success in this case is at most 1/2. Therefore

$$\Pr_{A,P_8}[\mathsf{Succ}] \leq Q(k)/N + \tfrac{1}{2} \cdot (1 - \tfrac{Q(k)}{N})$$

and the adversary's advantage in experiment P_8 is at most $Q(k)/N$. Claims 3.1–3.9 show that

$$\mathsf{Adv}_{A,P_0}(k) \leq Q(k)/N + \varepsilon(k)$$

for some negligible function $\varepsilon(\cdot)$ and therefore the original P_0 is a secure, password-only, key-exchange protocol. ■

Theorem 3.2 *Assuming (1) the hardness of the DDH problem for groups $\mathbb{G}$ defined by the output of $\mathcal{G}$; (2) the security of $(\mathcal{K}, \mathsf{Sign}, \mathsf{Vrfy})$ as a one-time signature scheme; and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 3.1 is a forward-secure, password-only, key-exchange protocol.*

Proof We refer to the formal specification of the protocol in Figures 3.2–3.4. Here, however, the adversary also has access to a Corrupt oracle as specified in Figure 3.8. As in the proof of Theorem 3.1,

```
Corrupt(Client, pw, Server) —
    if Client ∉ Client or Server ∉ Server
        return ⊥
    if pw ≠⊥
        pw_{Server,Client} := pw
    return pw_{Client}
```

Figure 3.8: Specification of the Corrupt oracle.

we imagine a simulator that runs the protocol for any adversary A. When the adversary completes its execution and outputs a bit b', the simulator can tell whether the adversary succeeds by checking

whether (1) a single Test query was made on instance Π_U^i; (2) acc_U^i was TRUE at the time of the Test query; (3) instance Π_U^i is fs-fresh; and (4) $b' = b$. Success of the adversary is denoted by event fsSucc. We refer to the real execution of the experiment as P_0.

In experiment P_0', the simulator interacts with the adversary as before except that the adversary does *not* succeed when any of the following occur:

1. Any of g_1, g_2, h, c, d are not generators of $\mathbb{G}$ (i.e., they are equal to 1).
2. At any point during the experiment, a verification key VK used by the simulator in responding to a Send_0 query is repeated.
3. At any point during the experiment, the adversary forges a new, valid message/signature pair for any verification key used by the simulator in responding to a Send_0 query.
4. At any point during the experiment, a value β used by the simulator in responding to Send_1 queries is repeated.
5. At any point during the experiment, a value β used by the simulator in responding to a Send_1 query (with $\textit{msg-out} = \langle \textit{Server}|E|F|G|I|J\rangle$) is equal to a value β used by the simulator in responding to a Send_2 query (with $\textit{msg-in} = \langle \textit{Server}'|E'|F'|G'|I'|J'\rangle$) and furthermore $\langle \textit{Server}|E|F|G|I\rangle \neq \langle \textit{Server}'|E'|F'|G'|I'\rangle$.

Since the adversary, by definition, cannot succeed once any of these events occur, we assume that the simulator immediately halts execution if any of these events occur. Using the same proof as for Claim 3.1, it is clear that $\mathsf{fsAdv}_{A,P_0}(k) \leq \mathsf{fsAdv}_{A,P_0'}(k) + \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$.

In experiment P_1, queries to the $\mathsf{Execute}$ oracle are handled differently. Upon receiving oracle query $\mathsf{Execute}(\textit{Client}, i, \textit{Server}, j)$, the simulator checks whether $pw_{Client} = pw_{Server,Client}$. If so, the values C and I used in responding to $\mathsf{Execute}$ queries are chosen independently at random from $\mathbb{G}$ and the session keys are computed as

$$\mathsf{sk}_{Client}^i := \mathsf{sk}_{Server}^j := A^{x_2} B^{y_2} (C/pw_{Server,Client})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{Client})^{z_1} J^{w_1}.$$

On the other hand, if $pw_{Client} \neq pw_{Server,Client}$ (i.e., as the result of a $\mathsf{Corrupt}$ oracle query), the values C and I are chosen independently at random from $\mathbb{G}$ and the session keys are computed as

$$\begin{aligned} \mathsf{sk}_{Client}^i &:= A^{x_2} B^{y_2} (C/pw_{Client})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{Client})^{z_1} J^{w_1} \\ \mathsf{sk}_{Server}^j &:= A^{x_2} B^{y_2} (C/pw_{Server,Client})^{z_2} D^{w_2} F^{x_1} G^{y_1} (I/pw_{Server,Client})^{z_1} J^{w_1}. \end{aligned}$$

Using a similar proof as for Claim 3.2, it can be shown that, under the DDH assumption, we have $|\mathsf{fsAdv}_{A,P_0'}(k) - \mathsf{fsAdv}_{A,P_1}(k)| \leq \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$.

In experiment P_2, queries to the $\mathsf{Execute}$ oracle are again handled differently. Upon receiving query $\mathsf{Execute}(\mathit{Client}, i, \mathit{Server}, j)$, the simulator checks whether $pw_{\mathit{Client}} = pw_{\mathit{Server,Client}}$. If so, $\mathsf{sk}^i_{\mathit{Client}}$ is chosen randomly from $\mathbb{G}$ and $\mathsf{sk}^j_{\mathit{Server}}$ is set equal to $\mathsf{sk}^i_{\mathit{Client}}$. Otherwise, both $\mathsf{sk}^i_{\mathit{Client}}$ and $\mathsf{sk}^j_{\mathit{Server}}$ are chosen independently at random from $\mathbb{G}$.

Claim 3.10 $|\mathsf{fsAdv}_{A,P_1}(k) - \mathsf{fsAdv}_{A,P_2}(k)| \leq \varepsilon(k)$ *for some negligible function* $\varepsilon(\cdot)$.

The claim follows from the negligible statistical difference between the distributions on the adversary's view in the two experiments. The case of $pw_{\mathit{Client}} = pw_{\mathit{Server,Client}}$ exactly parallels the proof of Claim 3.3. So, consider an invocation of the $\mathsf{Execute}$ oracle when $pw_{\mathit{Client}} \neq pw_{\mathit{Server,Client}}$ (for brevity, denote pw_{Client} by pw_C and $pw_{\mathit{Server,Client}}$ by $pw_{S,C}$). We may write $C = h^{r'_1} pw_C = h^{r''_1} pw_{S,C}$ and $I = h^{r'_2} pw_C = h^{r''_2} pw_{S,C}$ where $r'_1 \neq r''_1$ and $r'_2 \neq r''_2$. With all but negligible probability, we also have $r_1 \neq r'_1, r''_2$ and $r_2 \neq r'_2, r''_2$. Now, for any $\mu_1, \mu_2, \nu_1, \nu_2 \in \mathbb{G}$ and fixing the random choices for the remainder of experiment P_1, the probability over choice of $x_1, y_1, z_1, w_1, x_2, y_2, z_2, w_2$ that $E = \mu_1$, $K = \mu_2$, $\mathsf{sk}^i_{\mathit{Client}} = \nu_1$, and $\mathsf{sk}^j_{\mathit{Server}} = \nu_2$ is exactly the probability that

$$\log_{g_1} \mu_1 = x_2 + y_2 \cdot \log_{g_1} g_2 + z_2 \cdot \log_{g_1} h + w_2 \cdot \log_{g_1}(cd^\alpha) \tag{3.9}$$

$$\log_{g_1} \mu_2 = x_1 + y_1 \cdot \log_{g_1} g_2 + z_1 \cdot \log_{g_1} h + w_1 \cdot \log_{g_1}(cd^\beta) \tag{3.10}$$

$$\begin{aligned} \log_{g_1} \nu_1 = {} & x_1 \cdot r_2 + y_1 \cdot r_2 \log_{g_1} g_2 + z_1 \cdot r'_2 \log_{g_1} h + w_1 \cdot r_2 \log_{g_1}(cd^\beta) \\ & + x_2 \cdot r_1 + y_2 \cdot r_1 \log_{g_1} g_2 + z_2 \cdot r'_1 \log_{g_1} h + w_2 \cdot r_1 \log_{g_1}(cd^\alpha) \end{aligned} \tag{3.11}$$

$$\begin{aligned} \log_{g_1} \nu_2 = {} & x_1 \cdot r_2 + y_1 \cdot r_2 \log_{g_1} g_2 + z_1 \cdot r''_2 \log_{g_1} h + w_1 \cdot r_2 \log_{g_1}(cd^\beta) \\ & + x_2 \cdot r_1 + y_2 \cdot r_1 \log_{g_1} g_2 + z_2 \cdot r''_1 \log_{g_1} h + w_2 \cdot r_1 \log_{g_1}(cd^\alpha). \end{aligned} \tag{3.12}$$

Since (3.9)–(3.12) are linearly independent and not identically zero when $r_1 \neq r'_1, r''_2$ and $r_2 \neq r'_2, r''_2$, the values $E, K, \mathsf{sk}^i_{\mathit{Client}}$, and $\mathsf{sk}^j_{\mathit{Server}}$ are independently and uniformly distributed, independent of the rest of the experiment. □

In experiment P_3, the simulator runs the modified initialization procedure shown in Figure 3.6, with the values $\chi_1, \chi_2, \xi_1, \xi_2, \kappa$ stored as before. Define the terms *previously-used* and *new* as in the proof of Theorem 3.1. As before, a new *msg-in* for a query $\mathsf{Send}_2(\mathit{Client}, i, \langle \mathit{Server} | E | F | G | I | J \rangle)$ is said to *appear valid* only if $F^{\chi_1 + \beta\xi_1} G^{\chi_2 + \beta\xi_2} = J$ and $I / pw_{\mathit{Client}} = F^\kappa$. Otherwise, it *appears invalid*. A new *msg-in* for a query $\mathsf{Send}_1(\mathit{Server}, j, \langle \mathit{Client} | \mathsf{VK} | A | B | C | D \rangle)$ *appears valid* only if $A^{\chi_1 + \alpha\xi_1} B^{\chi_2 + \alpha\xi_2} = D$ and $C / pw_{\mathit{Server,Client}} = A^\kappa$, where the value of $pw_{\mathit{Server,Client}}$ is that *at the time of the* Send_1 *query* (this is an important point to bear in mind, since the value of $pw_{\mathit{Server,Client}}$ may change as a result of $\mathsf{Corrupt}$ queries).

In experiment P_3, queries to the Send_2 oracle are handled differently before any $\mathsf{Corrupt}$ queries have been made (after a $\mathsf{Corrupt}$ query is made, the behavior of the Send_2 oracle is as in experiment

P_2). Upon receiving query $\mathsf{Send}_2(\mathit{Client}, i, \mathit{msg\text{-}in})$, the simulator examines *msg-in*. If *msg-in* is new and appears invalid, the query is answered as in experiment P_2. If *msg-in* is new and appears valid, the query is answered as in experiment P_2 but the simulator stores the value $(\nabla, \mathsf{sk}^i_{Client})$ as the "session key". If *msg-in* is previously-used, the simulator checks for the Send_1 query following which *msg-in* was output (note that this query will be unique, since the experiment is aborted if a value β used by the Send_1 oracle repeats). Say this query was $\mathsf{Send}_1(\mathit{Server}, j, \mathit{msg\text{-}in}')$. If *msg-in*$'$ is new and appears invalid, the Send_2 query is answered as in experiment P_2. If *msg-in*$'$ is new and appears valid, the query is answered as in experiment P_2 but the simulator stores the value $(\nabla, \mathsf{sk}^i_{Client})$ for sk^i_{Client}.

Queries to the Send_3 oracle are also handled differently. Upon receiving a query of the form $\mathsf{Send}_3(\mathit{Server}, j, \mathit{msg\text{-}in})$ and assuming used^j_{Server} is true, the simulator first checks when the query $\mathsf{Send}_1(\mathit{Server}, j, \mathit{first\text{-}msg\text{-}in})$ was made. If this Send_1 query was made after any $\mathsf{Corrupt}$ query was made, the behavior of the Send_3 oracle is unchanged. Otherwise, the simulator checks *first-msg-in* and responds as in experiment P_2 unless *first-msg-in* is new and appears valid. When *first-msg-in* is new and valid the query is answered as in experiment P_2 but the simulator stores the value $(\nabla, \mathsf{sk}^j_{Server})$ as the "session key".

The Test and Reveal oracle queries are also handled differently. If the adversary queries $\mathsf{Test}(U, i)$ or $\mathsf{Reveal}(U, i)$ before any $\mathsf{Corrupt}$ queries have been made and the session key stored is of the form $(\nabla, \mathsf{sk}^i_U)$, the adversary is given ∇. If the Test or Reveal query is made after any $\mathsf{Corrupt}$ query has been made, the adversary is given the value sk^i_U. Test or Reveal queries where the session key is not stored along with ∇ are answered as in experiment P_2. Finally, the definition of success is changed. If the adversary ever receives the value ∇ in response to a Reveal or Test query, the adversary succeeds and the experiment is aborted (note that this can only occur before a $\mathsf{Corrupt}$ query has been made). Otherwise, the adversary may succeed, as before, by correctly guessing the bit b.

As in the proof of Theorem 3.1, we clearly have $\mathsf{fsAdv}_{A,P_2}(k) \leq \mathsf{fsAdv}_{A,P_3}(k)$ since the number of ways the adversary can succeed is increased.

In general, the actions of the Send_2 oracle will be modified only when the Send_2 query is made before any $\mathsf{Corrupt}$ queries have been made. Similarly, actions of the Send_3 oracle on query $\mathsf{Send}_3(\mathit{Server}, j, \mathit{msg\text{-}in})$ will be modified only if the query $\mathsf{Send}_1(\mathit{Server}, j, \mathit{first\text{-}msg\text{-}in})$ was made before any $\mathsf{Corrupt}$ queries were made (if no Send_1 query of this form was made, the query to Send_3 simply returns $\perp$). For clarity, however, we will still explicitly mention this condition when describing the modified experiments.

In experiment P_4, queries to the Send_3 oracle are handled differently. First, whenever the sim-

ulator responds to a Send_2 query, the simulator stores the values (K, β, x, y, z, w), where $K = g_1^x g_2^y h^z (cd^\beta)^w$. Upon receiving query $\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$ (as before, this assumes that query $\mathsf{Send}_1(\mathit{Server}, j, \mathit{first\text{-}msg\text{-}in})$ was made before any $\mathsf{Corrupt}$ queries), the simulator checks the value of $\mathit{first\text{-}msg\text{-}in} = \langle \mathit{Client}|\mathsf{VK}|A|B|C|D\rangle$ (if *first-msg-in* is not defined the query to Send_3 simply returns $\perp$ as in experiment P_3). If *first-msg-in* is new, the query is answered as in experiment P_3. If *first-msg-in* is previously-used and $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) = 0$, the query is answered as in experiment P_3 and the session key is not assigned a value. If *first-msg-in* is previously-used, $\mathsf{Vrfy}_{\mathsf{VK}}(\beta|K, \mathsf{Sig}) = 1$, and the experiment is not aborted, the simulator first checks whether there exists an i such that $\mathsf{sid}^i_{\mathit{Client}} = \mathsf{sid}^j_{\mathit{Server}}$ (if such an i exists, it must be unique since verification keys VK are not repeated during Send_0 queries). If so, $\mathsf{sk}^j_{\mathit{Server}}$ (if it is assigned a value at all) is assigned the value $\mathsf{sk}^i_{\mathit{Client}}$. Otherwise, let $\mathit{first\text{-}msg\text{-}out} = \langle \mathit{Server}|E|F|G|I|J\rangle$. The simulator must have stored values x', y', z', w' such that $K = g_1^{x'} g_2^{y'} h^{z'} (cd^\beta)^{w'}$ (this is true since the experiment is aborted if Sig is a valid signature on $\beta|K$ that was not output by the simulator following a Send_2 query). The session key (assuming it is assigned a value at all) is assigned the value:

$$\mathsf{sk}^j_{\mathit{Server}} := A^x B^y (C/pw)^z D^w F^{x'} G^{y'} (I/pw)^{z'} J^{w'}.$$

Claim 3.11 $\mathsf{fsAdv}_{A,P_4}(k) = \mathsf{fsAdv}_{A,P_3}(k)$.

The distribution on the adversary's view is identical in experiments P_3 and P_4; the proof is as for Claim 3.5. It is crucial here that the value $pw_{\mathit{Server},\mathit{Client}}$ used when responding to a Send_1 query is stored as part of the state and thus the same value is used subsequently when responding to a Send_3 query (cf. Figure 3.4). If this were not the case, the value of $pw_{\mathit{Server},\mathit{Client}}$ could change (as a result of a $\mathsf{Corrupt}$ query) sometime between the Send_1 and Send_3 queries. □

In experiment P_5, queries $\mathsf{Send}_3(\mathit{Server}, j, \mathit{msg\text{-}in})$ are handled differently (again, this assumes that the query $\mathsf{Send}_1(\mathit{Server}, j, \mathit{first\text{-}msg\text{-}in})$ was made before any $\mathsf{Corrupt}$ queries). Upon receiving query $\mathsf{Send}_3(\mathit{Server}, j, \langle K|\mathsf{Sig}\rangle)$, the simulator checks the value of *first-msg-in* (if *first-msg-in* is not defined the simulator returns $\perp$ as in experiment P_4). If *first-msg-in* is new and appears invalid and the session key is to be assigned a value, the session key is assigned a value randomly chosen in $\mathbb{G}$. Otherwise, the query is answered as in experiment P_4.

Claim 3.12 $\mathsf{fsAdv}_{A,P_5}(k) = \mathsf{fsAdv}_{A,P_4}(k)$.

The proof exactly follows the proof of Claim 3.6. Again, it is crucial that the same value $pw_{\mathit{Server},\mathit{Client}}$ is used during both the Send_1 and Send_3 queries for a given instance (this is achieved by storing $pw_{\mathit{Server},\mathit{Client}}$ as part of the state). □

In experiment P_6, queries to the Send_1 oracle are handled differently when a Send_1 query is made before any $\mathsf{Corrupt}$ queries (Send_1 queries are responded to as in experiment P_5 when they are made after a $\mathsf{Corrupt}$ query). Upon receiving query $\mathsf{Send}_1(\textit{Server}, j, \textit{msg-in})$, if *msg-in* is new and appears valid the query is answered as before. Otherwise, component I is computed as $h^r g_1^{N+1}$, where the dictionary of legal passwords is $\{1, \ldots, N\}$; note that g_1^{N+1} represents an invalid password since $N < q - 1$.

Claim 3.13 *Under the DDH assumption,* $|\mathsf{fsAdv}_{A,P_5}(k) - \mathsf{fsAdv}_{A,P_6}(k)| \leq \varepsilon(k)$*, for some negligible function* $\varepsilon(\cdot)$.

The proof of this claim exactly follows the proof of Claim 3.7. In particular, the claim follows from the non-malleability of the commitment scheme used. When a query $\mathsf{Send}_1(\textit{Server}, j, \textit{first-msg-in})$ is made before any $\mathsf{Corrupt}$ queries and *first-msg-in* is not both new and valid, the simulator will not require r in order to respond to the (subsequent) query $\mathsf{Send}_3(\textit{Server}, j, \textit{msg-in})$ in case this query is ever made. □

In contrast to the basic case when no $\mathsf{Corrupt}$ queries are allowed, the simulator *does* require r in case *first-msg-in* is both new and appears valid. The reason is the following: Assume the adversary queries $\mathsf{Send}_1(\textit{Server}, j, \langle \textit{Client} | \mathsf{VK} | A | B | C | D \rangle)$ (before any $\mathsf{Corrupt}$ queries have been made) where $\langle \textit{Client} | \mathsf{VK} | A | B | C | D \rangle$ is new and appears valid. Assume the adversary subsequently learns $pw_{Server,Client} = pw$ from a $\mathsf{Corrupt}$ query. The adversary might then query $\mathsf{Send}_3(\textit{Server}, j, \langle K | \mathsf{Sig} \rangle)$ followed by $\mathsf{Reveal}(\textit{Server}, j)$. In this case, the simulator must give the adversary the correct session key sk^j_{Server} — but this will be impossible without r.

In experiment P_7, queries to the Send_2 oracle are handled differently (when they are made before any $\mathsf{Corrupt}$ queries). Whenever *msg-in* is new and appears invalid, the session key (if it is assigned a value at all) is assigned a value chosen randomly from $\mathbb{G}$. If *msg-in* is previously-used, the simulator checks for the Send_1 query after which *msg-in* was output (note that this query will be unique, since values β used by the Send_1 oracle do not repeat). Say this query was $\mathsf{Send}_1(\textit{Server}, j, \textit{msg-in}')$. If *msg-in*$'$ is new and appears valid, the Send_2 query is answered as before. If *msg-in*$'$ is new and appears invalid, the session key (if it is assigned a value at all) is assigned a value chosen randomly from $\mathbb{G}$. Queries $\mathsf{Send}_3(\textit{Server}, j, \textit{msg-in})$ are also handled differently (assuming query $\mathsf{Send}_1(\textit{Server}, j, \textit{first-msg-in})$ was made before any $\mathsf{Corrupt}$ queries). If *first-msg-in* is previously-used, $\mathsf{Vrfy}_{\mathsf{VK}}(\beta | K, \mathsf{Sig}) = 1$, and there does not exist an i such that $\mathsf{sid}^i_{Client} = \mathsf{sid}^j_{Server}$, the session key is assigned a value chosen randomly from $\mathbb{G}$.

Claim 3.14 $\mathsf{fsAdv}_{A,P_7}(k) = \mathsf{fsAdv}_{A,P_6}(k)$.

The proof exactly follows that of Claim 3.8. □

Let $\mathsf{fsSucc1}$ denote the event that the adversary succeeds by receiving a value ∇ following a Test or Reveal query (note that this event can only occur before any $\mathsf{Corrupt}$ queries have been made). We have

$$\Pr_{A,P_7}[\mathsf{fsSucc}] = \Pr_{A,P_7}[\mathsf{fsSucc1}] + \Pr_{A,P_7}[\mathsf{fsSucc}|\overline{\mathsf{fsSucc1}}] \cdot \Pr_{A,P_7}[\overline{\mathsf{fsSucc1}}].$$

Event $\mathsf{fsSucc} \wedge \overline{\mathsf{fsSucc1}}$ can occur in one of two ways (by definition of fs-freshness): either (1) the adversary queries $\mathsf{Test}(U,i)$ before any $\mathsf{Corrupt}$ queries and does not receive ∇ in return; or (2) the adversary queries $\mathsf{Test}(U,i)$ after a $\mathsf{Corrupt}$ query and $\mathsf{acc}_U^i = \text{TRUE}$ but the adversary has never queried $\mathsf{Send}_n(U,i,M)$ for any n, M. In either case, sk_U^i is randomly chosen from $\mathbb{G}$ independent of the remainder of the experiment; therefore (assuming $\Pr_{A,P_7}[\overline{\mathsf{fsSucc1}}] \neq 0$) we must have $\Pr_{A,P_7}[\mathsf{fsSucc}|\overline{\mathsf{fsSucc1}}] = 1/2$. In other words

$$\Pr_{A,P_7}[\mathsf{fsSucc}] = 1/2 + 1/2 \cdot \Pr_{A,P_7}[\mathsf{fsSucc1}],$$

where we assume $\Pr_{A,P_7}[\overline{\mathsf{fsSucc1}}] \neq 0$. Below, we give an upper bound on $\Pr_{A,P_7}[\mathsf{fsSucc1}]$ which, in particular, will show that $\Pr_{A,P_7}[\overline{\mathsf{fsSucc1}}] \neq 0$.

We define experiment P_7' in which the adversary succeeds only if it receives a value ∇ in response to a Reveal or Test query. Clearly $\Pr_{A,P_7'}[\mathsf{fsSucc}] = \Pr_{A,P_7}[\mathsf{fsSucc1}]$ by definition of event $\mathsf{fsSucc1}$. Since the adversary cannot succeed in experiment P_7' once a $\mathsf{Corrupt}$ query is made, the simulator aborts the experiment if this is ever the case. For this reason, whenever the simulator would previously store values (∇, sk) as a "session key" (with sk being returned only in response to a Test or Reveal query *after* a $\mathsf{Corrupt}$ query), the simulator now need store only ∇.

In experiment P_8, queries to the Send_1 oracle are handled differently. Now, the simulator always computes I as $h^r g_1^{N+1}$ (i.e., even when *msg-in* is new and valid). That $|\mathsf{fsAdv}_{A,P_8}(k) - \mathsf{fsAdv}_{A,P_7'}(k)| \leq \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$ (under the DDH assumption) follows from a proof similar to that of Claim 3.7 (see also Claim 3.13). A key point is that when *msg-in* is new and appears valid, the simulator no longer need worry about simulating the adversary's view following a $\mathsf{Corrupt}$ query.

In experiment P_9, queries to the Send_0 are handled differently. Now, the simulator always computes C as $h^r g_1^{N+1}$. That $|\mathsf{fsAdv}_{A,P_9}(k) - \mathsf{fsAdv}_{A,P_8}(k)| \leq \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$ (under the DDH assumption) follows from a proof similar to that of Claim 3.9. In particular, since session keys computed during a Send_2 query are always either ∇ or are chosen randomly from $\mathbb{G}$, the value r is not needed by the simulator and hence we can use the simulator to break the Cramer-Shoup encryption scheme as in the proofs of Claims 3.7, 3.9, and 3.13.

The adversary's view in experiment P_9 is independent of the passwords chosen by the simulator until the adversary receives ∇ in response to a Reveal or Test query (in which case the adversary

succeeds). Thus, the probability that the adversary receives ∇ (which is exactly $\Pr_{A,P_9}[\mathsf{fsSucc}]$) is at most $Q(k)/N$. When $Q(k) < N$ (which is the only interesting case), we have $\Pr_{A,P_7}[\mathsf{fsSucc1}] < 1$ for k large enough. Therefore, for large enough values of k and for some negligible function $\varepsilon(\cdot)$, we have

$$\begin{aligned}
\Pr_{A,P_0}[\mathsf{fsSucc}] &\leq \Pr_{A,P_7}[\mathsf{fsSucc}] + \varepsilon(k) \\
&\leq 1/2 + 1/2 \cdot \Pr_{A,P_7}[\mathsf{fsSucc1}] + \varepsilon(k) \\
&\leq 1/2 + \frac{Q(k)}{2N} + \varepsilon(k)
\end{aligned}$$

and $\mathsf{fsAdv}_{A,P_0}(k) \leq Q(k)/N + 2 \cdot \varepsilon(k)$. In other words, the original P_0 is a forward-secure, password-only, key-exchange protocol. ■

Chapter 4

Non-Interactive and Non-Malleable Commitment

4.1 Introduction

Consider a setting in which two parties must each choose a course of action (by declaring their choice to a receiver), yet neither party should have the advantage of moving second and thereby basing their choice on the other player's selection. How can this be achieved? There are several natural solutions to this problem. One is to force both players to announce their choices to the receiver at exactly the same time. Clearly, this will achieve the desired result if absolute simultaneity can be achieved; in practice, however, guaranteeing this level of synchrony is difficult if not impossible. Another suggestion is to involve an "escrow agent" to whom each party communicates his choice. This escrow agent should not reveal either player's choice to the receiver until both players have communicated with him. Although this approach accomplishes the desired task, it involves an additional party who must be highly trusted by all others involved. This solution also places a heavy burden on the agent in case many parties want to use his services at the same time.

A particularly simple and elegant solution is to have both parties "commit" to their choice such that the following holds: (1) a commitment should reveal no information about the choice being committed to, yet (2) once a commitment is made, the committing party should be unable to change their committed value. A straightforward way to achieve this is to have a committing party write their choice on a piece of paper, seal it inside an envelope, and send the envelope to the receiver. This approach satisfies both requirements given above: the second party (even if he observes all communication between the other party and the receiver) sees only the envelope and gets no information about the contents inside; furthermore, the first party cannot change what is inside the envelope once it is sealed.

The preceding solution works when the parties and the receiver are physically close (and can

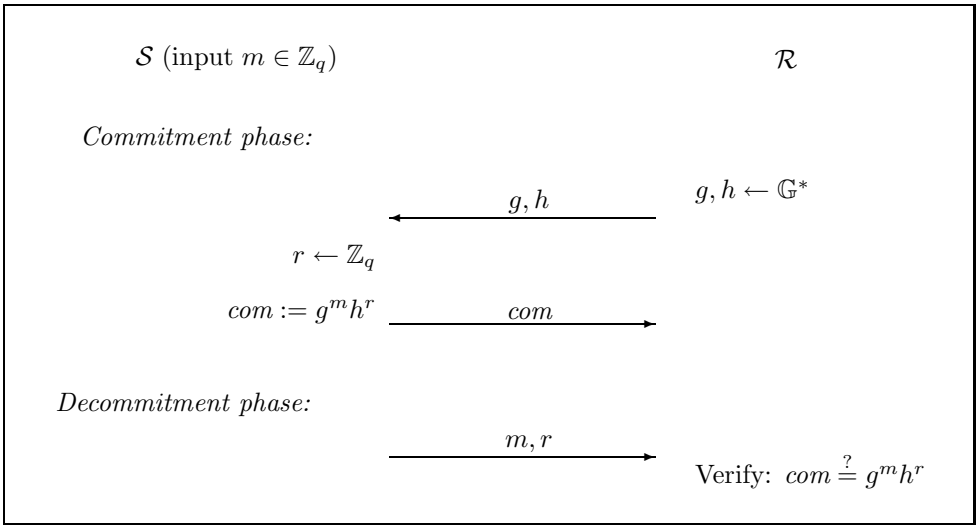

Figure 4.1: The Pedersen commitment scheme.

therefore quickly pass an envelope back and forth), but is not feasible in many other situations. A *commitment protocol* may be viewed as the cryptographic implementation of a secure envelope. Here, we can guarantee that the committed value is not revealed (*secrecy*) and that the committing party cannot change his mind (*binding*) under assumptions about the computational power of the parties.

An example will be instructive. In Figure 4.1, we illustrate the Pedersen commitment scheme [103] whose security is based on the hardness of computing discrete logarithms in group $\mathbb{G}$ of prime order q. First, the receiver $\mathcal{R}$ chooses two random generators $g, h \in \mathbb{G}$ and sends these to the sender $\mathcal{S}$. To commit to a message $m \in \mathbb{Z}_q$, $\mathcal{S}$ chooses a random value $r \in \mathbb{Z}_q$, computes $com = g^m h^r$, and sends com to $\mathcal{R}$. To decommit, the sender simply reveals m, r. Note that com is uniformly distributed in $\mathbb{G}$, independently of m, and therefore the commitment reveals no information about the committed message. On the other hand, assuming the hardness of computing discrete logarithms in $\mathbb{G}$, the Pedersen scheme is binding. To see this, note that given two legal decommitments $\langle m, r \rangle$ and $\langle m', r' \rangle$ to com, we have $g^m h^r = g^{m'} h^{r'}$. Thus, we may compute $g^{m-m'} = h^{r'-r}$ and $\log_g h = (m - m')/(r' - r) \bmod q$.

Two types of commitment schemes are primarily considered in the literature: perfectly-binding and perfectly-hiding (following [62] we refer to the former as *standard* and the latter as *perfect*). In a standard commitment scheme, each commitment is information-theoretically linked to only one possible (legal) decommitment value; on the other hand, the secrecy of the commitment is guaranteed only with respect to a computationally-bounded receiver. In a perfect commitment scheme, the secrecy of the commitment is information-theoretic while the binding property guarantees only that

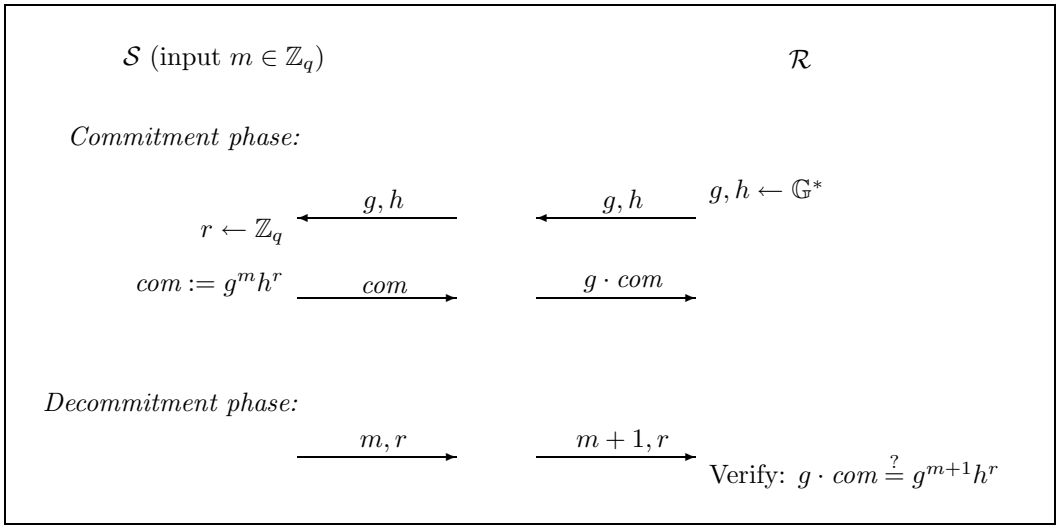

Figure 4.2: Man-in-the-middle attack on the Pedersen commitment scheme.

a computationally-bounded sender cannot find a commitment which can be opened in two possible ways. The type of commitment scheme to be used depends on the application; it may also depend on assumptions regarding the computational power of the participants. Furthermore, in some protocols (e.g., zero-knowledge proofs) certain commitments are never opened; information-theoretic privacy ensures that the committed data will remain hidden indefinitely.

Commitment protocols have become one of the most fundamental cryptographic primitives, and are used as sub-protocols in such applications as zero-knowledge proofs [67, 62], secure multi-party computation [66], and many others. Commitment protocols can also be used directly, for example, in remote (electronic) bidding. In this setting, parties bid by committing to a value; once bidding is complete, parties reveal their bids by decommitting.

In many situations, however, the secrecy and binding properties outlined previously do not fully capture everything one might expect from a secure envelope. For example, we do not expect a party to be able to commit to a value which is related in any way to a previously committed (and as yet unopened) value. Schemes in which commitment to a related value is possible are called *malleable*; schemes in which this is impossible are called *non-malleable*. As an illustration, in the bidding scenario it is unacceptable if one party can generate a valid commitment to $m + 1$ upon viewing a commitment to m. Note that the value of the original commitment may remain unknown (and thus secrecy need not be violated); in fact, the second party may only be able to decommit his bid after viewing a decommitment of the first.

Unfortunately, most known commitment protocols are easily susceptible to these types of man-in-the-middle attacks. For example, Figure 4.2 demonstrates the malleability of the Pedersen com-

mitment scheme. Here, the adversary changes the commitment *com* of $\mathcal{S}$ to $com' = g \cdot com$. At this point, the adversary has no idea what $\mathcal{S}$ has committed to, nor what he himself has committed to. In fact, the adversary will only be able to decommit after viewing the decommitment of $\mathcal{S}$. When $\mathcal{S}$ decommits to m, the adversary simply decommits to $m' = m + 1$. The receiver cannot even tell that a man-in-the-middle attack is taking place.[1]

One natural measure of efficiency for a commitment protocol is the communication complexity. The number of rounds is one measure of the communication complexity. Optimally, the commitment phase of a commitment protocol should consist of a single message from the sender to the receiver; such schemes are termed *non-interactive*. A second measure of the communication complexity is the bit complexity (i.e., size) of a commitment. This is particularly important when committing to a very large message such as the contents of a (large) database. Unfortunately, standard commitment schemes (even malleable ones) require commitment size at least $|M| + \omega(\log k)$, where $|M|$ is the message size and k is the security parameter. Perfect commitment schemes, on the other hand, offer the opportunity to achieve much shorter commitment lengths.

4.1.1 Previous Work

The commitment primitive has been extensively studied. Standard commitment has been shown to exist if and only if one-way functions exist [96, 77]. A perfect commitment scheme has been constructed assuming the existence of one-way permutations [97]. Both schemes have been designed in the interactive model (where no public information is available to the parties); the former, however, may be adapted to run in the public-parameters model (cf. Section 2.1.1). Efficient perfect commitment protocols, based on specific number-theoretic assumptions, are also known [103, 100].

Non-malleability of commitments was first explicitly considered by Dolev, Dwork, and Naor [44]. They also provided the first construction of a standard commitment scheme which is provably non-malleable. Although their protocol is constructed from the minimal assumption of a one-way function (in particular, without assuming any public parameters), it requires a poly-logarithmic number of rounds of interaction.[2] Assuming a public random string available to all participants, Di Crescenzo, Ishai, and Ostrovsky [40] construct the first *non-interactive*, non-malleable standard commitment scheme. Interestingly, their construction can be modified to give a non-interactive, non-malleable *perfect* commitment scheme. Unfortunately, the resulting commitments are large (i.e., $\mathcal{O}(|M| \cdot k)$), thus motivating the search for more efficient protocols. Furthermore, their protocol is

[1] In the present example, a receiver who obtains commitments and decommitments from *both* $\mathcal{S}$ and the adversary can tell that something unusual happened since the decommitments are correlated (for example, they both use the same r). However, it is possible for an adversary to prevent this detection by re-randomizing his commitment.

[2] Furthermore, their protocol allows an adversary to generate a different commitment to an identical value (unless user identities are assumed). The protocols we present do not have this drawback.

not computationally practical.

Efficient non-malleable commitment schemes, based on stronger (but standard) assumptions, have been given by Fischlin and Fischlin [58]. Like the construction of [40], these protocols require publicly-available parameters generated by a trusted party (in some cases this can be weakened to the assumption of a public random string). They describe non-malleable perfect commitment schemes based on either the discrete logarithm or RSA assumptions. Though efficient, these protocols require interaction between the sender and receiver.

Subsequent to the present work, a definition of "universally composable" commitment protocols (which implies, in particular, non-malleability) was introduced, and a provably-secure construction satisfying this definition was given in the common-random-string model [28]. Unfortunately, the given protocol is inefficient for commitment to more than a single bit. Other subsequent work, building on the paradigm described here, has demonstrated an efficient non-malleable commitment scheme based on the factoring assumption [59].

4.1.2 Our Contributions

We present the first efficient (in both computation and communication) constructions of non-interactive, non-malleable, *perfect* commitment schemes. We work in the setting in which public parameters are available to all participants (our discrete logarithm construction can be implemented in the public random string model using standard techniques). Previous constructions are either for the case of standard commitment [44, 40] or require interaction [44, 58]. Our constructions are based on the discrete logarithm or the RSA assumptions, and allow efficient, perfectly-hiding commitment to arbitrarily-large messages. The size of the resulting commitment is essentially optimal. The schemes described in [58], while able to handle large messages, require modifications which render them less efficient and result in statistical secrecy.

We also discuss the case of non-interactive, non-malleable, standard commitment and prove secure a construction based on trapdoor permutations that achieves commitment size $|M| + poly(k)$. The large commitment size of this construction (though near-optimal for standard commitment) serves as motivation for our consideration of perfect commitment schemes. Indeed, for arbitrarily-large messages, our perfect commitment schemes yield commitments of size $\mathcal{O}(k)$, where k is the security parameter (the commitment size is further improved in Section 4.5). All our schemes require only $poly(k)$ bits of public information, independent of the size of the committed message.

4.2 Definitions

Commitment schemes. A *non-interactive commitment scheme*[3] [40] in the public-parameters model is defined by a triple of probabilistic, polynomial-time algorithms $(\mathcal{TTP}, \mathcal{S}, \mathcal{R})$ which describe a two-phase protocol between a sender $\mathcal{S}$ and a receiver $\mathcal{R}$ such that the following is true. In the first phase (the commitment phase), the sender $\mathcal{S}$, given a public string σ output by a trusted third party $\mathcal{TTP}$, commits to a message m by computing a pair of strings (com, dec) and sending the commitment com to receiver $\mathcal{R}$. Given only σ and com, the receiver cannot determine any information about m (this is the *hiding property*). In the second phase (the decommitment phase) $\mathcal{S}$ reveals the decommitment dec to $\mathcal{R}$ and $\mathcal{R}$ checks whether the decommitment is valid. If it is not, $\mathcal{R}$ outputs a special string $\perp$, meaning that he rejects the decommitment from $\mathcal{S}$; otherwise, $\mathcal{R}$ can efficiently compute the message m and is convinced that m was indeed chosen by $\mathcal{S}$ in the first phase. It should be infeasible for $\mathcal{S}$ to generate a commitment which $\mathcal{S}$ can later decommit in more than one possible way (this is the *binding property*).

In a perfect commitment scheme, the committed message is hidden from the receiver in an information-theoretic sense. Thus, even an infinitely-powerful receiver cannot determine the message to which $\mathcal{S}$ has committed at the end of the first phase. On the other hand, the binding property only holds with respect to a computationally-bounded sender. Thus, a given commitment has possible (legal) decommitments to many messages, but a polynomial-time sender cannot explicitly find more than one such decommitment. In contrast, a standard commitment scheme prevents even an infinitely-powerful sender from decommitting a commitment in more than one possible way; however, at the end of the commitment phase the message is hidden only from a computationally-bounded receiver. Formal definitions follow:

Definition 4.1 *Let $(\mathcal{TTP}, \mathcal{S}, \mathcal{R})$ be a triple of probabilistic, polynomial time algorithms, k a security parameter, and $\{\mathcal{M}_\sigma\}_{\sigma \in \mathcal{TTP}(1^k), k \in \mathbb{N}}$ a collection of message spaces such that membership in $\mathcal{M}_\sigma$ is efficiently testable given σ output by $\mathcal{TTP}(1^k)$. We say that $(\mathcal{TTP}, \mathcal{S}, \mathcal{R})$ is a* non-interactive, perfect (resp. standard) commitment scheme over $\{\mathcal{M}_\sigma\}$ *if the following conditions hold:*

1. **(Meaningfulness)** *For all $k \in \mathbb{N}$, all σ output by $\mathcal{TTP}(1^k)$, and all $m \in \mathcal{M}_\sigma$:*

$$\Pr[(com, dec) \leftarrow \mathcal{S}(\sigma, m); m' \leftarrow \mathcal{R}(\sigma, com, dec) : m' = m] = 1.$$

2. **(Perfect (resp. computational) secrecy)** *For all computationally-unbounded (resp.* PPT*) distinguishing algorithms D, the following is identically zero (resp. negligible in k):*

$$\left|\Pr\left[\sigma \leftarrow \mathcal{TTP}(1^k); (m_0, m_1, s) \leftarrow D_1(\sigma); b \leftarrow \{0,1\};\right.\right.$$

[3] Because our constructions are non-interactive, we provide definitions for the non-interactive case only.

$$(com, dec) \leftarrow \mathcal{S}(\sigma, m_b) : D_2(com, s) = b] - 1/2|$$

3. **(Computational (resp. perfect) binding)** *For all* PPT *(resp. computationally unbounded) algorithms $\mathcal{S}'$, the following is negligible in k:*

$$\Pr\left[\sigma \leftarrow \mathcal{TTP}(1^k); (com, dec_1, dec_2) \leftarrow \mathcal{S}'(\sigma); m_1 \leftarrow \mathcal{R}(\sigma, com, dec_1); \right.$$
$$\left. m_2 \leftarrow \mathcal{R}(\sigma, com, dec_2) : m_1 \neq \perp \ \wedge m_2 \neq \perp \ \wedge m_1 \neq m_2\right].$$

NOTE. The symbol $\perp$ is reserved for an invalid decommitment (which includes a refusal to decommit). In particular, for all σ we have $\perp \notin \mathcal{M}_\sigma$.

Non-malleable commitment schemes. Two definitions of non-malleable commitment have appeared in the literature, both seeking to capture the following intuition of security: for any adversary who, after viewing a commitment to m, produces a commitment to a value m' which bears some relation to m, there exists a simulator performing at least as well in producing an m' related to m but *without viewing a commitment to* m (and thus having no information about the value of m). The difference between the two definitions lies in what it means for an adversary to"produce a commitment". In the original definition [44] (*non-malleability with respect to commitment*), generating a valid commitment to m' is sufficient. However, this definition does not apply to perfectly-hiding commitment schemes since for such schemes the value committed to by a commitment string is not well-defined. In the definition of [40, 58] (*non-malleability with respect to opening*), the adversary must also give a valid decommitment to m' after viewing the decommitment to m. Note that in the case of standard commitment, non-malleability with respect to commitment is a stronger notion of security.

Definition 4.2 *A non-interactive commitment scheme $(\mathcal{TTP}, \mathcal{S}, \mathcal{R})$ over $\{\mathcal{M}_\sigma\}$ is* ε-non-malleable with respect to opening *if, for all $\varepsilon > 0$ and every* PPT *algorithm $\mathcal{A}$, there exists a simulator $\mathcal{A}'$ running in probabilistic, poly$(k, 1/\varepsilon)$ time, such that for all polynomial-time computable, valid relations R (see note below), we have:*

$$\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) - \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k) \leq \varepsilon + negl(k),$$

for some negligible function negl$(\cdot)$; where:

$$\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) \stackrel{\mathrm{def}}{=}$$
$$\Pr\ \left[\sigma \leftarrow \mathcal{TTP}(1^k); (\mathcal{D}, s) \leftarrow \mathcal{A}_0(\sigma); m_1 \leftarrow \mathcal{D}; (com_1, dec_1) \leftarrow \mathcal{S}(\sigma, m_1); \right.$$
$$(com_2, s') \leftarrow \mathcal{A}_1(\sigma, com_1, s); dec_2 \leftarrow \mathcal{A}_2(\sigma, com_1, dec_1, s');$$
$$\left. m_2 \leftarrow \mathcal{R}(\sigma, com_2, dec_2) : com_1 \neq com_2 \ \wedge \ R(\sigma, \mathcal{D}, m_1, m_2) = 1\right]$$

and:

$$\widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k) \stackrel{\text{def}}{=}$$
$$\Pr\left[(\sigma, \mathcal{D}, m_2) \leftarrow \mathcal{A}'(1^k); m_1 \leftarrow \mathcal{D} : R(\sigma, \mathcal{D}, m_1, m_2) = 1\right].$$

($\mathcal{D}$ represents an efficiently sampleable distribution over $\mathcal{M}_\sigma$.)

NOTE. The definition of security above allows the simulator to do *arbitrarily better than* the adversary. The technical reason for this is that the adversary may simply refuse to decommit, even when it would have otherwise succeeded. In any case, if a simulator who has no information about m_1 can do *better* than an adversary who gets to see a commitment to m_1, the scheme still satisfies our intuitive notion of non-malleability.

VALID RELATIONS. Typically, we view a relation R as being defined on pairs of messages. However, there are a number of subtleties which need to be addressed. First, the adversary $\mathcal{A}$ should be allowed to choose the distribution $\mathcal{D}$, possibly depending upon the public parameters σ. Any reasonable definition of security thus requires the simulator to output a distribution as well. The simulator, however, must be prevented from choosing some "trivial" distribution for which it can always succeed. To handle this, we allow the relation R to take the public parameters σ and (a description of) a distribution $\mathcal{D}$ as additional parameters. To ensure that the distribution is defined with respect to the correct message space, we require that a valid relation satisfy $R(\sigma, \mathcal{D}, *, *) = 0$ if $\mathcal{D}$ is not a distribution over $\mathcal{M}_\sigma$. Furthermore, we require $R(\sigma, *, m_1, m_2) = 0$ if $m_1 \notin \mathcal{M}_\sigma$ or $m_2 \notin \mathcal{M}_\sigma$. In particular, this implies that $R(\sigma, \mathcal{D}, m, \perp) = R(\sigma, \mathcal{D}, \perp, m) = 0$ for all $\sigma, \mathcal{D}, m$.

We now turn to the definition of non-malleability with respect to commitment. For any standard commitment scheme, we may define the function $\mathsf{Decommit}(\sigma, \cdot)$ which takes a commitment as its second argument and returns the (unique) committed value in $\mathcal{M}_\sigma$ (or $\perp$, if the commitment is invalid). The perfect binding of the scheme implies that the function is well-defined (although not polynomial-time computable) except with negligible probability over choice of σ.

Definition 4.3 *A non-interactive, standard commitment scheme $(\mathcal{TTP}, \mathcal{S}, \mathcal{R})$ over message space $\{\mathcal{M}_\sigma\}$ is* non-malleable with respect to commitment *if, for every* PPT *algorithm $\mathcal{A}$, there exists a* PPT *simulator $\mathcal{A}'$, such that for all polynomial-time, valid relations R (see note above), we have:*

$$\left|\mathsf{Succ}^{\mathrm{NM-commit}}_{\mathcal{A},R}(k) - \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k)\right| \leq negl(k),$$

for some negligible function $negl(\cdot)$; where:

$$\mathsf{Succ}^{\mathrm{NM-commit}}_{\mathcal{A},R}(k) \stackrel{\text{def}}{=}$$

$$\Pr \quad \big[\sigma \leftarrow \mathcal{TTP}(1^k); (\mathcal{D}, s) \leftarrow \mathcal{A}_0(\sigma); m_1 \leftarrow \mathcal{D};$$
$$(com_1, dec_1) \leftarrow \mathcal{S}(\sigma, m_1); com_2 \leftarrow \mathcal{A}(\sigma, com_1, s);$$
$$m_2 = \mathsf{Decommit}(\sigma, com_2) : com_1 \neq com_2 \ \wedge \ R(\sigma, \mathcal{D}, m_1, m_2) = 1\big]$$

and $\widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k)$ *is defined as above.*

Equivocable commitment schemes. Our constructions of non-malleable, perfect commitment schemes use *equivocable commitment schemes* [2] as a building block; such schemes have been used previously in designing non-malleable commitment protocols [40]. Informally, an equivocable commitment scheme in the public-parameter model is one for which there exists an efficient equivocation algorithm Equiv, substituting for the trusted third party, which outputs public parameters σ and a commitment such that: (a) the distribution of σ, the commitment, and a decommitment to any message is exactly equivalent to their distribution in a real execution of the protocol; and (b) the commitment can be opened by Equiv in more than one possible way. We give a formal definition for the case of perfect commitment.

Definition 4.4 *A non-interactive, perfect commitment scheme* $(\mathcal{TTP}, \mathcal{S}, \mathcal{R})$ *over message space* $\{\mathcal{M}_\sigma\}$ *is* perfectly equivocable *if there exists a probabilistic, polynomial time* equivocable commitment generator Equiv *such that:*

1. $\mathsf{Equiv}_1(1^k)$ *outputs* (σ, com, s) *(where* s *represents state information).*
2. *For all* $k \in \mathbb{N}$*, the following distributions are equivalent:*

$$\{\sigma \leftarrow \mathcal{TTP}(1^k); m \leftarrow \mathcal{M}_\sigma; (com, dec) \leftarrow \mathcal{S}(\sigma, m) : (\sigma, com, dec, m)\}$$

$$\{(\sigma, com, s) \leftarrow \mathsf{Equiv}_1(1^k); m \leftarrow \mathcal{M}_\sigma; dec \leftarrow \mathsf{Equiv}_2(s, m) : (\sigma, com, dec, m)\}.$$

In particular, for all $k \in \mathbb{N}$, all (σ, com, s) output by $\mathsf{Equiv}_1(1^k)$, all $m \in \mathcal{M}_\sigma$, and all dec output by $\mathsf{Equiv}_2(s, m)$ we have $\mathcal{R}(\sigma, com, dec) = m$.

4.3 Non-Malleable Standard Commitment

We first examine the case of non-interactive, standard commitment. Note that the size of a standard commitment (even for malleable schemes) must be at least $|M| + \omega(\log k)$, where $|M|$ is the message length and k is the security parameter. Perfect binding implies that the size must be at least $|M|$, and semantic security requires that each message have $2^{\omega(\log k)} = \omega(\text{poly}(k))$ possible commitments associated with it.

The theorem below indicates that we can achieve roughly this bound for non-interactive, *non-malleable* standard commitment, assuming the existence of trapdoor permutations[4] (in the model with public parameters). The key realization is that a non-malleable public-key encryption scheme can also be used as a non-malleable standard commitment scheme. This connection between non-malleable public-key encryption and non-malleable commitment seems not to have been noticed before. Following Blum and Goldwasser [21, 68] (who consider the case of semantic security for public-key encryption), we construct a communication-efficient, non-malleable standard commitment scheme from the following components: first, we use a public-key encryption scheme $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ that is indistinguishable under an adaptive chosen-ciphertext attack (and hence non-malleable) [44, 6, 16]. Such a scheme can be based on any family of trapdoor permutations [44, 110, 38]. Next, we use a symmetric-key cryptosystem $(\mathcal{K}, \mathcal{E}^*, \mathcal{D}^*)$ which is indistinguishable under adaptive chosen-ciphertext attack; this can be based on any one-way function [44]. The commitment scheme works as follows: public parameters σ consist of a public key pk for the public-key cryptosystem. Commitment is done by choosing a random secret key K for the symmetric-key cryptosystem, encrypting K using the public-key cryptosystem, and then encrypting the committed message using the symmetric-key cryptosystem and key K. A commitment to M is thus

$$\mathcal{E}_{pk}(K) \circ \mathcal{E}^*_K(M). \tag{4.1}$$

Decommitment consists of revealing M and the random bits used to form the commitment. Commitment verification is done in the obvious way. Note that the commitment is actually a public-key encryption of the message M, although the receiver does not have the associated secret key (indeed, no party has it because σ was generated by $\mathcal{TTP}$). Furthermore, $|\sigma| = |pk| = poly(k)$ independent of the size of the committed message. We now prove the following:

Theorem 4.1 *Construction (4.1) is a non-interactive, standard commitment scheme (in the public-parameters model) that is non-malleable with respect to commitment and has commitment size $|M| + poly(k)$, where $|M|$ is the size of the committed message and k is the security parameter. Furthermore, (4.1) may be based on the existence of trapdoor permutations.*

Proof First note that for (4.1) to be perfectly binding we require that the decryption algorithms for both the public-key and symmetric-key cryptosystems have zero probability of decryption error. This is achieved, in particular, by the public-key and symmetric-key cryptosystems given in [44], which may be based on any family of trapdoor permutations. Thus, revealing the randomness used to generate the commitment perfectly binds the sender to the message.

[4]Recall that [40] achieves non-interactive, non-malleable, standard commitment assuming the existence of one-way functions. However, their scheme requires commitment size $\mathcal{O}(k \cdot |M|)$.

A proof of non-malleability with respect to commitment will immediately imply that the scheme is computationally hiding (this has been noted previously for the case of encryption [6, 16] and it is clear that a similar result holds for the case of commitment). Since any commitment to M using the scheme outlined above may also be viewed as an encryption of M under public key $pk = \sigma$, if we can prove that (4.1) constitutes a non-malleable public-key *encryption* scheme, we are done. Using the results of [6], it suffices to prove that (4.1) is secure under adaptive chosen-ciphertext attack.

Consider an adversary $\mathcal{A}$ attacking construction (4.1) under an adaptive chosen-ciphertext attack. Define adversary $\mathcal{B}$ using $\mathcal{A}$ as a black box to attack $\mathcal{E}$ under an adaptive chosen-ciphertext attack as follows ($\widetilde{\mathcal{D}}_{sk}(\cdot)$ denotes the decryption oracle for hybrid scheme (4.1)):

Algorithm $\mathcal{B}_1^{\mathcal{D}_{sk}(\cdot)}(pk)$ $(M_0, M_1, s) \leftarrow \mathcal{A}_1^{\widetilde{\mathcal{D}}_{sk}(\cdot)}(pk)$ $K \leftarrow \mathcal{K}(1^k)$ $M_0' := K; M_1' := 0^k$ return $(M_0', M_1', (K, M_0, M_1, s))$	Algorithm $\mathcal{B}_2^{\mathcal{D}_{sk}(\cdot)}(y, (K, M_0, M_1, s))$ $b \leftarrow \{0,1\}$ $C \leftarrow \mathcal{E}_K^*(M_b)$ $b' \leftarrow \mathcal{A}_2^{\widetilde{\mathcal{D}}_{sk,y\to K}(\cdot)}(y \circ C, s)$ if $b' = b$ return 0 else return 1

Decryption oracle queries of $\mathcal{A}_1$ are handled by $\mathcal{B}_1$ in the obvious way. The notation $\widetilde{\mathcal{D}}_{sk,y\to K}(\cdot)$ means that decryption oracle queries of $\mathcal{A}_2$ are handled by $\mathcal{B}_2$ as follows: if $\mathcal{A}_2$ requests decryption of ciphertext $y' \circ C'$, with $y' \neq y$, $\mathcal{B}_2$ submits y' to its decryption oracle, receives key K' in return, then computes $M' := \mathcal{D}_{K'}^*(C')$ and returns this answer to $\mathcal{A}_2$. If $\mathcal{A}_2$ submits ciphertext $y \circ C'$, note that $\mathcal{B}_2$ would *not* be allowed to submit y to its decryption oracle since $\mathcal{B}_2$ cannot ask for decryption of the challenge ciphertext. Instead, $\mathcal{B}_2$ "assumes" that y is an encryption of K, and computes the response $M := \mathcal{D}_K^*(C')$. Adaptive chosen-ciphertext security of $\mathcal{E}$ implies that the advantage of $\mathcal{B}$ is negligible; formally:

$$\begin{aligned}&\mathsf{Adv}_{\mathcal{B},\mathcal{E}}(k) \stackrel{\text{def}}{=} \\ &\quad \left| 2 \cdot \Pr\left[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (M_0, M_1, s) \leftarrow \mathcal{B}_1^{\mathcal{D}_{sk}(\cdot)}(pk); b \leftarrow \{0,1\}; \right.\right. \\ &\qquad \left.\left. y \leftarrow \mathcal{E}_{pk}(M_b) : \mathcal{B}_2^{\mathcal{D}_{sk}(\cdot)}(y, s) = b \right] - 1 \right| \leq \varepsilon_1(k), \end{aligned} \tag{4.2}$$

where Gen is the algorithm which generates public and private keys for $\mathcal{E}$ and $\varepsilon_1(\cdot)$ is a negligible function.

We now consider adversary $\mathcal{C}$ using $\mathcal{A}$ as a black box to attack $\mathcal{E}^*$ under an adaptive chosen-ciphertext attack.

Algorithm $\mathcal{C}_1^{\mathcal{D}_K^*(\cdot)}(1^k)$ $(pk, sk) \leftarrow \mathsf{Gen}(1^k)$ $(M_0, M_1, s) \leftarrow \mathcal{A}_1^{\widetilde{\mathcal{D}}_{sk}(\cdot)}(pk)$ $y \leftarrow \mathcal{E}_{pk}(0^k)$ return $(M_0, M_1, (y, sk, s))$	Algorithm $\mathcal{C}_2^{\mathcal{D}_K^*(\cdot)}(C, (y, sk, s))$ $b' \leftarrow \mathcal{A}_2^{\widetilde{\mathcal{D}}_{sk,y\to K}(\cdot)}(y \circ C, s)$ return b'

Decryption oracle queries of $\mathcal{A}_1$ are answered by $\mathcal{C}_1$ in the obvious way. The notation $\widetilde{\mathcal{D}}_{sk,y\to K}(\cdot)$, as before, means that decryption oracle queries of $\mathcal{A}_2$ are handled by $\mathcal{C}_2$ as follows: if $\mathcal{A}_2$ submits ciphertext $y' \circ C'$ to its decryption oracle with $y' \neq y$, $\mathcal{C}_2$ decrypts y' using key sk to get K' and then returns $M' := \mathcal{D}^*_{K'}(C')$ to $\mathcal{A}_2$. On the other hand, if $\mathcal{A}_2$ submits ciphertext $y \circ C'$, then $\mathcal{C}_2$ submits C' to its decryption oracle $\mathcal{D}^*_K(\cdot)$ and returns the result to $\mathcal{A}_2$ (we emphasize that $\mathcal{C}$ does not know the value of K). Adaptive chosen-ciphertext security of $\mathcal{E}^*$ implies that the advantage of $\mathcal{C}$ is negligible; formally:

$$\begin{aligned}&\mathsf{Adv}_{\mathcal{C},\mathcal{E}^*}(k) \stackrel{\text{def}}{=} \\ &\quad\Big|2 \cdot \Pr\Big[K \leftarrow \mathcal{K}(1^k); (M_0, M_1, s) \leftarrow \mathcal{C}_1^{\mathcal{D}^*_K(\cdot)}(1^k); b \leftarrow \{0,1\}; \\ &\qquad C \leftarrow \mathcal{E}^*_K(M_b) : \mathcal{C}_2^{\mathcal{D}^*_K(\cdot)}(C, s) = b\Big] - 1\Big| \leq \varepsilon_2(k)\end{aligned} \tag{4.3}$$

for some negligible function $\varepsilon_2(\cdot)$.

Define the following probabilities, parameterized by the security parameter k:

$$\begin{aligned}&p_{b,\mathsf{real}}(k) \stackrel{\text{def}}{=} \\ &\quad\Pr\Big[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (M_0, M_1, s) \leftarrow \mathcal{A}_1^{\widetilde{\mathcal{D}}_{sk}(\cdot)}(pk); K \leftarrow \mathcal{K}(1^k); \\ &\qquad y \leftarrow \mathcal{E}_{pk}(K); C \leftarrow \mathcal{E}_K(M_b) : \mathcal{A}_2^{\widetilde{\mathcal{D}}_{sk}(\cdot)}(y \circ C, s) = 0\Big] \\ &p_{b,\mathsf{fake}}(k) \stackrel{\text{def}}{=} \\ &\quad\Pr\Big[(pk, sk) \leftarrow \mathsf{Gen}(1^k); (M_0, M_1, s) \leftarrow \mathcal{A}_1^{\widetilde{\mathcal{D}}_{sk}(\cdot)}(pk); K \leftarrow \mathcal{K}(1^k); \\ &\qquad y \leftarrow \mathcal{E}_{pk}(0^k); C \leftarrow \mathcal{E}_K(M_b) : \mathcal{A}_2^{\widetilde{\mathcal{D}}_{sk,y\to K}(\cdot)}(y \circ C, s) = 0\Big].\end{aligned}$$

The crux of the proof is to note that when $\mathcal{D}_{sk}(y) = K$, oracle $\widetilde{\mathcal{D}}_{sk,y\to K}(\cdot)$ is equivalent to the real decryption oracle $\widetilde{\mathcal{D}}_{sk}(\cdot)$ for the hybrid encryption scheme. With this in mind, we may re-write (4.2) and (4.3) as $\mathsf{Adv}_{\mathcal{B},\mathcal{E}}(k) = 1/2 \cdot |p_{0,\mathsf{real}}(k) - p_{0,\mathsf{fake}}(k) + p_{1,\mathsf{fake}}(k) - p_{1,\mathsf{real}}(k)|$ and $\mathsf{Adv}_{\mathcal{C},\mathcal{E}^*}(k) = |p_{0,\mathsf{fake}}(k) - p_{1,\mathsf{fake}}(k)|$. Then:

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}(k) &\stackrel{\text{def}}{=} |p_{0,\mathsf{real}}(k) - p_{1,\mathsf{real}}(k)| \\ &= |p_{0,\mathsf{real}}(k) - p_{1,\mathsf{real}}(k) + p_{0,\mathsf{fake}}(k) - p_{1,\mathsf{fake}}(k) - p_{0,\mathsf{fake}}(k) + p_{1,\mathsf{fake}}(k)| \\ &\leq |p_{0,\mathsf{real}}(k) - p_{1,\mathsf{real}}(k) - p_{0,\mathsf{fake}}(k) + p_{1,\mathsf{fake}}(k)| + |p_{0,\mathsf{fake}}(k) - p_{1,\mathsf{fake}}(k)| \\ &= 2 \cdot \mathsf{Adv}_{\mathcal{B},\mathcal{E}}(k) + \mathsf{Adv}_{\mathcal{C},\mathcal{E}^*}(k) \\ &\leq 2 \cdot \varepsilon_1(k) + \varepsilon_2(k),\end{aligned}$$

and this last quantity is negligible.

To complete the proof, we note that $|K| = \mathit{poly}(k)$ and therefore $|\mathcal{E}_{pk}(K)| = \mathit{poly}(k)$. Furthermore, we can achieve $|\mathcal{E}^*_K(M)| = |M| + k$ using a stream cipher and a secure MAC (see [44]). The

total length of the commitment given by (4.1) is then $|M| + poly(k)$. ■

This theorem immediately implies the security (under the decisional Diffie-Hellman assumption) of construction (4.1) when using the efficient public-key cryptosystem of [36] and any adaptive chosen-ciphertext-secure symmetric-key cryptosystem $(\mathcal{K}, \mathcal{E}^*, \mathcal{D}^*)$. We note that the security requirements for the public- and private-key encryption schemes can be relaxed: $(\mathsf{Gen}, \mathcal{E}, \mathcal{D})$ is only required to be non-malleable under a chosen-plaintext attack (i.e., secure in the sense of NM-CPA) and $(\mathcal{K}, \mathcal{E}^*, \mathcal{D}^*)$ need only be indistinguishable under a P0 plaintext attack and an adaptive chosen-ciphertext attack (i.e., secure in the sense of IND-P0-C2); see [6, 84] for formal definitions. This is so because (4.1) is a non-malleable commitment scheme whenever it is secure in the sense of NM-CPA when viewed as a public-key encryption scheme (in the proof above, we show that (4.1) is secure in the sense of NM-CCA2). This allows for much greater efficiency since NM-CPA public-key cryptosystems can be constructed more efficiently than IND-CCA2 schemes [47] and IND-P0-C2 symmetric-key schemes may be deterministic.

We remark that the result in the theorem applies in the public random string model when chosen-ciphertext-secure *dense* [39] public-key encryption schemes are used.

4.4 Non-Malleable Perfect Commitment

The computationally-hiding commitment scheme presented above achieves commitment size $|M| + poly(k)$. This cannot be improved very much, since computationally-hiding commitments have size at least $|M|$. In this section (see also Section 4.5) we present perfectly-hiding commitment schemes that improve significantly on the commitment length, achieving commitment size $\mathcal{O}(k)$ for arbitrarily-large messages.

Both of our perfectly-hiding commitment schemes build on the paradigm established in [40], with modifications which substantially improve the efficiency. A commitment consists of three components $\langle A, B, Tag \rangle$. The first component A is a commitment to a random key r_1 for a one-time message authentication code (MAC). The second component B contains the actual commitment to the message m, using public parameters which depend upon the first component A. Finally, $Tag = \text{MAC}_{r_1}(B)$. An adversary who wishes to generate a commitment to a related value has two choices: he can either re-use A or use a different A'. If he re-uses A, with high probability he will be unable to generate a correct Tag for a different B', since he does not know the value r_1. On the other hand, if he uses a different A', the public parameters he is forced to use for his commitment B' will be different from those used for the original commitment; thus, the adversary will be able to decommit in only one way, regardless of how the original B is decommitted. In particular, if it is possible for a simulator to equivocate B for a particular choice of A, an adversary who uses

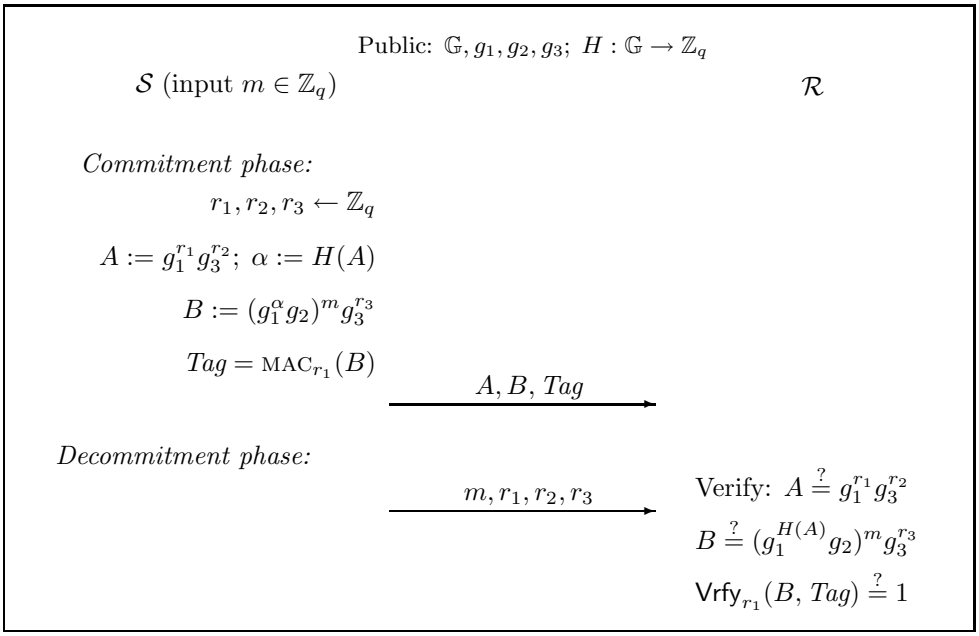

Figure 4.3: A non-malleable commitment scheme based on the discrete logarithm problem.

a different A' will be unable to equivocate B' (without breaking some computational assumption). We refer the reader to [40] for further discussion.

In [40], the dependence (upon A) of the public parameters used for commitment B was achieved via a "selector function," which results in public parameters whose size is dependent on the length of the committed message. Here, we exploit algebraic properties to drastically reduce the size of the public parameters and obtain a more efficient scheme, even in the case of large messages.

4.4.1 Construction Based on the Discrete Logarithm Assumption

The scheme discussed in this section works over any cyclic group $\mathbb{G}$ of prime order in which extracting discrete logarithms is hard but multiplication is easy. For concreteness, we assume an algorithm $\mathcal{G}$ that, on input 1^k, outputs primes p, q with $p = 2q + 1$ and $|q| = k$; we then take $\mathbb{G} \subseteq \mathbb{Z}_p^*$ to be the unique subgroup of order q.

Our starting point is the perfect commitment scheme of Pedersen[5] [103], shown in Figure 4.1 (see also the discussion in Section 4.1). The public parameters in our scheme are generated as follows. First, $\mathcal{TTP}(1^k)$ runs algorithm $\mathcal{G}(1^k)$ to generate primes p, q, thereby defining group $\mathbb{G}$ as discussed above. Next, $\mathcal{TTP}$ selects random generators g_1, g_2, g_3 of $\mathbb{G}$. Additionally, a random function $H : \mathbb{G} \to \mathbb{Z}_q$ is chosen from a family of universal one-way hash functions [98] according to

[5]Note that the Pedersen scheme can be made non-interactive by having generators g, h published as part of the public parameters.

algorithm $\mathsf{UOWH}(1^k)$. The output of $\mathcal{TTP}$ is (p, q, g_1, g_2, g_3, H).

To commit to a message $m \in \mathbb{Z}_q$ (cf. Figure 4.3), the sender first chooses random $r_1, r_2, r_3 \in \mathbb{Z}_q$. The sender forms the first component A by using g_1, g_3 to "commit" to r_1. The sender then computes $\alpha = H(A)$. The second component B is a Pedersen commitment to m with one important difference: the first generator used for the commitment depends upon α. That is, the sender uses Pedersen commitment with generators $(g_1^{\alpha} g_2)$ and g_3. Finally, a *Tag* of B is computed, using a secure MAC with key r_1. The security of the commitment scheme is described in the following Theorem:

Theorem 4.2 *Assuming (1) the hardness of the discrete logarithm problem for groups $\mathbb{G}$ defined by the output of $\mathcal{G}$, (2) the security of $(\mathcal{K}, \text{MAC}, \mathsf{Vrfy})$ as a message authentication code, and (3) the security of UOWH as a universal one-way hash family, the protocol of Figure 4.3 is an ε-non-malleable perfect commitment scheme over $\{\mathbb{Z}_q\}$ in the public-parameters model.*

Proof It is clear that the protocol is perfectly-hiding since B is uniformly distributed in group $\mathbb{G}$ independently of the message m. We now consider the question of computational binding. Say an adversary $\mathcal{A}$ exists which, when given the public parameters, can output a commitment $\langle A, B, \mathit{Tag}\rangle$ and two legal decommitments $\langle m, r_1, r_2, r_3\rangle$ and $\langle m', r_1', r_2', r_3'\rangle$, with $m \neq m'$. Then we can construct a second adversary $\mathcal{A}'$ which, given oracle access to $\mathcal{A}$, violates the computational binding of the standard Pedersen scheme (which holds assuming the discrete logarithm problem is hard [103]). On input $\mathbb{G}, g_1, g_2$, adversary $\mathcal{A}'$ chooses random $s \in \mathbb{Z}_q^*$, computes $g_3 = g_1^s$, selects a random H, and runs $\mathcal{A}(\mathbb{G}, g_1, g_2, g_2, H)$ to generate commitment $\langle A, B, \mathit{Tag}\rangle$ and decommitments $\langle m, r_1, r_2, r_3\rangle$ and $\langle m', r_1', r_2', r_3'\rangle$. Note that $\mathcal{A}$ is run on exactly the same distribution of inputs as it would receive in a real execution of the protocol. Now, $\mathcal{A}'$ computes $\alpha = H(A)$, and outputs B as its commitment along with decommitments $\langle \alpha m + s r_3, m\rangle$ and $\langle \alpha m' + s r_3', m'\rangle$. If the decommitments produced by $\mathcal{A}$ are legal decommitments to different values, then the decommitments output by $\mathcal{A}'$ are legal decommitments to different values. This proves the computational binding of the original protocol.

The proof of non-malleability is more involved, and we first provide some intuition. The simulator (which will do as well as the adversary without seeing any commitment) first generates public parameters which are distributed identically to the real experiment, but for which the simulator knows some trapdoor information allowing the simulator to perfectly equivocate its commitment (cf. Definition 4.4). The simulator generates a commitment *com* to a random message, gives this commitment to the adversary, and receives the commitment com_2 in return. The simulator now tries to get the adversary to decommit com_2 as some message; this is the message that will be output by the simulator. To get the adversary to open its commitment, the simulator decommits *com* to a random message and gives the decommitment to the adversary, repeating this step (rewinding

the adversary each time) a bounded number of times until the adversary opens[6] com_2. Since the simulator can perfectly equivocate its commitment, the adversary's view is equivalent to its view in the original experiment. Furthermore, we show that the adversary cannot equivocate its commitment com_2 without contradicting the discrete logarithm assumption. A complete proof follows.

We begin by describing an equivocable commitment generator Equiv that will be used as a subroutine by our simulator $\mathcal{A}'$:

$\mathsf{Equiv}_1(1^k)$
- $p, q \leftarrow \mathcal{G}(1^k)$
- $g_1, g_3 \leftarrow \mathbb{G}; H \leftarrow \mathsf{UOWH}(1^k)$
- $r_1, r_2, t, u \leftarrow \mathbb{Z}_q$
- $A := g_1^{r_1} g_3^{r_2}$; $\alpha := H(A)$
- $g_2 := g_1^{-\alpha} g_3^t$
- $\sigma := \langle p, q, g_1, g_2, g_3, H \rangle$
- $B := g_3^u$; $Tag := \mathrm{MAC}_{r_1}(B)$
- $com := \langle A, B, Tag \rangle$
- $s := \langle p, q, r_1, r_2, t, u \rangle$
- Output (σ, com, s)

$\mathsf{Equiv}_2(\langle p, q, r_1, r_2, t, u \rangle, m)$
- if $m \notin \mathbb{Z}_q$ output $\perp$
- $r_3 := u - tm \bmod q$
- $dec := \langle m, r_1, r_2, r_3 \rangle$
- Output dec

Note that Equiv satisfies Definition 4.4. Furthermore, p, q, g_1, g_3 can be chosen at random and given to Equiv; knowledge of $\log_{g_1} g_3$ is not necessary. This will be crucial for the proof of security.

We now describe the simulator $\mathcal{A}'$ in more detail. Fix ε, R. Let $com = \langle A, B, Tag \rangle$ be the commitment output by Equiv, and $com_2 = \langle A', B', Tag' \rangle$ be a commitment output by $\mathcal{A}_1$. Say event Collision occurs if $H(A) = H(A')$. The simulator runs Equiv_1 to generate public parameters σ and a commitment com. The adversary, given these values, outputs some commitment com_2. If event Collision occurs, the simulator simply outputs $\perp$. Otherwise, the simulator repeatedly "opens" com a bounded number of times until the adversary de-commits, in some legal way, to a message m_2. If this occurs, message m_2 (along with σ and $\mathcal{D}$) is then output by the simulator. If the adversary never de-commits in a legal way, the simulator simply outputs $\perp$. A formal description follows:

$\mathcal{A}'(1^k)$
- $(\sigma, com, s) \leftarrow \mathsf{Equiv}_1(1^k)$
- $(\mathcal{D}, s') \leftarrow \mathcal{A}_0(\sigma)$
- $(com_2, s'') \leftarrow \mathcal{A}_1(\sigma, com, s')$
- if Collision then output $(\sigma, \mathcal{D}, \perp)$
- Fix random coins $\omega in \Omega$
- Repeat at most $2\varepsilon^{-1} \ln 2\varepsilon^{-1}$ times:
  - $m \leftarrow \mathcal{D}$
  - $dec := \mathsf{Equiv}_2(s, m)$
  - $dec_2 := \mathcal{A}_2(\sigma, com, dec, s''; \omega)$
  - $m_2 := \mathcal{R}(\sigma, com_2, dec_2)$
  - if $m_2 \neq \perp$ break
- output $(\sigma, \mathcal{D}, m_2)$

[6] If the adversary never opens its commitment, the simulator outputs $\perp$.

We will show that the difference $\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) - \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k)$ (cf. Definition 4.2) is less than $\varepsilon + negl(k)$.

The simulator cannot succeed whenever Collision occurs since it outputs $m_2 = \perp$ in this case. This is not a problem, however, since the the probability that both Collision and a success for the adversary occur must be negligible. To see this, note that event Collision can occur in three ways:

1. $\langle A, B, Tag\rangle = \langle A', B', Tag'\rangle$. In this case, the adversary cannot succeed by definition.

2. $\langle A, B, Tag\rangle \neq \langle A', B', Tag'\rangle$ but $A = A'$. Say the adversary later gives legal decommitment $\langle m', r_1', r_2', r_3'\rangle$ for com_2 following legal decommitment $\langle m, r_1, r_2, r_3\rangle$ for com. There are two possibilities: either $\langle r_1, r_2\rangle = \langle r_1', r_2'\rangle$ or not. If they are not equal, then $\mathcal{A}$ has violated the computational binding property of the Pedersen scheme with generators g_1, g_3. If they are equal then the fact that $\mathsf{Vrfy}_{r_1}(B, Tag) = 1$ and $\langle B', Tag'\rangle \neq \langle B, Tag\rangle$ implies that $\mathcal{A}$ has violated the security of the MAC (note that r_1 is information-theoretically hidden from the adversary when com_2 is output). Either of these events can occur with only negligible probability. Thus, the adversary can give a legal decommitment to com_2 (and hence succeed) with only negligible probability.

3. $A \neq A'$ but $H(A) = H(A')$. In this case, $\mathcal{A}$ has violated the security of the universal one-way hash family (note that A may be computed by Equiv_1 before H is chosen). This can only occur with negligible probability.

From now on, assume that event Collision does not occur, since this can only contribute a negligible quantity to the difference of interest. Without loss of generality, we further assume that $\mathcal{D}$ output by $\mathcal{A}_0(\sigma)$ is always a valid distribution over $\mathcal{M}_\sigma \stackrel{\text{def}}{=} \mathbb{Z}_q$ since the adversary cannot succeed, by definition, when this is not the case. Straightforward manipulation, using the fact that Equiv is a perfectly equivocable commitment generator and $(\mathcal{TTP}, \mathcal{S}, \mathcal{R})$ is a perfect commitment scheme gives

$$\begin{aligned}
&\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) = \\
&\quad \Pr\big[(\sigma, com, s) \leftarrow \mathsf{Equiv}_1(1^k); (\mathcal{D}, s') \leftarrow \mathcal{A}_0(\sigma); (com_2, s'') \leftarrow \mathcal{A}_1(\sigma, com, s'); \\
&\qquad m_1 \leftarrow \mathcal{D}; \omega \leftarrow \Omega; dec := \mathsf{Equiv}_2(s, m_1); dec_2 := \mathcal{A}_2(\sigma, com, dec, s''; \omega); \\
&\qquad m_2 := \mathcal{R}(\sigma, com_2, dec_2) : R(\sigma, \mathcal{D}, m_1, m_2) = 1\big]
\end{aligned}$$

and

$$\begin{aligned}
&\widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k) = \\
&\quad \Pr\big[(\sigma, com, s) \leftarrow \mathsf{Equiv}_1(1^k); (\mathcal{D}, s') \leftarrow \mathcal{A}_0(\sigma); (com_2, s'') \leftarrow \mathcal{A}_1(\sigma, com, s');
\end{aligned}$$

$$m_1 \leftarrow \mathcal{D}; \omega \leftarrow \Omega; dec^* \leftarrow \mathsf{Equiv}_2(s, m^*); dec_2^* \leftarrow \mathcal{A}_2(\sigma, com, dec^*, s''; \omega);$$
$$m_2^* := \mathcal{R}(\sigma, com_2, dec_2^*) : R(\sigma, \mathcal{D}, m_1, m_2^*) = 1]\,,$$

where the notation m^*, dec^* represents the fact that the decommitment given to $\mathcal{A}_2$ was produced according to algorithm $\mathcal{A}'$. In particular, dec^* represents either the first decommitment given to $\mathcal{A}$ which resulted in $m_2^* \neq \perp$, or the $(2\varepsilon^{-1} \ln 2\varepsilon^{-1})^{\text{th}}$ decommitment given to $\mathcal{A}$ (if all decommitments up to then had $m_2^* = \perp$).

Define the tuple $(\sigma, com, s; \mathcal{D}; com_2, s''; \omega)$ as *good* if the following holds:

$$\Pr\left[m_1 \leftarrow \mathcal{D}; dec := \mathsf{Equiv}_2(s, m_1); dec_2 := \mathcal{A}_2(\sigma, com, dec, s''; \omega) : \right.$$
$$\left. \mathcal{R}(\sigma, com_2, dec_2) \neq \perp\right] \geq \varepsilon/2$$

(note that the above probability is over choice of m_1 only). Furthermore, define event Good as occurring when a *good* tuple is generated. For brevity, we denote by $\gamma \leftarrow \Gamma(1^k)$ generation of a random tuple via the following experiment:

$$(\sigma, com, s) \leftarrow \mathsf{Equiv}_1(1^k); (\mathcal{D}, s') \leftarrow \mathcal{A}_0(\sigma); (com_2, s'') \leftarrow \mathcal{A}_1(\sigma, com, s'); w \leftarrow \Omega.$$

With respect to a particular tuple γ, denote by $m_2 \leftarrow \mathcal{A}(\sigma, com, m)$ the sequence of events

$$dec \leftarrow \mathsf{Equiv}_2(s, m); dec_2 \leftarrow \mathcal{A}_2(\sigma, com, dec, s''; \omega); m_2 := \mathcal{R}(\sigma, com_2, dec_2)$$

(the outputs of Equiv_2 and $\mathcal{A}_2$, above, are random variables because we may have $m = m^*$ as discussed previously). We now have

$$\begin{aligned}
&\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) - \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k) = \\
&\quad \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\sigma, com_1, m_1) : R(\sigma, \mathcal{D}, m_1, m_2) \ \wedge\ \mathsf{Good}\right] \\
&\quad + \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\sigma, com_1, m_1) : R(\sigma, \mathcal{D}, m_1, m_2) \ \wedge\ \overline{\mathsf{Good}}\right] \\
&\quad - \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : R(\sigma, \mathcal{D}, m_1, m_2^*) \ \wedge\ \mathsf{Good}\right] \\
&\quad - \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : R(\sigma, \mathcal{D}, m_1, m_2^*) \ \wedge\ \overline{\mathsf{Good}}\right].
\end{aligned}$$

Since relation R cannot be true when $m_2 = \perp$, definition of event $\overline{\mathsf{Good}}$ shows that

$$\begin{aligned}
&\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) - \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k) \leq \\
&\quad \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\sigma, com_1, m_1) : R(\sigma, \mathcal{D}, m_1, m_2) \ \wedge\ \mathsf{Good}\right] \\
&\quad + \varepsilon/2 \\
&\quad - \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : R(\sigma, \mathcal{D}, m_1, m_2^*) \ \wedge\ \mathsf{Good}\right].
\end{aligned}$$

This, in turn, implies that

$$\begin{aligned}
&\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) - \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k) \leq \\
&\quad \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\sigma, com_1, m_1); m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : \right. \\
&\quad \left. R(\sigma, \mathcal{D}, m_1, m_2) \ \wedge \ \overline{R(\sigma, \mathcal{D}, m_1, m_2^*)} \ \wedge \ \mathsf{Good}\right] + \varepsilon/2,
\end{aligned}$$

which can be re-written as

$$\begin{aligned}
&\mathsf{Succ}^{\mathrm{NM}}_{\mathcal{A},R}(k) - \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k) \leq \\
&\quad \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\sigma, com_1, m_1); m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : \right. \\
&\qquad \left. R(\sigma, \mathcal{D}, m_1, m_2) \ \wedge \ \overline{R(\sigma, \mathcal{D}, m_1, m_2^*)} \ \wedge \ m_2^* = \perp \ \wedge \mathsf{Good}\right] \qquad (4.4) \\
&\quad + \Pr\left[\gamma \leftarrow \Gamma(1^k); m_1 \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\sigma, com_1, m_1); m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : \right. \\
&\qquad \left. R(\sigma, \mathcal{D}, m_1, m_2) \ \wedge \ \overline{R(\sigma, \mathcal{D}, m_1, m_2^*)} \ \wedge \ m_2^* \neq \perp \ \wedge \mathsf{Good}\right] \qquad (4.5) \\
&\quad + \varepsilon/2.
\end{aligned}$$

Note that if $\Pr[\gamma \leftarrow \Gamma(1^k) : \mathsf{Good}] = 0$ we are done, since the above expression is then equal to $\varepsilon/2$. Assuming that event Good occurs with non-zero probability, we now bound (4.4) and (4.5). First, notice that expression (4.4) is bounded from above by the probability that $m_2^* = \perp$. However, definition of event Good and a straightforward probability calculation show that:

$$\begin{aligned}
&\Pr\left[\gamma \leftarrow \Gamma(1^k); m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : m_2^* = \perp \ \wedge \mathsf{Good}\right] \\
&\quad \leq \quad \Pr\left[\gamma \leftarrow \Gamma(1^k); m_2^* \leftarrow \mathcal{A}(\sigma, com_1, m^*) : m_2^* = \perp \,|\mathsf{Good}\right] \\
&\quad < \quad (1 - \varepsilon/2)^{2\varepsilon^{-1} \ln 2\varepsilon^{-1}} \\
&\quad \leq \quad e^{\ln \varepsilon/2} \\
&\quad = \quad \varepsilon/2.
\end{aligned}$$

Finally, notice that for the event in expression (4.5) to occur, we must have $m_2 \neq \perp$ and $m_2 \neq m_2^*$. But this then gives a Pedersen commitment com_2 (using generators g_3 and $g_1^{\alpha'} g_2 = g_1^{(\alpha' - \alpha)} g_3^t$) which is decommitted in two different ways. This allows determination of $\log_{g_1} g_3$ as follows (recall that $\Delta \stackrel{\mathrm{def}}{=} \alpha' - \alpha \neq 0$): run Equiv using given generators g_1, g_3 to generate σ and com (recall that knowledge of $\log_{g_1} g_3$ is not necessary to run Equiv). The adversary $\mathcal{A}_1$ then produces a commitment com_2. Following the description of $\mathcal{A}'$, run $\mathcal{A}_2$ to obtain a decommitment to message m_2^*. Then, decommit once more to a randomly selected m_1 and give this as input to $\mathcal{A}_2$ to obtain a decommitment to m_2. If $m_2 \neq \perp$ and $m_2^* \neq \perp$ and $m_2 \neq m_2^*$ (which we call event $\mathsf{Success}$), then we have $\langle m_2, r_3 \rangle$ and $\langle m_2^*, r_3^* \rangle$ such that:

$$(g_1^{\Delta} g_3^t)^{m_2} g_3^{r_3} = (g_1^{\Delta} g_3^t)^{m_2^*} g_3^{r_3^*}$$

$$\Rightarrow g_1^{\Delta m_2} g_3^{t m_2 + r_3} = g_1^{\Delta m_2^*} g_3^{t m_2^* + r_3^*}$$
$$\Rightarrow \log_{g_1} g_3 = \frac{\Delta(m_2 - m_2^*)}{t(m_2^* - m_2) + r_3^* - r_3},$$

and we have then computed the desired discrete logarithm (the denominator in the above expression is non-zero as long as $\Delta \neq 0$). The probability of Success is bounded from below by expression (4.5); by assumption, however, the discrete logarithm problem is intractable and thus:

$$(4.5) \leq \Pr[\mathsf{Success}] \leq negl(k).$$

Putting everything together gives the desired result. ■

A slightly stronger security guarantee is possible. Let event Collision be defined as above. Let $p_{\mathcal{A}}(k)$ be the probability that the adversary in the original experiment succeeds and event Collision does not occur (note that for any PPT adversary, the probability that the adversary succeeds and Collision does occur is negligible as argued in the proof of Theorem 4.2). For a given simulator $\mathcal{A}'$, let $\widetilde{p}_{\mathcal{A}'}(k)$ denote the simulator's success probability; i.e., $\widetilde{p}_{\mathcal{A}'}(k) = \widetilde{\mathsf{Succ}}_{\mathcal{A}',R}(k)$. We will construct a simulator $\mathcal{A}'$ which has expected running time polynomial in $\frac{1}{p_{\mathcal{A}}(k)}$, such that $p_{\mathcal{A}}(k) - \widetilde{p}_{\mathcal{A}'}(k) \leq \varepsilon(k) \cdot \frac{1}{p_{\mathcal{A}}(k)}$ for some negligible function $\varepsilon(\cdot)$. In particular, if there exists some constant c such that $p_{\mathcal{A}}(k) > 1/k^c$ for all k, our simulator does (essentially) at least as well as the original adversary and runs in expected polynomial time.

The simulator $\mathcal{A}'$ is simple: it runs the adversary until the adversary succeeds and event Collision does not occur; it then outputs the values of $(\sigma, \mathcal{D}, m_2)$ at that time.

$\mathcal{A}'(1^k)$
  Repeat until $R(\sigma, \mathcal{D}, m, m_2) = 1$ and $\overline{\mathsf{Collision}}$:
    $(\sigma, com, s) \leftarrow \mathsf{Equiv}_1(1^k)$
    $(\mathcal{D}, s') \leftarrow \mathcal{A}_0(\sigma)$
    $(com_2, s'') \leftarrow \mathcal{A}_1(\sigma, com, s')$
    $m \leftarrow \mathcal{D}$
    $dec := \mathsf{Equiv}_2(s, m)$
    Choose random coins ω
    $dec_2 := \mathcal{A}_2(\sigma, com, dec, s''; \omega)$
    $m_2 := \mathcal{R}(\sigma, com_2, dec_2)$
  output $(\sigma, \mathcal{D}, m_2)$

(Algorithm Equiv is as defined previously.)

Our analysis proceeds assuming $p_{\mathcal{A}}(k) > 1/k^c$ for some constant c. In this case, the simulator clearly runs in expected polynomial time.

As in the proof of Theorem 4.2, we denote the generation of a tuple $\gamma = (\sigma, com, s; \mathcal{D}; com_2, s''; \omega)$ by the shorthand $\gamma \leftarrow \Gamma(1^k)$. We denote by $m_2 := \mathcal{A}(\gamma, m)$ the sequence of events:

$$dec := \mathsf{Equiv}_2(s, m); dec_2 \leftarrow \mathcal{A}_2(\sigma, com, dec, s''; \omega); m_2 := \mathcal{R}(\sigma, com_2, dec_2),$$

and, slightly abusing notation, denote by $(m_2, m_2') := \mathcal{A}(\gamma, m, m')$ the sequence of events:

$$m_2 := \mathcal{A}(\gamma, m); m_2' := \mathcal{A}(\gamma, m').$$

We also write $R(m, m_2)$ instead of $R(\sigma, \mathcal{D}, m, m_2)$ and assume that σ and $\mathcal{D}$ are implicitly defined during the course of the experiment. Finally, we abbreviate Collision by Coll.

The adversary's success probability may be expressed as

$$p_{\mathcal{A}}(k) = \Pr[\gamma \leftarrow \Gamma(1^k); m \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\gamma, m) : R(m, m_2) \wedge \overline{\mathsf{Coll}}],$$

while the success probability of the simulator may be written as

$$\begin{aligned} \widetilde{p}_{\mathcal{A}'}(k) &= \Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\gamma, m) : R(m', m_2) | R(m, m_2) \wedge \overline{\mathsf{Coll}}] \\ &= \frac{\Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\gamma, m) : R(m', m_2) \wedge R(m, m_2) \wedge \overline{\mathsf{Coll}}]}{p_{\mathcal{A}}(k)} \end{aligned}$$

Intuitively, we expect that for any given γ the adversary can only decommit in one (legal) way. This intuition is captured by the following lemma:

Lemma 4.1 *For all $\mathcal{A}$ such that $p_{\mathcal{A}}(k) > 1/k^c$ for some constant c, there exists some negligible $\varepsilon(\cdot)$ such that:*

$$\begin{aligned} &\Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; (m_2, m_2') := \mathcal{A}(\gamma, m, m') : R(m, m_2) \wedge R(m', m_2') \wedge \overline{\mathsf{Coll}}] < \\ &\quad \Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\gamma, m) : R(m, m_2) \wedge R(m', m_2) \wedge \overline{\mathsf{Coll}}] + \varepsilon(k). \end{aligned}$$

Proof For any two events A and B, we have $\Pr[A \wedge \overline{B}] \geq \Pr[A] - \Pr[B]$. Thus

$$\begin{aligned} &\Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; (m_2, m_2') := \mathcal{A}(\gamma, m, m') : R(m, m_2) \wedge R(m', m_2') \wedge \overline{R(m', m_2)} \wedge \overline{\mathsf{Coll}}] \geq \\ &\quad \Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; (m_2, m_2') := \mathcal{A}(\gamma, m, m') : R(m, m_2) \wedge R(m', m_2') \wedge \overline{\mathsf{Coll}}] \\ &\quad - \Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; m_2 := \mathcal{A}(\gamma, m) : R(m, m_2) \wedge R(m', m_2) \wedge \overline{\mathsf{Coll}}]. \end{aligned}$$

Event $R(m, m_2) \wedge R(m', m_2') \wedge \overline{R(m', m_2)}$ occurs only when $m_2 \neq m_2'$ yet $m_2 \neq \perp$ and $m_2' \neq \perp$. As in the proof of Theorem 4.2, however, such an event allows extraction of the discrete logarithm of g_3 with respect to g_1 in case event Coll does not occur. Since the entire experiment may be run in polynomial time, this probability must be negligible in k. ■

Therefore, for some negligible function $\varepsilon(\cdot)$ we have:

$$\widetilde{p}_{\mathcal{A}'}(k) \geq \frac{\Pr[\gamma \leftarrow \Gamma(1^k); m, m' \leftarrow \mathcal{D}; (m_2, m_2') := \mathcal{A}(\gamma, m, m') : R(m, m_2) \wedge R(m', m_2') \wedge \overline{\mathsf{Coll}}] - \varepsilon(k)}{p_{\mathcal{A}}(k)}.$$

We may re-write this as:

$$\widetilde{p}_{\mathcal{A}'}(k) \geq \frac{\left(\sum_{\gamma,m,m'} \Pr[\gamma = \Gamma(1^k) \wedge (m,m') = \mathcal{D}^2] \cdot \left(\Pr[m_2 := \mathcal{A}(\gamma, m) : R(m, m_2) \wedge \overline{\mathsf{Coll}}]\right)^2\right) - \varepsilon(k)}{p_{\mathcal{A}}(k)}.$$

Using the fact that for any random variable X we have $\mathbf{E}(X^2) \geq (\mathbf{E}(X))^2$, this gives:

$$\widetilde{p}_{\mathcal{A}'}(k) \geq \frac{\left(\sum_{\gamma,m,m'} \Pr[\gamma = \Gamma(1^k) \wedge (m,m') = \mathcal{D}^2] \cdot \Pr[m_2 := \mathcal{A}(\gamma, m) : R(m, m_2) \wedge \overline{\mathsf{Coll}}]\right)^2 - \varepsilon(k)}{p_{\mathcal{A}}(k)},$$

But this immediately yields

$$\widetilde{p}_{\mathcal{A}'}(k) \geq p_{\mathcal{A}}(k) - \frac{\varepsilon(k)}{p_{\mathcal{A}}(k)}$$

and therefore $p_{\mathcal{A}}(k) - \widetilde{p}_{\mathcal{A}'}(k) \leq \mathit{negl}(k)$ which is the desired result.

4.4.2 Construction Based on the RSA Assumption

Here, our starting point is the RSA-based perfect commitment scheme of Okamoto [100]. Let N be a product of two primes, and let $g \in \mathbb{Z}_N^*$ and e a prime number be given. Then, a commitment to a message $m \in \mathbb{Z}_e$ is generated by choosing a random $u \in \mathbb{Z}_N^*$ and forming the commitment $g^m u^e \bmod N$. It is easy to see that this scheme achieves information-theoretic secrecy. We use the following lemma to show that the scheme is computationally binding under the RSA assumption.

Lemma 4.2 *Fix N. For any $g \in \mathbb{Z}_N^*$ and non-zero e and m such that $\gcd(|m|, |e|) = 1$, given $u \in \mathbb{Z}_N^*$ such that $g^m = u^e \bmod N$, we may efficiently compute $g^{1/e} \bmod N$.*

Proof Without loss of generality, we may assume that $e, m > 0$; if, for example, $m < 0$ we can re-write the above equation as $(g^{-1})^{-m} = u^e \bmod N$. Since e and m are relatively prime, we may efficiently compute (using the extended Euclidean algorithm) integers a, b such that $am + be = 1$. But then:

$$\begin{aligned} g^1 &= g^{am} g^{be} \\ &= u^{ae} g^{be} \bmod N \\ &= \left(u^a g^b\right)^e \bmod N, \end{aligned}$$

and therefore $u^a g^b = g^{1/e} \bmod N$. ■

This immediately gives the following corollary:

Corollary 4.1 *([100]) The Okamoto scheme is computationally binding under the RSA assumption.*

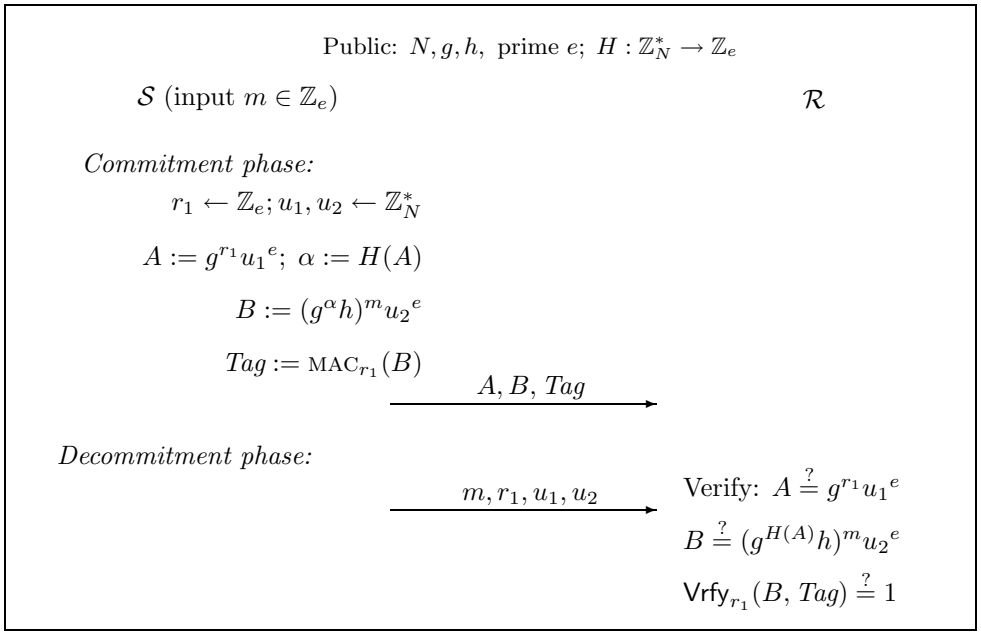

Figure 4.4: A non-malleable commitment scheme based on the RSA problem.

Proof Assume public parameters N, e, g as discussed above. Assume further that a given commitment can be opened to two messages $m_1, m_2 \in \mathbb{Z}_e$. Thus, we have $g^{m_1} u_1^e = g^{m_2} u_2^e \bmod N$ for some $u_1, u_2 \in \mathbb{Z}_N^*$. Re-writing this gives $g^{m_1 - m_2} = (u_2/u_1)^e \bmod N$. Since $|m_1 - m_2| < e$ and e is prime, $|m_1 - m_2|$ and e are relatively prime. Lemma 4.2 shows that we may then compute $g^{1/e} \bmod N$. ■

A complete description of our protocol appears in Figure 4.4. The intuition for the protocol is exactly the same as that for the discrete logarithm-based protocol given previously. The public parameters consist of N, a product of two large primes with $|N| = k$, along with a prime e with $|e| = \Theta(k)$ and two random elements $g, h \in \mathbb{Z}_N^*$. Also included is a function $H : \mathbb{Z}_N^* \to \mathbb{Z}_e$ chosen from a family of universal one-way hash functions. To commit to a message $m \in \mathbb{Z}_e$, the sender first chooses random $r_1 \in \mathbb{Z}_e$. The sender forms the first component A by "committing" to r_1 using the Okamoto scheme; this value will later be used to authenticate the second component. The sender calculates $\alpha = H(A)$, and then commits to message m using the Okamoto scheme, with one important difference: the element used for this commitment depends on α. That is, the sender performs Okamoto commitment with prime e and random element $g^\alpha h$. As before, this will be essential to the proof of security. Finally, a *Tag* of B is computed using r_1 from before. The following theorem describes the security of this construction:

Theorem 4.3 *Assuming (1) the hardness of the RSA problem for N generated by $\mathcal{TTP}$; (2) the security of $(\mathcal{K}, \text{MAC}, \mathsf{Vrfy})$ as a message authentication code; and (3) the security of* UOWH *as a*

universal one-way hash family, the protocol of Figure 4.4 is an ε-non-malleable perfect commitment scheme in the public-parameters model.

Proof The proof is substantially similar to that of Theorem 4.2; therefore, only the key differences are highlighted here. As before, we need to provide a perfectly equivocable commitment generator for our scheme (here, Gen represents the algorithm used by the trusted party to generate N and e):

$\mathsf{Equiv}_1(1^k)$	$\mathsf{Equiv}_2(\langle N, e, r_1, u_1, w_1, w_2\rangle, m)$
$N, e \leftarrow \mathsf{Gen}(1^k)$	if $m \notin \mathbb{Z}_e$ output $\perp$
$g \leftarrow \mathbb{Z}_N^*; H \leftarrow \mathsf{UOWH}(1^k)$	$u_2 = w_2 / w_1^m \bmod N$
$r_1 \leftarrow \mathbb{Z}_e; u_1, w_1, w_2 \leftarrow \mathbb{Z}_N^*$	$dec := \langle m, r_1, u_1, u_2\rangle$
$A := g^{r_1} u_1^e;\ \alpha := H(A)$	Output dec
$h := g^{-\alpha} w_1^e$	
$\sigma := \langle N, e, g, h; H\rangle$	
$B := w_2^e;\ Tag := \mathrm{MAC}_{r_1}(B)$	
$com := \langle A, B, Tag\rangle$	
$s := \langle N, e, r_1, u_1, w_1, w_2\rangle$	
Output (σ, com, s)	

Note that Equiv satisfies Definition 4.4. Furthermore, N, e, g can be chosen at random and given to Equiv; knowledge of $g^{1/e} \bmod N$ is not necessary. This will be crucial for the proof of security.

A simulator may be constructed as in the proof of Theorem 4.2, and the remainder of the proof essentially follows. In particular, we may define event COLLISION in an analogous manner as before and note that consideration of this event has only negligible effect on the difference of interest. Now, if the adversary can equivocate and open his commitment to two different messages when $\Delta \stackrel{\text{def}}{=} \alpha' - \alpha \neq 0$, then we may extract an e^{th} root of g as follows: upon obtaining $\langle m_2, u_2\rangle$ and $\langle m_2^*, u_2^*\rangle$ such that $(g^{\alpha'} h)^{m_2} u_2^e = (g^{\alpha'} h)^{m_2^*} (u_2^*)^e$, we derive (using the fact that $h = g^{-\alpha} w_1^e$ for known w_1):

$$\begin{aligned} \left(g^{\Delta} w_1^e\right)^{m_2} u_2^e &= \left(g^{\Delta} w_1^e\right)^{m_2^*} (u_2^*)^e \\ \Rightarrow g^{\Delta m_2} \left(w_1^{m_2} u_2\right)^e &= g^{\Delta m_2^*} \left(w_1^{m_2} u_2^*\right)^e \\ \Rightarrow g^{\Delta(m_2 - m_2^*)} &= \left(\frac{w_1^{m_2^*} u_2^*}{w_1^{m_2} u_2}\right)^e . \end{aligned}$$

Since $|\Delta| < e$, $|(m_2 - m_2^*)| < e$, and e is prime, we have that $\Delta \cdot (m_2 - m_2^*)$ and e are relatively prime. Application of Lemma 4.2 shows that we can then compute $g^{1/e}$. ■

4.5 Extensions

Arbitrarily-long messages. Theorems 4.2 and 4.3 hold even if the message is hashed before commitment (note that hashing the message before commitment is *not* known to be secure for an

arbitrary non-malleable commitment scheme; in fact, evidence to the contrary is given by the construction of [58]). To see this, note that Equiv can still perfectly equivocate to any (random) message M by first computing $m = \mathcal{H}(M)$ and then running the identical Equiv_2 algorithm. The simulator $\mathcal{A}'$ is also identical. The hash function $\mathcal{H}$ must be collision resistant for the binding property to hold, but no other assumptions about the hash function are necessary, and the scheme maintains perfect secrecy.[7]. The present schemes therefore give practical and provably-secure methods for non-malleable, perfect commitment to arbitrarily long messages.

Reducing the commitment size. Our schemes produce commitments $com = (A, B, Tag)$ of size (roughly) $2k$, where k is the length of a string representing a group element. However, one can replace this commitment with any string that uniquely binds the sender to *com*. At least two modifications in this vein seem useful:

- Using a collision-resistant hash-function h, we can replace the commitment *com* with $h(com)$. The decommitment phase is the same as before. This does not increase the computational cost of the protocol by much. The resulting commitment size is equal to the output length of a hash function believed to be collision-resistant. In particular, this allows us to achieve optimal commitment size $\omega(\log k)$, assuming an appropriate hash function. Note that hashing the commitment is *not* known to give provable security for general non-malleable commitment schemes, yet it *does* work (as can be seen by careful examination of the proof) for the particular constructions given here.

- By adding one more public parameter and making appropriate (small) modifications, we can (for example, in Figure 4.3) set the commitment to the *product* of A, B and Tag (assuming A is an extended-Pedersen commitment to r_1, r_2 and Tag is computed as $B^{r_1} g_3^{r_2}$, which is an information-theoretically secure MAC of B). This reduces the commitment length to k.

Unique identifiers. As mentioned in [44], in many situations there is a unique identifier associated with each user and using this can improve the efficiency of non-malleable primitives. This is also true of our perfect commitment schemes. For example, in our discrete-logarithm construction, if each user in the system has identifier $id \in \mathbb{Z}_q$, we can simplify the scheme by replacing α with id. An adversary who attempts to generate related commitments must do so with respect to *his* identifier $id' \neq id$. The commitment is now simply B, as the components A and Tag are no longer needed (their only role in the original protocol was to force an adversary to change α).

[7] This can be compared to [58] which requires added complications when using an arbitrary hash function and achieves only statistical secrecy.

Chapter 5

Non-Malleable and Concurrent (Interactive) Proofs of Knowledge

5.1 Introduction

A *proof of knowledge*, introduced by Goldwasser, Micali, and Rackoff [70], represents a formalization of the deceptively simple notion of "proving that you know something" to someone else. More formally, consider an arbitrary relation R which is computable in polynomial time. Say a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ have common input x, and $\mathcal{P}$ additionally knows a value w such that $R(x, w) = 1$. How can $\mathcal{P}$ prove to $\mathcal{V}$ that he indeed knows such a w? Of course, $\mathcal{P}$ can simply send w to $\mathcal{V}$, but, in many cases, this will reveal more information than $\mathcal{P}$ would like. One can try to construct other interactive protocols for accomplishing this task, but without a precise definition it is unclear how to proceed.

Indeed, defining the notion correctly has been difficult [70, 54, 116]. The effort to obtain the "right" definition culminated in the work of Bellare and Goldreich [8] which contains the now-standard definitional approach. Informally, and omitting many details, the definition states that an interactive protocol Π constitutes a proof of knowledge if, for any Turing machine $\mathcal{P}$ which successfully convinces a verifier $\mathcal{V}$ with "high" probability (where $\mathcal{V}$ executes Π), the value w may be *extracted* from $\mathcal{P}$ by an explicitly-given extraction algorithm (the *knowledge extractor*).[1]

We illustrate the above discussion with an example [111]. Let the common input to $\mathcal{P}$ and $\mathcal{V}$ be a finite, cyclic group $\mathbb{G}$ of prime order q, a generator g of $\mathbb{G}$, and a value $y \in \mathbb{G}$. Additionally, assume $\mathcal{P}$ has as input a value $x \in \mathbb{Z}_q$ such that $y = g^x$. Figure 5.1 shows an interactive protocol by which $\mathcal{P}$ can convince $\mathcal{V}$ that $\mathcal{P}$ in fact knows the discrete logarithm x of the common input y. To begin, $\mathcal{P}$ chooses a random value $r \in \mathbb{Z}_q$, computes $A = g^r$, and sends A to $\mathcal{V}$. Then, $\mathcal{V}$ chooses a

[1] *Non-interactive* proofs of knowledge are possible if the prover and verifier share a common random string [39]; however, since we do not require this notion here we omit further details.

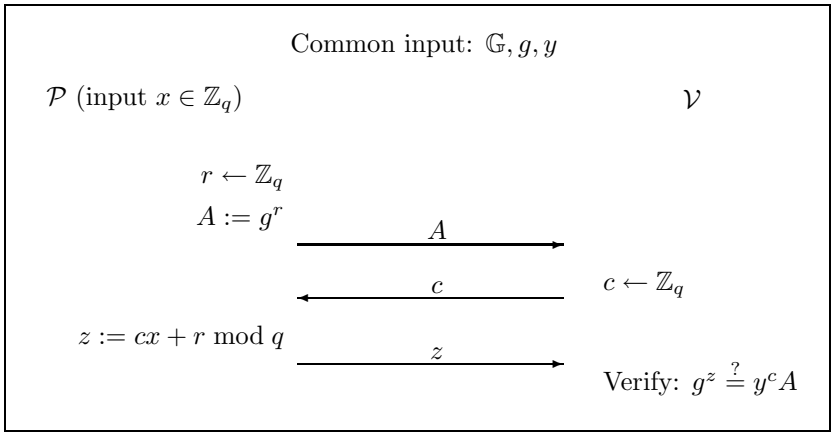

Figure 5.1: Proof of knowledge of a discrete logarithm.

random challenge $c \in \mathbb{Z}_q$ and sends c to $\mathcal{P}$. To respond, $\mathcal{P}$ computes $z = cx + r \bmod q$ and sends z back to the verifier. $\mathcal{V}$ checks that $g^z \stackrel{?}{=} y^c A$. Note that if $\mathcal{P}$ is honest (and really knows the correct value x) verification will always succeed since $g^z = g^{cx+r} = g^{xc} g^r = y^c A$.

To see intuitively why this is a proof of knowledge, note that there are two possibilities once $\mathcal{P}$ has sent A: either $\mathcal{P}$ is able to respond correctly to only one or fewer possible challenges, or $\mathcal{P}$ can respond correctly to two or more challenges. In the former case, $\mathcal{P}$'s probability of "fooling" $\mathcal{V}$ (who picks a random challenge) is at most $1/q$. In particular, if $|q| = k$ (where k is a security parameter) this probability is negligible. On the other hand, if $\mathcal{P}$ "knows" correct responses z_1, z_2 to the two distinct challenges c_1, c_2, this implies that $g^{z_1} = y^{c_1} A$ and $g^{z_2} = y^{c_2} A$. But then $g^{z_1 - z_2} = y^{c_1 - c_2}$ and hence $\mathcal{P}$ "could" efficiently compute $\log_g y = (z_1 - z_2)/(c_1 - c_2) \bmod q$ himself. In other words, if $\mathcal{P}$ has the ability to convince $\mathcal{V}$ with non-negligible probability, then $\mathcal{P}$, in some sense, already must be able to compute x himself. A formal proof that the protocol satisfies the formal definition of a proof of knowledge [8] is also possible.

Often, a protocol like that of Figure 5.1 is more useful than having $\mathcal{P}$ simply send x, since the protocol is in fact a *zero-knowledge* protocol as long as $\mathcal{V}$ is honest. To see this (informally), note that a random transcript of an execution of the protocol for any y can be efficiently simulated even without knowledge of $\log_g y$. To simulate, first pick random $c, z \in \mathbb{Z}_q$. Then, set $A = g^z / y^c$. The simulated transcript, which is distributed identically to a real transcript, is (A, c, z). Thus, the protocol reveals no information about x (beyond what can be computed in polynomial time from y alone) to an honest verifier $\mathcal{V}$.

Proofs of knowledge have a wide range of applications. They are crucial for secure two-party and multi-party computation [66], and have also been used to build interactive commitment protocols [44, 59] and identification schemes [56, 74, 111]. They may also be used to construct encryption

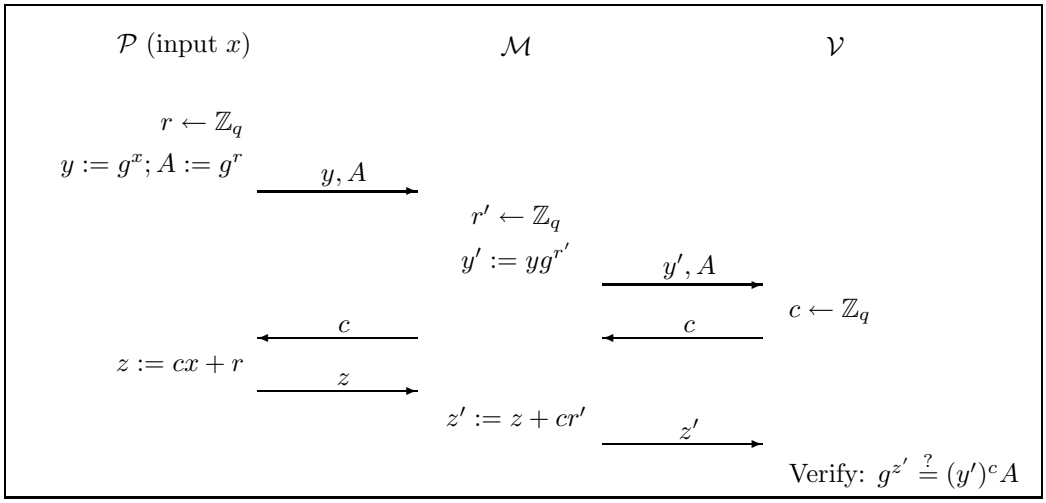

Figure 5.2: Man-in-the-middle attack on a proof of knowledge.

schemes with strong security properties [99, 44, 110, 38].

Unfortunately, the standard definition of a proof of knowledge is not sufficient in a network-based setting such as that considered in this work. More precisely, if an adversary $\mathcal{M}$ acts as a man-in-the-middle between $\mathcal{P}$ and $\mathcal{V}$, where $\mathcal{P}$ proves knowledge of x to $\mathcal{M}$ while $\mathcal{M}$ proves knowledge of x' to $\mathcal{V}$, the standard definition does not preclude $\mathcal{M}$ convincing $\mathcal{V}$ *even when* $\mathcal{M}$ *does not really know* x'. Proofs of knowledge in which such man-in-the-middle attacks are possible are called *malleable*. As an example, we demonstrate in Figure 5.2 that the protocol of Figure 5.1 is malleable. In Figure 5.2, we assume that $\mathcal{P}$, $\mathcal{M}$, and $\mathcal{V}$ all share the same group $\mathbb{G}$ and generator g as common input. Furthermore, for simplicity, the element y for which knowledge of $\log_g y$ is proved is included with the first message. In the figure, $\mathcal{P}$ proves knowledge of $x = \log_g y$ to $\mathcal{M}$, while $\mathcal{M}$ successfully proves knowledge of $x' = \log_g y'$ to $\mathcal{V}$; note, however, that $\mathcal{M}$ does not actually know x'!

For many suggested applications of proofs of knowledge, preventing such attacks is essential. To give just one example, it has been suggested [60, 75, 62] (following [99]) to use interactive (zero-knowledge) proofs of knowledge to achieve chosen-ciphertext-secure (interactive) public-key encryption via the following construction: to encrypt a message m using public key pk, the sender computes $C = \mathcal{E}_{pk}(m; r)$ for random r, and then executes an interactive proof-of-knowledge (with the receiver) of m and r. Unfortunately, while this construction is sufficient to achieve *non-adaptive* chosen-ciphertext security, it does not guarantee *adaptive* chosen-ciphertext security when the proof of knowledge is malleable.

The standard definition of a proof of knowledge is also not sufficient when multiple proofs are executed by multiple provers in a concurrent and asynchronous setting. To see why this is the

case, note that extraction of a valid witness from the prover typically requires the knowledge extractor to *rewind* the prover. However, if many proofs are being conducted simultaneously in an arbitrarily-interleaved manner, extracting all witnesses from all provers at the appropriate points of the execution[2] may require exponential time due to the nested rewindings. A similar problem arises in the simulation of concurrent zero-knowledge proofs; in that case, however, interaction with multiple *verifiers* is the source of the problem.

5.1.1 Our Contributions

We focus on proofs of *plaintext* knowledge (PPKs) in which a sender $\mathcal{S}$ proves knowledge of the contents m of a ciphertext C (a more formal definition is given below). However, it is clear that our definitions and constructions may be extended to proofs of knowledge for more general NP-relations. We give the first definition of non-malleability for *interactive* proofs of (plaintext) knowledge, and show efficient, non-malleable PPKs for the RSA [107], Rabin [104], Paillier [101], and El-Gamal [51] cryptosystems. We then highlight important applications of these PPKs to (1) chosen-ciphertext-secure, interactive encryption, (2) password-based authentication and key-exchange in the public-key model, (3) deniable authentication, and (4) identification. We construct efficient and practical protocols for each of these tasks, improving and extending previous work. Our results include:

- The first practical and non-malleable interactive encryption schemes based on the RSA, factoring, or (computational) composite residuosity assumptions.
- The first practical protocols for password-based authentication and key-exchange (in the public-key model) based on the RSA, factoring, or (computational) composite residuosity assumptions.
- The first practical protocols for deniable authentication based on the RSA, factoring, CDH, or (computational) composite residuosity assumptions. The round-complexities of our protocols are the same as in the only previous efficient solution based on DDH.
- The first 3-round identification scheme secure against man-in-the-middle attacks.

Of additional interest, our techniques provide a general methodology for constructing efficient, *non-malleable* (zero-knowledge) proofs of knowledge when shared parameters are available. Note that for the applications listed above, these parameters can simply be included as part of users' public keys. Furthermore, this work is the first to consider the issues arising in concurrent executions

[2] In our applications, a simulator will be required to extract the i^{th} witness from prover P_i *as soon as* P_i successfully completes his proof. Indeed, this is the source of the problem, since extraction of all witnesses at the end of the entire interaction (i.e., after all provers have completed their proofs) can be done in expected polynomial time.

of *proofs of knowledge*; previous work (e.g., [49]) considered concurrency only in the context of zero-knowledge.

5.1.2 Previous Work

Proofs of *plaintext* knowledge are explicitly considered by Aumann and Rabin [1] who provide an elegant solution for *any* public-key encryption scheme. Our solutions improve upon theirs in many respects: (1) by working with specific, number-theoretic assumptions we vastly improve the efficiency and round-complexity of our schemes; (2) we explicitly consider malleability and ensure that our solutions are non-malleable; (3) our protocols are secure even against a dishonest verifier, whereas [1] only considers security against an honest verifier (i.e., the intended recipient); (4) we explicitly handle concurrency and our protocols remain provably-secure under concurrent composition. More generally, every NP-relation has a (zero-knowledge) argument of knowledge assuming the existence of one-way functions [54]; furthermore, once public information is assumed (as we assume here) and assuming the existence of trapdoor permutations and dense cryptosystems, non-interactive zero-knowledge (NIZK) proofs of knowledge are also possible [39].

None of the above-mentioned solutions are non-malleable. Dolev, Dwork, and Naor [44] introduce definitions and constructions for non-malleable, zero-knowledge, interactive proofs. Sahai [110] subsequently considers the case of non-malleable, non-interactive, zero-knowledge proofs and proofs of knowledge; he provides definitions and constructions for the "single-theorem" case and shows extensions to the case of a bounded-polynomial number of proofs. De Santis, et al. [38] give definitions and constructions for *robust* non-interactive zero-knowledge proofs and proofs of knowledge (which are, in particular, non-malleable), improving upon previous work. Our definitions extend those of De Santis, et al. [38] to the interactive setting; interestingly, ours is the first work to explicitly consider non-malleability for *interactive* proofs of knowledge.

The above-mentioned works [44, 110, 38] show that, in principal, solutions to our problem exist. These solutions, however, are impractical. In particular, known non-malleable interactive proofs [44] require a poly-logarithmic number of rounds, while in the non-interactive setting practical NIZK proofs — let alone non-malleable ones — are not currently known for number-theoretic problems of interest (i.e., without reducing the problem to a general NP-complete language). The techniques outlined in this paper serve as a general method for achieving *practical* non-malleable (zero-knowledge) proofs of knowledge when public parameters are available, a problem which has not been previously considered.

We discuss previous work relating to non-malleable encryption, password-based key exchange, deniable authentication, and identification in the appropriate sections of this chapter.

5.1.3 Outline of the Chapter

We introduce definitions for proofs of plaintext knowledge (PPKs) and non-malleable PPKs in Section 5.2. As mentioned previously, the latter is the first definition of non-malleability for *interactive* proofs of knowledge. Sections 5.3.1–5.3.4 describe very efficient constructions of non-malleable PPKs based on the RSA assumption, the hardness of factoring, the composite residuosity assumption, and the DDH assumption, respectively. We then consider applications of our non-malleable PPKs. Sections 5.4.1–5.4.5 introduce definitions for non-malleable interactive encryption, password-based authentication and key exchange, deniable (message) authentication, and identification schemes secure against man-in-the-middle attacks. These sections also describe practical constructions (based on our non-malleable PPKs) for these tasks.

5.2 Definitions and Preliminaries

This section includes definitions specifically related to PPKs and non-malleable PPKs only. Other definitions appear in the relevant sections of this chapter.

Non-malleable proofs of plaintext knowledge. The definitions given in this section focus on proofs of plaintext knowledge, yet they may be easily extended to proofs of knowledge for general NP-relations. We assume a non-interactive public-key encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. The encryption of message m under public key pk using randomness r to give ciphertext C is denoted as $C := \mathcal{E}_{pk}(m; r)$. In this case, we say that tuple (m, r) is a *witness to the decryption of* C *under* pk. For convenience, we assume that $|pk| = k$, where k is the security parameter. We let the notation $\langle A(a), B(b)\rangle(c)$ be the random variable denoting the output of B following an execution of an interactive protocol between A (with private input a) and B (with private input b) on joint input c, where A and B have uniformly-distributed random tapes.

A *proof of plaintext knowledge* (PPK) allows a sender $\mathcal{S}$ to prove knowledge of a witness to the decryption of some ciphertext C to a receiver $\mathcal{R}$. Both $\mathcal{S}$ and $\mathcal{R}$ have an additional joint input σ; in practice, this may be published along with $\mathcal{R}$'s public key pk.[3] To be useful, a PPK should additionally ensure that no information about m is revealed, either to the receiver (which is important if the receiver does not have the secret key) or to an eavesdropper. So that no information about m is revealed, a PPK is required to be zero-knowledge in the following sense: as mentioned above, our PPKs use parameters σ (known to all parties); these parameters will be generated by an algorithm $\mathcal{G}(pk)$. We require the existence of a simulator $\mathcal{SIM}$ which takes pk as input and outputs parameters σ whose distribution will be equivalent to the output of $\mathcal{G}(pk)$. Furthermore, given any

[3] It is important to note that there is no incentive for $\mathcal{R}$ to cheat when choosing σ.

valid ciphertext C (but no witness to its decryption), $\mathcal{SIM}$ must be able to perfectly simulate a PPK of C with any (malicious) receiver $\mathcal{R}'$ using parameters σ.

Our definitions build on the standard one for proofs of knowledge [8], except that our protocols are technically *arguments* of knowledge and we therefore restrict ourselves to consideration of provers running in probabilistic, polynomial time.

Definition 5.1 *Let $\Pi = (\mathcal{G}, \mathcal{S}, \mathcal{R})$ be a tuple of* PPT *algorithms. We say Π is a* proof of plaintext knowledge (PPK) *for encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ if the following conditions hold:*

(Completeness) *For all pk output by $\mathcal{K}(1^k)$, all σ output by $\mathcal{G}(pk)$, and all C with witness w to the decryption of C under pk we have $\langle \mathcal{S}(w), \mathcal{R} \rangle(pk, \sigma, C) = 1$ (when $\mathcal{R}$ outputs 1 we say it* accepts*).*

(Perfect zero-knowledge) *There exists a* PPT simulator $\mathcal{SIM}$ *such that, for all pk output by $\mathcal{K}(1^k)$, all computationally-unbounded $\mathcal{R}'$, and all m, r, the following distributions are equivalent:*

$$\{\sigma \leftarrow \mathcal{G}(pk); C := \mathcal{E}_{pk}(m; r) : \langle \mathcal{S}(m, r), \mathcal{R}' \rangle(pk, \sigma, C)\}$$

$$\{(\sigma, s) \leftarrow \mathcal{SIM}_1(pk); C := \mathcal{E}_{pk}(m; r) : \langle \mathcal{SIM}_2(s), \mathcal{R}' \rangle(pk, \sigma, C)\}.$$

(Witness extraction) *There exists a function $\kappa : \{0,1\}^* \to [0,1]$, a negligible function $\varepsilon(\cdot)$, and an expected-polynomial-time* knowledge extractor $\mathcal{KE}$ *such that, for all* PPT *algorithms $\mathcal{S}'$, with all but negligible probability over pk output by $\mathcal{K}(1^k)$, σ output by $\mathcal{G}(pk)$, and uniformly-distributed r, machine $\mathcal{KE}$ satisfies the following:*

> *Denote by $p_{pk,\sigma,r}$ the probability that $\mathcal{R}$ accepts when interacting with $\mathcal{S}'$ (using random tape r) on joint input pk, σ, C (where C is chosen by $\mathcal{S}'$). On input pk, σ, and access to $\mathcal{S}'_r$, the probability that $\mathcal{KE}$ outputs a witness to the decryption of C under pk is at least:*
>
> $$p_{pk,\sigma,r} - \kappa(pk) - \varepsilon(|pk|).$$

The zero-knowledge property stated above is quite strong: $\mathcal{SIM}_2$ achieves a perfect simulation without rewinding $\mathcal{R}'$. This definition is met by our constructions. More generally, one may weaken the zero-knowledge requirement to allow, for example, computational indistinguishability where $\mathcal{SIM}_2$ is given oracle access to $\mathcal{R}'$ (i.e., is allowed to rewind $\mathcal{R}'$). While this is an interesting direction, we do not pursue such a definition here.

A non-malleable PPK should satisfy the intuition that "anything proven by a man-in-the-middle adversary $\mathcal{M}$ is known by $\mathcal{M}$ (unless $\mathcal{M}$ simply copies a proof)." To formalize this idea, we allow $\mathcal{M}$ to interact with a simulator (whose existence is guaranteed by Definition 5.1) while simultaneously interacting with a (real) receiver $\mathcal{R}$. The goal of $\mathcal{M}$ is to output ciphertexts C, C' (which may be

chosen adaptively) and then successfully complete a PPK of C' to $\mathcal{R}$ while the simulator is executing a PPK of C to $\mathcal{M}$. The following definition states (informally) that if $\mathcal{R}$ accepts $\mathcal{M}$'s proof — yet the transcripts of the two proofs are different — then a knowledge extractor $\mathcal{KE}^*$ can extract a witness to the decryption of C'. The reason we have $\mathcal{M}$ interact with the simulator instead of the real sender $\mathcal{S}$ is that we must ensure that the knowledge is actually extracted from $\mathcal{M}$, and not from the real sender. This definition is based on the ideas of [38], who define a similar notion in the non-interactive setting.

Definition 5.2 *PPK $(\mathcal{G}, \mathcal{S}, \mathcal{R})$ is* non-malleable *if there exists a simulator $\mathcal{SIM}$ (satisfying the relevant portion of Definition 5.1), a function $\kappa^* : \{0,1\}^* \to [0,1]$, a negligible function $\varepsilon^*(\cdot)$, and an expected-polynomial-time* knowledge extractor *$\mathcal{KE}^*$ such that, for all* PPT *algorithms $\mathcal{M}$, with all but negligible probability over pk output by $\mathcal{K}(1^k)$, σ, s output by $\mathcal{SIM}_1(pk)$, and uniformly-distributed r, r', machine $\mathcal{KE}^*$ satisfies the following:*

> *Assume $\mathcal{M}$ (using random tape r') acts as a receiver with $\mathcal{SIM}_2(s; r)$ on joint input pk, σ, C and simultaneously as a sender with $\mathcal{R}$ on joint input pk, σ, C' (where C is a valid ciphertext and C, C' are adaptively chosen by $\mathcal{M}$). Let the transcripts of these two interactions be π and π'. Denote by p^* the probability (over the random tape of $\mathcal{R}$) that $\mathcal{R}$ accepts in the above interaction and $\pi \neq \pi'$. On input pk, σ, s, r, and access to $\mathcal{M}_{r'}$, the probability that $\mathcal{KE}^*$ outputs a witness to the decryption of C' under pk is at least:*
>
> $$p^* - \kappa^*(pk) - \varepsilon^*(|pk|).$$

We note that our definitions of zero-knowledge (in Definition 5.1) and non-malleability (in Definition 5.2) both consider the single-theorem case. The definitions may be modified for the multi-theorem case; however, the present definitions suffice for our intended applications.

Σ-protocols. Since we use Σ-protocols [31] in an essential way as part of our constructions, we briefly review their definition here. A Σ-protocol is a pair of PPT algorithms $(\mathcal{P}, \mathcal{V})$ which defines a three-move interactive protocol between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, where the prover sends the first message. Furthermore, the message q sent by $\mathcal{V}$ is a random challenge (without loss of generality, we may assume it is the contents of $\mathcal{V}$'s random tape). Let R be a binary relation computable in polynomial-time. Define L_R as the set of all y such that there exists an x with $R(y, x) = 1$. The string x is called a *witness* for y. The common input to $\mathcal{P}$ and $\mathcal{V}$ will be y, while x is known only to $\mathcal{P}$. Let (A, q, z) be a transcript of the conversation between $\mathcal{P}$ and $\mathcal{V}$. Upon completion of the protocol, the verifier outputs the single bit $\varphi(y, A, q, z)$, where $\varphi(\cdot)$ is an efficient, publicly-computable predicate with 1 denoting acceptance and 0 denoting rejection. If $\varphi(y, A, q, z) = 1$, we say that (A, q, z) is an *accepting conversation for y*.

All Σ-protocols we consider satisfy *special soundness* and *special honest-verifier zero-knowledge* (special-HVZK). For a particular y, let (A, q, z) and (A, q', z') denote two accepting conversations with $q \neq q'$. Special soundness implies that $y \in L_R$ and furthermore, that on input y and these two conversations, one can efficiently compute an x such that $R(y, x) = 1$. Special-HVZK means that there exists a PPT simulator $\mathcal{S}$ that, on input $y \in L_R$ and a randomly-chosen q, can generate a conversation (A, q, z) which is identically-distributed to a real conversation between $\mathcal{P}$ and $\mathcal{V}$ in which q is the challenge sent by $\mathcal{V}$. Note that the protocol of Figure 5.1 is a Σ-protocol satisfying the above requirements.

A note on complexity assumptions. Our proofs of security require hardness assumptions with respect to adversaries permitted to run in *expected polynomial time*. For example, we assume that the RSA function cannot be inverted with more than negligible probability by any expected-polynomial-time algorithm. The reason for this is our reliance on constant-round proofs of knowledge, for which only expected-polynomial-time knowledge extractors are currently known. Security definitions of *protocols*, however, are stated with respect to PPT adversaries.

5.3 Non-Malleable Proofs of Plaintext Knowledge

Our constructions follow a common paradigm. Recall the parameter σ which is shared by the sender and receiver and which is used as a common input during execution of the PPK. Embedded in σ will be a particular value y for which the simulator knows a witness x such that $R(y, x) = 1$. A PPK for ciphertext C will be a witness indistinguishable proof of knowledge of *either* a witness to the decryption of C *or* a witness for y, using the generic techniques for constructing such proofs [35]. Note that soundness of the protocol is not affected since a PPT adversary cannot derive a witness for y, while ZK simulation is easy (since the simulator knows the witness x).

As stated, this simple approach does not suffice to achieve non-malleability. To see why, consider a simulator interacting with man-in-the-middle $\mathcal{M}$ while $\mathcal{M}$ simultaneously interacts with verifier $\mathcal{V}$. Since the simulator must simulate a proof of "w or x" (where w is the witness to the decryption of the ciphertext C chosen by the adversary), the simulator must know x. However, if we use the knowledge extractor to extract from $\mathcal{M}$, who is proving "w' or x" (where w' is the witness to the decryption of C'), there is nothing which precludes extracting x! Note that without initial knowledge of x the simulator cannot properly perform the simulation; yet, if the simulator initially knows x, there is no contradiction in extracting this value from $\mathcal{M}$. Thus, a more careful approach is needed.

To overcome this obstacle, we borrow a technique from Chapter 4. Namely, the value y will depend on a parameter α which $\mathcal{M}$ cannot re-use; thus, the simulator proves knowledge of "w or x_α" while $\mathcal{M}$ is forced to prove knowledge of "w' or $x_{\alpha'}$", for some $\alpha' \neq \alpha$. This will be secure as

long as the following conditions hold: (1) it is possible for the simulator to know the witness x_α; yet (2) learning $x_{\alpha'}$ for *any* $\alpha' \neq \alpha$ results in a contradiction; furthermore, (3) $\mathcal{M}$ cannot duplicate the value α used by the simulator. Details follow in the remainder of this section.

5.3.1 Construction for the RSA Cryptosystem

We briefly review the RSA cryptosystem, extended to allow encryption of ℓ-bit messages using the techniques of Blum and Goldwasser [21]. The public key N is chosen as a product of two random $k/2$-bit primes (where k is the security parameter), and e is a prime number such that $|e| = O(k)$.[4] Let $hc(\cdot)$ be a hard-core bit [64] for the RSA permutation (so that, given r^e, $hc(r)$ is computationally indistinguishable from random; note that $hc(\cdot)$ may depend on information included with the public parameters), and define $hc^*(r) \stackrel{\text{def}}{=} hc(r^{e^{\ell-1}}) \circ \cdots \circ hc(r^e) \circ hc(r)$. Encryption of ℓ-bit message m is done by choosing a random element $r \in \mathbb{Z}_N^*$, computing $C = r^{e^\ell} \bmod N$, and sending $\langle C, c \stackrel{\text{def}}{=} hc^*(r) \oplus m \rangle$. It is easily shown that this scheme is semantically secure under the RSA assumption.

Our protocol uses a Σ-protocol for proving knowledge of e^ℓ-th roots which extends a previously-given Σ-protocol for proving knowledge of e-th roots [74]. To prove knowledge of $r = C^{1/e^\ell}$, the prover chooses a random element $r_1 \in \mathbb{Z}_N^*$ and sends $A = r_1^{e^\ell}$ to the verifier. The verifier replies with a challenge q selected randomly from $\mathbb{Z}_e$. The prover responds with $R = r^q r_1$ and the receiver verifies that $R^{e^\ell} \stackrel{?}{=} C^q A$. To see that special soundness holds, consider two accepting conversations (A, q, R) and (A, q', R'). Since $R^{e^\ell} = C^q A$ and $(R')^{e^\ell} = C^{q'} A$ we have $(R/R')^{e^\ell} = C^{q-q'}$. Noting that $|q - q'|$ is relatively prime to e^ℓ, Lemma 4.2 shows that the desired witness C^{1/e^ℓ} may be efficiently computed. Special-HVZK is demonstrated by the simulator which, on input C and a "target" challenge q, generates $A = R^{e^\ell}/C^q$ for random $R \in \mathbb{Z}_N^*$ and outputs transcript (A, q, R).

We now describe the non-malleable PPK in detail (cf. Figure 5.3). The public parameters σ (which may be included with the public key) are generated by selecting two random elements $g, h \in \mathbb{Z}_N^*$. Additionally, a function $H : \{0,1\}^* \to \mathbb{Z}_e$ from a family of universal one-way hash functions is chosen at random. Once σ is established, a PPK for ciphertext $\langle C, c \rangle$ proceeds as follows: first, a key-generation algorithm for a one-time signature scheme is run to yield verification key VK and signing key SK, and $\alpha = H(\mathsf{VK})$ is computed. The PPK will be a witness indistinguishable proof of knowledge of *either* $r = C^{1/e^\ell}$ *or* $x_\alpha \stackrel{\text{def}}{=} (g^\alpha h)^{1/e}$, following the paradigm of [35]. The sender chooses random elements $r_1, R_2 \in \mathbb{Z}_N^*$ along with a random element $q_2 \in \mathbb{Z}_e$. The sender then executes a real proof of knowledge of C^{1/e^ℓ} (using the known witness) and a simulated proof of knowledge of $(g^\alpha h)^{1/e}$ for challenge q_2 (using the simulator guaranteed by the special-HVZK property of the Σ-protocol). In more detail, element A_1 is computed as $r_1^{e^\ell}$ while A_2 is computed

[4] Below, we mention a modification of the protocol for the case of small e (e.g., $e = 3$).

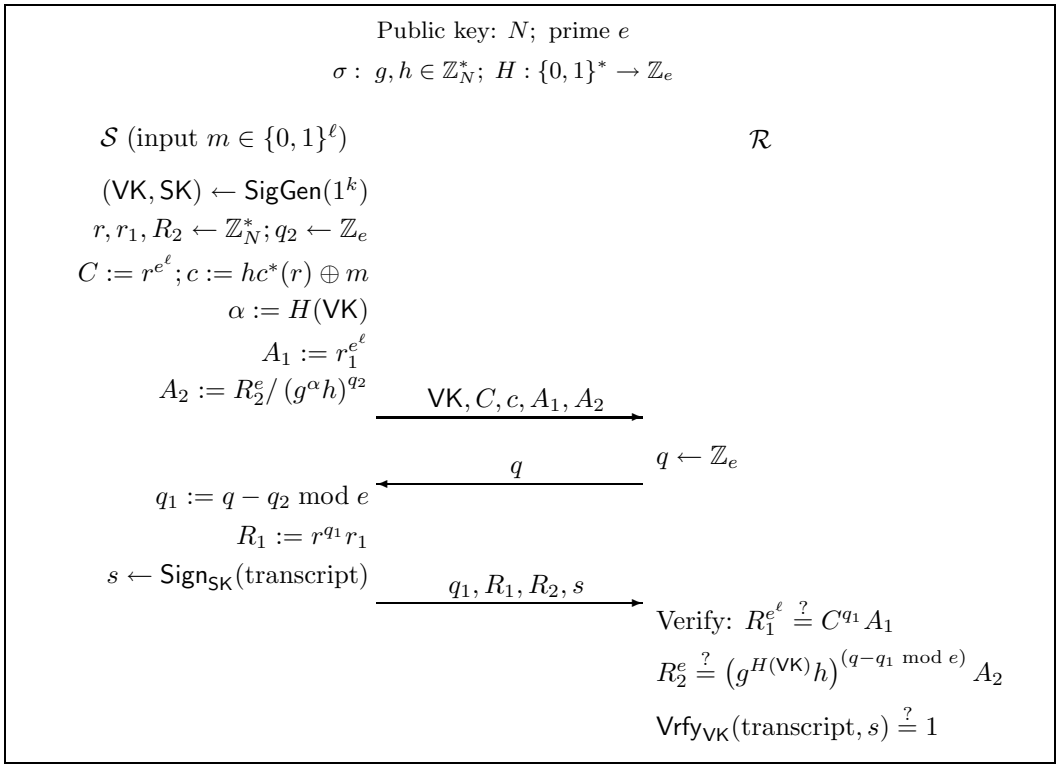

Figure 5.3: Non-malleable PPK for the RSA cryptosystem.

as $R_2^e/(g^\alpha h)^{q_2}$. These values are sent (along with VK, C, c) as the first message of the PPK. The receiver chooses challenge $q \in \mathbb{Z}_e$ as before. The sender sets $q_1 = q - q_2 \bmod e$ and answers with $R_1 = r^{q_1} r_1$ (completing the "real" proof of knowledge with challenge q_1) and R_2 (completing the "simulated" proof of knowledge with challenge q_2). The values q_1, R_1, R_2 are sent to the receiver. To complete the proof, the sender signs a transcript of the entire execution of the PPK (including C, c but not including VK itself) using SK and sends the signature to the receiver. The receiver verifies the correctness of the proofs by checking that $R_1^{e^\ell} \stackrel{?}{=} C^{q_1} A_1$ and $R_2^e \stackrel{?}{=} (g^\alpha h)^{(q - q_1 \bmod e)} A_2$. Finally, the receiver verifies the correctness of the signature on the transcript.

Theorem 5.1 *Assuming the hardness of the RSA problem for expected-polynomial-time algorithms, the protocol of Figure 5.3 is a PPK (with $\kappa(pk) = 1/e$) for the RSA encryption scheme outlined above.*

Proof We show that the given protocol satisfies Definition 5.1. Completeness is trivial. Simulatability (zero-knowledge) is achieved by making sure the simulator knows appropriate secret information about σ. For example, $\mathcal{SIM}_1(N, e)$ may choose random elements x, y and a hash function H and output $\sigma = \langle x^e, y^e, H \rangle$. Note that σ has the correct distribution. When $\mathcal{SIM}_2$ is

requested to run the PPK on any ciphertext, it can do so easily (without rewinding the potentially dishonest receiver) because it knows $(g^{\alpha}h)^{1/e}$ for any α. Witness indistinguishability of the proof (cf. [35]) implies that the simulated transcript is distributed identically to a real transcript.

The witness extraction property follows from the stronger result proved in Theorem 5.2. ■

Theorem 5.2 *Assuming (1) the hardness of the RSA problem for expected-polynomial-time algorithms, (2) the security of* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme, and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 5.3 is a non-malleable PPK (with* $\kappa^*(pk) = 1/e$*) for the RSA encryption scheme outlined above.*

Proof We use the following simulator (which is different from the one given in the proof of Theorem 5.1): $\mathcal{SIM}_1(N, e)$ chooses random hash function H, runs $\mathsf{SigGen}(1^k)$ to generate $(\mathsf{VK}, \mathsf{SK})$, and computes $\alpha = H(\mathsf{VK})$. Random elements $g, x \in \mathbb{Z}_N^*$ are chosen, and h is set equal to $g^{-\alpha}x^e$. Finally, $\sigma = \langle g, h, H\rangle$ is output along with state information $\mathsf{state} = \langle \mathsf{VK}, \mathsf{SK}, x\rangle$. Note that σ output by $\mathcal{SIM}_1$ has the correct distribution. Furthermore, given state, simulation of a single PPK by $\mathcal{SIM}_2$ for any ciphertext is easy: simply use verification key VK and then $(g^{\alpha}h)^{1/e}$ is known (cf. the proof of Theorem 5.1). Note that the simulation is perfect due to the witness indistinguishability of the proof.

Fix pk, σ, state, and randomness r for $\mathcal{SIM}_2$ (recall that ciphertext $\langle C, c\rangle$, for which $\mathcal{SIM}_2$ will be required to prove a witness, is chosen adaptively by $\mathcal{M}$). We are given adversary $\mathcal{M}$ using (unknown) random tape r' who interacts with both $\mathcal{SIM}_2(\mathsf{state}; r)$ and honest receiver $\mathcal{R}$. Once the challenge q' of $\mathcal{R}$ is fixed, the entire interaction is completely determined; thus, we may define $\pi(q')$ as the transcript of the conversation between $\mathcal{SIM}_2(\mathsf{state}; r)$ and $\mathcal{M}_{r'}$ when q' is the challenge of $\mathcal{R}$; analogously, we define $\pi'(q')$ as the transcript of the conversation between $\mathcal{M}_{r'}$ and $\mathcal{R}$ when q' is the challenge of $\mathcal{R}$. If certain messages have not been sent (e.g., $\mathcal{M}$ never sends a challenge q to $\mathcal{SIM}_2$) we simply set those messages to $\perp$.

The knowledge extractor $\mathcal{KE}^*$ is given $pk, \sigma, \mathsf{state}, r$, and access to $\mathcal{M}_{r'}$. When we say that $\mathcal{KE}^*$ *runs* $\mathcal{M}_{r'}$ *with challenge* q we mean that $\mathcal{KE}^*$ interacts with $\mathcal{M}_{r'}$ by running algorithm $\mathcal{SIM}_2(\mathsf{state}; r)$ and sending challenge q for $\mathcal{R}$. We stress that interleaving of messages (i.e., scheduling of messages to/from $\mathcal{R}$ and $\mathcal{SIM}_2$) is completely determined by $\mathcal{M}_{r'}$. $\mathcal{KE}^*$ operates as follows: First, $\mathcal{KE}^*$ picks a random value $q'^1 \in \mathbb{Z}_e$ and runs $\mathcal{M}_{r'}$ with challenge q'^1 (cf. Figure 5.4). If $\pi'(q'^1)$ is not accepting, or if $\pi'(q'^1) = \pi(q'^1)$, stop and output $\perp$. Otherwise, run the following:

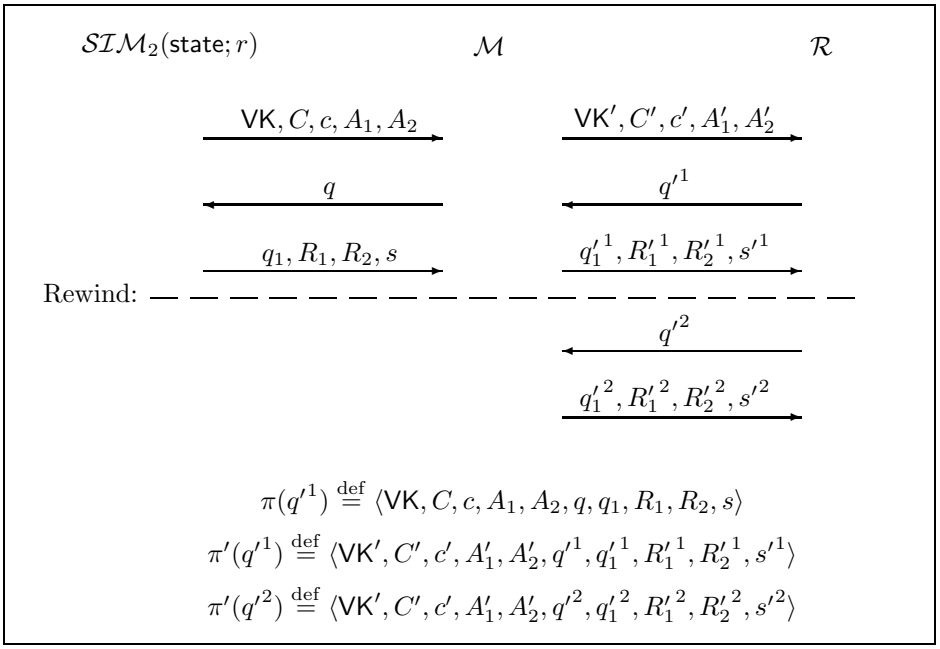

Figure 5.4: Knowledge extraction.

For $i = 0$ to $e - 1$:
 $q'^2 \leftarrow \mathbb{Z}_e$
 Run $\mathcal{M}_{r'}$ with challenge q'^2
 If $\pi'(q'^2)$ is accepting and $\pi(q'^2) \neq \pi'(q'^2)$ and $q'^2 \neq q'^1$:
  Output $\pi'(q'^2)$ and stop
 Run $\mathcal{M}_{r'}$ with challenge i
 If $\pi'(i)$ is accepting and $\pi(i) \neq \pi'(i)$ and $i \neq q'^1$:
  Output $\pi'(i)$ and stop
Output $\perp$ and stop.

At this point, $\mathcal{KE}^*$ has either output $\perp$ or has the transcripts $\pi(q'^1), \pi'(q'^1), \pi'(q'^2)$, where $\pi(q'^1) \neq \pi'(q'^1)$ and $\pi'(q'^1), \pi'(q'^2)$ are accepting transcripts with $q'^1 \neq q'^2$ (see Figure 5.4).

We first verify that the expected running time of $\mathcal{KE}^*$ until this point is polynomial in k. Fix $pk, \sigma, \mathsf{state}$, and r as above. Let p^* be the probability (over challenges q sent by $\mathcal{R}$) that $\mathcal{M}_{r'}$ gives a valid proof and $\pi(q) \neq \pi'(q)$. If $p^* > 1/e$, the expected number of iterations of the loop above (assuming this loop is executed) is at most $2/p^*$; furthermore, the probability of executing this loop (which is only done following an initial success)is exactly p^*. Thus, the expected running time is upper-bounded by $p^* \cdot \frac{poly(k)}{p^*} = poly(k)$, where $poly(k)$ is an upper bound on the running time of $\mathcal{M}$. On the other hand, if $p^* \leq 1/e$, the number of iterations of the loop above is at most e, yet the probability of executing the loop is at most $1/e$. Thus, the expected running time of Ext in this case is at most $\frac{1}{e} \cdot e \cdot poly(k) = poly(k)$.

Let Good be the event that $\mathcal{KE}^*$ does not output $\perp$. In this case, let the transcripts be as

indicated in Figure 5.4. Note that the probability of event Good is exactly p^* when $p^* > 1/e$ and 0 otherwise. In either case, we have $\Pr[\mathsf{Good}] \geq p^* - 1/e$.

Assuming event Good occurs, $\pi'({q'}^1)$ and $\pi'({q'}^2)$ are accepting transcripts with ${q'}^1 \neq {q'}^2$ and therefore we must have either ${q'_1}^1 \neq {q'_1}^2$ or ${q'}^1 - {q'_1}^1 \neq {q'}^2 - {q'_1}^2 \bmod e$ (or possibly both). In case ${q'}^1 - {q'_1}^1 = {q'}^2 - {q'_1}^2 \bmod e$ (denote this event by Real) and hence ${q'_1}^1 \neq {q'_1}^2$, we have the two equations:

$$\begin{aligned} \left({R'_1}^1\right)^{e^\ell} &= (C')^{{q'_1}^1} A'_1 \bmod N \\ \left({R'_1}^2\right)^{e^\ell} &= (C')^{{q'_1}^2} A'_1 \bmod N. \end{aligned}$$

Therefore, $(\Delta_R)^{e^\ell} = (C')^{\Delta_q}$, where $\Delta_R \stackrel{\text{def}}{=} {R'_1}^1 / {R'_1}^2$ (we assume all values in $\mathbb{Z}_N$ are in fact invertible; if not, N may be immediately factored and e^{th} roots easily computed) and $\Delta_q \stackrel{\text{def}}{=} {q'_1}^1 - {q'_1}^2$. Lemma 4.2 shows that this allows computation of $r' \stackrel{\text{def}}{=} (C')^{1/e^\ell}$. Note that once r' is known, $\mathcal{KE}^*$ may compute $hc^*(r')$ and hence determine the entire witness to the decryption of ciphertext $\langle C', c' \rangle$.

On the other hand, if ${q'}^1 - {q'_1}^1 \neq {q'}^2 - {q'_1}^2 \bmod e$ (denote this event by Fake), we have the two equations:

$$\begin{aligned} \left({R'_2}^1\right)^{e} &= \left(g^{\alpha'} h\right)^{{q'}^1 - {q'_1}^1} A'_2 \\ \left({R'_2}^2\right)^{e} &= \left(g^{\alpha'} h\right)^{{q'}^2 - {q'_1}^2} A'_2, \end{aligned}$$

where $\alpha' \stackrel{\text{def}}{=} H(\mathsf{VK}')$, from which $\mathcal{KE}^*$ may compute $(g^{\alpha'} h)^{1/e}$ (using the same techniques as above). Since $\mathsf{Good} = \mathsf{Real} \cup \mathsf{Fake}$, we have $\Pr[\mathsf{Real}] + \Pr[\mathsf{Fake}] \geq p^* - 1/e$. To complete the proof (since $\mathsf{Real} \cap \mathsf{Fake} = \emptyset$), we show that $\Pr[\mathsf{Fake}]$ (where the probability is over the random tape ω of $\mathcal{KE}^*$) is less than some negligible function with all but negligible probability over choice of $pk, \sigma, \mathsf{state}, r, r'$. We establish this via the following sequence of claims:

Claim 5.1 $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[\mathsf{VK} = \mathsf{VK}' \wedge \mathsf{Good}]$ *is negligible.*

Consider algorithm Forge which takes input VK, has access to a signing oracle, and runs as follows: emulating $\mathcal{KE}^*$, algorithm Forge generates pk, σ, and parameters state and r for $\mathcal{SIM}_2$. However, Forge does not run the key-generation procedure for the one-time signature scheme, but instead uses VK. Next, Forge runs the initial portion of $\mathcal{KE}^*$ by choosing random r' for $\mathcal{M}$, choosing q' randomly from $\mathbb{Z}_e$, and running $\mathcal{M}_{r'}$ with challenge q'. When a signature under VK is needed (during the single execution of $\mathcal{SIM}_2$), Forge obtains the required signature from its signing oracle. Once $\mathcal{M}_{r'}$ completes its execution, Forge stops and outputs the transcript $\pi'(q')$.

The probability that the transcript output by Forge contains a valid forgery under VK is at least $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[\mathsf{VK} = \mathsf{VK}' \wedge \mathsf{Good}]$. However, the security of the one-time signature scheme

guarantees that this is negligible. Note that, since no rewinding is involved and consequently Forge runs in strict polynomial time, the signature scheme need only be secure against PPT adversaries. □

Claim 5.2 $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[\mathsf{VK} \neq \mathsf{VK}' \wedge H(\mathsf{VK}) = H(\mathsf{VK}') \wedge \mathsf{Good}]$ *is negligible.*

The proof is similar to that of Claim 5.1, but is based on the security of the family of universal one-way hash functions. As in Claim 5.1, the family of universal one-way hash functions need only be secure against PPT adversaries. Details omitted. □

Claim 5.3 $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[H(\mathsf{VK}) \neq H(\mathsf{VK}') \wedge \mathsf{Fake}]$ *is negligible.*

Note that algorithm $\mathcal{KE}^*$, as described previously, does not require any secret information about N, e, g in order to run. Thus, we can consider the expected polynomial-time algorithm $\mathcal{KE}'$ which takes as input a modulus N, a prime e, and a (random) element $g \in \mathbb{Z}_N^*$ and otherwise runs identically to $\mathcal{KE}^*$. Clearly, the probability that both Fake and $H(\mathsf{VK}) \neq H(\mathsf{VK}')$ occur remains unchanged.

Let $\alpha = H(\mathsf{VK})$ and $\alpha' = H(\mathsf{VK}')$; define $\Delta \stackrel{\text{def}}{=} \alpha' - \alpha \neq 0$. By definition of event Fake, $\mathcal{KE}'$ may compute y' such that $(y')^e = g^{\alpha'} h$; but then:

$$y' \stackrel{\text{def}}{=} \left(g^{\alpha'} h\right)^{1/e} = \left(g^{\Delta} x^e\right)^{1/e} = \left(g^{\Delta}\right)^{1/e} x,$$

and therefore $y \stackrel{\text{def}}{=} y'/x$ satisfies $y^e = g^{\Delta}$. Note that $|\Delta|$ and e are relatively prime since $\Delta \in (-e, e)$. Thus, Lemma 4.2 shows that $g^{1/e}$ may be efficiently computed. In other words, whenever Fake and $\alpha' \neq \alpha$ occur, algorithm $\mathcal{KE}'$ inverts the given RSA instance g. Therefore, under the RSA assumption, $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[H(\mathsf{VK}) \neq H(\mathsf{VK}') \wedge \mathsf{Fake}]$ must be negligible. □

To complete the proof, note that the above claims imply that $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[\mathsf{Fake}] < \varepsilon(k)$ for some negligible function $\varepsilon(\cdot)$. But then:

$$\varepsilon(k) > \Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[\mathsf{Fake}] > \sqrt{\varepsilon(k)} \cdot \Pr_{pk,\sigma,\mathsf{state},r,r'}\left[\Pr_{\omega}[\mathsf{Fake}] > \sqrt{\varepsilon(k)}\right],$$

and hence $\Pr_{pk,\sigma,\mathsf{state},r,r'}\left[\Pr_{\omega}[\mathsf{Fake}] \leq \sqrt{\varepsilon(k)}\right] \geq 1 - \sqrt{\varepsilon(k)}$. In other words, with all but negligible probability over choice of $pk, \sigma, \mathsf{state}, r, r'$, the probability of event Fake is negligible ■

Using small values of e. For reasons of efficiency, RSA encryption is often performed using small (prime) values of e; e.g., $e = 3$. The protocol above may be adapted for this case by using the Σ-protocol given in [33, Section 6.3], which works for all e. Details are left to the reader.

5.3.2 Construction for the Rabin Cryptosystem

We propose a variant of the Rabin cryptosystem [104] for ℓ-bit messages. The public key N is chosen as a product of two primes p, q where $|p| = |q| = k/2$ (k is the security parameter) and $p, q = 3 \bmod 4$; such N are called *Blum integers*. Additionally, a positive integer t, where $t = \Theta(k)$, is included with the public key. Let $\mathcal{QR}_N$ denote the set of quadratic residues modulo N and let $\mathcal{J}_N(x)$, for $x \in \mathbb{Z}_N^*$, denote the Jacobi symbol of x. Note that $\mathcal{J}_N(x)$ can be efficiently computed even without knowledge of the factorization of N; furthermore, since N is a Blum integer, for all $x \in \mathcal{QR}_N$ we have $\mathcal{J}_N(x) = \mathcal{J}_N(-x) = 1$. It is well known that the squaring permutation on $\mathcal{QR}_N$ is one-way if and only if factoring Blum integers is hard [104]. Let $hc(\cdot)$ denote a hard-core bit of this permutation, and for $r \in \mathcal{QR}_N$ define $hc^*(r) \stackrel{\text{def}}{=} hc(r^{2^{\ell-1}}) \circ \cdots \circ hc(r^2) \circ hc(r)$. Encryption of ℓ-bit message m is performed by choosing random $r \in \mathbb{Z}_N^*$ and random bit b, computing $C = (-1)^b r^{2^{t+\ell}}$, and sending $\langle C, c \stackrel{\text{def}}{=} hc^*(r^{2^t}) \oplus m \rangle$ (note that $r^{2^t} \in \mathcal{QR}_N$). To decrypt ciphertext $\langle C, c \rangle$ (assuming the factorization of N is known), first check that $\mathcal{J}_N(C) = 1$; if not, output $\perp$. Otherwise, compute the unique element $r' \in \mathcal{QR}_N$ such that $(r')^{2^\ell} = \pm C$ and output $c \oplus hc^*(r')$. Semantic security of this scheme may be based on the hardness of factoring Blum integers, following [104].

Our PPK is based on a Σ-protocol for proving knowledge of 2^ℓ-th roots that, as far as we know, has not appeared previously. Given an element C with $\mathcal{J}_N(C) = 1$, successful execution of the protocol proves knowledge of $r' \in \mathcal{QR}_N$ such that $(r')^{2^\ell} = \pm C$; however, a real prover is required to know $r \in \mathbb{Z}_N^*$ such that $r^{2^{t+\ell}} = \pm C$. The Σ-protocol proceeds as follows: the prover first chooses random $r_1 \in \mathbb{Z}_N^*$ and sends $A = r_1^{2^{t+\ell+1}}$. The verifier responds with random challenge $q \in \mathbb{Z}_{2^t}$. The prover then answers with $R = r^q r_1$ and the verifier checks that $R^{2^{t+\ell+1}} \stackrel{?}{=} AC^{2q}$. Special-HVZK follows from the following simulator: given C and challenge q, choose random $R \in \mathbb{Z}_N^*$ and compute $A = R^{2^{t+\ell+1}}/C^{2q}$; the simulated transcript is (A, q, R). Special soundness will be clear from the proof of Theorem 5.3.

Using this basic Σ-protocol, we construct a non-malleable PPK following the paradigm outlined in Section 5.3.1. We give a high-level description of this PPK (cf. Figure 5.5): the public parameters σ are generated by selecting two random elements $g, h \in \mathcal{QR}_N$ (note that this may be done efficiently even without knowledge of the factorization of N). Additionally, a function $H : \{0,1\}^* \to \mathbb{Z}_{2^t}$ from a family of universal one-way hash functions is chosen at random. The PPK for ciphertext $\langle C, c \rangle$ begins by having the prover generate VK and SK for a one-time signature scheme and computing $\alpha = H(\mathsf{VK})$. The PPK will be a witness indistinguishable proof of knowledge of *either* $r \in \mathcal{QR}_N$ such that $r^{2^\ell} = \pm C$ *or* $x_\alpha \in \mathcal{QR}_N$ such that $x_\alpha^2 = g^\alpha h$.

Theorem 5.3 *Assuming (1) the hardness of factoring Blum integers for expected-polynomial-time*

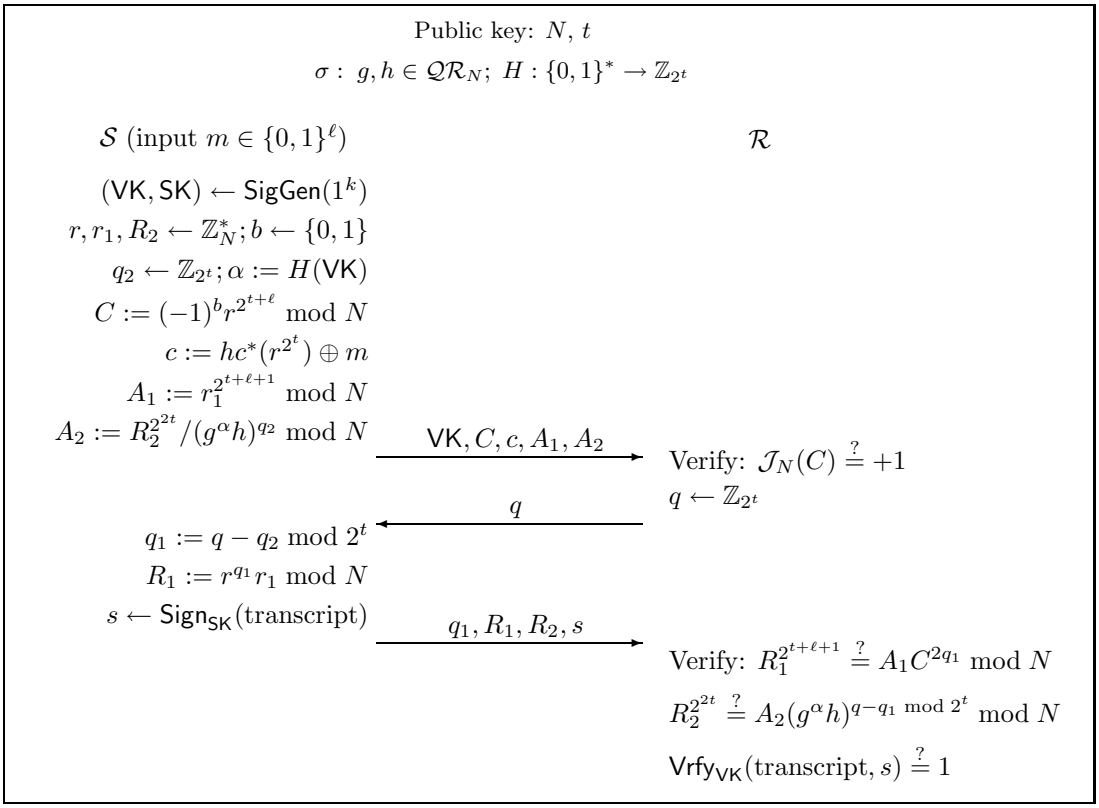

Figure 5.5: Non-malleable PPK for a Rabin-like cryptosystem.

algorithms, (2) the security of $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme, and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 5.5 is a non-malleable PPK (with* $\kappa(pk) = 1/2^t$*) for the Rabin-like encryption scheme outlined above.*

Proof The proof that the scheme is a PPK is similar to the proof of Theorem 5.1, and is omitted. The proof of non-malleability follows the outline of the proof of Theorem 5.2, so we only sketch the key differences here.

Our simulator is as follows: $\mathcal{SIM}_1(N, t)$ chooses random hash function H, runs the key generation algorithm for the one-time signature scheme to obtain $(\mathsf{VK}, \mathsf{SK})$, and computes $\alpha = H(\mathsf{VK})$. Random elements $g, x \in \mathcal{QR}_N$ are chosen, and h is set equal to $g^{-\alpha} x^{2^{2t}}$. Finally, $\sigma = \langle g, h, H \rangle$ is output along with state information $\mathsf{state} = \langle \mathsf{VK}, \mathsf{SK}, x \rangle$. Note that σ output by $\mathcal{SIM}_1$ is identically distributed to σ in a real execution. Furthermore, given state, simulation of a single PPK by $\mathcal{SIM}_2$ for any valid ciphertext is easy since $x \in \mathcal{QR}_N$ such that $x^{2^{2t}} = g^\alpha h$ is known.

Fix pk, σ, state, and randomness r for $\mathcal{SIM}_2$. Recall that ciphertext $\langle C, c \rangle$ for which $\mathcal{SIM}_2$ will be required to prove a witness, is chosen adaptively by $\mathcal{M}$; furthermore, a ciphertext is valid if and only if $\mathcal{J}_N(C) = 1$, which can be efficiently and publicly verified even without knowledge of the

factorization of N. We are given adversary $\mathcal{M}$ using (unknown) random tape r' who interacts with both $\mathcal{SIM}_2(\mathsf{state}; r)$ and honest receiver $\mathcal{R}$. For any query q' sent by $\mathcal{R}$, define $\pi(q')$ and $\pi'(q')$ as in the proof of Theorem 5.2. As in the proof of Theorem 5.2, we define a knowledge extractor $\mathcal{KE}^*$ which runs in expected polynomial time (the proof is similar to that given previously) and outputs either $\perp$ or $\langle \pi(q'^1), \pi'(q'^1), \pi'(q'^2) \rangle$, where $\pi(q'^1) \neq \pi'(q'^1)$ and $\pi'(q'^1), \pi'(q'^2)$ are accepting transcripts with $q'^1 \neq q'^2$. Let Good be the event that $\mathcal{KE}^*$ does not output $\perp$. By a similar proof as before, we have $\Pr[\mathsf{Good}] \geq p^* - 1/2^t$.

Assuming event Good occurs, $\pi'(q'^1)$ and $\pi'(q'^2)$ are accepting transcripts with $q'^1 \neq q'^2$ and therefore we must have either $q_1'^1 \neq q_1'^2$ or $q'^1 - q_1'^1 \neq q'^2 - q_1'^2 \bmod 2^t$ (or possibly both). In case $q'^1 - q_1'^1 = q'^2 - q_1'^2 \bmod 2^t$ and hence $q_1'^1 \neq q_1'^2$, we show how to compute a value $r \in \mathcal{QR}_N$ such that $r^{2^\ell} = \pm C'$. From the accepting transcripts, we have the two equations:

$$\begin{aligned} \left(R_1'^1\right)^{2^{t+\ell+1}} &= (C')^{2q_1'^1} A_1' \bmod N \\ \left(R_1'^2\right)^{2^{t+\ell+1}} &= (C')^{2q_1'^2} A_1' \bmod N. \end{aligned}$$

Together, these imply $(\Delta_R)^{2^{t+\ell+1}} = (C')^{2\Delta_q}$, where $\Delta_R \stackrel{\text{def}}{=} R_1'^1 / R_1'^2$ (we assume all values in $\mathbb{Z}_N$ are invertible; if not, N may be factored and the desired witness computed easily) and $\Delta_q \stackrel{\text{def}}{=} q_1'^1 - q_1'^2 \neq 0$. Without loss of generality, assume $\Delta_q > 0$ (if not, we can always re-write the above equation so this holds). Since $\Delta_q < 2^t$, we must have $\gcd(2^{t+\ell+1}, \Delta_q) = 2^i$ with $0 \leq i < t$. We may then efficiently compute integers a, b such that $a2^{t+\ell+1} + b\Delta_q = 2^i$. Then:

$$((C')^2)^{2^i} = ((C')^2)^{a2^{t+\ell+1} + b\Delta_q} = ((C')^{2a}(\Delta_R)^b)^{2^{t+\ell+1}} = \gamma^{2^{t+\ell+1}} \bmod N, \tag{5.1}$$

where $\gamma \stackrel{\text{def}}{=} (C')^{2a}(\Delta_R)^b$. If $i = 0$, we are done since $r = \gamma^{2^t}$ satisfies $r \in \mathcal{QR}_N$ and $r^{2^\ell} = \pm C'$. Otherwise, using the fact that squaring is a permutation on $\mathcal{QR}_N$, (5.1) implies that:

$$(C')^2 = (\gamma^2)^{2^{t+\ell-i}} = (\gamma^{2^{t+\ell-i}})^2 \bmod N.$$

Since $\mathcal{J}_N(C') = 1$, we have $\pm C' = \gamma^{2^{t+\ell-i}} = (\gamma^{2^{t-i}})^{2^\ell}$. Note that, since $t - i > 0$, $\gamma^{2^{t-i}}$ can be efficiently computed; furthermore, $\gamma^{2^{t-i}} \in \mathcal{QR}_N$. Thus, $r = \gamma^{2^{t-i}}$ is the desired value.

On the other hand, denote event $q'^1 - q_1'^1 \neq q'^2 - q_1'^2 \bmod 2^t$ by Fake. To complete the proof, we show that $\Pr[\mathsf{Fake}]$ (where the probability is over the random tape ω of Ext) is less than some negligible function with all but negligible probability over choice of $pk, \sigma, \mathsf{state}, r, r'$. We establish this using Claims 5.1 and 5.2 from the proof of Theorem 5.2 along with the following:

Claim 5.4 $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[H(\mathsf{VK}) \neq H(\mathsf{VK}') \wedge \mathsf{Fake}]$ *is negligible.*

Note that algorithm $\mathcal{KE}^*$ does not require any information beyond N, t, and g in order to run; in particular, element $g \in \mathcal{QR}_N$ may be selected at random. Thus, we can consider the expected

polynomial-time algorithm $\mathcal{KE}'$ which takes input modulus N, t, and (random) element $g \in \mathcal{QR}_N$ and otherwise runs identically to $\mathcal{KE}^*$. Clearly, the probability that both Fake and $H(\mathsf{VK}) \neq H(\mathsf{VK}')$ occur remains unchanged.

We show that $r \in \mathcal{QR}_N$ such that $r^2 = g$ may be computed when both Fake and $H(\mathsf{VK}) \neq H(\mathsf{VK}')$ occur. Since inverting the squaring permutation on $\mathcal{QR}_N$ is hard under the factoring assumption, this will give the desired contradiction. Let $\alpha = H(\mathsf{VK})$ and $\alpha' = H(\mathsf{VK}')$; define $\Delta_\alpha \stackrel{\text{def}}{=} \alpha' - \alpha \neq 0$. From the two accepting transcripts, we have the two equations:

$$\begin{aligned} \left(R_2'^{\,1}\right)^{2^{2t}} &= \left(g^{\alpha'}h\right)^{q'^1 - q_1'^1 \bmod 2^t} A_2' \bmod N \\ \left(R_2'^{\,2}\right)^{2^{2t}} &= \left(g^{\alpha'}h\right)^{q'^2 - q_1'^2 \bmod 2^t} A_2' \bmod N, \end{aligned}$$

which yield:

$$\begin{aligned} \left(\Delta_R\right)^{2^{2t}} &= \left(g^{\alpha'}h\right)^{\Delta_q} \\ &= \left(g^{\Delta_\alpha} x^{2^{2t}}\right)^{\Delta_q} \bmod N, \end{aligned} \tag{5.2}$$

where $\Delta_R \stackrel{\text{def}}{=} R_2'^{\,1}/R_2'^{\,2}$ and $\Delta_q \stackrel{\text{def}}{=} q'^1 - q_1'^1 - q'^2 + q_1'^2 \neq 0 \bmod 2^t$. Re-arranging (5.2) gives:

$$g^{\Delta_\alpha \Delta_q} = \left(x^{-\Delta_q}\Delta_R\right)^{2^{2t}}.$$

Without loss of generality, assume $\Delta_\alpha \Delta_q > 0$ (if not, we can re-write the equation above so that this holds). Since $\Delta_\alpha, \Delta_q < 2^t$, we must have $\gcd(2^{2t}, \Delta_\alpha \Delta_q) = 2^i$ with $0 \leq i < 2t-1$. We may then efficiently compute integers a, b such that $a2^{2t} + b\Delta_\alpha\Delta_q = 2^i$. Then:

$$g^{2^i} = g^{a2^{2t} + b\Delta_\alpha\Delta_q} = (g^a(\Delta_R x^{-\Delta_q})^b)^{2^{2t}} = \gamma^{2^{2t}}, \tag{5.3}$$

where $\gamma \stackrel{\text{def}}{=} g^a(\Delta_R x^{-\Delta_q})^b$. If $i = 0$, we have computed a square root $\gamma^{2^{2t-1}} \in \mathcal{QR}_N$ of g. Otherwise, we may re-write (5.3) as:

$$g^{2^i} = (\gamma^2)^{2^{2t-1}}. \tag{5.4}$$

Using the fact that squaring is a permutation over $\mathcal{QR}_N$, (5.4) implies that:

$$g = (\gamma^2)^{2^{2t-i-1}},$$

and hence $\gamma^{2^{2t-i-1}} \in \mathcal{QR}_N$ is the desired square root of g.

Thus, under the factoring assumption, $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[H(\mathsf{VK}) \neq H(\mathsf{VK}') \wedge \mathsf{Fake}]$ must be negligible. The remainder of the proof follows the proof of Theorem 5.2 exactly. ■

5.3.3 Construction for the Paillier Cryptosystem

We provide a brief review of the Paillier cryptosystem [101]. The public key N is a product of two primes p, q where $|p| = |q| = k$ (where k is the security parameter). Additionally, the public key contains an element $g \in \mathbb{Z}_{N^2}^*$ such that the order of g (in $\mathbb{Z}_{N^2}^*$) is a non-zero multiple of N. It can be shown that the function $f_g : \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$ defined by $f_g(x, y) = g^x y^N \bmod N^2$ is a bijection. For $C \in \mathbb{Z}_{N^2}^*$, we define by $[C]_g$ the unique $x \in \mathbb{Z}_N$ for which there exists a $y \in \mathbb{Z}_N^*$ such that $f_g(x, y) = C$. The *decisional composite residuosity assumption* states that the distributions $\{C \leftarrow \mathbb{Z}_{N^2}^* : (C, [C]_g)\}$ and $\{C \leftarrow \mathbb{Z}_{N^2}^*, x \leftarrow \mathbb{Z}_N : (C, x)\}$ are computationally indistinguishable. This naturally gives rise to the following encryption scheme whose semantic security is equivalent to this assumption: to encrypt message $m \in \mathbb{Z}_N$, choose a random $y \in \mathbb{Z}_N^*$ and send $C = g^m y^N \bmod N^2$. It can be shown that, given the factorization of N, decryption may be done efficiently [101]. We note that an encryption scheme based on the *computational* composite residuosity assumption (i.e., the assumed hardness of computing $[C]_g$ for random C) is also possible using the hardcore bits of $f_g(\cdot)$; the PPK given below can be modified for this case in a straightforward way. In either case, the security of the PPK itself depends on the weaker computational assumption.

We build on a Σ-protocol [34] that, given C, proves knowledge of m, y such that $C = g^m y^N \bmod N^2$. The basic Σ-protocol proceeds as follows: the prover chooses random $x \in \mathbb{Z}_N$, $u \in \mathbb{Z}_{N^2}^*$ and sends $B = g^x u^N \bmod N^2$ as the first message. The verifier responds with a random challenge $q \in \mathbb{Z}_N$, and the prover answers with $w = x + qm \bmod N$ and $z = u y^q g^t \bmod N^2$, where t is such that $x + qm = w + tN$ (over the integers). The verifier then checks that $g^w z^N \stackrel{?}{=} B C^q \bmod N^2$. Special-HVZK follows from the following simulator: given C and challenge q, choose random $w \in \mathbb{Z}_N, z \in \mathbb{Z}_{N^2}$ and set $B = g^w z^N / C^q \bmod N^2$; the simulated transcript is (B, q, w, z). Special soundness will be clear from the proof of Theorem 5.4.

Using this basic Σ-protocol, we construct a non-malleable PPK following the paradigm outlined in Section 5.3.1. We give a high-level description of this PPK (cf. Figure 5.6): the public parameters σ are generated by selecting two random elements $h_0, h_1 \in \mathbb{Z}_{N^2}^*$. Additionally, a function $H : \{0,1\}^* \to \mathbb{Z}_N$ from a family of universal one-way hash functions is chosen at random. The PPK for ciphertext C begins by having the prover generate VK and SK for a one-time signature scheme and computing $\alpha = H(\mathsf{VK})$. The PPK will be a witness indistinguishable proof of knowledge of $(m, y) \in \mathbb{Z}_N \times \mathbb{Z}_N^*$ such that *either* $g^m y^N = C \bmod N^2$ *or* $g^m y^N = h_0^\alpha h_1 \bmod N^2$.

Theorem 5.4 *Assuming (1) the hardness of the computational composite residuosity assumption for expected-polynomial-time algorithms, (2) the security of* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme, and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure*

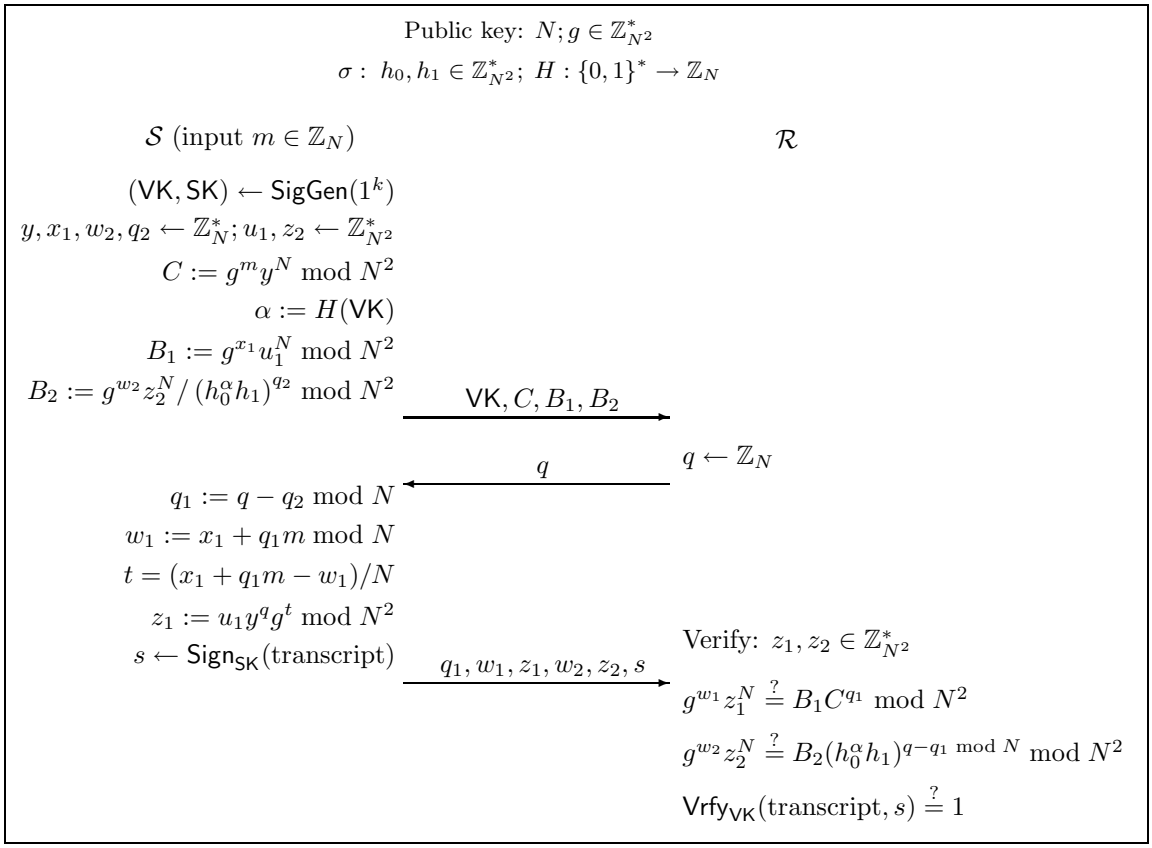

Figure 5.6: Non-malleable PPK for the Paillier cryptosystem.

5.6 is a non-malleable PPK (with $\kappa(pk) = 1/N$) for the Paillier encryption scheme.

Proof The proof that the scheme is a PPK is similar to the proof of Theorem 5.1, and is omitted. The proof of non-malleability follows the outline of the proof of Theorem 5.2, so we only sketch the key differences here.

Our simulator is as follows: $\mathcal{SIM}_1(N, g)$ chooses random hash function H, runs the key generation algorithm for the one-time signature scheme to obtain $(\mathsf{VK}, \mathsf{SK})$, and computes $\alpha = H(\mathsf{VK})$. Random elements $h_0 \in \mathbb{Z}^*_{N^2}$, $m' \in \mathbb{Z}_N$, and $y' \in \mathbb{Z}^*_N$ are chosen, and h_1 is set equal to $h_0^{-\alpha} g^{m'} (y')^N$. Finally, $\sigma = \langle h_0, h_1, H \rangle$ is output along with state information $\mathsf{state} = \langle \mathsf{VK}, \mathsf{SK}, m', y' \rangle$. Note that σ output by $\mathcal{SIM}_1$ is identically distributed to σ in a real execution. Furthermore, given state, simulation of a single PPK by $\mathcal{SIM}_2$ for any valid ciphertext is easy since m', y' such that $g^{m'} (y')^N = h_0^\alpha h_1$ are known.

Fix pk, σ, state, and randomness r for $\mathcal{SIM}_2$. Recall that ciphertext C for which $\mathcal{SIM}_2$ will be required to prove a witness is chosen adaptively by $\mathcal{M}$. We are given adversary $\mathcal{M}$ using (unknown) random tape r' who interacts with both $\mathcal{SIM}_2(\mathsf{state}; r)$ and honest receiver $\mathcal{R}$. For any query q

sent by $\mathcal{R}$, define $\pi(q)$ and $\pi'(q)$ as in the proof of Theorem 5.2. As in the proof of Theorem 5.2, we define a knowledge extractor $\mathcal{KE}^*$ which runs in expected polynomial time (the proof is similar to that given previously) and outputs either $\perp$ or $\langle \pi(q'^1), \pi'(q'^1), \pi'(q'^2) \rangle$, where $\pi(q'^1) \neq \pi'(q'^1)$ and $\pi'(q'^1), \pi'(q'^2)$ are accepting transcripts with $q'^1 \neq q'^2$. Let Good be the event that $\mathcal{KE}^*$ does not output $\perp$. As above, we have $\Pr[\mathsf{Good}] \geq p^* - 1/N$.

Assuming event Good occurs, $\pi'(q'^1)$ and $\pi'(q'^2)$ are accepting and therefore we must have either $q_1'^1 \neq q_1'^2$ or $q'^1 - q_1'^1 \neq q'^2 - q_1'^2 \bmod N$ (or possibly both). In case $q'^1 - q_1'^1 = q'^2 - q_1'^2 \bmod N$ and hence $q_1'^1 \neq q_1'^2$, we show how to compute a values m, y such that $g^m y^N = C' \bmod N^2$. From the accepting transcripts, we have the two equations:

$$\begin{aligned} g^{w_1'^1}(z_1'^1)^N &= B_1'(C')^{q_1'^1} \bmod N^2 \\ g^{w_1'^2}(z_1'^2)^N &= B_1'(C')^{q_1'^2} \bmod N^2. \end{aligned}$$

Together, these imply $g^{\Delta_w}(\Delta_z)^N = (C')^{\Delta_q}$, where $\Delta_w \stackrel{\text{def}}{=} w_1'^1 - w_1'^2$, $\Delta_z \stackrel{\text{def}}{=} z_1'^1 / z_1'^2$ (recall $z_1'^1, z_1'^2 \in \mathbb{Z}_{N^2}^*$), and $\Delta_q \stackrel{\text{def}}{=} q_1'^1 - q_1'^2$. Without loss of generality, assume $\Delta_q > 0$ (if not, we can always rearrange the above equation so this holds). If $\gcd(\Delta_q, N) \neq 1$, then we have found a factor of N and, in particular, can use the factorization of N to compute the desired m, y [101]. Otherwise, we may compute integers a, b such that $a\Delta_q + bN = 1$; then:

$$C' = (C')^{a\Delta_q + bN} = g^{a\Delta_w}((\Delta_z)^a (C')^b)^N \bmod N^2,$$

giving the desired values $m = a\Delta_w \bmod N$ and $y = (\Delta_z)^a (C')^b \bmod N^2$.

On the other hand, denote event $q'^1 - q_1'^1 \neq q'^2 - q_1'^2 \bmod N$ by Fake. To complete the proof, we show that $\Pr[\mathsf{Fake}]$ (where the probability is over the random tape ω of $\mathcal{KE}^*$) is less than some negligible function with all but negligible probability over choice of $pk, \sigma, \mathsf{state}, r, r'$. We establish this using Claims 5.1 and 5.2 from the proof of Theorem 5.2 along with the following:

Claim 5.5 $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[H(\mathsf{VK}) \neq H(\mathsf{VK}') \wedge \mathsf{Fake}]$ *is negligible.*

Note that algorithm $\mathcal{KE}^*$ does not require any information beyond N, g, h_0 in order to run; in particular, element h_0 may be selected at random. Thus, we can consider the expected polynomial-time algorithm $\mathcal{KE}'$ which takes input modulus N and (random) elements $g, h_0 \in \mathbb{Z}_{N^2}^*$ and otherwise runs identically to $\mathcal{KE}^*$. Clearly, the probability that both Fake and $H(\mathsf{VK}) \neq H(\mathsf{VK}')$ occur remains unchanged.

We show that m, y such that $g^m y^r = h_0$ may be efficiently computed when both Fake and $H(\mathsf{VK}) \neq H(\mathsf{VK}')$ occur, contradicting the computational composite residuosity assumption. Let $\alpha = H(\mathsf{VK})$ and $\alpha' = H(\mathsf{VK}')$; define $\Delta_\alpha \stackrel{\text{def}}{=} \alpha' - \alpha \neq 0$. From the two accepting transcripts, we

have the two equations:

$$\begin{aligned} g^{{w_2'}^1}({z_2'}^1)^N &= B_2'(h_0^{\alpha'}h_1)^{{q'}^1-{q_1'}^1} \bmod N^2 \\ g^{{w_2'}^2}({z_2'}^2)^N &= B_2'(h_0^{\alpha'}h_1)^{{q'}^2-{q_1'}^2} \bmod N^2, \end{aligned}$$

which yield

$$\begin{aligned} g^{\Delta_w}(\Delta_z)^N &= (h_0^{\alpha'}h_1)^{\Delta_q} \\ &= (h_0^{\Delta_\alpha}g^{m'}(y')^N)^{\Delta_q} \bmod N^2, \end{aligned} \tag{5.5}$$

where $\Delta_w \stackrel{\text{def}}{=} {w_2'}^1 - {w_2'}^2$, $\Delta_z \stackrel{\text{def}}{=} {z_2'}^1/{z_2'}^2$, and $\Delta_q \stackrel{\text{def}}{=} {q'}^1 - {q_1'}^1 - {q'}^2 + {q_1'}^2$. Re-arranging (5.5) gives

$$h_0^{\Delta_\alpha\Delta_q} = g^{\Delta_w - m'\Delta_q}(\Delta_z/(y')^{\Delta_q})^N \bmod N^2$$

(recall $y' \in \mathbb{Z}_N^*$ and is therefore invertible modulo N^2). Without loss of generality, assume $\Delta_\alpha\Delta_q > 0$ (if not, we can re-write the equation above so this holds). If $\gcd(\Delta_\alpha\Delta_q, N) \neq 1$, then we have found a factor of N and, in particular, can use the factorization of N to compute the desired m, y. Otherwise, we may compute integers a, b such that $a\Delta_\alpha\Delta_q + bN = 1$; then

$$h_0 = h_0^{a\Delta_\alpha\Delta_q + bN} = g^{a\Delta_w - am'\Delta_q}((\Delta_z/(y')^{\Delta_q})^a h_0^b)^N,$$

yielding the desired values $m = a\Delta_w - am'\Delta_q \bmod N$ and $y = (\Delta_z/(y')^{\Delta_q})^a h_0^b \bmod N^2$.

The remainder of the proof exactly follows the proof of Theorem 5.2. ■

5.3.4 Construction for the El Gamal Cryptosystem

We briefly review the El Gamal cryptosystem [51]. Given primes p, q such that $p = 2q+1$ and $|q| = k$ (where k is a security parameter), we may define finite, cyclic group $\mathbb{G}$ as the unique subgroup of $\mathbb{Z}_p^*$ with order q. The public key of the El Gamal scheme is created by choosing a random generator $g_0 \in \mathbb{G}$, choosing a random $x \in \mathbb{Z}_q$, and setting $g_1 = g_0^x$. The public key is p, q, g_0, g_1. To encrypt a message $m \in \mathbb{G}$, select a random $y \in \mathbb{Z}_q$ and send $\langle g_0^y, g_1^y m\rangle$. Decryption of $\langle C_0, C_1\rangle$ is simply done by computing $m = C_1/C_0^x$. Semantic security of this scheme is equivalent to the DDH assumption. We note that an encryption scheme whose security may be based on the CDH assumption is also possible (using hard-core bits of the Diffie-Hellman key-exchange protocol), and the PPK given below may be easily extended for this case.

Our protocol builds on the Σ-protocol for proving knowledge of discrete logarithms given in Figure 5.1. Our non-malleable PPK (cf. Figure 5.7) requires public parameters σ consisting of a generator $h \in \mathbb{G}$. Additionally, a function $H : \{0,1\}^* \to \mathbb{Z}_q$ from a family of universal one-way hash functions is chosen at random. The PPK for ciphertext C_0, C_1 begins by having the prover generate

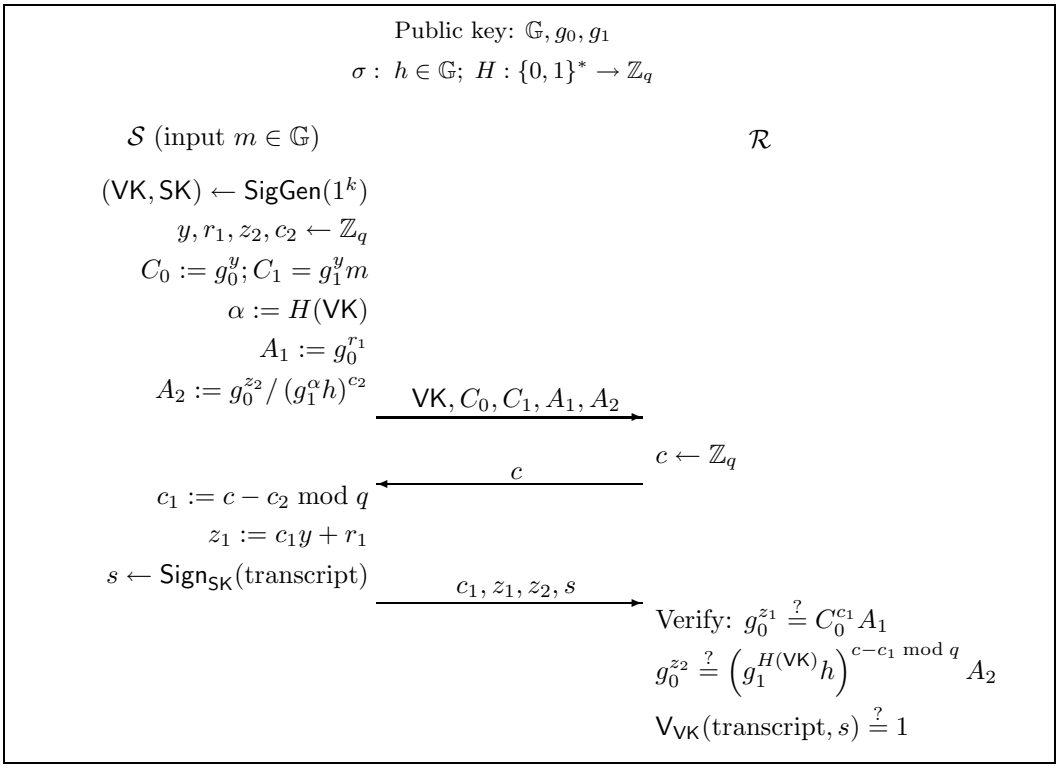

Figure 5.7: Non-malleable PPK for the El Gamal cryptosystem.

VK and SK for a one-time signature scheme and computing $\alpha = H(\mathsf{VK})$. The PPK will be a witness indistinguishable proof of knowledge of $y \in \mathbb{Z}_q$ such that *either* $g_0^y = C_0$ *or* $g_0^y = g_1^\alpha h$.

Theorem 5.5 *Assuming (1) the hardness of the discrete logarithm problem in* $\mathbb{G}$ *for expected-polynomial-time algorithms, (2) the security of* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme, and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 5.7 is a non-malleable PPK (with* $\kappa(pk) = 1/q$*) for the El Gamal encryption scheme.*

Proof The proof that the scheme is a PPK is similar to the proof of Theorem 5.1, and is omitted. Proof of non-malleability follows the outline of the proof of Theorem 5.2, so we only sketch the key differences here.

Our simulator is as follows: $\mathcal{SIM}_1(p, q, g_0, g_1)$ chooses random hash function H, runs the key-generation algorithm for the one-time signature scheme to obtain $(\mathsf{VK}, \mathsf{SK})$, and computes $\alpha = H(\mathsf{VK})$. Random $x \in \mathbb{Z}_q$ is chosen, and h is set equal to $g_1^{-\alpha} g_0^x$. Finally, $\sigma = \langle h, H \rangle$ is output along with state information $\mathsf{state} = \langle \mathsf{VK}, \mathsf{SK}, x \rangle$. Note that σ output by $\mathcal{SIM}_1$ is identically distributed to σ in a real execution. Furthermore, given state, simulation of a single PPK by $\mathcal{SIM}_2$ for any valid ciphertext is easy since $\log_{g_0}(g_1^\alpha h)$ is known.

Fix pk, σ, state, and randomness r for $\mathcal{SIM}_2$. Recall that ciphertext C_0, C_1 for which $\mathcal{SIM}_2$ will be required to prove a witness is chosen adaptively by $\mathcal{M}$. We are given adversary $\mathcal{M}$ using (unknown) random tape r' who interacts with both $\mathcal{SIM}_2(\mathsf{state}; r)$ and honest receiver $\mathcal{R}$. For any query c sent by $\mathcal{R}$, define $\pi(c)$ and $\pi'(c)$ as in the proof of Theorem 5.2. As in the proof of Theorem 5.2, we may define a knowledge extractor $\mathcal{KE}^*$ which runs in expected polynomial time (the proof is similar to that given previously) and outputs either $\perp$ or $\langle \pi(c'^1), \pi'(c'^1), \pi'(c'^2) \rangle$, where $\pi(c'^1) \neq \pi'(c'^1)$ and $\pi'(c'^1), \pi'(c'^2)$ are accepting transcripts with $c'^1 \neq c'^2$. Let Good be the event that $\mathcal{KE}^*$ does not output $\perp$. By a similar proof as above, we have $\Pr[\mathsf{Good}] \geq p^* - 1/q$.

Assuming event Good occurs, $\pi'(c'^1)$ and $\pi'(c'^2)$ are accepting and therefore we must have either ${c'_1}^1 \neq {c'_1}^2$ or $c'^1 - {c'_1}^1 \neq c'^2 - {c'_1}^2 \bmod q$ (or possibly both). In case $c'^1 - {c'_1}^1 = c'^2 - {c'_1}^2 \bmod q$ and hence ${c'_1}^1 \neq {c'_1}^2$, we show how to compute a value r such that $r = \log_{g_0} C'_0$. From the accepting transcripts, we have the two equations:

$$\begin{aligned} g_0^{{z'_1}^1} &= (C'_0)^{{c'_1}^1} A'_1 \\ g_0^{{z'_1}^2} &= (C'_0)^{{c'_1}^2} A'_1. \end{aligned}$$

Together, these imply $g_0^{\Delta_z} = (C'_0)^{\Delta_c}$, where $\Delta_z \stackrel{\text{def}}{=} {z'_1}^1 - {z'_1}^2$ and $\Delta_c \stackrel{\text{def}}{=} {c'_1}^1 - {c'_2}^2$. Hence, $\log_{g_0} C'_0 = \Delta_z / \Delta_c$ and we are done.

On the other hand, if $c'^1 - {c'_1}^1 \neq c'^2 - {c'_1}^2 \bmod q$, we denote this event by Fake. To complete the proof, we show that $\Pr[\mathsf{Fake}]$ (where the probability is over the random tape ω of $\mathcal{KE}^*$) is less than some negligible function with all but negligible probability over choice of $pk, \sigma, \mathsf{state}, r, r'$. We establish this using Claims 5.1 and 5.2 from the proof of Theorem 5.2 along with the following:

Claim 5.6 $\Pr_{pk,\sigma,\mathsf{state},r,r',\omega}[H(\mathsf{VK}) \neq H(\mathsf{VK}') \wedge \mathsf{Fake}]$ *is negligible.*

Note that algorithm $\mathcal{KE}^*$ does not require any information about g_0 or g_1 in order to run; in particular, elements g_0, g_1 may be selected at random. Thus, we can consider the expected polynomial-time algorithm $\mathcal{KE}'$ which takes input modulus p, q, g_0, g_1 and otherwise runs identically to $\mathcal{KE}^*$. Clearly, the probability that both Fake and $H(\mathsf{VK}) \neq H(\mathsf{VK}')$ occur remains unchanged.

We show that $\log_{g_0} g_1$ may be computed when both Fake and $H(\mathsf{VK}) \neq H(\mathsf{VK}')$ occur, contradicting the discrete logarithm assumption. Let $\alpha = H(\mathsf{VK})$ and $\alpha' = H(\mathsf{VK}')$; define $\Delta_\alpha \stackrel{\text{def}}{=} \alpha' - \alpha \neq 0$. From the two accepting transcripts, we have the two equations:

$$\begin{aligned} g_0^{{z'_2}^1} &= (g_1^{\alpha'} h)^{{c'}^1 - {c'_1}^1} A'_2 \\ g_0^{{z'_2}^2} &= (g_1^{\alpha'} h)^{{c'}^2 - {c'_1}^2} A'_2, \end{aligned}$$

which yield:

$$g_0^{\Delta_z} = (g_1^{\alpha'} h)^{\Delta_c}$$

$$= (g_1^{\Delta_\alpha} g_0^x)^{\Delta_c}$$
$$= g_1^{\Delta_\alpha \Delta_c} g_0^{x\Delta_c},$$

where $\Delta_z \stackrel{\text{def}}{=} {z_2'}^1 - {z_2'}^2$ and $\Delta_c \stackrel{\text{def}}{=} {c'}^1 - {c_1'}^1 - {c'}^2 + {c_1'}^2 \neq 0 \bmod q$. From this, we may immediately conclude that $\log_{g_0} g_1 = \frac{\Delta_z - x\Delta_c}{\Delta_\alpha \Delta_c}$.

The remainder of the proof exactly follows that of Theorem 5.2. ■

5.4 Applications

In this section, we discuss applications of non-malleable PPKs to the construction of (1) chosen-ciphertext-secure, interactive encryption protocols, (2) password-based authentication and key exchange protocols in the public-key model, (3) strong deniable-authentication protocols, and (4) identification protocols secure against man-in-the-middle attacks. In each case, we show how the protocols of the previous section may be used for the intended application.

Concurrent composition. In our intended applications, the man-in-the-middle adversary may conduct multiple PPKs *concurrently* and witness extraction will be required from each such execution; furthermore, extraction of this witness is typically required as soon as the relevant proof is completed. If arbitrary interleaving of the proofs is allowed, extracting all witnesses may require exponential time due to the nested rewinding of the prover (a similar problem is encountered in simulation of concurrent zero-knowledge proofs [49]). To avoid this problem, we introduce *timing constraints* [49] in our protocols. These are explained in detail in the relevant sections.

5.4.1 Chosen-Ciphertext-Secure Interactive Encryption

Previous work. Definitions for chosen-ciphertext-secure public-key encryption were given by Naor and Yung [99] and and Rackoff and Simon [105]. Naor and Yung also give a construction achieving non-adaptive chosen-ciphertext security [99]. The notion of non-malleable public-key encryption was put forth by Dolev, Dwork, and Naor [44]. The first construction of a non-malleable (and hence chosen-ciphertext-secure [16]) public-key encryption[5] scheme was given in [44], and improved constructions appear in [110, 38]. These constructions, however, are based on general assumptions and are therefore impractical. Efficient non-malleable encryption schemes are known in the random oracle model (e.g., OAEP [14]); we work in the standard model only. Prior to this work, the only efficient non-malleable encryption scheme in the standard model was [36], whose security is based on the DDH assumption. Subsequent to the present work, Cramer and Shoup [37] have proposed non-malleable encryption schemes based on alternate assumptions; yet, it is important to note that

[5] Unless stated otherwise, "encryption" refers to non-interactive encryption.

the security of these schemes is based on the hardness of *decisional* problems, whereas we present schemes whose security may be based on the hardness of *computational* problems.

Chosen-ciphertext security for *interactive* public-key encryption has been considered previously [60, 75, 62], although formal definitions do not appear until [44]. Using an interactive PPK to achieve chosen-ciphertext-secure, interactive public-key encryption has been previously proposed [60, 75, 62]; however, such an approach is *not* secure against adaptive chosen-ciphertext attacks unless a non-malleable PPK is used. A practical, non-malleable interactive public-key encryption scheme (which does not use proofs of knowledge) is given by [44]; however, this protocol requires a signature from the receiver, making it unsuitable for use in a deniable authentication protocol (see below). Moreover, this protocol [44] requires the receiver — for each encrypted message — to (1) compute an existentially unforgeable signature and (2) run the key generation procedure for a public-key encryption scheme (often the most computationally intensive step). Our protocols, optimized for particular number-theoretic assumptions, are more efficient.

Definitions. A number of definitional approaches to chosen-ciphertext security in the interactive setting are possible. For example, the notion of non-malleability [44] may be extended for the case of interactive encryption. An oracle-based definition is also possible, and we sketch such a definition here.[6]

We have a sender, a receiver (where the receiver has published public-key pk), and a man-in-the-middle adversary $\mathcal{M}$ who controls all communication between them (cf. Chapter 2). To model this, we define an *encryption oracle* and a *decryption oracle* to which $\mathcal{M}$ is given access. The encryption oracle $\mathcal{E}_{b,pk}$ plays the role of the sender. The adversary may interact with this oracle multiple times at various points during its execution, and may interleave requests to this oracle with requests to the decryption oracle in an arbitrary manner. At the outset of protocol execution, the encryption oracle picks a bit b at random. An instance of the adversary's interaction with the oracle proceeds as follows: first, the adversary chooses two messages m_0, m_1 and sends these to $\mathcal{E}_{b,pk}$. The encryption oracle then executes the encryption protocol for message m_b. The adversary, however, need not act as an honest receiver. Since the adversary may have multiple concurrent interactions with $\mathcal{E}_{b,pk}$, the oracle must maintain state between the adversary's oracle calls. Formally, each instance of the encryption oracle is associated with a unique label; furthermore, each message the adversary sends to the oracle must include a label indicating to which encryption-instance the message corresponds. When $\mathcal{E}_{b,pk}$ sends the final message for a given instance of its execution, we say that instance is *completed*.

[6] To obtain an equivalent definition using the language of non-malleability, we would need to define a notion of non-malleability with respect to *vectors* of ciphertexts. Such a definition becomes cumbersome to work with in the interactive setting.

The *decryption oracle* $\mathcal{D}_{sk}$ plays the role of a receiver. Since the adversary may perform multiple concurrent executions of the protocol with the oracle, $\mathcal{D}_{sk}$ must also record state between oracle calls, and each message sent by the adversary must also include a label indicating to which decryption-instance the message corresponds (as above). The adversary need not act as an honest sender. Each time a given decryption-instance is completed, the decryption oracle computes the decryption (using the secret key) and sends the resulting message (or $\perp$, if the transcript was invalid) to the adversary.

The adversary succeeds if it can guess the bit b. Clearly, some limitations must be placed on the adversary's access to the decryption oracle or else the adversary may simply forward messages between $\mathcal{E}_{b,pk}$ and $\mathcal{D}_{sk}$ and therefore trivially determine b. At any point during the adversary's execution, the set of transcripts of completed encryption-instances of $\mathcal{E}_{b,pk}$ is well defined. Additionally, when a decryption-instance of $\mathcal{D}_{sk}$ is completed, the transcript of that interaction is well defined. Upon completing a decryption-instance, let $\{\pi_1, \ldots, \pi_\ell\}$ denote the transcripts (not including instance labels) of all completed encryption-instances. We allow the adversary to receive the decryption corresponding to a decryption-instance with transcript π' only if $\pi' \neq \pi_i$ for $1 \leq i \leq \ell$.

Definition 5.3 *Let* $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ *be an interactive, public-key encryption scheme. We say that* Π *is* CCA2-secure *if, for any* PPT *adversary* A*, the following is negligible (in* k*):*

$$\left|\Pr\left[(sk, pk) \leftarrow \mathcal{K}(1^k); b \leftarrow \{0,1\} : A^{\mathcal{E}_{b,pk}, \mathcal{D}_{sk}}(1^k, pk) = b\right] - 1/2\right|,$$

where A*'s access to* $\mathcal{D}_{sk}$ *is restricted as discussed above.*

A straightforward hybrid argument shows that it is sufficient to consider adversaries which are allowed only a single access to the encryption oracle. We consider this type of adversary in what follows.

Constructions. The protocols of Figures 5.3, 5.5–5.7 are in fact chosen-ciphertext-secure interactive encryption schemes (under the relevant assumptions) when the adversary is given *sequential* access to the decryption oracle. More precisely, given a semantically-secure encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ (which may be the RSA, Rabin, Paillier, or El Gamal scheme) and non-malleable PPK $(\mathcal{G}, \mathcal{S}, \mathcal{R})$ for this encryption scheme, the interactive, chosen-ciphertext-secure encryption scheme $(\mathcal{K}', \mathcal{E}', \mathcal{D}')$ is defined as follows:

- $\mathcal{K}'(1^k)$ runs $\mathcal{K}(1^k)$ to generate pk, sk. Additionally, $\mathcal{G}(pk)$ is run to give σ. The public key pk' is $\langle pk, \sigma \rangle$ and the secret key is sk.
- To encrypt message m under public key $pk' = \langle pk, \sigma \rangle$, the sender computes $C \leftarrow \mathcal{E}_{pk}(m)$ and then executes algorithm $\mathcal{S}$ for C using parameters σ (i.e., the sender proves knowledge of a witness to the decryption of C).

- To decrypt, the receiver uses $\mathcal{R}$ to determine whether to accept or reject the proof of ciphertext C. If the receiver accepts (we say the proof *succeeds*), the receiver outputs $\mathcal{D}_{sk}(C)$. If the proof is rejected (we say the proof *fails*), the receiver outputs $\perp$.

As mentioned previously, timing constraints are needed to ensure security against an adversary who is given *concurrent* access to the decryption oracle. In this case, we require that $\mathcal{S}$ respond to the challenge (i.e., send the third message of the protocol) within time α from when the second message of the protocol is sent. If $\mathcal{S}$ does not respond in this time, the proof is rejected. Additionally, the protocols are modified so that a fourth message is sent from the receiver to the sender; this message is simply an *acknowledgment* message which is ack if the sender's proof was verified to be correct and $\perp$ otherwise. Furthermore, $\mathcal{R}$ delays the sending of this message until at least time β has elapsed from when the second message of the protocol was sent (with $\beta > \alpha$).

We stress that, when concurrent access to the decryption oracle is allowed, the decryption oracle enforces the above timing constraints by (1) rejecting any proofs for which more than time α has elapsed between sending the second message and receiving the third message, and (2) the decrypted ciphertext is not given to the adversary until after the acknowledgment message is sent (in particular, until time β has elapsed since sending the second message).

Theorem 5.6 *Assuming (1) the hardness of the RSA problem for expected-polynomial-time algorithms, (2) the security of* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme, and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 5.3 (with* $|e| = \Theta(k)$*) is an interactive encryption scheme secure against sequential chosen-ciphertext attacks. If timing constraints are enforced as outlined above, the protocol is secure against concurrent chosen-ciphertext attacks.*

Proof We prove security for the more challenging case of concurrent access to the decryption oracle. The protocol Π of Figure 5.3 is a PPK for encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ in which $\mathcal{K}(1^k)$ outputs as the public key a k-bit modulus N and a k-bit prime e. Encryption of ℓ-bit message m is done by choosing random $r \in \mathbb{Z}_N^*$ and sending $\widetilde{C} = \langle r^{e^\ell}, hc^*(r) \oplus m \rangle$, where $hc^*(\cdot)$ is a hard-core function for the RSA permutation. Assuming the hardness of the RSA problem for expected-polynomial-time algorithms, $(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is semantically secure against expected-polynomial-time adversaries. We transform any PPT adversary $\mathcal{A}$ mounting a CCA2 attack against Π into an expected-polynomial-time adversary $\mathcal{A}'$ attacking the semantic security of $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. Furthermore, we show that the advantage of $\mathcal{A}'$ is not negligible if the advantage of $\mathcal{A}$ is not negligible. This will immediately imply CCA2 security of Π.

Let $t(k)$, which is polynomial in k, be a bound on the number of times $\mathcal{A}$ accesses the decryption

oracle when run on security parameter 1^k; without loss of generality, we assume that $t(k) \geq k$ (for convenience, in the remainder of the proof we suppress the dependence on k and simply write t). In the real experiment Expt_0, the final output b' of $\mathcal{A}$ is completely determined by $pk' = \langle pk, \sigma \rangle$, random coins r' for $\mathcal{A}$, the vector of challenges $\vec{q} = q_1, \ldots, q_t$ used during the t instances $\mathcal{A}$ interacts with the decryption oracle, and the randomness used by the encryption oracle (this includes the bit b, the randomness ω used for the encryption, and the randomness used for execution of the PPK). Let Succ denote the event that $b' = b$, and let $\Pr_0[\mathsf{Succ}]$ denote the probability of this event in the real experiment.

We modify the real experiment, giving Expt_1, as follows. Key generation is done by running $\mathcal{K}(1^k)$ to generate pk, sk. Additionally, $\mathcal{SIM}_1(pk)$ (cf. the proof of Theorem 5.2) is run to generate parameters σ and state (in the real experiment σ was generated by $\mathcal{G}(pk)$). The public key pk' is $\langle pk, \sigma \rangle$ and the secret key is sk. The adversary's calls to the decryption oracle are handled as in Expt_0 (in particular, any ciphertext $\widetilde{C}$ may be decrypted since sk is known), but the adversary's encryption oracle call will be handled differently. When $\mathcal{A}$ calls the encryption oracle on messages m_0, m_1, we pick b randomly, compute $\widetilde{C}^* = \mathcal{E}_{pk}(m_b; \omega)$ for random ω, and simulate the PPK for $\widetilde{C}^*$ using algorithm $\mathcal{SIM}_2(\mathsf{state}; r)$ with randomly-chosen r. Now, the final output b' of $\mathcal{A}$ is completely determined by $pk' = \langle pk, \sigma \rangle$, random coins r' for $\mathcal{A}$, the vector of challenges $\vec{q}$ used by the decryption oracle, the values b and ω used in computing $\widetilde{C}^*$, and the values state and r used by $\mathcal{SIM}_2$ in simulating the encryption oracle. Since $\mathcal{SIM}$ yields a perfect simulation of a real execution of the PPK, we have $\Pr_1[\mathsf{Succ}] = \Pr_0[\mathsf{Succ}]$, where the first probability refers to the probability of an event in Expt_1.

We now describe our adversary $\mathcal{A}'$ attacking the semantic security of $(\mathcal{K}, \mathcal{E}, \mathcal{D})$. Given the public key pk, adversary $\mathcal{A}'$ runs $\mathcal{SIM}_1(pk)$ to generate parameters σ and state. $\mathcal{A}'$ then fixes the randomness r' of $\mathcal{A}$, and runs $\mathcal{A}$ on input $pk' = \langle pk, \sigma \rangle$. Simulation of the encryption oracle for $\mathcal{A}$ is done as follows: when $\mathcal{A}$ submits two messages m_0, m_1, adversary $\mathcal{A}'$ simply forwards these to its encryption oracle and receives in return a ciphertext $\widetilde{C}^*$ (note that the encryption oracle thus implicitly defines values b^* and ω^*). Then, $\mathcal{A}'$ simulates the PPK for $\widetilde{C}^*$ using algorithm $\mathcal{SIM}_2(\mathsf{state}; r)$ for randomly-chosen coins r. Simulation of the decryption oracle for $\mathcal{A}$ is done by choosing a random vector of queries $\vec{q}^*$ and attempting to extract the relevant witnesses (in expected polynomial time) from the PPKs given by $\mathcal{A}$. Details of the simulation are described below. In case the simulation is successful, the final output b' is just the final output of $\mathcal{A}$; if the simulation is not successful, the final output b' is a randomly-chosen bit. As we show below, the simulation will succeed with sufficiently high probability such that if the advantage of $\mathcal{A}$ (in attacking the CCA2 security of Π) is not negligible then the advantage of $\mathcal{A}'$ (in attacking the semantic security of

$(\mathcal{K}, \mathcal{E}, \mathcal{D})$ is not negligible as well. This will complete the proof.

It remains to show how to simulate the decryption oracle. Our proof requires techniques used in an analysis of concurrent composition of zero-knowledge proofs [49]. We assume that $\mathcal{A}$ controls the scheduling of all messages to and from all the oracles; so, for example, the decryption oracle does not send its next message until $\mathcal{A}$ requests it. Define the i^{th} *instance* of the decryption oracle as the i^{th} time $\mathcal{A}$ requests the second message (i.e., the challenge) of the PPK be sent by the decryption oracle. In any transcript of the execution of $\mathcal{A}$, we let C_i denote the ciphertext sent by $\mathcal{A}$ in the i^{th} instance of the decryption oracle. For any instance of the decryption oracle, we say the instance *succeeds* if (1) an honest receiver would accept the instance, (2) the transcript of the instance is different from the transcript (if it yet exists) of the interaction of $\mathcal{A}$ with the encryption oracle, and (3) the timing constraints are satisfied for that instance. Otherwise, we say the instance *fails*.

Recall that the simulator has values $\langle pk, \sigma, r', \vec{q}^*, \mathsf{state}, r\rangle$ and has access to an encryption oracle which, on input m_0, m_1, outputs $\mathcal{E}_{pk}(m_{b^*}; \omega^*)$ for random b^* and ω^*. Note that the value state defines a key VK which is used by $\mathcal{SIM}_2$ when giving its simulated proof. The values $pk, \sigma, r', \mathsf{state}$, and r are fixed throughout the simulation. When we say the simulator *interacts with* $\mathcal{A}$ *using* $\langle \vec{q}, \omega, b\rangle$ we mean that the simulator runs $\mathcal{A}$ as in Expt_1; that is, encryption oracle query m_0, m_1 is answered by encrypting m_b using randomness ω and then running $\mathcal{SIM}_2(\mathsf{state}; r)$, and the challenge sent by the i^{th} instance of the decryption oracle is q_i. We note that decryption requests cannot be immediately satisfied; this will not be a problem, as we show below.

We begin with simulation of the first instance. The simulator chooses random $\vec{q}, \omega, b$ and interacts with $\mathcal{A}$ using $\langle \vec{q}, \omega, b\rangle$. If $\mathcal{A}$ makes a call m_0, m_1 to the encryption oracle *before* q_1 is requested, the simulator forwards m_0, m_1 to its encryption oracle and receives in return a ciphertext $\widetilde{C}^*$; we then say the *ciphertext is defined at instance 1*. Once the ciphertext is defined, the simulator no longer needs to choose values ω, b and, in effect, interacts with $\mathcal{A}$ using $\langle \vec{q}, \omega^*, b^*\rangle$. If the verification key VK_1 used by $\mathcal{A}$ in the first instance is equal to VK defined by state, the first instance is declared *conditionally delinquent* and the simulator proceeds to simulation of instance 2.

If the ciphertext is not defined at instance 1, the simulator interacts with $\mathcal{A}$ using $\langle \vec{q}, \omega, b\rangle$ until the first instance either succeeds or conclusively fails. Note that decryption of ciphertexts $\widetilde{C}_i$ with $i > 1$ is not required since such a request would imply that time β has elapsed since the sending of the second message of instance i, but this would mean that time α has already elapsed since the second message of instance 1 was sent (and therefore the first instance has either succeeded or failed by that point). If the first instance succeeds, the simulator proceeds with witness extraction as described below. If the first instance fails, the simulator chooses new, random $\vec{q}, \omega, b$ and interacts with $\mathcal{A}$ using $\langle \vec{q}, \omega, b\rangle$. This is repeated for a total of at most t^4/τ times (using new, random $\vec{q}, \omega, b$ each time) or

until the first instance succeeds, where $\tau = \tau(k)$ is an inverse polynomial whose value we will fix at the end of the proof and $t = t(k)$ is a bound on the number of times $\mathcal{A}$ interacts with the decryption oracle. If the first instance ever succeeds, the simulator proceeds with witness extraction. Otherwise, the first instance is declared *conditionally delinquent* and the simulator proceeds to simulation of the second instance.

If the ciphertext is defined at instance 1, the simulator proceeds as above, but uses values ω^*, b^* to interact with $\mathcal{A}$ (where these values are defined by ciphertext $\widetilde{C}^*$ received from the simulator's encryption oracle, as discussed previously).

If the first instance ever succeeds, witness extraction will be performed. Assume the first instance succeeded when interacting with $\mathcal{A}$ using $\langle \vec{q}, \omega, b \rangle$. The simulator then does the following:

For $n = 0$ to $e - 1$:
  $q_1' \leftarrow \mathbb{Z}_e$
  Interact with $\mathcal{A}$ using $\langle q_1', q_2, \ldots, q_t, \omega, b \rangle$
    If the first instance succeeds and $q_1' \neq q_1$, output the transcript and stop
  Interact with $\mathcal{A}$ using $\langle n, q_2, \ldots, q_t, \omega, b \rangle$
    if the first instance succeeds and $n \neq q_1$, output the transcript and stop
Output $\perp$ and stop

If $\perp$ is not output, the simulator attempts to compute a witness to the decryption of $\widetilde{C}_1$ as in the proof of Theorem 5.2. If such a witness is computed, we say the first instance is *extracted* and the simulator proceeds with simulation of instance 2. If $\perp$ is output, or if $\perp$ is not output but a witness to the decryption of $\widetilde{C}_1$ cannot be computed, the entire simulation is aborted; we call this a *failure to extract*.

In general, when we are ready to simulate the i^{th} instance (assuming the entire simulation has not been aborted), each of the first $i-1$ instances has been classified as either extracted or conditionally delinquent. If instance j is extracted, the simulator knows the decryption of $\widetilde{C}_j$ and can send it to $\mathcal{A}$ upon successful completion of that instance in the current simulation. On the other hand, if instance j is classified as conditionally delinquent, then with sufficiently high probability that instance will never succeed.

We say the *ciphertext is defined before instance i* if, for some $j \leq i$, the ciphertext is defined at j. At the beginning of simulation of the i^{th} instance, if the ciphertext is not defined before instance $i-1$, the simulator chooses random $q_i, \ldots, q_t, \omega, b$ and interacts with $\mathcal{A}$ using $\langle q_1^*, \ldots q_{i-1}^*, q_i, \ldots, q_t, \omega, b \rangle$. If $\mathcal{A}$ makes a call m_0, m_1 to the decryption oracle before q_i is requested, the simulator forwards m_0, m_1 to its encryption oracle and receives in return a ciphertext $\widetilde{C}^*$; we then say the ciphertext is defined at instance i. If the verification key VK_i used by $\mathcal{A}$ during the i^{th} instance of the decryption oracle is equal to VK defined by state, the i^{th} instance is declared *conditionally delinquent* and the simulator proceeds with simulation of the next instance.

If the ciphertext is not defined before instance i, the simulator continues to interact with $\mathcal{A}$ using $\langle q_1^*, \ldots q_{i-1}^*, q_i, \ldots, q_t, \omega, b\rangle$ until the i^{th} instance either succeeds or conclusively fails. If a success occurs, witness extraction is performed as described below. In case the i^{th} instance fails, the simulator chooses new, random $q_i, \ldots, q_t, \omega, b$ and interacts with $\mathcal{A}$ using $\langle q_1^*, \ldots, q_{i-1}^*, q_i, \ldots, q_t, \omega, b\rangle$. This is repeated for a total of at most t^4/τ times or until the i^{th} instance succeeds. If the i^{th} instance ever succeeds, the simulator proceeds with witness extraction as described below. Otherwise, the i^{th} instance is declared *conditionally delinquent* and the simulator proceeds to simulation of the next instance.

Note that during simulation of instance i, decryption of ciphertexts C_j with $j > i$ is not required since such a request would imply that time β has elapsed since the sending of the second message of instance j, but this would mean that time α has already elapsed since the second message of instance i was sent (and therefore the i^{th} instance has either succeeded or failed by that point). However, the simulator may be required to decrypt ciphertext $\widetilde{C}_j$ with $j < i$. In case instance j is extracted, this is no problem, since the simulator knows the witness to the decryption of $\widetilde{C}_j$. On the other hand, when j is conditionally delinquent, there is a problem. We handle this as follows: if conditionally delinquent instance j succeeds *before* q_i is sent, the entire simulation is aborted; we call this a *classification failure*. If a conditionally delinquent instance j succeeds *after* q_i is sent, we consider this an *exceptional event at instance j during simulation of i*, and do not include it in the count of failed trials. However, if $3t^3/\tau$ such exceptional events occur for any j, the entire simulation is aborted; we call this an *exception at j during simulation of i*.

If the ciphertext is defined before instance i, the simulator proceeds as above but using ω^*, b^* (where these values are defined by the ciphertext C^* obtained from the simulator's encryption oracle, as discussed previously).

If the i^{th} instance ever succeeds, witness extraction will be performed. Assume the first instance succeeded when interacting with $\mathcal{A}$ using $\langle \vec{q}, \omega, b\rangle$. The simulator then does the following (*success* here means that the i^{th} instance succeeds and no exceptional events occurred during the interaction with $\mathcal{A}$):

For $n = 0$ to $e - 1$:
  $q_i' \leftarrow \mathbb{Z}_e$
  Interact with $\mathcal{A}$ using $\langle q_1, q_{i-1}, q_i', q_{i+1}, \ldots, q_t, \omega, b\rangle$
    If the first instance succeeds and $q_i' \neq q_i$, output the transcript and stop
  Interact with $\mathcal{A}$ using $\langle q_1, q_{i-1}, n, q_{i+1}, \ldots, q_t, \omega, b\rangle$
    if the first instance succeeds and $n \neq q_i$, output the transcript and stop
Output $\perp$ and stop

If $\perp$ is not output, the simulator attempts to compute a witness to the decryption of $\widetilde{C}_i$ as in the proof of Theorem 5.2. If such a witness is computed, we say the i^{th} instance is *extracted* and

proceed with simulation of the next instance. If $\perp$ is output, or if $\perp$ is not output but a witness to the decryption of $\widetilde{C}_i$ cannot be computed, the entire simulation is aborted; we call this a *failure to extract*.

Once all t instances have been simulated, if the ciphertext is not defined before t, the simulator simply interacts with $\mathcal{A}$ on input $\langle \vec{q}^*, \perp, \perp \rangle$ until $\mathcal{A}$ makes call m_0, m_1 to the encryption oracle. The simulator forwards these values to its encryption oracle, receiving in return $\widetilde{C}^*$. Simulation of the encryption oracle for $\mathcal{A}$ is done using $\mathcal{SIM}_2$, as above. In this case, we say the ciphertext is defined at $t+1$.

As long as the entire simulation is not aborted, the result is a perfect simulation of the view of $\mathcal{A}$ in Expt_1 with random variables $\Omega^* \stackrel{\text{def}}{=} \langle pk, \sigma, r', \mathsf{state}, r, w^*, b^* \rangle$. We now show that: (1) the expected running time of the above simulation is polynomial in t and $1/\tau$, and (2) with all but negligible probability over Ω^* and for some negligible function $\mu(\cdot)$, the simulation fails with probability at most $3\tau/4 + \mu(k)$. Fixing $\tau(k)$ to an appropriate inverse polynomial function then yields a correct simulation with sufficiently high probability.

Claim 5.7 *The expected running time of the simulation is polynomial in t and $1/\tau$.*

For simulation of each instance, at most $4t^4/\tau$ trials are run before either aborting, declaring the instance conditionally delinquent, or attempting to extract. Say extraction is attempted at instance i because the i^{th} instance succeeded (and no exceptional events occurred) when interacting with $\mathcal{A}$ using $\langle \vec{q}, \omega, b \rangle$. Let p_i denote the probability, over choice of q_i', that the i^{th} instance succeeds and no exceptional events occur when interacting with $\mathcal{A}$ using $\langle q_1, \ldots, q_{i-1}, q_i', q_{i+1}, \ldots, q_t, \omega, b \rangle$. If $p_i > 1/e$, extraction requires expected number of steps at most $2/p_i$. Since extraction with these values of $\vec{q}, \omega, b$ is performed with probability at most $p_i \cdot (4t^4/\tau)$, the contribution to the expected running time is at most $(4t^4 p_i/\tau) \cdot 2/p_i = 8t^4/\tau$. If $p_i \leq 1/e$, extraction requires at most e steps, and the contribution to the expected running time is then at most $(4t^4/e\tau) \cdot e = 4t^4/\tau$. In either case, the contribution to the expected running time for simulation of any instance is polynomial in t and $1/\tau$. Since there are at most t instances, the entire simulation has expected running time polynomial in t and $1/\tau$. □

Claim 5.8 *With all but negligible probability over Ω^*, the probability of a* classification failure *is at most $\tau/4 + \varepsilon_4(k)$, where $\varepsilon_4(\cdot)$ is negligible.*

The analysis follows [49]. Say the ciphertext is defined at instance v, where $1 \leq v \leq t+1$. For $i < v$, define the *values defined before i* as $q_1^*, \ldots, q_{i-1}^*$ and the *variables not defined before i* as variables $q_i, \ldots, q_t, \omega, b$; for $i \geq v$, define the *values defined before i* as $q_1^*, \ldots, q_{i-1}^*, \omega^*, b^*$ and the *variables*

not defined before i as $q_i, \ldots, q_t$. For each i, recursively define d_i as the probability (over variables not defined before i) that instance i succeeds, conditioned on the values defined before i and on the event that no delinquent instance j ($j < i$) succeeds. Define an instance i to be *delinquent* if d_i is at most $\tau/4t^3$. We now compute the probability that an instance which is not delinquent is classified as conditionally delinquent.

If an instance i is declared conditionally delinquent because $\mathsf{VK}_i = \mathsf{VK}$, then, with all but negligible probability over Ω^*, the probability that instance i succeeds is negligible. If not, the security of the one-time signature scheme is violated with non-negligible probability during Expt_1 (details omitted). If an instance is declared conditionally delinquent for failing too many trials, then, if $d_i > \tau/4t^3$, the probability that none of the t^4/τ trials succeeded is at most

$$\left(1 - \frac{\tau}{4t^3}\right)^{t^4/\tau} \leq e^{-t/4},$$

which is negligible.

Assuming that all instances declared conditionally delinquent are in fact delinquent, the probability of a classification failure during a given instance is at most $\tau/4t^3$, and hence the probability of a classification failure occurring is at most $t \cdot \tau/4t^3 \leq \tau/4$. □

Claim 5.9 *With all but negligible probability over* Ω^**, the probability of a* failure to extract *is negligible.*

A failure to extract occurs for one of two reasons: (1) the extraction algorithm cannot generate two different accepting transcripts or (2) the extraction algorithm generates two different accepting transcripts but cannot extract a witness to the decryption of the relevant ciphertext. Say extraction is attempted at instance i because the i^{th} instance succeeded (and no exceptional events occurred) when interacting with $\mathcal{A}$ using $\langle \vec{q}, \omega, b \rangle$. Let p_i denote (as in Claim 5.7) the probability, over choice of q_i', that the i^{th} instance succeeds and no exceptional events occur when interacting with $\mathcal{A}$ using $\langle q_1, \ldots, q_{i-1}, q_i', q_{i+1}, \ldots, q_t, \omega, b \rangle$. If case (1) occurs, this implies that $p_i = 1/e$. But in this case, extraction at this instance and with these values of $\vec{q}, \omega, b$ is performed only with probability at most $(4t^4/\tau) \cdot (1/e)$, which is negligible. Furthermore, since $\mathsf{VK}_i \neq \mathsf{VK}$ (otherwise instance i is declared conditionally delinquent), the techniques of the proof of Theorem 5.3 imply that, with all but negligible probability over Ω^*, case (2) occurs with only negligible probability. (If not, an RSA root may be extracted in expected polynomial time with non-negligible probability.) □

Claim 5.10 *With all but negligible probability over* Ω^**, the probability of abort due to an* exception at j during simulation of i *(for any* i, j*) is at most* $\tau/2 + \varepsilon_5(k)$*, where* $\varepsilon_5(\cdot)$ *is negligible.*

Fix i and j with $i > j$. Define $d'_{i,j}$ as the probability (over variables not defined before i) that instance j succeeds (conditioned on the values defined before i). An exception at j during simulation of i means that, during simulation of i, conditionally delinquent instance j succeeded at least $3t^3/\tau$ times out of at most $4t^4/\tau$ trials. We claim that if this happens, then, with all but negligible probability, $d'_{i,j}$ is at least $1/2t$. If this were not the case, letting X be a random variable denoting the number of successes in $4t^4/\tau$ trials and μ denote the actual value of $d'_{i,j}$, the Chernoff bound shows that:

$$\begin{aligned} \Pr[X > \frac{3}{2} \cdot \mu 4t^4/\tau] &< \Pr[X > 3t^3/\tau] \\ &< (e/(3/2)^3)^{t^3/\tau}, \end{aligned}$$

and since $e/(3/2)^3 < 1$, this expression is negligible. Assuming instance j is in fact delinquent (which is true with all but negligible probability; see the proof of Claim 5.8), the probability (over variables not defined before j) that $d'_{i,j} \geq 1/2t$ is at most $\tau/2t^2$. Summing over all t^2 possible choices of i and j yields the desired result. □

Assume the advantage of $\mathcal{A}$ in Expt_1 is not negligible. This implies the existence of some constant c such that, for infinitely many values of k,

$$\left| \Pr_1[\mathsf{Succ}] - \frac{1}{2} \right| > 1/k^c.$$

Set $\tau(k) = 1/2k^c$. The probability of a correct simulation is then at least $1 - 1/2k^c - \mu(k)$ for some negligible function $\mu(\cdot)$. Let Sim denote the event that a successful simulation occurs. Then:

$$\begin{aligned} \mathsf{Adv}_{\mathcal{A}'}(k) &= \left| \Pr[\mathsf{Succ} \wedge \mathsf{Sim}] + \Pr[\mathsf{Succ} \wedge \overline{\mathsf{Sim}}] - \frac{1}{2} \right| \\ &= \left| \Pr_1[\mathsf{Succ} \wedge \mathsf{Sim}] + \Pr[\mathsf{Succ} | \overline{\mathsf{Sim}}] \Pr[\overline{\mathsf{Sim}}] - \frac{1}{2} \right| \\ &\geq \left| \Pr_1[\mathsf{Succ}] - \Pr_1[\mathsf{Succ} \wedge \overline{\mathsf{Sim}}] - \frac{1}{2} \right| - \frac{1}{2} \cdot (\frac{1}{2k^c} + \mu(k)) \\ &\geq \left| \Pr_1[\mathsf{Succ}] - \frac{1}{2} \right| - \frac{3}{2} \cdot (\frac{1}{2k^c} + \mu(k)), \end{aligned}$$

and then for infinitely many values of k we have $\mathsf{Adv}_{\mathcal{A}'}(k) \geq 1/8k^c$ so that this quantity is not negligible. This completes the proof of the Theorem. ■

For completeness, we state the following theorems (in each case, we must also assume the security of $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ as a one-time signature scheme and the security of UOWH as a universal one-way hash family):

Theorem 5.7 *Assuming the hardness of factoring Blum integers for expected-polynomial-time algorithms, the protocol of Figure 5.5 (with $t = \Theta(k)$) is an interactive encryption scheme secure against sequential chosen-ciphertext attacks. If timing constraints are enforced, the resulting protocol is secure against concurrent chosen-ciphertext attacks.*

Theorem 5.8 *Assuming the hardness of the decisional*[7] *composite residuosity problem for expected-polynomial-time algorithms, the protocol of Figure 5.6 is an interactive encryption scheme secure against sequential chosen-ciphertext attacks. If timing constraints are enforced, the resulting protocol is secure against concurrent chosen-ciphertext attacks.*

Theorem 5.9 *Assuming the hardness of the DDH*[8] *problem for expected-polynomial-time algorithms, the protocol of Figure 5.7 is an interactive encryption scheme secure against sequential chosen-ciphertext attacks. If timing constraints are enforced, the resulting protocol is secure against concurrent chosen-ciphertext attacks.*

These schemes are in fact quite efficient, especially in comparison with the underlying semantically secure schemes. For example, the RSA-based scheme of Figure 5.3 for encryption of ℓ-bit messages requires only $2\ell+4$ exponentiations compared to the ℓ exponentiations required by the basic scheme. Furthermore, the additional exponentiations may be done in a preprocessing stage before the message to be sent is known.

5.4.2 Password-Based Authentication and Key Exchange

Password-based authentication and key exchange in the public-key setting were first formally modeled by Halevi and Krawczyk [76]; Boyarsky [22] extends the model to the multi-party setting (the model and definition of security for this setting are essentially identical to that presented in Chapter 3, except that the adversary is now additionally given the public keys of the servers). Protocols for password-based authentication may be constructed from any chosen-ciphertext-secure encryption scheme [76, 22]. Let pw be the password of the user which is stored by the server. A password-based authentication protocol using a (non-interactive) chosen-ciphertext-secure encryption scheme has the server send a random, sufficiently-long nonce n to the user, who replies with an encryption of $pw \circ n$ (actually, this brief description suppresses details which are unimportant for the discussion which follows; see [76, 22]). The server decrypts and verifies correctness of the password and the nonce; the nonce is necessary to prevent replay attacks. When an interactive, chosen-ciphertext-secure encryption scheme is used, the nonce is not necessary if the probability that a server repeats its messages is negligible [22]. In this case, authentication proceeds by simply having the user perform a (random) encryption of pw.

Password-based key-exchange protocols (with or without mutual authentication) may also be constructed using any chosen-ciphertext-secure encryption scheme [76, 22]. Here, for example, the

[7] As mentioned in Section 5.3.3, a construction secure under the (weaker) *computational* variant of this assumption is also possible.

[8] As mentioned in Section 5.3.4, a construction secure under the (weaker) CDH assumption is also possible.

user responds to a random nonce n with an encryption of $pw \circ n \circ k$, where k is the key to be shared (this achieves one-way authentication only; mutual authentication can be achieved with an additional round). As above, when an interactive chosen-ciphertext-secure encryption scheme is used, the nonce is not necessary when the probability of repeat messages from the server is negligible.

The only previously-known efficient and provably-secure implementations use the public-key encryption scheme of [36] whose security relies on the DDH assumption (the interactive solution of [44] may also be used, but, as mentioned previously, the solutions presented here are more efficient). Our techniques allow efficient implementation of these protocols based on a wider class of assumptions.

For the chosen-ciphertext-secure schemes given in the previous section, the probability that a server (acting as a receiver for the given encryption scheme) repeats a challenge is negligible. Figures 5.3, 5.5–5.7 therefore immediately yield efficient, 3-round, password-based authentication or key-exchange protocols in the public-key model. Security of these protocols may be based on the hardness of factoring, the RSA problem, or the decisional composite residuosity assumption. We stress that these are the first efficient and provably-secure constructions based on assumptions other than DDH.

5.4.3 Deniable Authentication

Previous work. Deniable authentication was first considered in [44], and a formal definition appears in [49]. The strongest notion of security requires the existence of a simulator which, given access only to a malicious verifier, can output a transcript which is indistinguishable from an interaction of the verifier with the actual prover. Constructions based on any non-malleable encryption scheme are known [49, 50, 48]. However, these protocols are not secure (in general) when a non-malleable *interactive* encryption scheme is used. For example, the non-malleable, interactive encryption scheme of [44] requires a signature from the prover and hence the resulting deniable authentication protocol is not simulatable (this problem is pointed out explicitly by Dwork, et al. [49]). Thus, the only previously-known, efficient deniable-authentication protocol which is secure under the strongest definition of security uses the construction of Dwork et al. [49] instantiated with the Cramer-Shoup encryption scheme [36]. Our constructions have the same round-complexity and efficiency, but their security may be based on a larger (and, in some cases, weaker) class of assumptions.

Definitions. We begin with a review of the definition that appears in [49, 48]. We have a prover $\mathcal{P}$ who has established a public key using key-generation algorithm $\mathcal{K}$ and is willing to authenticate messages to a verifier $\mathcal{V}$; however, $\mathcal{P}$ is *not* willing to allow the verifier to convince a third party (after the fact) that $\mathcal{P}$ authenticated anything. This is captured by ensuring that a transcript of an execution of the authentication protocol can be efficiently simulated without any access to $\mathcal{P}$. We also

require that no malicious adversary will be able to impersonate $\mathcal{P}$. More specifically, an adversary $\mathcal{M}$ (acting as man-in-the-middle between $\mathcal{P}$ and a verifier) should not be able to authenticate a message m to the verifier which $\mathcal{P}$ does not authenticate for $\mathcal{M}$.

Definition 5.4 *Let $\Pi = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ be a tuple of* PPT *algorithms. We say Π is a* strong deniable-authentication protocol *over message space $\mathbb{M} = \{M_{pk}\}_{pk \in \mathcal{K}(1^k), k \in \mathbb{N}}$ if it satisfies the following:*

(Completeness) *For all (pk, sk) output by $\mathcal{K}(1^k)$ and all $m \in M_{pk}$, we have $\langle \mathcal{P}_{sk}(m), \mathcal{V} \rangle(pk) = m$ (when $\mathcal{V}$ does not output $\perp$ we say it* accepts*).*

(Soundness) *Let $\mathcal{M}$ be a* PPT *adversary with oracle access to $\mathcal{P}$, where $\mathcal{P}$ authenticates any polynomial number of messages chosen adaptively by $\mathcal{M}$. Then the following is negligible in k:*

$$\Pr[(pk, sk) \leftarrow \mathcal{K}(1^k); m \leftarrow \langle \mathcal{M}^{\mathcal{P}_{sk}(\cdot)}, \mathcal{V} \rangle : m \notin \{m_1, \ldots, m_\ell\} \wedge m \neq \perp],$$

where $m_1, \ldots, m_\ell$ are the messages authenticated by $\mathcal{P}$.

(Strong deniability) *Let $\mathcal{V}'$ be a* PPT *adversary interacting with $\mathcal{P}$, where $\mathcal{P}$ authenticates any polynomial number of messages chosen adaptively by $\mathcal{V}'$. There exists an expected-polynomial-time simulator $\mathcal{SIM}$ with black-box (rewind) access to $\mathcal{V}'$ such that, with all but negligible probability over pk output by $\mathcal{K}(1^k)$, the following distributions are statistically indistinguishable:*

$$\{pk, \mathcal{SIM}(pk)\}$$
$$\{pk, {\mathcal{V}'}^{\mathcal{P}_{sk}}(pk)\}.$$

We also consider deniable-authentication protocols with a slightly weaker guarantee on their deniability.

Definition 5.5 *Let $\Pi = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ be a tuple of* PPT *algorithms. We say Π is a* strong ε-deniable-authentication protocol *over message space $\mathbb{M} = \{M_{pk}\}$ if it satisfies completeness and soundness as in Definition 5.4 in addition to the following:*

(Strong ε-deniability) *Let $\mathcal{V}'$ be a* PPT *adversary interacting with $\mathcal{P}$, where $\mathcal{P}$ authenticates any polynomial number of messages chosen adaptively by $\mathcal{V}'$. There exists a negligible function $\mu(\cdot)$ and a simulator $\mathcal{SIM}$ with black-box (rewind) access to $\mathcal{V}'$ such that, for all $\varepsilon > 0$, the expected running time of $\mathcal{SIM}(pk, \varepsilon)$ is polynomial in k and $1/\varepsilon$ and, with all but negligible probability over pk output by $\mathcal{K}(1^k)$, the following distributions have statistical difference at most $\varepsilon + \mu(k)$:*

$$\{pk, \mathcal{SIM}(pk, \varepsilon)\}$$
$$\left\{pk, {\mathcal{V}'}^{\mathcal{P}_{sk}}(pk)\right\}.$$

A relaxation of the above definitions which has been considered previously (e.g., [49]) allows the simulator $\mathcal{SIM}$ to have access to $\mathcal{P}_{sk}$ when producing the simulated transcript, but $\mathcal{P}$ only authenticates some fixed sequence of messages independent of those chosen by $\mathcal{V}'$. We call this *weak deniability*. In practice, weak deniability may not be acceptable because the protocol then leaves an undeniable trace that P authenticated *something* (even if not revealing *what*). However, $\mathcal{P}$ may want to deny that any such interaction took place. Note that previous solutions based on non-malleable encryption [44, 49, 50, 48] only achieve weak deniability when using the interactive non-malleable encryption scheme suggested by [44] (which requires a signature from $\mathcal{P}$).

Constructions. The protocols of Section 5.3 may be easily adapted to give deniable-authentication protocols whose security rests on the one-wayness of the appropriate encryption scheme for random messages; semantic security of the encryption scheme is not necessary. This yields very efficient deniable-authentication protocols since, for example, we may use the "simple" RSA encryption scheme in which r is encrypted as $r^e \bmod N$ (under the RSA assumption, this scheme is one-way for random messages). Furthermore, efficient deniable-authentication protocols may be constructed using weaker assumptions; for example, using the CDH assumption instead of the DDH assumption. Below, we improve the efficiency of these schemes even further, although in this case the resulting protocols are only secure over polynomially-large message spaces.

We first present the paradigm for construction of protocols which are secure over exponentially-large message spaces. The basic idea is for the receiver to give a non-malleable PPK for a ciphertext C encrypted using an encryption scheme which is one-way for random messages. Additionally, the message m which is being authenticated is included in the transcript and is signed along with everything else. Assuming the receiver's proof succeeds, the prover authenticates the message by responding with the decryption of C.

Figure 5.8 shows an example of this approach applied to the non-malleable PPK for RSA encryption. The public key of the prover $\mathcal{P}$ is an RSA modulus N, a prime e (with $|e| = \Theta(k)$), elements $g, h \in \mathbb{Z}_N^*$, and a hash function H chosen randomly from a family of universal one-way hash functions. Additionally, the prover has secret key d such that $de = 1 \bmod \varphi(N)$. The verifier $\mathcal{V}$ has message m taken from an arbitrary message space (of course, $|m|$ must be polynomial in the security parameter). To have m authenticated by $\mathcal{P}$, the verifier chooses a random $y \in \mathbb{Z}_N^*$, computes $C = y^e$, and then performs a non-malleable proof of knowledge of the witness y to the decryption of C (as in Figure 5.3). Additionally, the message m is sent as the first message of the protocol, and is signed along with the rest of the transcript. If the verifier's proof succeeds, the prover computes C^d and sends this value to the verifier. If the proof does not succeed, the prover simply replies with $\perp$.

As in the case of interactive encryption in Section 5.4.1, timing constraints are needed when

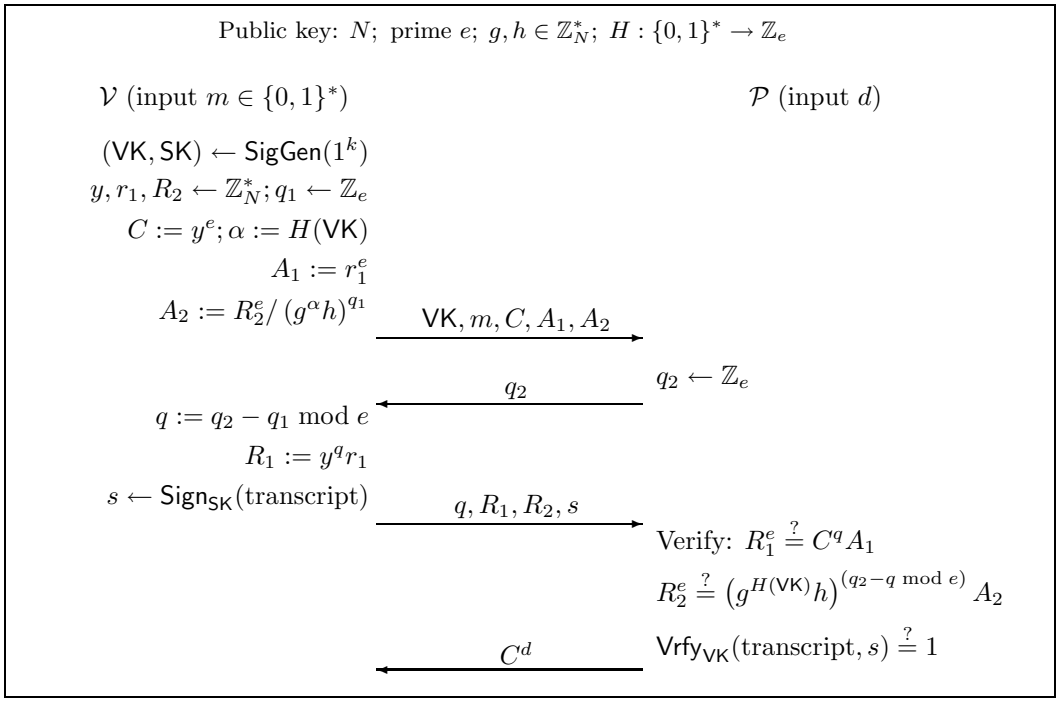

Figure 5.8: A deniable-authentication protocol based on RSA.

concurrent access to the prover is allowed. In this case, we require that the verifier respond to the challenge (i.e., send the third message of the protocol) within time α from when the challenge was sent. If $\mathcal{V}$ does not respond within this time, the proof is rejected. Additionally, the last message of the protocol is not sent by the prover until at least time β has elapsed since sending the challenge (clearly, we must have $\beta > \alpha$).

Theorem 5.10 *Assuming (1) the hardness of the RSA problem for expected-polynomial-time algorithms, (2) the security of* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme, and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 5.8 is a strong deniable-authentication protocol, over an arbitrary message space, for adversaries given sequential access to the prover. If timing constraints are enforced as outlined above, the protocol is a strong ε-deniable-authentication protocol for adversaries given concurrent access to the prover.*

Proof We assume familiarity with the proofs of Theorems 5.3 and 5.6. We consider the more challenging case of concurrent access to the prover; results for the case of sequential access may be derived from the arguments given here. Correctness of the protocol is immediate. We demonstrate soundness of the protocol using the same techniques as in the proof of Theorem 5.6. Given a PPT adversary $\mathcal{M}$, we construct an expected-polynomial-time adversary $\mathcal{A}'$ who attacks the one-wayness of the encryption scheme for a random message. In particular, adversary $\mathcal{A}'$ takes as input modulus

N and a random $C \in \mathbb{Z}_N^*$ and attempts to compute $y = C^{1/e} \bmod N$. Define Succ as the event that $\mathcal{M}$ authenticates a message for $\mathcal{V}$ which was not authenticated by $\mathcal{P}$ (cf. Definition 5.4). We show that the success probability of $\mathcal{A}'$ in inverting C will be negligible if and only if $\Pr[\mathsf{Succ}]$ is negligible. This immediately implies soundness of the deniable-authentication protocol.

Let $t(k)$, which is polynomial in k, be a bound on the number of times $\mathcal{M}$ accesses the prover when run on security parameter 1^k; without loss of generality, we assume that $t(k) \geq k$ (for convenience, in the remainder of the proof we suppress the dependence on k and simply write t). Let pk denote the components N, e of the public key and let σ denote components g, h, H. In the real experiment Expt_0, the actions of $\mathcal{M}$ are completely determined by $pk' = \langle pk, \sigma \rangle$, random coins r' for $\mathcal{M}$, the vector of challenges $\vec{q} = q_1, \ldots, q_t$ used during the t interactions of $\mathcal{M}$ with the prover, and the randomness used by the verifier (this includes the randomness y used to generate C as well as the randomness used for execution of the PPK). Let $\Pr[\mathsf{Succ}_0]$ denote the probability of event Succ in the real experiment.

We modify the real experiment giving Expt_1, as follows. Component pk of the public key is generated normally, along with the corresponding secret key sk; however, σ is generated using $\mathcal{SIM}_1(pk)$, which also generates state. The adversary $\mathcal{M}$ is run on input $pk' = \langle pk, \sigma \rangle$. The adversary's calls to the prover are handled as in Expt_0 (in particular, any ciphertext C may be decrypted since sk is known), but the adversary's call to the verifier will be handled differently. When $\mathcal{M}$ calls the verifier on message m, we now compute $C = y^e$ for random y and simulate the PPK for C (including m in the transcript) using algorithm $\mathcal{SIM}_2(\mathsf{state}; r)$ for randomly-chosen r. Now, the actions of $\mathcal{M}$ are completely determined by pk', random coins r' for $\mathcal{M}$, the vector of challenges $\vec{q} = q_1, \ldots, q_t$ used by the prover, the value y, and the values state and r used by $\mathcal{SIM}_2$ in simulating the verifier. Since $\mathcal{SIM}$ yields a perfect simulation of a real execution of the PPK (cf. Theorem 5.2), we have $\Pr[\mathsf{Succ}_1] = \Pr[\mathsf{Succ}_0]$, where the first probability refers to the probability of event Succ in Expt_1.

For the final experiment Expt_2, interaction with the prover will be handled by extracting witnesses to the decryption of the various ciphertexts from the proofs given by $\mathcal{M}$ (in particular, sk will not be used). Let $\Omega \stackrel{\text{def}}{=} \langle pk, \sigma, r', \vec{q}, \mathsf{state}, r \rangle$ and let $\Omega^* \stackrel{\text{def}}{=} \langle \Omega, y \rangle$. We show below an expected-polynomial-time simulator which takes Ω and C as input (where $C = y^e$). For any particular Ω^*, let p_{Ω^*} be the probability that the simulator succeeds in simulating the execution of $\mathcal{M}$ in Expt_1 with random variables Ω^*. Then for some negligible function $\mu(\cdot)$ and with all but negligible probability over Ω^*, we will have $p_{\Omega^*} > 1/2 - \mu(k)$. As in the proof of Theorem 5.6, this implies that $\Pr[\mathsf{Succ}_2]$ is at least $1/2 \cdot \Pr[\mathsf{Succ}_1] - \varepsilon_1(k)$, for some negligible function $\varepsilon_1(\cdot)$.

Before giving the details of the simulation, we note that Expt_2 immediately suggests an adversary

$\mathcal{A}'$ attacking the one-wayness of RSA encryption. Given a public key pk and a ciphertext C, adversary $\mathcal{A}'$ runs $\mathcal{SIM}_1(pk)$ to generate parameters σ and state. $\mathcal{A}'$ then fixes the randomness r' of $\mathcal{M}$, and runs $\mathcal{M}$ on input $pk' = \langle pk, \sigma \rangle$. Simulation of the verifier for $\mathcal{M}$ is done as follows: when $\mathcal{M}$ gives message m, adversary $\mathcal{A}'$ uses $\mathcal{SIM}_2(\mathsf{state}; r)$, for randomly-chosen r, to simulate a PPK for C and includes m in the transcript. The decryption oracle will be simulated as in Expt_2 (details of which appear below). $\mathcal{A}'$ outputs whatever value $\mathcal{M}$ sends to the verifier as the final message of that interaction. Note that the probability that $\mathcal{A}'$ successfully outputs $C^{1/e}$ is exactly $\Pr[\mathsf{Succ}_2]$. One-wayness of RSA implies that this is negligible.

Simulation of $\mathcal{P}$ in Expt_2 may be done exactly as in the proof of Theorem 5.6. In fact, the proof here is even easier since the ciphertext C to be inverted is known to $\mathcal{A}'$ at the beginning of the simulation; therefore, we do not need to worry about the instance at which C is defined. In brief, the simulator is given values $\langle pk, \sigma, r', \vec{q}^*, \mathsf{state}, r, C \rangle$ (where $C = y^e$ for some y unknown to the simulator). The values $pk, \sigma, r', \mathsf{state}, r$, and C are fixed throughout the simulation. When we say the simulator *interacts with* $\mathcal{M}$ *using* $\langle \vec{q} \rangle$ we mean that the simulator runs $\mathcal{M}$ as in Expt_1; that is, if $\mathcal{M}$ submits message m to $\mathcal{V}$, this query is answered by using $\mathcal{SIM}_2(\mathsf{state}; r)$ to simulate a PPK of C and including m in the transcript, and the challenge sent by the i^{th} instance of the prover is q_i.

When we are ready to simulate the i^{th} instance (assuming the entire simulation has not been aborted), each of the first $i-1$ instances has been classified as either extracted or conditionally delinquent. The simulator chooses random $q_i, \ldots, q_t$ and interacts with $\mathcal{M}$ using $\langle q_1^*, \ldots, q_{i-1}^*, q_i, \ldots, q_t \rangle$ until either (1) the i^{th} instance succeeds or (2) the i^{th} instance fails. If success occurs, witness extraction is performed as in the proof of Theorem 5.6. Otherwise, the above process is repeated at most t^4/τ times; if the i^{th} instance never succeeds, it is classified as conditionally delinquent.

The remaining details of the simulation may be derived from the proof of Theorem 5.6. This completes the proof of soundness.

For the proof of strong ε-deniability, note that the proof of Theorem 5.6 actually gives, for all $\tau > 0$, a simulator which has expected running time polynomial in k and $1/\tau$; furthermore, with all but negligible probability over Ω^* the simulation is aborted with probability at most τ plus a negligible quantity. The modified simulator for the present context, as described above, inherits this property. This immediately implies strong ε-deniability.

We briefly sketch the proof of strong deniability for the sequential case (where timing constraints are not used). Here, the simulator is given values pk', r', and $\vec{q}^*$ and must simulate the interaction of a malicious verifier $\mathcal{V}^*$ with $\mathcal{P}$. To do so, the simulator fixes pk, σ, sets the random tape of $\mathcal{V}^*$ to r' and gives pk' to $\mathcal{V}^*$ as input, and interacts with $\mathcal{V}^*$ using $\vec{q}^*$. To simulate the i^{th} instance (assuming all previous instances have been simulated and the entire transcript has not been aborted), the

simulator interacts with $\mathcal{V}^*$ using $\vec{q}^*$ until $\mathcal{V}^*$ sends the third message of the i^{th} instance. If this instance fails, the simulator responds with $\perp$ and continues with simulation of the next instance. If this instance succeeds, the simulator runs the witness extraction procedure as in the proof of Theorem 5.6. If witness extraction fails, the simulation is aborted; we call this a *failure to extract*. If witness extraction succeeds, the simulator now knows $C_i^{1/e}$ and can therefore continue with simulation of the next instance.

If an instance j fails when interacting with $\mathcal{V}^*$ using $q_1, \ldots, q_j, q_{j+1}, \ldots, q_t$ then this instance always fails when interacting with $\mathcal{V}^*$ using $q_1, \ldots, q_j, q'_{j+1}, \ldots, q'_t$ for arbitrary $q'_{j+1}, \ldots, q'_t$; this follows from the sequential access of $\mathcal{V}^*$. Therefore, no classification failures or exceptional events can occur. Thus, the simulation is aborted only following a failure to extract; however, as shown in the proof of Theorem 5.6, the probability of a failure to extract is negligible. ■

For completeness, we state the following theorems (in each case, we must also assume the security of $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ as a one-time signature scheme and the security of UOWH as a universal one-way hash family):

Theorem 5.11 *Assuming the hardness of factoring Blum integers for expected-polynomial-time algorithms, the protocol of Figure 5.5 may be adapted, as above, to give a strong deniable-authentication protocol (over an arbitrary message space) for adversaries given sequential access to the prover. If timing constraints are enforced, the protocol is an strong ε-deniable-authentication protocol for adversaries given concurrent access to the prover.*

Theorem 5.12 *Assuming the hardness of the computational composite residuosity problem for expected-polynomial-time algorithms, the protocol of Figure 5.6 may be adapted, as above, to give a strong deniable-authentication protocol (over an arbitrary message space) for adversaries given sequential access to the prover. If timing constraints are enforced, the protocol is a strong ε-deniable-authentication protocol for adversaries given concurrent access to the prover.*

Theorem 5.13 *Assuming the hardness of the computational Diffie-Hellman problem for expected-polynomial-time algorithms, the protocol of Figure 5.7 may be adapted, as above, to give a strong deniable-authentication protocol (over an arbitrary message space) for adversaries given sequential access to the prover. If timing constraints are enforced, the protocol is a strong ε-deniable-authentication protocol for adversaries given concurrent access to the prover.*

We stress that these protocols are quite practical. For example, the protocol implied by Theorem 5.13 has the same round-complexity, requires fewer exponentiations, has a shorter public key, and is based on a weaker assumption than the most efficient, previously-known protocol for strong deniable

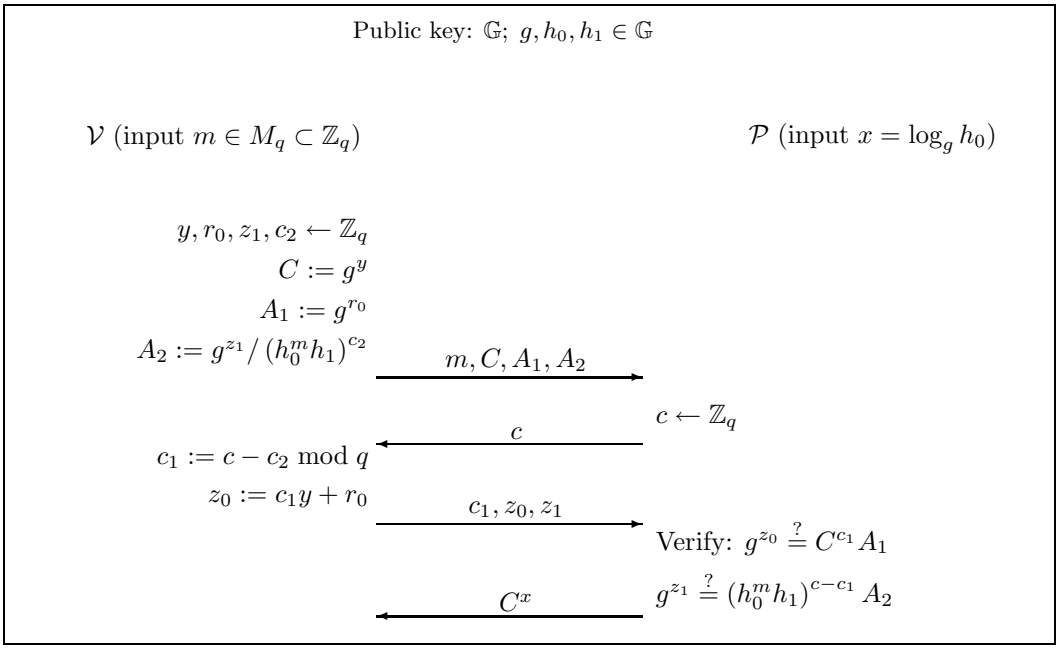

Figure 5.9: A deniable-authentication protocol for polynomial-size message spaces.

authentication (i.e., the protocol of [49] instantiated with the Cramer-Shoup encryption scheme [36]). No previous efficient protocols were known based on the RSA, factoring, or computational/decisional composite residuosity assumptions.

The efficiency of the above constructions may be improved further; however, the resulting protocols are secure over polynomial-sized message spaces only.[9] In Figure 5.9, we illustrate the improvement for the non-malleable PPK of Figure 5.7 (similar modifications to Figures 5.3, 5.5, and 5.6 yield protocols whose security may be based on the RSA, factoring, or computational composite residuosity assumptions). In the improved protocol, we make use of the fact that the adversary cannot re-use the value m which the adversary attempts to authenticate for the verifier (since, to break the security of the scheme, the adversary must authenticate a message which the prover does not authenticate). This eliminates the need to use a verification key VK, and eliminates the need for a one-time signature on the transcript. However, since the adversary chooses m (and this value must be guessed by the simulator in advance; see proof below), the present schemes are only secure when the message space is polynomial-size.

Theorem 5.14 *Assuming the hardness of the CDH problem for expected-polynomial-time algorithms, the protocol of Figure 5.9 is a strong deniable-authentication protocol (over message space $\{M_q\}$ where $M_q \subset \mathbb{Z}_q$ and $|M_q|$ is polynomial in k) for adversaries given sequential access to the*

[9] In Section 5.4.5, we modify the protocol based on the CDH assumption to achieve security over an exponentially-large message space.

prover. If timing constraints are enforced, the protocol is a strong ε-deniable-authentication protocol for adversaries given concurrent access to the prover.

Proof Completeness is obvious. Strong deniability in the sequential case and strong ε-deniability in the concurrent case follow from the arguments in the proof of Theorem 5.10. We prove soundness of the protocol for the more difficult, concurrent case.

Given a PPT adversary $\mathcal{M}$, we construct an expected-polynomial-time algorithm A which solves an instance of the CDH problem. Let Succ be the event that $\mathcal{M}$ authenticates a message for $\mathcal{V}$ which was not authenticated by $\mathcal{P}$ (cf. Definition 5.4). We show that the success probability of A in solving a random CDH instance is negligible if and only if $\Pr[\mathsf{Succ}]$ is negligible. This immediately implies soundness of the deniable-authentication protocol.

Let the size of M_q be $p(k)$ (where $p(\cdot)$ is polynomial in the security parameter), and let $t(k)$ (where $t(\cdot)$ is polynomial in k), be a bound on the number of times $\mathcal{M}$ accesses the prover when run on security parameter 1^k; without loss of generality, we assume that $t(k) \geq k$ (for convenience, in the remainder of the proof we suppress the dependence on k and simply write p, t). Let pk denote the values $\mathbb{G}, g, h_0, h_1$ which constitute the public key. In the real experiment Expt_0, the actions of $\mathcal{M}$ are completely determined by pk, random coins r' for $\mathcal{M}$, the vector of challenges $\vec{c} = c_1, \ldots, c_t$ used during the t interactions of $\mathcal{M}$ with the prover, and the randomness used by the verifier (this includes the randomness y used to generate C as well as the randomness ω_r used for the real PPK). Let $\Pr[\mathsf{Succ}_0]$ denote the probability of event Succ in the real experiment.

We next modify the real experiment, giving Expt_1, as follows. Public key pk is generated by choosing g at random in $\mathbb{G}$, choosing random $m^* \in M$ and $x, r \in \mathbb{Z}_q$, and setting $h_0 = g^x$ and $h_1 = h_0^{m^*} g^r$. The adversary $\mathcal{M}$ is run on input $\mathbb{G}, g, h_0, h_1$. The adversary's calls to the prover are handled as in Expt_0 (this can be done efficiently since $\log_g h_0$ is known), but the adversary's call to the verifier will be handled differently. If $\mathcal{M}$ ever asks the prover to authenticate message m^*, abort. Furthermore, if $\mathcal{M}$ calls the verifier on message m with $m \neq m^*$, abort. (Without loss of generality, we assume that if $\mathcal{M}$ calls the verifier on message m^*, then $\mathcal{M}$ never asks the prover to authenticate m^*; note that $\mathcal{M}$, by definition, cannot succeed if this occurs.) If $m = m^*$, choose $C^* \in \mathbb{G}$ at random, and simulate a PPK for C^* as in the proof of Theorem 5.5. In particular, choose $z_0, r_1, c_1 \in \mathbb{Z}_q$ at random, set $A_1 = g^{z_0}/(C^*)^{c_1}$ and $A_2 = g^{r_1}$, and send m^*, C^*, A_1, A_2 as the first message. Upon receiving query c from $\mathcal{M}$, set $c_2 = c - c_1$ and compute $z_1 = c_2 r + r_1$; the values c_1, z_0, z_1 are sent as the response. Note that the actions of $\mathcal{M}$ are now completely determined by $pk, r', \vec{c}, C^*, \omega_f$, where ω_f denotes the randomness used for the simulated PPK. Let $\Pr[\mathsf{Succ}_1]$ denote the probability of event Succ in this experiment. Since pk hides all information about the choice of m^*, the probability of not aborting is exactly $1/p$; furthermore, since the simulated proof

is identically-distributed to a real proof, we have $\Pr[\mathsf{Succ}_1] = 1/p \cdot \Pr[\mathsf{Succ}_0]$.

For the final experiment Expt_2, actions of the prover will be simulated by extracting $\log_g C_i$ from the proofs given by $\mathcal{M}$ in its various interactions with the prover (in particular, $\log_g h_0$ will not be used). Let $\Omega \stackrel{\text{def}}{=} \langle pk, r', \vec{c}, C^*, \omega_f \rangle$. We show below an expected-polynomial-time simulator which takes Ω as input; furthermore, for all Ω and some negligible $\varepsilon_1(\cdot)$, the simulator succeeds in simulating the execution of $\mathcal{M}$ in Expt_1 with random variables Ω with probability at least $1/2 - \varepsilon_1(k)$. Therefore, $\Pr[\mathsf{Succ}_2] > 1/2 \cdot \Pr[\mathsf{Succ}_1] - \varepsilon_1(k)$. In particular (recalling that p is polynomial in k), if $\Pr[\mathsf{Succ}_0]$ is non-negligible, then so is $\Pr[\mathsf{Succ}_2]$.

Before giving the details of the simulation, we note that Expt_2 immediately suggests an adversary A which solves the CDH problem. Given g, h_0, C^* as input, A fixes random $r', \vec{c}$, and ω_f, and runs $\mathcal{M}$ as in Expt_2. Note that this can be efficiently done (given the simulator we describe below); in particular $\log_g h_0$ is not needed. Finally, A outputs whatever value $\mathcal{M}$ sends to the verifier as the final message of that interaction. The probability that A solves the given CDH instance is exactly $\Pr[\mathsf{Succ}_2]$. Hardness of the CDH problem implies that this is negligible.

It remains to show how to simulate the actions of the prover. Define the i^{th} *instance* of the prover as the i^{th} time that $\mathcal{A}$ requests the second message of the PPK (i.e., the challenge) be sent by the prover. In any transcript of the execution of $\mathcal{M}$, we let C_i denote the second component of the first message of the i^{th} instance of the prover; define m_i similarly. For any instance of the prover, we say the instance *succeeds* if (1) an honest prover would accept the instance, (2) the timing constraints are satisfied for that instance. Otherwise, we say the instance *fails*.

The simulator is given values $\langle pk, r', \vec{c}^*, C^*, \omega_f^* \rangle$; furthermore, the value m^* is as defined above for Expt_1. The values pk, r' are fixed throughout the simulation. When we say the simulator *interacts with* $\mathcal{M}$ *using* $\langle \vec{c}, y, \omega_r, C, \omega_f \rangle$ we mean the following:

- If $C, \omega_f = \perp$ and $y, \omega_r \neq \perp$, the simulator runs $\mathcal{M}$ using $\vec{c}$ as the challenges of the prover. When $\mathcal{M}$ interacts with the verifier, the simulator runs the honest protocol for the verifier, sending $C = g^y$ and executing a real PPK using coins ω_r.

- If $y, \omega_r = \perp$ and $C, \omega_f \neq \perp$, the simulator runs $\mathcal{M}$ using $\vec{c}$ as the challenges of the prover. When $\mathcal{M}$ interacts with the verifier, the simulator sends C^* and executes the simulated PPK using coins ω_f. Note that such a simulation is only possible if m^* is the message $\mathcal{M}$ sends to the verifier.

Decryption requests cannot be immediately satisfied; this will not be a problem, as we show below.

To begin, the simulator chooses random $\vec{q}, y, \omega_r$ and interacts with $\mathcal{M}$ using $\langle \vec{c}, y, \omega_r, \perp, \perp \rangle$. If $\mathcal{M}$ sends message m to the verifier and $m \neq m^*$, the simulation is aborted. If $\mathcal{M}$ sends message m

to the verifier and $m = m^*$, we say *the message is defined at instance 1.* Otherwise, the message is not defined at instance 1.

If the message is not defined at instance 1, the simulator interacts with $\mathcal{M}$ using $\langle \vec{c}, y, \omega_r, \perp, \perp \rangle$. If $m_1 = m^*$, the simulation is aborted. Otherwise, interaction continues until either (1) the first instance succeeds or (2) the first instance fails. Note that the simulator cannot yet simulate the prover for instances $j > i$; however, this is not required, since a request for such a simulation implies that the first instance has already failed due to the timing constraints.

If the first instance ever succeeds, witness extraction is performed as described below. In case the first instance fails, the simulator chooses new, random $\vec{c}, y, \omega_r$ and interacts with $\mathcal{M}$ using $\langle \vec{c}, y, \omega_r, \perp, \perp \rangle$. This is repeated at most t^4/τ times (using new, random $\vec{q}, \omega, b$ each time) or until the first instance succeeds, where τ is a constant whose value we will fix at the end of the proof. If the first instance ever succeeds, the simulator proceeds with witness extraction as described below. Otherwise, the first instance is declared *conditionally delinquent* and the simulator proceeds to simulation of the second instance as described below.

If the message is defined at instance 1, the simulator proceeds as above, but interacts with $\mathcal{M}$ using $\langle \vec{c}, \perp, \perp, C^*, \omega_f^* \rangle$.

If the first instance ever succeeds, witness extraction will be performed. Assume the first instance succeeded when interacting with $\mathcal{M}$ using $\langle \vec{c}, y, \omega_r, C, \omega_f \rangle$. The simulator does the following:

For $n = 0$ to $q - 1$:
  $c_1' \leftarrow \mathbb{Z}_q$
  Interact with $\mathcal{M}$ using $\langle c_1', c_2, \ldots, c_t, y, \omega_r, C, \omega_f \rangle$
    If the first instance succeeds and $c_1' \neq c_1$, output the transcript and stop
  If $g^n = C_1$ output n and stop

If a value in $\mathbb{Z}_q$ is output, we immediately have $y_1 (= \log_g C_1)$. If a second transcript is output, the simulator computes (cf. the proof of Theorem 5.5) either y_1 or $\log_g(h_0^{m_1} h_1)$ for $m_1 \neq m^*$ (recall the simulation is aborted if $m_i = m^*$ is ever sent by $\mathcal{M}$ during any of the instances). In this second case, the simulator may then compute $x = \log_g h_0$. In either case, the simulator can now simulate the first instance by responding with either C_1^x or $h_0^{y_1}$ and we say the instance is *extracted*.

In general, when we are ready to simulate the i^{th} instance (assuming the entire simulation has not been aborted), each of the first $i-1$ instances has been classified as either extracted or conditionally delinquent. If instance j is extracted, then the simulator can send the appropriate value to $\mathcal{M}$ upon successful completion of that instance in the current simulation. On the other hand, if instance j is classified as conditionally delinquent, then with sufficiently high probability that instance will never succeed.

We say the message is *defined before instance* i if, for some $j \leq i$, the message is defined at instance j. At the beginning of simulation of the i^{th} instance, if the message is not defined

before instance $i-1$, the simulator chooses random $c_i, \ldots, c_t, y, \omega_r$ and interacts with $\mathcal{M}$ using $\langle c_1^*, \ldots c_{i-1}^*, c_i, \ldots, c_t, y, \omega_r, \perp, \perp \rangle$. If $\mathcal{M}$ sends message m to the verifier and $m \neq m^*$, the simulation is aborted. If $\mathcal{M}$ sends message m to the verifier and $m = m^*$, we say *the message is defined at instance i*. Otherwise, the message is not defined at instance i.

If the message is not defined before instance i, the simulator continues to interact with $\mathcal{M}$ using $\langle c_1^*, \ldots c_{i-1}^*, c_i, \ldots, c_t, y, \omega_r, \perp, \perp \rangle$. If $m_i = m^*$, the simulation is aborted. Otherwise, interaction continues until either: (1) the i^{th} instance succeeds or (2) the i^{th} instance fails. This is repeated for a total of at most t^4/τ times or until the i^{th} instance succeeds. If the i^{th} instance ever succeeds, the simulator proceeds with witness extraction similar to what was described above (details omitted). Otherwise, the i^{th} instance is declared *conditionally delinquent* and the simulator proceeds to simulation of the next instance.

Note that during simulation of instance i, simulation of instance j with $j > i$ is not required since such a request would imply that time the i^{th} instance has already failed due to timing constraints. However, the simulator may be required to simulate instances j with $j < i$. In case instance j is extracted, this is not problem. On the other hand, if j is conditionally delinquent, simulation cannot continue. We handle this as follows: if conditionally delinquent instance j succeeds *before* c_i is sent, the entire simulation is aborted; we call this a *classification failure*. If a conditionally delinquent instance j succeeds *after* c_i is sent, we consider this an *exceptional event at instance j during simulation of i*, and do not include it in the count of failed trials. However, if $3t^3/\tau$ such exceptional events occur for any j, the entire simulation is aborted; we call this an *exception at j during simulation of i*.

If the message is defined before instance i, the simulator proceeds as above, but interacts with $\mathcal{M}$ using $\langle c_1^*, \ldots, c_{i-1}^*, c_i, \ldots, c_t, \perp, \perp, C^*, \omega_f^* \rangle$.

Once all t instances have been simulated in this way, if the message is not defined before instance t, then the simulator interacts with $\mathcal{M}$ using $\langle \vec{c}^*, \perp, \perp, C^*, \omega_f^* \rangle$ until completion.

Claim 5.11 *The expected running time of the simulation is polynomial in t and $1/\tau$.*

The proof is as for Claim 5.7. □

For the following two claims, it is important to note that for *any* $\vec{c}$, the distribution of actions of $\mathcal{M}$ when the simulator interacts with $\mathcal{M}$ using $\langle \vec{c}, y, \omega_r, \perp, \perp \rangle$ (for random choice of y, ω_r) is identical to the distribution of actions of $\mathcal{M}$ when the simulator interacts with $\mathcal{M}$ using $\langle \vec{c}, \perp, \perp, C, \omega_f \rangle$ (for random choice of C, ω_f). This follows from the fact that the simulated PPK is distributed identically to a real PPK.

Claim 5.12 *The probability of a* classification failure *is at most $\tau/4 + \varepsilon_2(k)$, where $\varepsilon_2(\cdot)$ is negligible.*

The proof is as for Claim 5.8, except that the claim holds for all Ω since the adversary is assumed not to ask the prover to authenticate m^* once that value is given to the verifier (in Claim 5.8, $\mathcal{M}$ was "prevented" from re-using VK by virtue of the security of the signature scheme). □

Claim 5.13 *The probability of abort due to an* exception at j during simulation of i *(for any i, j) is at most $\tau/2 + \varepsilon_3(k)$, where $\varepsilon_3(\cdot)$ is negligible.*

The proof is as in Claim 5.10, except that, as above, the claim holds for all Ω. □

Note that the probability of a failure to extract is 0 in this case; however, for the protocols based on RSA, factoring, or Paillier, this probability would be negligible with all but negligible probability over Ω.

As in the proof of Theorem 5.6, setting $\tau = 2/3$ yields the desired simulation. ■

5.4.4 Identification

Definitions and preliminaries. Identification protocols satisfying various notions of security are known [56, 111, 74, 100, 31]. Only recently, however, has a definition of security against man-in-the-middle attacks been given by Bellare, et al. [7]. They provide practical, 4-round protocols and less practical, 2-round solutions which are secure under this definition. Motivated by this recent work, we introduce a definition of security against man-in-the-middle attacks that is weaker than that considered previously [7]. In particular, we do not allow $\mathcal{P}'$ to invoke multiple concurrent executions of $\mathcal{P}$ while $\mathcal{P}'$ is interacting with $\mathcal{V}$, and we do not consider reset attacks. We believe this approach is reasonable for most network-based settings since (1) $\mathcal{P}$ may simply refuse to execute multiple instances of the protocol simultaneously and (2) reset attacks are not an issue in a network-based setting.

Definition 5.6 *Let $\Pi = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ be a tuple of* PPT *algorithms. We say Π is an* identification scheme secure against man-in-the-middle attacks *if the following conditions hold:*

(Correctness) *For all (pk, sk) output by $\mathcal{K}(1^k)$, we have $\langle \mathcal{P}_{sk}, \mathcal{V} \rangle(pk) = 1$ (where $\mathcal{P}_{sk}$ denotes $\mathcal{P}(sk)$).*

(Security) *For all* PPT *adversaries $\mathcal{P}' = (\mathcal{P}'_1, \mathcal{P}'_2)$, the following is negligible (in k):*

$$\Pr\left[(pk, sk) \leftarrow \mathcal{K}(1^k); \mathsf{state} \leftarrow {\mathcal{P}'_1}^{\mathcal{P}_{sk}}(pk) : \langle {\mathcal{P}'_2}^{\mathcal{P}_{sk}}(\mathsf{state}), \mathcal{V} \rangle(pk) = 1 \wedge \pi \neq \pi'\right],$$

where $\mathcal{P}'_2$ may access only a single instance of $\mathcal{P}_{sk}$, π is defined as the transcript of the interaction between $\mathcal{P}_{sk}$ and $\mathcal{P}'_2$, and π' is defined as the transcript of the interaction between $\mathcal{P}'_2$ and $\mathcal{V}$.

We also define the notion of a simulatable identification scheme. Although such a definition is fairly standard and seems to have motivated the design of previous identification schemes, no such definition (in the context of identification schemes) has previously appeared.

Definition 5.7 *Let $\Pi = (\mathcal{K}, \mathcal{P}, \mathcal{V})$ be an identification scheme and let $\mathcal{V}^*$ denote the algorithm which runs $\mathcal{V}$ honestly and then outputs the entire transcript of the interaction.* Π *is* simulatable *if there exists a* PPT *simulator $\mathcal{SIM}$ such that, for all (pk, sk) output by $\mathcal{K}(1^k)$, the distribution $\{pk, \mathcal{SIM}(pk)\}$ is equivalent to the distribution $\{pk, \langle \mathcal{P}_{sk}, \mathcal{V}^* \rangle (pk)\}$.*

For the remainder of this chapter we let $\mathbb{G}$ denote a finite, cyclic group of prime order q. We review some properties of this group.

Definition 5.8 *Let $g_1, g_2, h \in \mathbb{G}$. Then $(x, y) \in \mathbb{Z}_q^2$ is a* representation of h with respect to g_1, g_2 *if $g_1^x g_2^y = h$. When g_1, g_2 are clear from context, we simply say that (x, y) is a representation of h. Representations (x_0, y_0) and (x_1, y_1) are* distinct *if and only if $x_0 \neq x_1$ and $y_0 \neq y_1$.*

Fact 5.1 *Let $g_1, g_2 \in \mathbb{G}$ be generators of $\mathbb{G}$. Every $h \in \mathbb{G}$ has q distinct representations with respect to g_1, g_2.*

Proof Since g_1 is a generator, there exist α, β such that $g_1^\alpha = g_2$ and $g_1^\beta = h$. Furthermore, $\alpha \neq 0$ since g_2 is a generator. The set $\{(\beta - \alpha y, y)\}_{y \in \mathbb{Z}_q}$ consists of exactly q distinct representations of h. Indeed:

$$\begin{aligned} g_1^{\beta - \alpha y} g_2^y &= (g_1^\beta)(g_1^\alpha)^{-y} g_2^y \\ &= h g_2^{-y} g_2^y \\ &= h, \end{aligned}$$

and, for $y \neq y'$, we have $\beta - \alpha y \neq \beta - \alpha y'$. ■

We now show that learning two distinct representations of any element with respect to the same g_1, g_2 is equivalent to learning $\log_{g_1} g_2$.

Lemma 5.1 *Given two distinct representations $(a_0, b_0), (a_1, b_1)$ of any value $h \in \mathbb{G}$ with respect to $g_1, g_2 \in \mathbb{G}$, the value $\log_{g_1} g_2$ may be efficiently computed.*

Proof Since $g_1^{a_0} g_2^{b_0} = g_1^{a_1} g_2^{b_1}$, algebraic manipulation implies that $\log_{g_1} g_2 = (a_0 - a_1)/(b_1 - b_0)$. Note that this value may be efficiently computed since $\mathbb{Z}_q$ is a field and $b_1 - b_0 \neq 0$. ■

Our construction. Our construction follows the paradigm outlined in the previous sections, yet new techniques are needed for the present setting. A public key for the identification protocol may

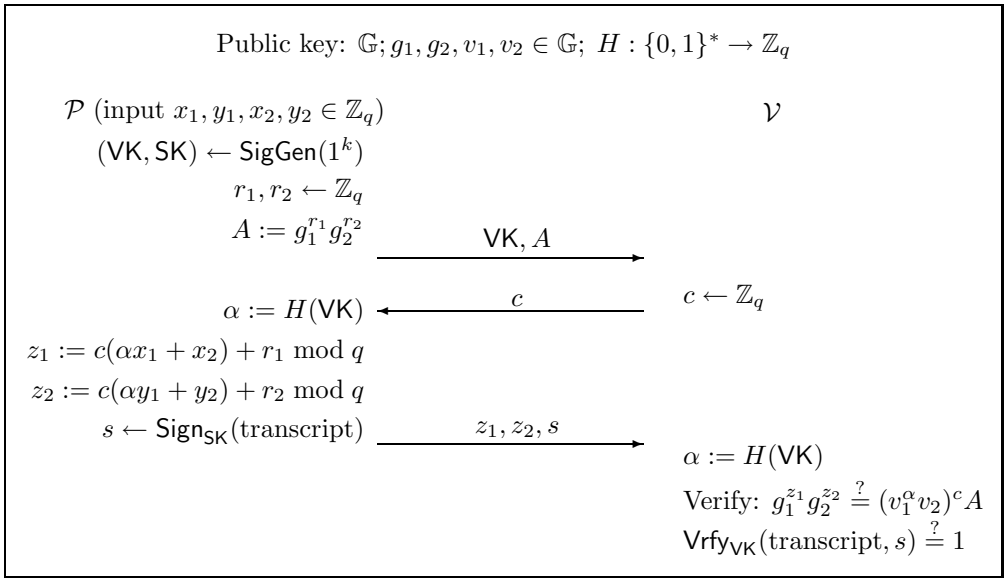

Figure 5.10: Identification scheme secure against man-in-the-middle attacks.

be viewed as containing values g, h such that a simulator knows a witness $x = \log_g h$ and can then run the identification protocol as the prover when interacting with the adversary $\mathcal{P}'$. Furthermore, this value h will depend on a parameter α which $\mathcal{P}'$ cannot re-use; thus the simulator will prove knowledge of $\log_g h_\alpha$ while the man-in-the-middle $\mathcal{P}'$ will be forced to prove knowledge of $\log_g h_{\alpha'}$ for some $\alpha' \neq \alpha$. This will be secure as long as the following hold: (1) it is possible for the simulator to know the witness $\log_g h_\alpha$ yet (2) learning $\log_g h_{\alpha'}$ for *any* $\alpha' \neq \alpha$ will allow the simulator to break some computationally-hard problem; furthermore, (3) the value of α used by the simulator cannot be used in any other execution of the protocol.

So far, this is exactly as in Sections 5.4.1–5.4.3. However, in the context of identification schemes, a proof of security using the above paradigm runs into trouble. The problem is that the simulator is required to simulate polynomially-many executions of the identification protocol with $\mathcal{P}'$ before the actual man-in-the-middle attack begins. Recall that in the case of proofs of knowledge (and their application to public-key encryption schemes and deniable-authentication protocols), the simulator is required to simulate only a *single* proof of knowledge for a particular value α which is fixed by the simulator in advance. Simulating many executions seems to contradict the requirements listed above: if the simulator cannot re-use the value α yet must simulate more than one execution of the protocol, it appears the simulator must then know $\log_g h_{\alpha_1}$ and $\log_g h_{\alpha_2}$ for $\alpha_1 \neq \alpha_2$. However, if the simulator already knows two such values, it seems the simulator has already broken the computational assumption on which the security of the scheme relies!

We overcome this seeming contradiction by extending the Okamoto-Schnorr [111, 100] identifi-

cation scheme; the modified protocol is shown in Figure 5.10. In the simulation, the simulator will know a witness (x_α, y_α) which is a representation of h_α with respect to two generators g_1, g_2; in fact, the simulator will know such a witness for *all* h_α. Yet, $\mathcal{P}'$ will not know (in an information-theoretic sense) which witnesses the simulator knows. Security will follow from the following, modified requirements: (1) the simulator knows a witness (which is a representation of h_α) for all α; thus, the simulator can simulate polynomially-many executions of the protocol. Let α_0 be the value of α used by the simulator during the execution of $\mathcal{P}_2'$ (i.e., the man-in-the-middle portion of the attack); then: (2) for any $\alpha' \neq \alpha_0$, $\mathcal{P}_2'$ cannot tell which representation of $h_{\alpha'}$ the simulator knows and furthermore (3) learning two witnesses for any α (in particular, for α') allows the simulator to break some computationally-hard problem. Note that, since there are exponentially-many possible representations for any h_α, this immediately implies that, with all but negligible probability, if the simulator can extract from $\mathcal{P}'$ a representation of $h_{\alpha'}$ with $\alpha' \neq \alpha_0$, then the simulator breaks the computationally-hard problem. Finally, we require (4) that $\mathcal{P}'$ cannot re-use the value α_0 during its impersonation attempt.

We briefly describe the protocol of Figure 5.10. The public key consists of a finite, cyclic group $\mathbb{G}$ and random elements $g_1, g_2, v_1, v_2 \in \mathbb{G}$, along with a function $H : \{0,1\}^* \to \mathbb{Z}_q$ chosen at random from a family of universal one-way hash functions. The prover $\mathcal{P}$ knows values $x_1, y_1, x_2, y_2 \in \mathbb{Z}_q$ such that $v_1 = g_1^{x_1} g_2^{y_1}$ and $v_2 = g_1^{x_2} g_2^{y_2}$. Execution of the identification scheme begins by having $\mathcal{P}$ generate keys $(\mathsf{VK}, \mathsf{SK})$ for a one-time signature scheme. Then, $\mathcal{P}$ runs the Okamoto-Schnorr identification protocol [111, 100] using public key $v_1^\alpha v_2$ and secret key (i.e., representation) $(\alpha x_1 + x_2, \alpha y_1 + y_2)$, where $\alpha \stackrel{\text{def}}{=} H(\mathsf{VK})$. The entire transcript is signed using SK. Note that this protocol inherits the witness indistinguishability of the Okamoto-Schnorr protocol.

Theorem 5.15 *Assuming (1) the hardness of the discrete logarithm problem in* $\mathbb{G}$ *for expected polynomial-time algorithms, (2) the security of* $(\mathsf{SigGen}, \mathsf{Sign}, \mathsf{Vrfy})$ *as a one-time signature scheme, and (3) the security of* UOWH *as a universal one-way hash family, the protocol of Figure 5.10 is a simulatable identification protocol secure against man-in-the-middle attacks.*

Proof The proof of simulatability is straightforward, as the protocol of Figure 5.10 is honest-verifier zero-knowledge. Therefore, we focus on the proof of security against man-in-the-middle attacks.

Given a PPT adversary $\mathcal{P}' = (\mathcal{P}_1', \mathcal{P}_2')$, let $\Pr[\mathsf{Succ}]$ be its probability of success (by *success* we mean that $\mathcal{V}$ accepts and that a different transcript is used by $\mathcal{P}'$; cf. Definition 5.6). In other words,

$$\begin{aligned}\Pr[\mathsf{Succ}] \stackrel{\text{def}}{=} \Pr[&g_1, g_2 \leftarrow \mathbb{G}; x_1, y_1, x_2, y_2 \leftarrow \mathbb{Z}_q; v_1 := g_1^{x_1} g_2^{y_1}; v_2 := g_1^{x_2} g_2^{y_2}; H \leftarrow \mathsf{UOWH}(1^k);\\ &pk := (\mathbb{G}, g_1, g_2, v_1, v_2, H); \mathsf{state} \leftarrow {\mathcal{P}_1'}^{\mathcal{P}_{sk}}(pk) : \langle {\mathcal{P}_2'}^{\mathcal{P}_{sk}}(\mathsf{state}), \mathcal{V}\rangle(pk) = 1 \wedge \pi \neq \pi'].\end{aligned}$$

We construct an expected polynomial-time algorithm $\mathcal{A}$ which uses $\mathcal{P}'$ to break the discrete logarithm problem; $\mathcal{A}$'s probability of computing the desired discrete logarithm will be negligibly close to $\Pr[\mathsf{Succ}]$. This immediately implies that $\Pr[\mathsf{Succ}]$ is negligible.

$\mathcal{A}$ proceeds as follows: on input $g_1, g_2 \in \mathbb{G}$ (where A must compute $\log_{g_1} g_2$), $\mathcal{A}$ picks random x_1, x_2, y_1, y_2, computes $v_1 = g_1^{x_1} g_2^{y_1}$ and $v_2 = g_1^{x_2} g_2^{y_2}$, chooses random hash function H, fixes random coins for $\mathcal{P}'_1$ and $\mathcal{P}'_2$, and runs $\mathcal{P}'_1$ on input $pk = (\mathbb{G}, g_1, g_2, v_1, v_2, H)$. Note that the distribution of this public key is exactly the same as in a real experiment (in fact, the key generation procedure is identical); furthermore, since $\mathcal{A}$ has a valid secret key (which is generated exactly as in a real experiment), $\mathcal{A}$ can perfectly simulate the interaction of the real prover with $\mathcal{P}'_1$. Hence the view of $\mathcal{P}'_1$ in its interaction with $\mathcal{A}$ is identically distributed to its view in an execution of the real experiment.

Once $\mathcal{P}'_1$ has completed its execution, $\mathcal{A}$ runs $\mathcal{P}'_2$ on the value state which was output by $\mathcal{P}'_1$. $\mathcal{A}$ chooses random coins ω' with which to simulating an honest prover (as above), and $\mathcal{A}$ additionally plays the role of an honest verifier $\mathcal{V}$. If $\mathcal{P}'_2$ does not succeed in its impersonation attempt, $\mathcal{A}$ simply aborts. Since $\mathcal{A}$ perfectly simulates the joint view of $(\mathcal{P}'_1, \mathcal{P}'_2)$, the probability of not aborting is exactly p.

If $\mathcal{P}'_2$ succeeds, let VK be the verification key used by $\mathcal{A}$ when simulating the honest prover during execution of $\mathcal{P}'_2$, let $\alpha \stackrel{\text{def}}{=} H(\mathsf{VK})$, and let $\langle \mathsf{VK}', A', c', z'_1, z'_2, s' \rangle$ be the transcript of the interaction between $\mathcal{P}'_2$ and $\mathcal{V}$. Note that even if $\mathcal{A}$ was not requested to simulate the honest prover, values VK, α are still well-defined by the random coins ω' chosen by $\mathcal{A}$. We now show how $\mathcal{A}$ can compute $\log_{g_1} g_2$ with all but negligible probability.

Fixing pk, sk, state, random coins ω for $\mathcal{P}'_2$, and random coins ω' used by $\mathcal{A}$ in simulating the honest prover, $\mathcal{A}$ proceeds to extract a witness from P'_2 using the following extraction algorithm:

For $i = 1$ to q:
  Check whether $(g_1)^i = g_2$; if so, output i and stop
  $c_i \leftarrow \mathbb{Z}_q$
  Run P'_2, sending challenge c_i for $\mathcal{V}$
    If P'_2 succeeds and $c_i \neq c'$, output the transcript
      of the interaction between P'_2 and $\mathcal{V}$ and stop

We first verify that $\mathcal{A}$ runs in expected polynomial time. With $\Omega \stackrel{\text{def}}{=} (pk, sk, \mathsf{state}, \omega, \omega')$ fixed as above, define p_Ω as the probability, over random challenges sent by $\mathcal{V}$, that $\mathcal{P}'_2$ succeeds. For each fixed value of Ω, the probability that the extraction algorithm is run is exactly p_Ω (since this algorithm is run only following a success by $\mathcal{P}'_2$). If $p_\Omega > 1/q$, the expected number of iterations of the loop is at most $2/p_\Omega$, so the contribution to the expected running time is at most $p_\Omega \cdot \frac{2 \cdot poly(k)}{p_\Omega} = 2 \cdot poly(k)$, where $poly(\cdot)$ is a bound on the running time of $\mathcal{P}'_2$. On the other hand, if $p_\Omega \leq 1/q$, the extraction procedure halts after at most q iterations, and the contribution to the running time is

therefore at most $1/q \cdot q \cdot poly(k) = poly(k)$.

When the extraction algorithm is run, it always outputs either the desired discrete logarithm $\log_{g_1} g_2$ or a second accepting transcript $\langle \mathsf{VK}', A', c'', z_1'', z_2'', s'' \rangle$ with $c'' \neq c'$. In case a second accepting transcript is output, we show how the desired discrete logarithm can be computed with all but negligible probability. Given the two accepting transcripts, we must have:

$$\begin{aligned} g_1^{z_1'} g_2^{z_2'} &= (v_1^{\alpha'} v_2)^{c'} A' \\ g_1^{z_1''} g_2^{z_2''} &= (v_1^{\alpha'} v_2)^{c''} A', \end{aligned}$$

where $\alpha' \stackrel{\text{def}}{=} H(\mathsf{VK}')$. These equations imply:

$$g_1^{\Delta_{z_1}} g_2^{\Delta_{z_2}} = (v_1^{\alpha'} v_2)^{\Delta_c},$$

where $\Delta_{z_1} \stackrel{\text{def}}{=} z_1' - z_1''$, $\Delta_{z_2} \stackrel{\text{def}}{=} z_2' - z_2''$, and $\Delta_c \stackrel{\text{def}}{=} c' - c''$. Since $\Delta_c \neq 0$, this means that:

$$g_1^{\Delta_{z_1}/\Delta_c} g_2^{\Delta_{z_2}/\Delta_c} = v_1^{\alpha'} v_2,$$

and $(\Delta_{z_1}/\Delta_c, \Delta_{z_2}/\Delta_c)$ is a representation of $v_1^{\alpha'} v_2$. From the secret key, $\mathcal{A}$ already knows a representation $(\alpha' x_1 + x_2, \alpha' y_1 + y_2)$ of $v_1^{\alpha'} v_2$. We show that with all but negligible probability these two representations are distinct.

Denote by Good the event that the extraction algorithm is run; by definition, we have $\Pr[\mathsf{Good}] = \Pr[\mathsf{Succ}]$. Note that $\Pr[\mathsf{VK}' = \mathsf{VK} \wedge \mathsf{Good}]$ is negligible. This follows from the assumed security of the one-time signature scheme (details omitted). Similarly, $\Pr[\mathsf{VK}' \neq \mathsf{VK} \wedge \alpha' = \alpha \wedge \mathsf{Good}]$ is negligible, by the assumed security of the family of universal one-way hash functions. Thus, $\Pr[\alpha' \neq \alpha \wedge \mathsf{Good}]$ is negligibly close to $\Pr[\mathsf{Succ}]$.

Given that event $(\mathsf{Good} \wedge \alpha \neq \alpha')$ occurs, and that two accepting transcripts are output by the extraction algorithm, the probability that $(\alpha' x_1 + x_2, \alpha' y_1 + y_2) = (\Delta_{z_1}/\Delta_c, \Delta_{z_2}/\Delta_c)$ is exactly $1/q$ (which is negligible). This is implied by the following facts:

Fact 5.2 *Given the entire view of $\mathcal{P}'$, throughout the entire experiment described above (including rewinding), the values x_1, y_1, x_2, y_2 known by the simulator are uniformly distributed, subject to:*

$$x_1 + \gamma y_1 = \log_{g_1} v_1 \tag{5.6}$$

$$x_2 + \gamma y_2 = \log_{g_1} v_2 \tag{5.7}$$

$$\alpha x_1 + x_2 = w, \tag{5.8}$$

where $\gamma \stackrel{\text{def}}{=} \log_{g_1} g_2$ and w is some fixed value.

Given the view of $\mathcal{P}_1'$, the values x_1, y_1, x_2, y_2 are uniformly distributed, subject to (5.6)–(5.7); this follows from the perfect witness indistinguishability of the protocol. As for the view of $\mathcal{P}_2'$, note that all $\mathcal{A}$ requires in order to simulate execution of the honest prover during this stage is the representation $(w, w') \stackrel{\text{def}}{=} (\alpha x_1 + x_2, \alpha y_1 + y_2)$ of $v_1^\alpha v_2$. In other words, the values of x_1, x_2, y_1, y_2 themselves are immaterial. Thus, x_1, x_2, y_1, y_2 are further constrained only by (5.8) and $\alpha y_1 + y_2 = w'$. However, this last equation may be derived from (5.6)–(5.8) since we also have $w + \gamma w' = \log_{g_1}(v_1^\alpha v_2)$. Note that witness indistinguishability does *not* imply that the *entire* view of $\mathcal{P}_2'$ is independent of the representation used by $\mathcal{A}$; the reason for this is that $\mathcal{A}$ rewinds $\mathcal{P}_2'$ during the extraction procedure, and the protocol does not remain witness indistinguishable once multiple proofs with the same initial value A are given. □

Fact 5.3 *When $\alpha' \neq \alpha$, the equations (5.6)–(5.8) and $\alpha' x_1 + x_2 = u$ are linearly independent for any value u. A similar statement holds for the equation $\alpha' y_1 + y_2 = u'$.*

Thus, given that event Good occurs, the two representations above are distinct with probability $(1 - 1/q)$; if they are distinct, Lemma 5.1 shows that $\log_{g_1} g_2$ may be computed. The probability of $\mathcal{A}$'s computing the desired discrete logarithm is then negligibly close to $(1 - 1/q) \cdot \Pr[\mathsf{Succ}]$, and $\Pr[\mathsf{Succ}]$ must therefore be negligible. ■

5.4.5 Deniable Authentication with Improved Efficiency

Definitions of security for deniable-authentication protocols are given in Section 5.4.3. We apply the techniques of Section 5.4.4 to construct an efficient deniable authentication protocol whose security may be based on the CDH assumption. Our protocol has the same round-complexity and comparable efficiency to the most efficient previous solution (the construction of [49] instantiated with the Cramer-Shoup encryption scheme [36]), but may be based on a weaker assumption. Furthermore, our solutions offers an alternate paradigm for constructing deniable authentication protocols.

Recall Figure 5.9, which illustrates a paradigm for constructing efficient deniable-authentication protocols for *polynomial-size* message spaces. Unfortunately, the proof of security given in Theorem 5.14 fails when trying to extend it to exponential-size message spaces. It is instructive to see where this failure lies. The proof requires a simulator who can simulate $\mathcal{V}$ for an arbitrary message chosen by the adversary. If the message space is polynomial-size, a message to be simulated can be chosen in advance by the simulator; with non-negligible probability, the message later chosen by the adversary will be equal to that for which a simulation is possible. However, if the message space is exponentially-large, the probability that the adversary will choose the message for which simulation is possible is negligible.

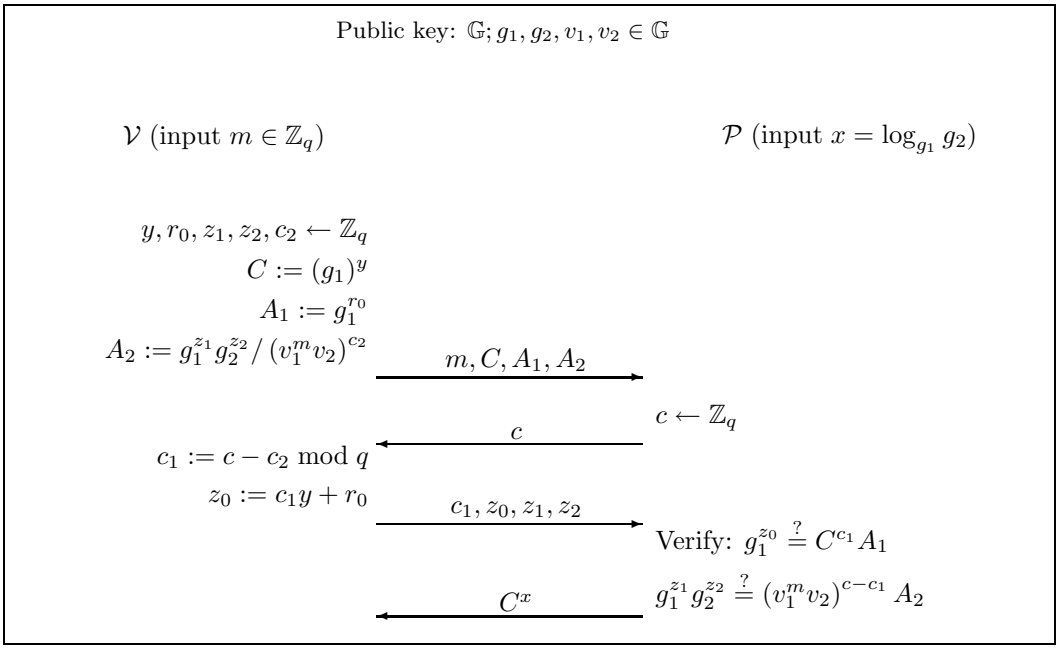

Figure 5.11: A deniable-authentication protocol for exponential-size message spaces.

The techniques of the previous section help get around this problem. We can in fact construct a simulator who can simulate execution of the protocol for *any* message chosen by the adversary. Besides facilitating the proof of security, a tighter reduction is achieved.

The deniable-authentication protocol is given in Figure 5.11. The public key consists of a finite, cyclic group $\mathbb{G}$, and elements g_1, v_1, v_2 randomly chosen from $\mathbb{G}$. The value $x \in \mathbb{Z}_q$ is randomly chosen, and $g_2 = g_1^x$ is added to the public key. The prover stores the value x as the secret key of the scheme. To have message $m \in \mathbb{Z}_q$ authenticated by $\mathcal{P}$, the verifier chooses a random $y \in \mathbb{Z}_q$, computes $C = g_0^y$, and then performs a witness indistinguishable proof of knowledge of *either* the discrete logarithm $\log_{g_1} C$ *or* a representation of $v_1^m v_2$ with respect to g_1 and g_2. If the proof succeeds, the prover computes C^x and sends this value back to the verifier. The verifier can check whether this value is correct by verifying that it is equal to g_2^y.

Theorem 5.16 *Assuming the hardness of the CDH problem for expected-polynomial-time algorithms, the protocol of Figure 5.11 is a strong deniable-authentication protocol (over message space $\mathbb{Z}_q$) under sequential access to the prover. Furthermore, it is a strong ε-deniable-authentication protocol under concurrent access to the prover.*

Proof Since a complete proof of security for the protocol of Figure 5.9 (over a polynomial-size message space) is given in Theorem 5.14, we merely sketch the differences which make the protocol of Figure 5.11 secure over an exponential-size message space. Completeness is clear. For soundness

and strong deniability, we focus on the case of sequential access to the prover; concurrent access to the prover may be handled using timing constraints as discussed at the beginning of Section 5.4.

We now show that the protocol is strongly deniable; note that our proof assumes only the hardness of the discrete logarithm problem for expected-polynomial-time algorithms. For simplicity, we show how a single interaction of the malicious verifier $\mathcal{V}'$ with the prover can be simulated to within negligible statistical difference; this immediately implies simulation to within negligible statistical difference for polynomially-many sequential interactions. The simulator $\mathcal{SIM}$ proceeds as follows: given public key $pk = \langle \mathbb{G}, g_1, g_2, v_1, v_2 \rangle$ and access to verifier $\mathcal{V}'$, $\mathcal{SIM}$ fixes the random coins of $\mathcal{V}'$ and simulates the interaction of $\mathcal{V}'$ with the real $\mathcal{P}$ for the first three messages. In other words, $\mathcal{SIM}$ replies to the initial message of $\mathcal{V}'$ with a random challenge c. Let the third message, sent by $\mathcal{V}'$, be c_1, z_0, z_1, z_2. If this partial transcript is not accepting (i.e., verification as shown in Figure 5.11 fails), $\mathcal{SIM}$ simply outputs the transcript $\langle m, C, A_1, A_2, c, c_1, z_0, z_1, z_2, \perp \rangle$ and stops. Otherwise, $\mathcal{SIM}$ will attempt to extract $y \stackrel{\text{def}}{=} \log_{g_1} C$ using the procedure outlined below; if this value is extracted, $\mathcal{SIM}$ outputs the transcript $\langle m, C, A_1, A_2, c, c_1, z_0, z_1, z_2, g_2^y \rangle$. Let Good be the event that verification of the transcript succeeds, and let $\mathsf{Extract}$ denote the event that $\mathcal{SIM}$ extracts y. Clearly, the above results in a perfect simulation unless both Good and $\overline{\mathsf{Extract}}$ occur. We show that $\Pr[\overline{\mathsf{Extract}} \wedge \mathsf{Good}]$ is negligible, where the probability is over choice of pk and random coins of $\mathcal{V}'$ and $\mathcal{SIM}$.

To extract y, $\mathcal{SIM}$ runs the following extraction algorithm:

For $i = 1$ to q:
  Check whether $(g_1)^i = C$; if so, output i and stop
  $c' \leftarrow \mathbb{Z}_q$
  Run V', sending challenge c' for $\mathcal{P}$
    If $c' \neq c$ and verification (as in Figure 5.11) succeeds,
      output the values c_1', z_0', z_1', z_2' sent by $\mathcal{V}'$

We verify that extraction takes expected-polynomial time. Fixing pk and random coins for $\mathcal{V}'$, let p denote the probability (over random challenges sent by $\mathcal{P}$) that verification of the proof of $\mathcal{V}'$ succeeds. If $p > 1/q$, the expected number of iterations of the loop above is at most $2/p$, so the contribution to the expected running time is at most $p \cdot \frac{2 \cdot poly(k)}{p} = 2 \cdot poly(k)$, where $poly(\cdot)$ is a bound on the running time of $\mathcal{V}'$. On the other hand, if $p \leq 1/q$, the extraction procedure halts after at most q iterations, and the contribution to the running time is therefore at most $1/q \cdot q \cdot poly(k) = poly(k)$.

When the extraction algorithm is run, it always outputs either the desired discrete logarithm y or a second accepting (partial) transcript $\langle m, C, A_1, A_2, c', c_1', z_0', z_1', z_2' \rangle$ with $c' \neq c$. In case a second accepting (partial) transcript is output, we show how the desired discrete logarithm can be computed with all but negligible probability (over pk and random coins of $\mathcal{V}'$). Given the two accepting (partial) transcripts, we must have either $c_1 \neq c_1'$ or else $c - c_1 \neq c' - c_1' \bmod q$ (or possibly

both). In case $c - c_1 = c' - c'_1$, we must have $c_1 \neq c'_1$, and the following two equations

$$\begin{aligned} g_1^{z_0} &= C^{c_1} A_1 \\ g_1^{z'_0} &= C^{c'_1} A_1 \end{aligned}$$

imply that $g_1^{z_0 - z'_0} = C^{c_1 - c'_1}$ (with $c_1 - c'_1 \neq 0$). Therefore $\log_{g_1} C = (z_0 - z'_0)/(c_1 - c'_1)$.

We show that $\Pr[c - c_1 \neq c' - c'_1 \wedge \mathsf{Good}]$ is negligible. Note that, in this case, the equations

$$\begin{aligned} g_1^{z_1} g_2^{z_2} &= (v_1^m v_2)^{c - c_1} A_2 \\ g_1^{z'_1} g_2^{z'_2} &= (v_1^m v_2)^{c' - c'_1} A_2 \end{aligned}$$

imply that:

$$g_1^{\Delta_{z_1}} g_2^{\Delta_{z_2}} = (v_1^m v_2)^{\Delta_c},$$

where $\Delta_{z_1} \stackrel{\text{def}}{=} z_1 - z'_1$, $\Delta_{z_2} \stackrel{\text{def}}{=} z_2 - z'_2$, and $\Delta_c \stackrel{\text{def}}{=} c - c_1 - c' + c'_1 \neq 0$. Simple algebraic manipulation shows that this allows efficient computation of a representation $(\Delta_{z_1}/\Delta_c, \Delta_{z_2}/\Delta_c)$ of $v_1^m v_2$.

If $\Pr[c - c_1 \neq c' - c'_1 \wedge \mathsf{Good}]$ is not negligible, then we can construct an expected-polynomial-time algorithm A which computes discrete logarithms with non-negligible probability. A, on input g_1, g_2 for which $\log_{g_1} g_2$ must be computed, generates public key pk for the deniable-authentication scheme by choosing random $x_1, x_2, y_1, y_2 \in \mathbb{Z}_q$ and setting $v_1 = g_1^{x_1} g_2^{y_1}$ and $v_2 = g_1^{x_2} g_2^{y_2}$. A then interacts with $\mathcal{V}'$ exactly as $\mathcal{SIM}$ interacts with $\mathcal{V}'$, above. If both Good and $c - c_1 \neq c' - c'_1$ occur, then A can compute a representation $(\Delta_{z_1}/\Delta_c, \Delta_{z_2}/\Delta_c)$ of $v_1^m v_2$, for some m, as above. However, note that A already knows a representation $(mx_1 + x_2, my_1 + y_2)$ for $v_1^m v_2$. Furthermore, given the view of $\mathcal{V}'$, the values x_1, x_2, y_1, y_2 are uniformly distributed, subject to:

$$x_1 + \gamma y_1 = \log_{g_1} v_1 \tag{5.9}$$

$$x_2 + \gamma y_2 = \log_{g_1} v_2, \tag{5.10}$$

where $\gamma \stackrel{\text{def}}{=} \log_{g_1} g_2$. Furthermore, for any $m, u \in \mathbb{Z}_q$, the equations (5.9), (5.10), and $mx_1 + x_2 = u$ are linearly independent. Therefore, with all but negligible probability $1/q$, we have that representations $(\Delta_{z_1}/\Delta_c, \Delta_{z_2}/\Delta_c)$ and $(mx_1 + x_2, my_1 + y_2)$ are distinct. When this occurs, Lemma 5.1 shows that A may compute $\log_{g_1} g_2$, which (by assumption) must occur with negligible probability.

We have thus shown that $(1 - 1/q)\Pr[c - c_1 \neq c' - c'_1 \wedge \mathsf{Good}]$ is negligible. This implies that $\Pr[c - c_1 \neq c' - c'_1 \wedge \mathsf{Good}]$ is negligible. Note that event $\overline{\mathsf{Extract}} \wedge \mathsf{Good}$ occurs exactly when $(c - c_1 \neq c' - c'_1 \wedge \mathsf{Good})$ occurs; therefore $\Pr[\overline{\mathsf{Extract}} \wedge \mathsf{Good}]$ is negligible as well. This completes the proof of strong deniability.

The proof of soundness is very similar to the proof of Theorem 5.14, and we therefore sketch only the differences here. Let pk denote the values $\mathbb{G}, g_1, g_2, v_1, v_2$ which constitute the public key.

In the real experiment Expt_0, the actions of $\mathcal{M}$ are completely determined by pk, random coins r' for $\mathcal{M}$, the vector of challenges $\vec{c} = c_1, \ldots, c_t$ used during the t interactions of $\mathcal{M}$ with the prover, and the randomness used by the verifier (this includes the randomness y used to generate C as well as the randomness ω_r used for the real PPK). Let $\Pr[\mathsf{Succ}_0]$ denote the probability of event Succ in the real experiment.

We next modify the real experiment, giving Expt_1, as follows. Given $\mathbb{G}$, public key pk is generated by choosing g_1 at random in $\mathbb{G}$, choosing r, x_1, x_2, y_1, y_2 at random in $\mathbb{Z}_q$, and setting $g_2 = g_1^r$, $v_1 = g_1^{x_1} g_2^{y_1}$, and $v_2 = g_1^{x_2} g_2^{y_2}$. The adversary $\mathcal{M}$ is run on input pk. The adversary's calls to the prover are handled as in Expt_0 (this can be done efficiently since $\log_{g_1} g_2$ is known), but the adversary's call to the verifier will be handled differently. For any message m which $\mathcal{M}$ sends to the verifier, choose $C^* \in \mathbb{G}$ at random, and simulate a PPK for C^* as in the proof of Theorem 5.15 (details omitted). Let $\Pr[\mathsf{Succ}_1]$ denote the probability of event Succ in this experiment. Since the simulated proof is identically-distributed to a real proof, we have $\Pr[\mathsf{Succ}_1] = \Pr[\mathsf{Succ}_0]$. This is a key difference from the protocol of Figure 5.9, since the PPK can here be simulated for *any* value m chosen by $\mathcal{M}$ (and we do not lose the factor $1/\mu$, where μ is the size of the message space).

For the final experiment Expt_2, actions of the prover will be simulated by extracting $\log_{g_1} C_i$ from the proofs given by $\mathcal{M}$ in its various interactions with the prover (in particular, $\log_{g_1} g_2$ will not be used). Let $\Omega \stackrel{\text{def}}{=} \langle pk, r', \vec{c}, C^*, \omega_f \rangle$, where ω_f are the random coins used for simulation of the PPK. We show below an expected-polynomial-time simulator which takes Ω as input; furthermore, for some negligible $\varepsilon_2(\cdot)$ and all Ω, the simulator succeeds in simulating the execution of $\mathcal{M}$ in Expt_1 (with random variables Ω) with probability at least $1/2 - \varepsilon_2(k)$. Therefore, $\Pr[\mathsf{Succ}_2] > 1/2 \cdot \Pr[\mathsf{Succ}_1] - \varepsilon_2(k)$. In particular this implies that if $\Pr[\mathsf{Succ}_0]$ is non-negligible, then so is $\Pr[\mathsf{Succ}_2]$.

Before giving the details of the simulation, we note that Expt_2 immediately suggests an adversary A which solves the CDH problem. Given g, h_0, C^* as input, A fixes random $r', \vec{c}$, and ω_f, and runs $\mathcal{M}$ as in Expt_2. The probability that A solves the given CDH instance is exactly $\Pr[\mathsf{Succ}_2]$. Hardness of the CDH problem implies that this is negligible.

The simulation is similar to the simulation used in the proof of Theorem 5.14, with two exceptions: (1) the simulator never aborts because $m \neq m^*$ (where m is the message given by $\mathcal{M}$ to the verifier), or because $m_i = m^*$ during some instance (indeed, a value m^* is not defined in advance of the simulation here), (2) witness extraction is slightly different, and we will need to take into account the probability of a failure to extract. We stress that, even though the simulator is able to simulate the protocol for many different values of m, such simulation is done only once the message *is defined* (see proof of Theorem 5.14 for definition of this term). When the message is not defined, the

simulator runs a real execution of the protocol for a value C' where $\log_{g_1} C'$ is known. This will be crucial for the proof of security. We denote the message which is eventually defined by m^*.

Assume the i^{th} instance succeeds when the simulator interacts with $\mathcal{M}$ using $\langle \vec{c}, y, \omega_r, C, \omega_f \rangle$. The simulator does the following:

For $n = 0$ to $q - 1$:
  $c'_i \leftarrow \mathbb{Z}_q$
  Interact with $\mathcal{M}$ using $\langle c_1, \ldots, c_{i-1}, c'_i, c_{i+1}, \ldots, c_t, y, \omega_r, C, \omega_f \rangle$
    If the first instance succeeds and $c'_i \neq c_i$, output the transcript and stop
  If $g_1^n = C_i$ output n and stop

If a value $n \in \mathbb{Z}_q$ is output, then $n = \log_g C_i$. If a second transcript is output, the simulator computes (cf. the proof of Theorem 5.15) either $y_i = \log_{g_1} C_i$ or a representation (a, b) of $v_1^{m'} v_2$ with respect to g_0, g_1. We show below that, with all but negligible probability, the latter allows the simulator to compute $x = \log_{g_0} g_1$. If this value cannot be computed, the simulation is aborted due to *failure to extract*. In any case, with all but negligible probability, the simulator can now simulate the i^{th} instance by sending either the value C_i^x or $g_1^{y_i}$ and we say the instance is *extracted*.

We now prove that learning a representation (a, b) of $v_1^{m'} v_2$, as above, allows the simulator to compute $\log_{g_0} g_1$ with all but negligible probability. Consider the first time extraction is performed. Note that the simulator already knows a representation $(m'x_1 + x_2, m'y_1 + y_2)$ of $v_1^{m'} v_2$. We argue (as in the proof of Theorem 5.15) that these representations are different except with probability $1/q$. Indeed, given the entire view of $\mathcal{M}$ (including rewinding), the values x_1, x_2, y_1, y_2 are uniformly distributed, subject to:

$$x_1 + \gamma y_1 = \log_{g_1} v_1 \tag{5.11}$$

$$x_2 + \gamma y_2 = \log_{g_1} v_2, \tag{5.12}$$

$$m^* x_1 + x_2 = w, \tag{5.13}$$

where $\gamma \stackrel{\text{def}}{=} \log_{g_1} g_2$, and w is some fixed value. Equations (5.11) and (5.12) are derived from pk itself. Furthermore, for any $m' \neq m^*$ and any $u \in \mathbb{Z}_q$, equations (5.11)–(5.13) and $m'x_1 + x_2 = u$ are linearly independent; note that we must have $m^* \neq m'$ since we assume that $\mathcal{M}$ does not ask the prover to authenticate the message given to the verifier. Thus, in the first witness extraction, the probability of a *failure to extract* is at most $1/q$.

Arguing in this way, we see that the probability of a failure to extract during the j^{th} witness extraction is at most $1/(q-j)$. Summing over at most t witness extractions (where $t(\cdot)$ is polynomial), shows that the probability of a failure to extract ever occurring is negligible.

The remainder of the proof is as in the proof of Theorem 5.14. ■

Chapter 6

Conclusions

Throughout this dissertation, we have argued that man-in-the-middle attacks represent an important and potentially damaging class of attacks that must be taken into account when designing and implementing real-world protocols. We have shown explicit examples of such attacks on many seemingly innocuous protocols; these examples clearly illustrate the pitfalls of constructing protocols without considering the possibility of active attacks. The results of this thesis also demonstrate the importance of rigorous proofs of security against active attacks, in a well-defined and robust model.

And yet, more than a decade after the notion of non-malleability was introduced (in the conference version of [44]), it remains difficult and challenging to construct provably-secure protocols protecting against active attacks. Even more elusive has been the goal of designing protocols that are provably secure against an active adversary yet efficient enough to be used in practice.

The work included in this dissertation represents a step toward that goal. We give very *practical* protocols for password-only authenticated key exchange, non-interactive commitment, interactive public-key encryption, password-based authenticated key exchange, and deniable authentication. More importantly, we believe the techniques and methods outlined here will find further applications to the construction of provably-secure protocols for other tasks.

Much work in this area remains to be done. Some compelling problems that come to mind include:

Password-only authenticated key exchange. The protocol given in Chapter 3 is the only known practical and provably secure protocol for password-only authenticated key exchange. The security of this protocol is based on the decisional Diffie-Hellman assumption. One interesting direction is to design a practical protocol whose security is based on, for example, the RSA assumption. It would also be useful to formalize security for password-only protocols in the model of Canetti [27] and to design a protocol secure in this model.

Non-malleable commitment. The protocols given in Chapter 4 all work in the *public parameters model*. The original scheme of Dolev, Dwork, and Naor [44] is thus far the only known example of a non-malleable commitment scheme operating in the "vanilla" model (without public parameters). It is a important question whether the round complexity of their solution can be improved using any reasonable cryptographic assumptions.

Even in the public parameters model, some intriguing questions remain. Perhaps most striking is the following: non-malleable and non-interactive *standard* commitment can be done with commitment size $\mathcal{O}(|M|k)$ based on the existence of one-way functions [40] and with commitment size $|M| + \mathit{poly}(k)$ based on the existence of trapdoor permutations (cf. Chapter 4). It seems counter-intuitive that the trapdoor property is necessary to achieve a reduced commitment size. Yet, we do not know how to substantially reduce the commitment size when only one-way functions are assumed.

Non-malleable proofs of knowledge and applications. Definition 5.2 seems to capture what is meant by a "non-malleable" interactive proof of knowledge. Yet, we were unable to prove secure any generic construction of, say, an interactive encryption scheme from an *arbitrary* non-malleable PPK. Instead, we were only able to prove such a construction secure for the *specific* non-malleable PPKs given in Chapter 5. This is indeed an unsatisfying state of affairs. It seems that the flaw lies in the definition itself (counterexamples that seem to satisfy Definition 5.2 yet fail to yield a secure interactive encryption scheme can be constructed), but we were unable to extend the definition in a reasonable way so as to obtain any sort of composition theorem. Resolving this issue seems a promising future direction.

Bibliography

[1] Y. Aumann and M.O. Rabin. A Proof of Plaintext Knowledge Protocol and Applications. Manuscript, June 2001.

[2] D. Beaver. Adaptive Zero-Knowledge and Computational Equivocation. *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, ACM, 1996, pp. 629–638.

[3] M. Bellare, A. Boldyreva, and S. Micali. Public-Key Encryption in a Multi-User Setting: Security Proofs and Improvements. *Advances in Cryptology — Eurocrypt 2000*, LNCS 1807, B. Preneel ed., Springer-Verlag, 2000, pp. 259–274.

[4] M. Bellare, R. Canetti, and H. Krawczyk. A Modular Approach to the Design and Analysis of Authentication and Key Exchange Protocols. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, ACM, 1998, pp. 419–428.

[5] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. *Proceedings of the 38th Annual Symposium on Foundations of Computer Science*, IEEE, 1997, pp. 394–403.

[6] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations Among Notions of Security for Public-Key Encryption Schemes. *Advances in Cryptology — CRYPTO '98*, LNCS 1462, H. Krawczyk ed., Springer-Verlag, 1998, pp. 26–45.

[7] M. Bellare, M. Fischlin, S. Goldwasser, and S. Micali. Identification Protocols Secure Against Reset Attacks. *Advances in Cryptology — Eurocrypt 2001*, LNCS 2045, B. Pfitzmann ed., Springer-Verlag, 2001, pp. 495–511.

[8] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. *Advances in Cryptology — CRYPTO '92*, LNCS 740, E.F. Brickell, ed., Springer-Verlag, 1992, pp. 390–420.

[9] M. Bellare, S. Halevi, A. Sahai, and S. Vadhan. Many-to-One Trapdoor Functions and Their Relation to Public-Key Cryptosystems. *Advances in Cryptology — CRYPTO '98*, LNCS 1462, H. Krawczyk ed., Springer-Verlag, 1998, pp. 283–298.

[10] M. Bellare, J. Kilian, and P. Rogaway. The Security of the Cipher Block Chaining Message Authentication Mode. *Journal of Computer and System Sciences* 61(3): 352–399 (2000).

[11] M. Bellare, D. Pointcheval, and P. Rogaway. Authenticated Key Exchange Secure Against Dictionary Attacks. *Advances in Cryptology — Eurocrypt 2000*, LNCS 1807, B. Preneel ed., Springer-Verlag, 2000, pp. 139–155.

[12] M. Bellare and P. Rogaway. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. *Proceedings of the First Annual Conference on Computer and Communications Security*, ACM, 1993, pp. 62–73.

[13] M. Bellare and P. Rogaway. Entity Authentication and Key Distribution. *Advances in Cryptology — CRYPTO '93*, LNCS 773, D.R. Stinson ed., Springer-Verlag, 1993, pp. 232–249.

[14] M. Bellare and P. Rogaway. Optimal Asymmetric Encryption. *Advances in Cryptology — Eurocrypt '94*, LNCS 950, A. De Santis ed., Springer-Verlag, 1994, pp. 92–111.

[15] M. Bellare and P. Rogaway. Provably-Secure Session Key Distribution: the Three Party Case. *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, ACM, 1995, pp. 57–66.

[16] M. Bellare and A. Sahai. Non-Malleable Encryption: Equivalence between Two Notions and an Indistinguishability-Based Characterization. *Advances in Cryptology — CRYPTO '99*, LNCS 1666, M. Wiener ed., Springer-Verlag, 1999, pp. 519–536.

[17] S.M. Bellovin and M. Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. *IEEE Symposium on Research in Security and Privacy*, IEEE, 1992, pp. 72–84.

[18] R. Bird, I. Gopal, A. Herzberg, P. Janson, S. Kutten, R. Molva, and M. Yung. Systematic Design of Two-Party Authentication Protocols. *Advances in Cryptology — CRYPTO '91*, LNCS 576, J. Feigenbaum ed., Springer-Verlag, 1991, pp. 44–61.

[19] M. Blum. How to Prove a Theorem So No One Else Can Claim It. *Proceedings of the International Congress of Mathematicians*, 1986, pp. 1444–1451.

[20] M. Blum, P. Feldman, and S. Micali. Non-Interactive Zero-Knowledge and Its Applications. *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, ACM, 1988, pp. 103–112.

[21] M. Blum and S. Goldwasser. An Efficient Probabilistic Public-Key Encryption Scheme which Hides All Partial Information. *Advances in Cryptology — CRYPTO '84*, LNCS 196, G.R. Blakley and D. Chaum, eds., Springer-Verlag, 1984, pp. 289–302.

[22] M. Boyarsky. Public-Key Cryptography and Password Protocols: The Multi-User Case. *Proceedings of the 7th Annual Conference on Computer and Communications Security*, ACM, 1999, pp. 63–72.

[23] V. Boyko. On All-or-Nothing Transforms and Password-Authenticated Key Exchange Protocols. PhD Thesis, MIT, Department of Electrical Engineering and Computer Science. Cambridge, MA, 2000.

[24] V. Boyko, P. MacKenzie, and S. Patel. Provably-Secure Password-Authenticated Key Exchange Using Diffie-Hellman. *Advances in Cryptology — Eurocrypt 2000*, LNCS 1807, B. Preneel ed., Springer-Verlag, 2000, pp. 156–171.

[25] G. Brassard, C. Crépeau, and M. Yung. Constant-Round Perfect-Zero-Knowledge Computationally-Convincing Protocols. *Theoretical Computer Science* 84(1): 23–52 (1991).

[26] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1): 143–202 (2000).

[27] R. Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science*, IEEE, 2001, pp. 136–145.

[28] R. Canetti and M. Fischlin. Universally Composable Commitments. *Advances in Cryptology — CRYPTO 2001*, LNCS 2139, J. Kilian ed., Springer-Verlag, 2001, pp. 19–40.

[29] R. Canetti, O. Goldreich, and S. Halevi. The Random Oracle Methodology, Revisited. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, ACM, 1998, pp. 209–218.

[30] R. Canetti and H. Krawczyk. Analysis of Key-Exchange Protocols and their Use for Building Secure Channels. *Advances in Cryptology — Eurocrypt 2001*, LNCS 2045, B. Pfitzmann ed., Springer-Verlag, 2001, pp. 453–474.

[31] R. Cramer. Modular Design of Secure Yet Practical Cryptographic Protocols. PhD Thesis, CWI and University of Amsterdam, 1996.

[32] R. Cramer and I. Damgård. Fast and Secure Immunization against Adaptive Man-in-the-Middle Impersonation. *Advances in Cryptology — Eurocrypt '97*, LNCS 1233, W. Fumy ed., Springer-Verlag, 1997, pp. 75–87.

[33] R. Cramer, I. Damgård, and P. MacKenzie. Efficient Zero-Knowledge Proofs of Knowledge Without Intractability Assumptions. *Public Key Cryptography — PKC 2000*, LNCS 1751, H. Imai and Y. Zheng, eds., Springer-Verlag, 2000, pp. 354–372.

[34] R. Cramer, I. Damgård, and J.B. Nielsen. Multiparty Computation from Threshold Homomorphic Encryption. *Advances in Cryptology — Eurocrypt 2001*, LNCS 2045, B. Pfitzmann ed., Springer-Verlag, 2001, pp. 280–299.

[35] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. *Advances in Cryptology — CRYPTO '94*, LNCS 839, Y. Desmedt ed., Springer-Verlag, 1994, pp. 174–187.

[36] R. Cramer and V. Shoup. A Practical Public Key Cryptosystem Provably Secure Against Chosen Ciphertext Attack. *Advances in Cryptology — CRYPTO '98*, LNCS 1462, H. Krawczyk ed., Springer-Verlag, 1998, pp. 13–25.

[37] R. Cramer and V. Shoup. Universal Hash Proofs and a Paradigm for Adaptive Chosen-Ciphertext-Secure Public-Key Encryption. To appear, Eurocrypt 2002.

[38] A. De Santis, G. Di Crescenzo, R. Ostrovsky, G. Persiano, and A. Sahai. Robust Non-Interactive Zero Knowledge. *Advances in Cryptology — CRYPTO 2001*, LNCS 2139, J. Kilian ed., Springer-Verlag, 2001, pp. 566-598.

[39] A. De Santis and G. Persiano. Zero-Knowledge Proofs of Knowledge Without Interaction. *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science*, IEEE, 1992, pp. 427–436.

[40] G. Di Crescenzo, Y. Ishai, and R. Ostrovsky. Non-Interactive and Non-Malleable Commitment. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, ACM, 1998, pp. 141–150.

[41] G. Di Crescenzo, J. Katz, R. Ostrovsky, and A. Smith. Efficient and Non-Interactive, Non-Malleable Commitment. *Advances in Cryptology — Eurocrypt 2001*, LNCS 2045, B. Pfitzmann ed., Springer-Verlag, 2001, pp. 40–59.

[42] W. Diffie and M. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6): 644–654 (1976).

[43] W. Diffie, P. van Oorschot, and M. Wiener. Authentication and Authenticated Key Exchanges. *Designs, Codes, and Cryptography*, 2(2): 107–125 (1992).

[44] D. Dolev, C. Dwork, and M. Naor. Nonmalleable Cryptography. *SIAM Journal of Computing* 30(2): 391–437 (2000).

[45] D. Dolev, S. Even, and R. Karp. On the Security of Ping-Pong Protocols. *Information and Control* 55(1–3): 57–68 (1982).

[46] D. Dolev and A.C. Yao. On the Security of Public-Key Protocols. *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science*, IEEE, 1981, pp. 350–357.

[47] C. Dwork. The Non-Malleability Lectures. Manuscript, Spring 1999.

[48] C. Dwork and M. Naor. Zaps and Their Applications. *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, IEEE, 2000, pp. 283–293.

[49] C. Dwork, M. Naor, and A. Sahai. Concurrent Zero-Knowledge. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, ACM, 1998, pp. 409–418.

[50] C. Dwork and A. Sahai. Concurrent Zero-Knowledge: Reducing the Need for Timing Constraints. *Advances in Cryptology — CRYPTO '98*, LNCS 1462, H. Krawczyk ed., Springer-Verlag, 1998, pp. 442–457.

[51] T. El Gamal. A Public-Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory* 31(4): 469–472 (1985).

[52] S. Even and O. Goldreich. On the Security of Multi-Party Ping-Pong Protocols. *Proceedings of the 24th Annual Symposium on Foundations of Computer Science*, IEEE, 1983, pp. 34–39.

[53] S. Even, O. Goldreich, and S. Micali. On-Line/Off-Line Digital Signatures. *Journal of Cryptology* 9(1): 35–67 (1996).

[54] U. Feige, A. Fiat, and A. Shamir. Zero-Knowledge Proofs of Identity. *Journal of Cryptology* 1(2): 77–94 (1988).

[55] U. Feige and A. Shamir. Zero-Knowledge Proofs of Knowledge in Two Rounds. *Advances in Cryptology — CRYPTO '89*, LNCS 435, G. Brassard ed., Springer-Verlag, 1990, pp. 526–544.

[56] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *Advances in Cryptology — CRYPTO '86*, LNCS 263, A. Odlyzko ed., Springer-Verlag, 1986, pp. 186–194.

[57] FIPS 113, "Computer Data Authentication," Federal Information Processing Standards Publication 113, U.S. Department of Commerce/National Bureau of Standards, National Technical Information Service, Springfield, VA, 1985.

[58] M. Fischlin and R. Fischlin. Efficient Non-Malleable Commitment Schemes. *Advances in Cryptology — CRYPTO 2000*, LNCS 1880, M. Bellare ed., Springer-Verlag, 2000, pp. 413–431.

[59] M. Fischlin and R. Fischlin. The Representation Problem Based on Factoring. To appear, *The Cryptographer's Track at RSA Conference 2002*, 2002.

[60] Z. Galil, S. Haber, and M. Yung. Symmetric Public-Key Encryption. *Advances in Cryptology — CRYPTO '85*, LNCS 218, H.C. Williams ed., Springer-Verlag, 1986, pp. 128–137.

[61] O. Goldreich. Secure Multi-Party Computation. Manuscript, 1998. Available at `http://www.wisdom.weizmann.ac.il/~oded/pp.html`.

[62] O. Goldreich. "Foundation of Cryptography (Basic Tools)" Cambridge University Press, 2001. Preliminary versions of volumes 2 and 3 are available at `http://www.wisdom.weizmann.ac.il/~oded/frag.html`.

[63] O. Goldreich, S. Goldwasser, and S. Micali. On the Cryptographic Applications of Random Functions. *Advances in Cryptology — CRYPTO '84*, LNCS 196, G.R. Blakley and D. Chaum eds., Springer-Verlag, 1985, pp. 276–288.

[64] O. Goldreich and L.A. Levin. A Hard-Core Predicate for all One-Way Functions. *Proceedings of the 21st ACM Symposium on Theory of Computing*, ACM, 1989, pp. 25–32.

[65] O. Goldreich and Y. Lindell. Session-Key Generation Using Human Passwords Only. *Advances in Cryptology — CRYPTO 2001*, LNCS 2139, J. Kilian ed., Springer-Verlag, 2001, pp. 408–432.

[66] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game or a Completeness Theorem for Protocols with Honest Majority. *Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, ACM, 1987, pp. 218–229.

[67] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Yield Nothing but their Validity or All Languages in NP have Zero-Knowledge Proof Systems. *Journal of the ACM* 38(3): 691–729 (1991).

[68] S. Goldwasser. Probabilistic Encryption: Theory and Applications. PhD Thesis, University of California, Berkeley, 1984.

[69] S. Goldwasser and S. Micali. Probabilistic Encryption. *Journal of Computer and System Sciences* 28(2): 270–299 (1984).

[70] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing* 18(1): 186–208 (1989).

[71] S. Goldwasser, S. Micali, and R. Rivest. A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. *SIAM Journal on Computing* 17(2): 281–308 (1988).

[72] L. Gong, T.M.A. Lomas, R.M. Needham, and J.H. Saltzer. Protecting Poorly-Chosen Secrets from Guessing Attacks. *IEEE Journal on Selected Areas in Communications*, 11(5): 648–656 (1993).

[73] L. Gong. Optimal Authentication Protocols Resistant to Password Guessing Attacks. *Proceedings of the 8th IEEE Computer Security Foundations Workshop*, 1995, pp. 24–29.

[74] L.C. Guillou and J.-J. Quisquater. A Practical Zero-Knowledge Protocol Fitted to Security Microprocessors Minimizing Both Transmission and Memory. *Advances in Cryptology — Eurocrypt '88*, LNCS 330, C. Gunther ed., Springer-Verlag, 1988, pp. 123–128.

[75] S. Haber. Multi-Party Cryptographic Computations: Techniques and Applications. PhD Thesis, Columbia University, 1987.

[76] S. Halevi and H. Krawczyk. Public-Key Cryptography and Password Protocols. *ACM Transactions on Information and System Security*, 2(3): 230–268 (1999).

[77] J. Håstad, R. Impagliazzo, L.A. Levin, and M. Luby. A Pseudorandom Generator from any One-Way Function. *SIAM Journal on Computing* 28(4): 1364–1396 (1999).

[78] R. Impagliazzo and M. Luby. One-Way Functions are Essential for Complexity-Based Cryptography. *Proceedings of the 30th Annual Symposium on Foundations of Computer Science*, IEEE, 1989, pp. 230–235.

[79] D. Jablon. Strong Password-Only Authenticated Key Exchange. *ACM Computer Communications Review*, 26(5): 5–20 (1996).

[80] D. Jablon. Extended Password Key Exchange Protocols Immune to Dictionary Attack. *Proceedings of WET-ICE '97*, IEEE Computer Society, 1997, pp. 248–255.

[81] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated-Verifier Proofs and Their Applications. *Advances in Cryptology — Eurocrypt '96*, LNCS 1070, U. Maurer ed., Springer-Verlag, 1996, pp. 143–154.

[82] J. Katz, R. Ostrovsky, and M. Rabin. Identity-Based Non-Interactive Zero-Knowledge. Manuscript, November 2001.

[83] J. Katz, R. Ostrovsky, and M. Yung. Efficient Password-Authenticated Key Exchange Using Human-Memorable Passwords. *Advances in Cryptology — Eurocrypt 2001*, LNCS 2045, B. Pfitzmann ed., Springer-Verlag, 2001, pp. 475–494.

[84] J. Katz and M. Yung. Complete Characterization of Security Notions for Probabilistic Private-Key Encryption. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, ACM, 2000, pp. 245–254.

[85] J.H. Kim, D.R. Simon, and P. Tetali. Limits on the Efficiency of One-Way Permutation-Based Hash Functions. *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, IEEE, 1999, pp. 535–542.

[86] H. Krawczyk. SKEME: A Versatile Secure Key-Exchange Mechanism for the Internet. *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, 1996, pp. 114–127.

[87] L. Lamport. Constructing Digital Signatures from a One-Way Function. Technical Report CSL-98, SRI International, Palo Alto, 1979.

[88] T.M.A. Lomas, L. Gong, J.H. Saltzer, and R.M. Needham. Reducing Risks from Poorly-Chosen Keys. *ACM Operating Systems Review*, 23(5): 14–18 (1989).

[89] S. Lucks. Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys. *Proceedings of the Security Protocols Workshop*, LNCS 1361, B. Christianson, B. Crispo, T.M.A. Lomas, and M. Roe eds., Springer-Verlag, 1997, pp. 79–90.

[90] P. MacKenzie. More Efficient Password-Authenticated Key Exchange. *Progress in Cryptology — CT-RSA 2001*, LNCS 2020, D. Naccache ed., Springer-Verlag, 2001, pp. 361–377.

[91] P. MacKenzie. On the Security of the SPEKE Password-Authenticated Key-Exchange Protocol. Manuscript, 2001.

[92] P. MacKenzie, S. Patel, and R. Swaminathan. Password-Authenticated Key Exchange Based on RSA. *Advances in Cryptology — Asiacrypt 2000*, LNCS 1976, T. Okamoto ed., Springer-Verlag, 2000, pp. 599–613.

[93] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1999.

[94] R.C. Merkle. A Certified Digital Signature. *Advances in Cryptology — CRYPTO '89*, LNCS 435, G. Brassard, ed., Springer-Verlag, 1990, pp. 218–238.

[95] S. Micali and P. Rogaway. Secure Computation. *Advances in Cryptology — CRYPTO '91*, LNCS 576, J. Feigenbaum ed., Springer-Verlag, 1991, pp. 392–404.

[96] M. Naor. Bit Commitment Using Pseudorandomness. *Journal of Cryptology* 4(2): 151–158 (1991).

[97] M. Naor, R. Ostrovsky, R, Venkatesan, and M. Yung. Perfect Zero-Knowledge Arguments for NP can be Based on General Complexity Assumptions. *Advances in Cryptology — CRYPTO '92*, LNCS 740, E. Brickell, ed., Springer-Verlag, 1993, pp. 196–214.

[98] M. Naor and M. Yung. Universal One-Way Hash Functions and Their Cryptographic Applications. *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, ACM, 1989, pp. 33–43.

[99] M. Naor and M. Yung. Public-Key Cryptosystems Provably Secure against Chosen Ciphertext Attacks. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, ACM, 1990, pp. 427–437.

[100] T. Okamoto. Provably Secure and Practical Identification Schemes and Corresponding Signature Schemes. *Advances in Cryptology — CRYPTO '92*, LNCS 740, E. Brickell ed., Springer-Verlag, 1993, pp. 31–53.

[101] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. *Advances in Cryptology — Eurocrypt '99*, LNCS 1592, J. Stern ed., Springer-Verlag, 1999, pp. 223–238.

[102] S. Patel. Number-Theoretic Attacks on Secure Password Schemes. *Proceedings of the IEEE Symposium on Research in Security and Privacy*, IEEE, 1997, pp. 236–247.

[103] T.P. Pedersen. Non-Interactive and Information-Theoretic-Secure Verifiable Secret Sharing. *Advances in Cryptology — CRYPTO '91*, LNCS 576, J. Feigenbaum ed., Springer-Verlag, 1991, pp. 129–140.

[104] M. Rabin. Digitalized Signatures and Public-Key Functions as Intractable as Factorization. Technical Report MIT/LCS/TR-212, MIT Laboratory for Computer Science, January 1979.

[105] C. Rackoff and D. Simon. Non-Interactive Zero-Knowledge Proofs of Knowledge and Chosen Ciphertext Attack. *Advances in Cryptology — CRYPTO '91*, LNCS 576, J. Feigenbaum ed., Springer-Verlag, 1991, pp. 433–444.

[106] R. Rivest and A. Shamir. How to Expose an Eavesdropper. *Communications of the ACM* 27(4): 393–395 (1984).

[107] R. Rivest, A. Shamir, and L.M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM* 21(2): 120–126 (1978).

[108] J. Rompel. One-Way Functions are Necessary and Sufficient for Secure Signatures. *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing*, ACM, 1990, pp. 387–394.

[109] A. Russell. Necessary and Sufficient Conditions for Collision-Free Hashing. *Journal of Cryptology* 8(2): 87–100 (1995).

[110] A. Sahai. Non-Malleable, Non-Interactive Zero Knowledge and Adaptive Chosen-Ciphertext Security. *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, IEEE, 1999, pp. 543–553.

[111] C.P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3): 161–174 (1991).

[112] A. Shamir and Y. Tauman. Improved Online/Offline Signature Schemes. *Advances in Cryptology — CRYPTO 2001*, LNCS 2139, J. Kilian ed., Springer-Verlag, 2001, pp. 355–367.

[113] V. Shoup. Why Chosen Ciphertext Security Matters. IBM Research Report RZ-3076, November, 1998.

[114] V. Shoup. On Formal Models for Secure Key Exchange. Available at `http://eprint.iacr.org/1999/012`.

[115] M. Steiner, G. Tsudik, and M. Waidner. Refinement and Extension of Encrypted Key Exchange. *ACM Operating Systems Review*, 29(3): 22–30 (1995).

[116] M. Tompa and H. Woll. Random Self-Reducibility and Zero-Knowledge Interactive Proofs of Possession of Information. *Proceedings of the 28th Annual Symposium on Foundations of Computer Science*, IEEE, 1987, pp. 472–482.

[117] T. Wu. The Secure Remote Password Protocol. *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, 1998, pp. 97–111.

[118] T. Wu. A Real-World Analysis of Kerberos Password Security. *Proceedings of the Internet Society Symposium on Network and Distributed System Security*, 1999, pp. 13–22.

[119] A.C. Yao. Protocols for Secure Computations. *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, IEEE, 1982, pp. 160–164.