Theory and Practice of Secret Commitment

by

Shai Halevi

B.A. Computer Science, Technion - Israel Institute of Technology, 1991

M.Sc. Computer Science, Technion - Israel Institute of Technology, 1993

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1997

Author ....[illegible].......................................
Department of Electrical Engineering and Computer Science
May 2, 1997

Certified by ....[illegible]..
Silvio Micali
Professor of Computer Science
Thesis Supervisor

Accepted by ....[illegible]........................
Arthur C. Smith
Chair, Departmental Committee on Graduate Students

Theory and Practice of Secret Commitment

by

Shai Halevi

Submitted to the Department of Electrical Engineering and Computer Science
on May 2, 1997, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

In this thesis we study a cryptographic primitive called *secret commitment schemes.* A secret commitment scheme is an electronic way to temporarily hide a value that cannot be changed. Such a protocol emulates the following two-stage process. In Stage 1, one party (called the Sender) locks a message in a box, and sends the locked box to another party (called the Receiver). In Stage 2, the Sender provides the Receiver with the key to the box, thus enabling him to learn the original message. Protocols for secret-commitment are used in many scenarios in cryptography, and quite a few variants of these schemes were discussed in the literature.

In the thesis we develop a general formal definition of secret commitment schemes which includes all the different variants as special cases, and demonstrate some basic properties of these variants. We also describe two constructions of these protocols which are more efficient than previous ones in the literature. The first construction is algebraic, and is based on the hardness of integer factorization, and it improves on previous schemes in that it does not require a complex set-up mechanism. The other is a more generic construction which is based on message-digest functions and is very efficient in terms of both communication and computations.

Thesis Supervisor: Silvio Micali

Title: Professor of Computer Science

Acknowledgments

First I wish to thank my advisor, Silvio Micali, for his guidance and support. Silvio's continuous encouragement and sound advice has been an invaluable resource for me throughout my studies in MIT. His never-ending enthusiasm and non-orthodox views make working with him an exciting experience.

Next I would like to thank Oded Goldreich and Shafi Goldwasser for teaching me so much about science and cryptography via classes, joint work, and many conversations. I also wish to thank them for serving on my thesis committee (and for helping me understand why I should write a thesis).

I am thankful to many people who influenced my studies and research. A very partial list of people with whom I had the opportunity to work and interact includes Shai Ben-David, Amir Ben-Dor, Ran Canetti, Benny Chor, Guy Even, Hugo Krawczyk, Erez Petrank, Tal Rabin, Ron Rivest and Assaf Schuster. I would also like to thank Amos Beimel, Rosario Gennaro, Stas Jarecki, Jon Kleinberg, Tal Malkin and Daniele Micciancio, for their friendship and for sharing knowledge and ideas in many conversations.

Special thanks to my mother Le'ah and my father Yiga'el, for all their love and support over the past 31 years.

And finally, I wish to thank (if this is the right word) my wife Tzipi, for putting up with all this through seven years and two continents, and for being a continuous source of happiness in my life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

... the opposite of an introduction, my dear Pooh, was a Contradiction;

A.A. Milne / The House At Pooh Corner

1.1 Preface: Information Security and Cryptography

In the last two decades we witness an "information revolution". These decades are characterized by an explosive growth in the availability and use of digital communication which is rapidly changing the ways we deal with information, and implies many far reaching effects on the way we live and work. The rapid advances in information technology which we witness these days offer new and exciting ways to communicate and interact with each another. Applications like automated telephone services, e-mail, and on-line databases are already a reality. In the near future we can expect to see electronic-commerce, video-conferencing and many more applications becoming widely available. All these new applications can be used to make our lives better, but at the same time they also pose new threats of being misused or abused. Combating these threats is the goal of information security.

Because of the immense variety of ways in which information is used (and can be misused), information security encompasses very many different aspects and scenarios.

These range from the simplest cases with only a single party and no interaction, to the ever-so-delicate protocols with many parties in different roles and conflicting interests. A few examples of scenarios where information security is a concern include

- A party may wish to protect its private information from being accessed by others. For example, a computer user may wish to make sure that its private files cannot be read or modified by anybody else.

- Two parties may wish to protect an exchange of information between them from begin accessed by others. For example, two computer users may wish to carry out an e-mail correspondence in such a way that no one else is able to read it, and so that they are assured of the integrity of the exchange.

- Even in the case of only two parties, we may have more involved scenarios. For example, consider an "identification" scenario where Party A tries to establish the identity of Party B via electronic communication. In this scenario, Party A needs a way to verify the identity of Party B, while Party B may wish to reveal as little as possible about itself in the process (e.g., to prevent Party A from using what it learns in this process in order to impersonate Party B at some later time). In this scenario, the parties may already have to follow a more involved protocol, spanning several exchanges of messages.

- Things become even more complicated when there are more than two parties involved. For example, a typical credit-card transaction may involve four parties or more (buyer, seller, credit-card company and a few banks), each with a different set of interests. Such a transaction may span a few separate protocols (i.e., establishing the relations among all the participants, issuing the credit card, the transaction itself, and the transfer of funds). An even more complex example is electronic elections, where the tally has to be computed accurately without violating the privacy of the voters. This last scenario may potentially involve millions of users.

Cryptography and Cryptographic Primitives. Roughly speaking, cryptography is the algorithmic part of information security. The role of cryptography is to distill the underlying problems in these real-life scenarios, and to design *cryptographic algorithms and protocols* to solve these problems. After two decades of active research, we now have a wide variety of cryptographic protocols to match the wealth of information security needs, ranging from encryption algorithms which protect sensitive data from being read, to generic protocols for multi-party computations which (in principle) can be used for complex tasks like electronic elections. Still, cryptographers keep inventing new protocols, either for solving the same problems in more efficient ways, or for solving new problems which arise from new applications.

As it turns out, there are several "building-blocks" which are used in a many different protocols for many different tasks. These building-blocks are called *cryptographic primitives.* These cryptographic primitives constitute a "bag of tools" which we can use in the design of many different protocols. This is very much like the way a steel-door and a lock can be used in a variety of "physical security" settings. Also, just like a lock can be build in several different ways (e.g., key lock, combination lock, card reader lock, etc.) these primitives too can typically be implemented in many different ways.

In this thesis we consider one such primitive, called a *secret commitment scheme.* This is a two-party protocol which can be used in some applications, as well as a building block inside larger cryptographic primitives.

1.2 Secret Commitment

A commitment scheme is a protocol between two parties, called the Sender and the Receiver. In this protocol, the Sender bound itself to a specific message in front of the Receiver, so that this message cannot be changed later without the Receiver knowing it. Clearly, one way to achieve this commitment is by having the Sender send the message itself to the Receiver. There are, however, situations where the message should be kept (temporarily) secret from the Receiver. Bounding the Sender to the message without revealing it to the Receiver is the goal of a secret commitment scheme.

It is a useful to think of a secret commitment scheme as emulating, by means of a protocol, the process of delivering a secret message to the Receiver in a locked box: Once the Receiver has the box, it is assured that the Sender cannot change the secret message in it, but the Sender is assured that the Receiver does not know the secret message yet. At some later time, the Receiver may be given the key to the box, and then it can open it and read the secret message in it.

A secret commitment protocol can therefore be viewed as comprised of two phases. The first phase – called the *Commit phase* – emulates the delivery of the locked box. When this phase is completed successfully, the Sender can not change the secret message any more, but the Receiver still can not read it. The second phase – called the *De-commit phase* emulates the delivery of the key. The Receiver can now open the box, verify that it indeed contains some message, and read this message.

Two Naive Attempts

To gain some intuition to the notion of secret commitment protocols, it may be useful to look at the following two simple constructions which seem to be doing "more or less what we need". As we explain below, however, none of these constructions actually achieves the security goals which we require from a secret commitment protocol.

Using Message-Digest Functions. A *message-digest function* is a function which takes as input a string of arbitrary length and produces as output a "digest" of some fixed length. The useful property of these functions is that it is typically very hard to find two different input strings which map to the same digest. Message-digest functions (such as SHA-1 [14] and RIPEMD-160 [13]) are used in may applications, and quite a few constructions of such functions are discussed in the literature (see [36] for a comprehensive discussion about these functions).

A conceivable way to use a message-digest function for secret commitment is to have the sender apply this function to its secret message and send the digest to the Receiver in the Commit phase. This way, the Sender is bound to its secret message since it is hard for it to find another string with the same digest, but the Receiver cannot get the secret message yet since it is "scrambled" via the message-digest function. In the de-commit phase the Sender sends the secret message itself to the Receiver, who can check that an application of the function to the string yields the right digest.

Although "reasonable" at first sight, the above construction is in fact *not secure enough.*[1] First, it is not at all clear that the message-digest function indeed scrambles the secret message so as to prevent the Receiver from understanding it. Moreover, it is clear that upon receiving the output of the function, the Receiver may dismiss possible candidate messages from consideration by checking that applying the function to them yields a different output. Thus, if there is a fairly small set of possible messages to begin with, the Receiver can find the right one by means of elimination. We stress that since we view secret commitment protocols as cryptographic primitive, we must ensure that they work in *any application* (e.g., even if only a single bit in the secret message is not known to the Receiver).

Using One-Way Permutations. Another construction can be obtained by using *one-way permutations.* A one-way permutation is a permutation over some domain (i.e., a function from the domain to itself which is one-to-one and onto), which is easy to compute but hard to invert. Namely, given the input to the function if is easy to figure out what the output is, but given the output is is very hard to find the input.

If the set of possible secret messages is contained in the domain of a one-way permutation, we may try to get a secret commitment by having the Sender apply the permutation to its string and send the output to the Receiver in the commit phase. The Sender is now bound to its secret since there is no other string which yields the same output. Notice that this is somewhat different than the previous construction, where the Sender is only bound to its secret as long as it cannot perform some computational task (namely finding two strings with the same digest). On the other hand, the Receiver cannot deduce the secret from the commitment since the

[1]Nevertheless, we show in Chapter 5 how this simple scheme can be fixed to yield a secure and efficient secret commitment protocol.

permutation is hard to invert. As before, in the de-commit phase the Sender sends the secret itself to the Receiver, who can check that an application of the permutation to it yields the right output.

In terms of secrecy, this scheme suffers from similar drawbacks as the previous one. Although the permutation may be hard to invert, it may very well "leak" some information about the secret. Moreover, we still have the problem that the Receiver is able to eliminate potential secrets from consideration by checking that an application of the one-way permutation to them yields the wrong output.

1.3 Applications of Secret Commitment

Secret Commitment plays an important role in the design of cryptographic protocols. Below we describe a few of their most important applications

Time Stamping. In many communication scenarios, there is a need for time-stamping mechanism, in which a "certification server" certifies the arrival time of documents. Possible examples of such scenarios include bidding for a contract (where all the bids must arrive by some specified deadline), problem-solving contests (where the first solution wins a prize), etc. In some of these scenarios we may need to keep the contents of the documents secret from the certification server. For example, when bidding for a contract, we may not trust the recipient to keep the bid from leaking to the competitors. (Alternatively, even if do we trust the certification server itself, we may not have a secure channel on which to send the document.)

A good way to use the certification server while keeping the documents secret is to have the server certify commitments for the documents (i.e., the box from above) instead of the documents themselves. Since the commitment bounds the Sender to its document, a time-stamp on the commitment is as good as a time-stamp on the document itself. At the same time, since we use a secret commitment, it does not reveal the contents of this document.

Ensuring Randomness. Just about every cryptographic protocol uses randomness to achieve security. Namely, the security of the parties in the protocol relies on the premise that some strings of bits which are used in the protocol were "chosen at random". Perhaps the simplest example is when two parties exchange a secret key which is to be used for encrypting future communication between them. When using this key, both parties rely on the premise that this key is "chosen at random" and therefore cannot be guessed by an outsider. The question then arises - how to pick these strings so that both parties can be assured of their randomness?

Using secret commitment as a mean of ensuring randomness was the first example in the literature of the use of secret commitment inside a larger primitive. It was shown by Blum [4] (and also in Blum-Micali [5]) how to use secret commitment in a protocol for "tossing coins over the phone". The idea in this protocol is that each party can pick its own random string, and the string which is used is the bitwise exclusive-or of the two strings. The only problem here is how to let the parties tell each other about their strings, without giving one of them the ability to determine its string only after seeing the string of the other. Here is where we use secret commitment. First, each party commits to its string. Only after this commitment is completed, the parties go on to exchange their strings. This way, we can be sure that the choices of the two strings are made independently, and thus - the outcome is random as long as at least one of the parties choose its string at random.

Zero-Knowledge Proofs. Zero-knowledge proofs, introduced by Goldwasser Micali and Rackoff in [21] are protocols between two parties, in which one party (the Prover) convinces the other (the Verifier) of the validity of some assertion, without revealing any information about how to prove this assertion. Zero-knowledge protocols - in turn - are used in many cryptographic settings.

Though it is not intuitively clear how secret commitment can be useful for such proofs, Goldreich, Micali and Wigderson demonstrated in [19], that secret commitment can be used to prove any NP type assertion in zero-knowledge. The same paradigm was employed by Brassard, Chaum and Crépeau in the slightly different setting of argument systems [1, 2].

1.4 Security Guarantees of Secret Commitment

A protocol for secret commitment must satisfy two requirements, corresponding to the security needs of the two parties involved. The *secrecy requirement* is that the Receiver cannot read the secret message before the de-commit phase, and the *commitment requirement* is that the Sender cannot alter the contents of the secret message after the commit phase. Each of these requirements can be satisfied either *computationally* (i.e., under the assumption that some computational task is infeasible for one of the parties), or *unconditionally* (i.e., without any such assumptions).

To demonstrate the difference between these notions, recall the two naive examples from above. In the first example (using message digest) the commitment requirement was satisfied only as long as the Sender was not able to find two strings with the same digest. However, since there exist many different strings with the same digest, then we must rely on the assumption that the Sender is unable to find two such strings. In the second example (using one-way permutations), no such assumption is necessary since the function we use is one-to-one and thus – for any output value there exists only one string which yields this output value. When we must make assumptions on the computational ability of the parties, we refer to these assumptions as the *computational assumptions underlying the security of the protocol.*

It is easy to see - and we prove it in Chapter 3 - that no protocol can have both unconditional commitment and unconditional secrecy. There are, however, constructions which achieve unconditional secrecy, and others which achieve unconditional commitment.

1.4.1 Deciding on Security Guarantees

A few aspects should be considered when deciding which type of secret commitment to use for a particular application.

1. In some applications, one of the parties has considerably more computing power than the other. If the strong party is the Sender then there is an advantage in using unconditionally committing protocol, and if the strong party is the Receiver then there is an advantage in using unconditionally secret commitment. When doing this, we can sometimes work with smaller security parameters, since the computational assumption which we make is that some task is infeasible for the weak party. Therefore, we may be able to obtain more efficient schemes without compromising security.

2. In some application we may care more about the security of one party than about the security of the other (e.g., say that violating the secrecy causes only a slight discomfort to the Sender, while violating the commitment may cause a significant harm to the Receiver). In this case, we may choose a commitment scheme in which breaking the underlying assumption only effects the interests of the "less important" party.

3. In some settings, it may be sufficient to guarantee the security of one party only for a limited time. As an example for this scenario, consider a setting in which many participants submit secrets documents to a competition, in which a single "winner" is chosen. Then, only the winner is required to reveal its document. Since the winner is announced soon after the submission of the secrets, then the participants do not have much time to try and violate the commitment requirement. On the other hand, we would like the secrets of the losers to remain secret "forever".

   When this happens, it makes sense to choose a commitment scheme so that the party whose security should be guaranteed for a longer time is protected unconditionally, and the other party is only protected computationally. For instance, in the example above we may wish to use an unconditionally secret commitment. This may also enable us to use a smaller security parameter, since the Senders has only very little time to try and break the commitment.

4. There are theoretical applications in which one must use one type of commitment schemes or the other to yield the desired result. For instance, in the zero-knowledge proofs for NP [19], one must use unconditionally binding commitments, while obtaining prefect zero-knowledge arguments for NP requires unconditionally secret commitment [1, 2, 26, 31].

1.5 Communication Models

Recall that a secret commitment is a protocol between two parties, the Sender and the Receiver. In a construction of such commitment, therefore, we must take into account the ways these parties can interact with each other. There are several possible communication models which we can consider. A few such models are described next.

1. **Interactive model**. In this model, both the Sender and the Receiver remain on-line throughout the protocol and can freely interact with each other. In this scenario, both the Commit and De-commit phases may consist of several messages which are sent back and forth between the parties.

2. **Non-interactive model**. In this model, the Receiver only operates off-line, and thus cannot send messages to the Sender. This means that both phases of the protocol consist only of messages sent from the Sender to the Receiver. (In fact, we can view each phase as consisting of a *single message* from the Sender to the Receiver.)

   A protocol which can be implemented in this model may be preferable to a protocol which requires interaction even in cases where the Receiver is capable of interacting with the Sender, since eliminating the need for interaction may make such a protocol more efficient and easier to implement. However, as we show in Section 3.4, there are settings where such protocols simply do not exist.

3. **Non-interactive model with initialization phase**. This model is somewhere in between the previous two models, in that it only allows a limited form of interaction. Specifically, in this model we have an additional phase before the Commit

and De-commit phases, called the *Initialization phase*, where a few "system parameters" are determined. For example, in a protocol which is based on the assumption that factoring is hard, the Initialization phase can be used to choose a random composite number (which is then assumed to be hard to factor).

We stress that these system parameters are set independently of the secret message which is committed to (i.e., we can think of the Initialization phase as taking place before the Sender has any secret to commit to). After these system parameters are set we require that the Commit and De-commit phases be non-interactive (but both the Sender and the Receiver can use the system parameters which were set in the initialization phase). Within this model, there are a few variants, depending on how the initialization phase can be implemented.

(a) Initialization by a Trusted Party. In this scenario there is a randomized algorithm for generating the system parameters, and there is a third party which is trusted to run this algorithm and send the results to both the Sender and the Receiver.

(b) Interactive Initialization. In this scenario there is no trusted party, but the Sender and the Receiver can interact in order to generate the system parameters. The initialization algorithm from above is thus replaced with an interactive two party protocol. The Sender and Receiver execute the initialization protocol first, and each of them keep its state between the Initialization and the other phases.

We stress again that this model is different than the model of interactive commitment, since the Initialization phase is independent of any secret message, and thus can be carried out ahead of time. The Commit and De-commit phases themselves are non-interactive.

Note that any protocol which can be implemented in this model can also be implemented in the trusted party model (by having the trusted party run the protocol with itself and send the results to Sender and the Receiver).

(c) Initialization with a Public Directory. In this model the Sender and Receiver cannot interact in the initialization, and there is no trusted party either. Instead, there is a public directory service which is only trusted to keep values which are sent to it by the parties and to make them publicly available. In the Initialization phase, the parties can therefore deposit some values in the directory service, and they later can retrieve each other's values for use in the Commit and De-commit phases.[2]

Most of the efficient secret commitment protocols (including the two in Chapters 4 and 5) work in this model. Typically, in these protocols only the Receiver deposits a value to the directory, and both parties use this value in the other phases. Also, it is clear that any protocol which can be implemented in this model can also be implemented in any of the two previous models.

1.6 Efficiency of Secret Commitment

As for any other protocol, there are three important resources to consider when designing commitment schemes, namely interaction, communication, and computation.

1. **Interaction.** Protocols are typically interactive because their parties communicate by exchanging messages back and forth. As we discussed above, however, there are cases where interaction is simply not possible, and even when it is possible - it may be very expensive. Therefore, it is preferable to have non-interactive schemes, or at least schemes with as little interaction as possible.

2. **Communication and State.** Another important resource in a protocol is the number of bits sent by the parties. In a commitment scheme, we measure the number of bits against the length of the secret message and the security parameter. Also, in a multi-phase protocol we may want to minimize the size of the state that the parties need to maintain between the different phases. We

[2]In reality, implementing such a directory service involves many complex constraints and requirements. In this thesis we ignore all these issues, as they are not directly relevant to secret commitment.

therefore would like to obtain schemes in which the number of communicated bits and state bits is as low as possible.

3. **Computation.** A third important resource is the amount of (local) computation that the parties need to spend during the protocol. Clearly, we want protocols which require as little effort as possible from the parties.

   (We stress that the computational efficiency objective is totally unrelated to the computational assumptions which we make on the parties. The later deals with the amount of effort it takes a "cheating party" to violate the security of the other party, while the former deals with the amount of effort it takes a honest party to execute its role in the protocol.)

1.7 Previous Work

Many protocols for secret commitment were discussed in the literature. We review these protocols below, and briefly describe their properties. In the discussion below we separate these protocols according to their security guarantees. In Table 1.1 we sketch some properties of protocols in the literature, as compared to the ones which we present in this thesis.

1.7.1 Protocols with Unconditional Secrecy

Many of the protocols in the literature which offer unconditional secrecy and computational commitment are based on number-theoretic constructions. The first such protocol was suggested by Blum [4], who described a secret commitment protocol for one bit, where the underlying computational assumption is that the Sender is not capable of factoring large integers. Blum's scheme calls for one or two modular multiplications and a k-bit commit string for every bit which is being committed to (where k is the size of the composite modulus).

A more efficient construction, which is based on the same computational assumption, is implicit in the work of Goldwasser, Micali and Rivest [22]. Their claw-free permutation-pairs enables one to commit to long secrets using about the same amount of local computation as in Blum's scheme, but to send only a k-bit commit string, regardless of the length of the secret. A similar construction was described explicitly in [1, 2][3]. The GMR-based construction was used since then in many other works (e.g. [3, 7, 10]).

One common problem of both the above constructions, however, is that they rely on composite numbers of a special form (i.e., product of two primes which are both congruent to 3 mod 4). Thus they require a special initialization procedure in which these special-form numbers are established. It should be noted that we can not let any of the parties pick these numbers on its own, since the security of both parties depend on the proper choice of the numbers. We also note that there is no known algorithm which lets the other party check that a given composite number is of the right form (although it is easy to check that the composite number is congruent to 1 mod 4, it is not known how to verify that it is indeed a product of exactly two primes). In Chapter 4 we describe a protocol which eliminates this problem.

Several other constructions are based on the computational assumption that the Sender in not capable of of extracting discrete-logarithms. In particular, [2] also show how to use claw-free permutation-pairs which are based on the hardness of the discrete-log problem for secret commitment. This construction, however, requires one modular exponentiation per bit in the secret. Pedersen [33] and Chaum, van-Heijst and Pfitzmann [7], described a construction in which the Sender can commit to a secret of length k (where k is the size of the prime modulus) by performing two modular exponentiations and sending a k-bit commit string.

There were also a few constructions of secret commitment protocols using more generic computational assumptions. Naor, Ostrovsky, Venkatesan and Yung [31] described a construction which can use any one-way permutation. Their scheme calls for $2k$ rounds of communication and one application of the one-way permutation for each bit which is being committed to. Finally a construction that uses message-digest

[3]The description in [1, 2] is stated in terms of a commitment to a single bit, but it can easily be generalized to allow committing to many bits.

functions (which is somewhat similar to the first "naive example" from above) was first mentioned by Naor and Yung in [30], and was later improved by Damgård, Pedersen, Pfitzmann [12] and by Halevi and Micali [24]. This protocol is described in Chapter 5. It calls for one application of the digest function to the secret and has a commit string of size k (where k is the security parameter) regardless of the secret length.

1.7.2 Protocols with Unconditional Commitment

A protocol for secret commitment which achieve computational secrecy and unconditional commitment (which is somewhat similar to the second "naive example" from above) was first presented in [5]. Although it was described in terms of the discreet logarithm problem, it can in fact use any one-way permutation. This protocol uses hard-core bits of any one-way permutation and requires one application of the one-way permutation for each bit which is being committed to. This scheme is described in [17, Construction 6.21]. In [29], Naor presented another construction which can be implemented using any pseudo-random generator (or, equivalently, any one-way function [25]). In the commit phase of Naor's scheme, the Sender generates a pseudorandom string of length linear in the length of the secret and sends a string of the same length to the Receiver.

1.8 Contributions of this Thesis

In this thesis we develop further the notion of secret commitment. The contribution of this thesis is two fold.

Formulation. In Chapter 3 we give a formal and general definition for the notion of secret commitment, and discuss a few subtleties in these definitions. Although several definitions of secret commitment can be found in the literature, none of them is general enough so as to capture all the different variants of this notion (in particular, none of these definitions includes the notion of an Initialization phase).

Constructions. In Chapters 4 and 5 we describe constructions of secret commitment protocols which have some advantages over previous ones. Specifically, in Chapter 4 we show how to construct a protocol which uses the Goldwasser-Micali-Rivest permutation pairs to obtain unconditional secrecy and computational commitment, under the assumption that the Sender cannot factor large composite integers. The advantage of our construction over previous ones is in the Initialization phase. The security of previous scheme rely on the use of composite integers whose prime factorization is unknown to the Sender, but for which it is known that this prime factorization is of a special form. Therefore, an interactive Initialization phase is required in order to convince the Sender that the prime factorization is of the right form without actually revealing it. Our protocol is unique in that the form of these composite integers does not effects the security of the Sender, so they can be selected by the Receiver. Thus, the interactive initialization is not needed.

In Chapter 5 we present another construction for secret commitment protocol which can use any message-digest function. This construction obtain unconditional secrecy and computational commitment, under the assumption that the Sender cannot find two different strings with the same digest. This construction is very efficient in terms of interaction, communication and computation. It is essentially the same as the construction in [12], but we provide a more direct proof of security for it.

Committing to an n-bit string with security-parameter k

Protocols with Unconditional Secrecy

the scheme	computational assumption	#-rounds in Commit-phase	length of commit-string	computation	Initialization
Blum [4]	factoring Blum-integers	1	O(k n)	n modular-multiplications	Interactive
NOVY [31]	one-way permutations	2k	O(k n)	n one-way permutations & nk^2 XOR's	No initialization
Pedersen [7,33]	solving discrete-log	1	O(max{k,n})	O(max{k,n}) modular-multiplications	Directory
GMR-based [2]	factoring Blum-integers	1	O(k)	O(n) modular-multiplications	Interactive
Chapter 4 (GMR-based)	**factoring Blum-integers**	**1**	**O(k)**	**O(k+n) modular-multiplications**	**Directory**
Chapter 5 (also [12])	**message-digest functions**	**1**	**O(k)**	**digest of n+O(k) bits & universal hash O(k) bits**	**Directory**

Protocols with Unconditional Commitment

the scheme	computational assumption	#-rounds in Commit-phase	length of commit-string	computation	Initialization
Blum-micali [5]	one-way permutations	1	O(k n)	n one-way permutations	No initialization
Naor [29]	pseudorandom generators	1	O(n)	error-correction encoding & O(n) pseudorandom bits	Directory

Table 1.1: Comparison between secret commitment protocols.

Chapter 2

Preliminaries

Pooh looked at his two paws. He knew that one of them was the right, and he knew that when you had decided which one of them was the right, then the other one was the left, but he never could remember how to begin.

A.A. Milne / The House At Pooh Corner

2.1 Basic Notations and Definitions

Throughout this thesis, the set of non-negative integers is denoted by $\mathcal{Z}^+$. For any non-negative integer $k \in \mathcal{Z}^+$, we denote by $\{0,1\}^k$ the set of binary-strings of length k, and by $\{0,1\}^{\leq k}$ the set of binary-strings of length at most k. We use $\{0,1\}^*$ to denote the set of all finite-length binary strings.

The notation $|\cdot|$ is used in three different standard ways which can easily be distinguished by the context: For a binary string $s \in \{0,1\}^*$, we denote by $|s|$ the number of bits in s. For a finite set S, we denote by $|S|$ the number of elements in S. Finally, if x is a real number then $|x|$ denotes the absolute value of x.

Distributions and statistical difference. A *probability distribution* (or just a distribution) over a countable set S is a function $D : S \longrightarrow [0,1]$ for which

$$\sum_{s \in S} D(s) = 1 \tag{2.1}$$

The *support set* of a distribution D is the set of elements $s \in S$ for which $D(s) > 0$. If D_1, D_2 are two distributions over S, the *statistical difference* between D_1 and D_2, denoted $\|D_1 - D_2\|$, is defined as

$$\|D_1 - D_2\| \overset{\text{def}}{=} \frac{1}{2} \sum_{s \in S} |D_1(s) - D_2(s)| \tag{2.2}$$

We note that for any two distributions D_1, D_2, we always have $0 \le \|D_1 - D_2\| \le 1$.

Syntax for probabilistic experiments. In the sequel we denote by $\alpha \leftarrow D$ the experiment in which we pick α according to the distribution D. Sometimes we also write $\alpha \leftarrow S$ (where S is a set) to denote picking α uniformly at random from the set S. For a predicate $p(\cdots)$, we use the syntax

$$\Pr\left[\begin{array}{l} \alpha_1 \leftarrow D_1 \\ \alpha_2 \leftarrow D_2 \\ \vdots \\ p(\alpha_1, \alpha_2, \ldots) \end{array}\right]$$

to denote the probability of the event in which we first pick α_1 from D_1, then α_2 from D_2, and so on, and in which at the end the α_i's chosen this way happen to satisfy the predicate p. In the simple case where the experiment consists of picking one element from one distribution, we use either of the notations

$$\Pr\left[\begin{array}{l} \alpha \leftarrow D \\ p(\alpha) \end{array}\right] \quad \text{or} \quad \Pr[p(D)] \quad \text{or} \quad \Pr_{\alpha \leftarrow D}[p(a)]$$

These notations are used interchangeably, depending on which looks more natural in a certain context.

2.2 Algorithms and Protocols

A probabilistic algorithm is an algorithm which is given - in addition to the usual input - a string of uniformly and independently chosen bits. For any input, the random choice of these bits induces a probability distribution over the behavior of the algorithm. This string of bits is often called the *random tape* of A. If A is a probabilistic algorithm and x is a possible input for this algorithm, then we denote by $A(x)$ the distribution over the possible outputs of A, which is induced by running A on input x. (Below we assume that A halts on any input and any random tape. This would be true for all the algorithm which are of interest to us in this thesis.)

A *probabilistic interactive algorithm* can be viewed as a (standard) probabilistic algorithm A which is run over and over again, retaining its state from one execution to the next. Every time after A produces an output, it is given a new input and then it is executed again from its current state, until it produces its next output.[1] We call each execution of A in this process a *communication round.* We view the first input to A in this process as its "real input" and the rest of the inputs as the messages which A receives during this execution. The last output of A is considered to be its "real output" and the other outputs are the messages which A sends.

We describe a two-party protocol by an ordered pair of probabilistic interactive algorithms (A, B). If (A, B) is a protocol and x, y are possible inputs for A and B, respectively, then we use the notation $\{A(x) \leftrightarrow B(y)\}$ to denote the distribution over the possible outputs which is induced by running the protocol (A, B) when A is given input x and B is given input y. The notation

$$(\alpha, \beta) \leftarrow \{A(x) \leftrightarrow B(y)\}$$

describes the experiment in which we run the protocol (A, B) on inputs (x, y), and then denote the output of A by α and the output of B by β.

[1]There should also be some syntax for termination: say that this process ends when A produces an output which ends in the special symbol $\dashv$. We ignore this matter from here on.

Don't-care values. In some cases we are only interested in the output value of one of the parties in a protocol. In these cases we denote the output of the other party by '$*$'. For example, if s is some binary string, then the expression

$$\{A(x) \leftrightarrow B(y)\} = (*, s)$$

denotes the event in which we run the protocol (A, B) on inputs (x, y) and the output of B happens to be s. That is, the output of A in this experiment is ignored.

Transcripts of protocols. If (A, B) is a protocol and (x, y) are inputs to this protocol, then a *transcript* of an execution of (A, B) on (x, y) is a sequence which consists of all the messages that were sent between the parties in this execution. We denote by $\mathbf{trans}\{A(x) \leftrightarrow B(y)\}$ the distribution over the possible transcripts which is induced by running the protocol (A, B) on inputs (x, y).

If t is any sequence of messages, A is an interactive algorithm and x is an input, we say that t *is consistent with* $A(x)$ if there exists an execution of $A(x)$ (with some partner) which generates the transcript t. Namely, if there exist an algorithm B and an input y so that the distribution $\mathbf{trans}\{A(x) \leftrightarrow B(y)\}$ assigns non-zero probability to t. (We note that this is equivalent to the condition that there exists some choice r of random tape of A such that if A is run on input x and random tape r, and if in each communication round i, A is given the $2i$'th message in t, then it produces in that round the $2i + 1$'st message in t.)

Multi-phase protocols. As we explained in the Introduction, sometimes we logically view a protocol as consisting of a few phases, where each of these phases accomplishes a certain task. In order to talk about multi-phase protocols, we view each phase as a protocol by itself, and the combined protocol is then viewed as a composition of all these phase protocols. A sequence of protocols (A_1, B_1), (A_2, B_2), (A_3, B_3), ... are composed into one protocol by giving the output of each party in the i'th protocol to the same party in the $i + 1$'st protocol. We denote an execution of this protocol on inputs (x, y) by

$$\begin{aligned}(\alpha_1, \beta_1) &\leftarrow \{A_1(x) \leftrightarrow B_1(y)\}\\(\alpha_2, \beta_2) &\leftarrow \{A_2(\alpha_1) \leftrightarrow B_2(\beta_1)\}\\(\alpha_3, \beta_3) &\leftarrow \{A_3(\alpha_2) \leftrightarrow B_3(\beta_2)\}\\&\dots\end{aligned}$$

When considering a multi-phase protocol, we sometimes give an input value to a party at an intermediate phase of the protocol. For this we use a similar syntax to the above. For example, if (A_1, B_1) and (A_2, B_2) are two phases of a protocol and x, y, z are inputs strings, then the notation

$$\begin{aligned}(\alpha_1, *) &\leftarrow \{A_1(x) \leftrightarrow B_1(y)\}\\(\alpha_2, \beta_2) &\leftarrow \{C(\alpha_1) \leftrightarrow D(z)\}\end{aligned}$$

describes the experiment in which we first run protocol (A_1, B_1) on inputs (x, y), and denote the outputs of A_1 by α_1, and then run protocol (A_2, B_2) on inputs (α_1, z), and denote the output of the parties by (α_2, β_2).

When we use this notation, we sometimes stress the difference between pre-specified parameters (such as x, y, z above) and the random outcomes in the experiment (such as $\alpha_1, \alpha_2, \beta_2$ above), by denoting the non-random parameters by small English letters and the random outcomes by small Greek letters.

Polynomial-time algorithms. An algorithm A is *Probabilistic Polynomial-Time* (PPT) if there exists some polynomial $Q : \mathcal{Z}^+ \to \mathcal{Z}^+$ such that for any input string $x \in \{0,1\}^*$, the running-time of A on x is always less than $Q(|x|)$.

An interactive algorithm is said to be PPT if there exists a polynomial as above which bounds the running time of A in each communication round. Namely, for any input x and sequence of messages $m_1, m_2, \dots$, the running time of A in the i'th round is always less than $Q(|x|)$.

2.3 Security

In most cryptographic protocols, there is always a slight chance that a "cheating party" will be able to compromise the security of the other party (e.g., by being very lucky and guessing the secret held by the other party). Thus, the goal in most protocols is not to make cheating impossible, but only to make the probability of cheating very small. We quantify what "very small" means via the notions of security-parameter and negligible functions. These are discussed next.

Negligible functions. We say that a function $f : \mathcal{Z}^+ \to [0,1]$ is *negligible* if as k gets larger, $f(k)$ goes to zero faster than any fixed polynomial in $1/k$. Formally,

$$\forall c > 0 \; \exists k_c \in \mathcal{Z}^+ \text{ so that } \forall k > k_c, \; f(k) < k^{-c} \tag{2.3}$$

Security-parameter. A *security parameter* in a cryptographic protocol is simply an integer that is used in the protocol to denote the required "level of security". The behavior of the parties in the protocol may depend on the value of the security parameter (typically, this value determines the length of the cryptographic keys that are used by the parties). Formally, the security parameter is given as input to the parties in the cryptographic protocol. This security parameter is encoded in unary (so the security-parameter $k \in \mathcal{Z}^+$ is denoted by 1^k), and the parties are allowed to work in time which is polynomial in this security parameter.

The "cheating probability" in such protocol is considered to be small enough if it is negligible in the security parameter which is used by the parties. Usually, such protocols can only guarantee this small "cheating probability" as long as the the length of the other inputs of the parties is polynomial in the security parameter.

Computational assumptions. As we explained in the Introduction, the security guarantees of cryptographic protocols may depend on the assumptions which we make on the computational power of the parties. Notice that when we deal with a "cheating party", we typically have no knowledge about the algorithm which this "cheating party" uses. Instead, we may only have some bound on the computational resources which are available for it. We therefore want to guarantee that the probability of cheating using these bounded resources (in any way) is low enough.

We formalize the notion of bounded resources using the notion of *complexity classes* of algorithm. We consider a class $\mathcal{C}$ of all the algorithms which only use some bounded resources, and allow a cheating party to pick any algorithm from this class. We formalize our security guarantee by saying that any algorithm from this class has only very low probability of cheating. In this thesis, the only classes of algorithm which we consider are the class of all the algorithms, and the class of PPT algorithms.[2]

Complexity conjectures vs. computational assumptions. It is worth stressing the difference between the computational assumptions discussed above and the complexity conjectures which we sometimes rely on when proving security of protocols. A complexity conjecture is a mathematical assertion of the form "no algorithm in class $\mathcal{C}$ can solve problem X" (e.g., "no PPT algorithm can factor large integers with non-negligible probability"). Such a conjecture is either true or false (and the reason we call it a conjecture is because as of today, we still do not know which is the case). On the other hand, the computational assumptions from above assert that a party in a protocol belong to some complexity class. This assertion does not have a well-defined truth value, since it depends on the specific application in which this protocol is used.

To further clarify this distinction, assume that we have a protocol which is secure as long as one of the parties cannot factor large integers (such a protocol is described in Chapter 4). Assume now that we were able to prove the conjecture that no PPT algorithm can factor large integers with non-negligible probability. Notice now that even in this case, to argue that the protocol is secure for a specific application *we still need to make the computational assumption* that this party is bounded to probabilistic

[2]The definitions in the thesis also make sense for any other "reasonable" complexity classes (as long as the running time is restricted to be sub-exponential in the security parameter).

polynomial time. (For example, this party might still be able to violate the security if using a "quantum-computer" [39], assuming such computers are at all possible.)

Auxiliary-inputs and non-uniformity. When a protocol is used in a larger environment (e.g., as a sub-protocol inside some larger protocol), this environment may contain details which are not used by this particular algorithm or protocol, but are still available for the parties running this protocol. Note that although the honest parties does not use these details, a "cheating party" may try to use them to break the scheme. For example, when using a secret commitment protocol inside a larger protocol, the parties may have some history at the point where the secret commitment protocol is involved. This history may include such details as previous commitments to other secret messages, cryptographic keys which are used for other parts of the larger protocol, etc. It makes sense, therefore, to strengthen the security guarantees of the protocol so that they still hold even if the parties try to use these additional details. Syntactically, we represent these details as additional inputs – called *auxiliary inputs* – which are available for the algorithms used by a "cheating party".

If the environment in which this protocol is used is very complex, or if this environment is not specified, then we may want to claim that the security guarantees hold *for all auxiliary inputs.* This forces us to view the algorithms used by the "cheating party" as non-uniform ones: A non-uniform algorithm can be viewed in this context as an algorithm which is given a fixed auxiliary input string, which is independent of all the other inputs to the protocol, except the security parameter. (Equivalently, non-uniform algorithms can be formulated via the notion of sequences of boolean circuits. See for example [32, Section 11.4].)

Chapter 3

Definitions

... it is a thing which you can easily explain twice before anybody knows what you are talking about.

A.A. Milne / The House At Pooh Corner

In this chapter we define formally the notion of a commitment scheme, and show how variants of secret commitment fit within the framework of the general definition. We also demonstrate a few basic properties of these variants.

(We note that although these scheme should probably be called secret commitment schemes – since they have elements of secrecy, as well as commitment – they are traditionally called just commitment schemes, so from now on we stick to this notion.)

3.1 Syntax and Validity

Syntactically, a commitment scheme is combined of three two-party protocols. Below we refer to the parties in these protocols as *Sender* and *Receiver*, and we call the protocols themselves *Initialization phase*, *Commit phase* and *De-commit phase.*

In the Initialization phase both parties are given the same input, which is the security parameter of the system, encoded in unary (we denote this by 1^k). The parties maintain a state between the Initialization and the Commit phase, which (as stated in Chapter 2) is represented by having the parties output this state at the end of Initialization phase and use it as an input to the commit phase.

In addition to this state, in the Commit phase the Sender is also given another input (denoted by m), which is the secret message to be committed to. As usual, to maintain a state between the Commit and De-commit phase, the parties output their state at the end of the Commit phase and use it as an input in the De-commit phase.

Finally, at the end of the De-commit phase the Receiver either outputs a string (which is supposedly the secret message of the Sender), or outputs the special symbol $\perp$ (meaning that the commitment was not opened correctly).

In the sequel, we sometimes refer to the states of the Sender and the Receiver after the Initialization phase as the *system parameters*, the state of the Receiver after the Commit phase as the *commit string*, and the state of the Sender after the Commit phase as the *de-commit string.*

The Validity Condition. Before we can even start talking about security, we first need to make sure that if all the parties are honest, then after the De-commit phase the Receiver is able to learn the secret message. This is captured by the following definition.

Definition 1 *Let $(I_S, I_R), (C_S, C_R), (D_S, D_R)$ be three protocols, where all the algorithms involved are PPT. We say that these protocols comprise a* commitment scheme *if the following requirement hold*

Validity: *For every $k \in \mathcal{Z}^+$ and every $m \in \{0,1\}^*$,*

$$\Pr\left[\begin{array}{rcl} (\rho_s, \rho_r) & \leftarrow & \{I_S(1^k) \leftrightarrow I_R(1^k)\} \\ (\tau_s, \tau_r) & \leftarrow & \{C_S(\rho_s, m) \leftrightarrow C_R(\rho_r)\} \\ (*,\ \sigma\ ) & \leftarrow & \{C_S(\tau_s) \leftrightarrow C_R(\tau_r)\} \\ & m = \sigma & \end{array}\right] = 1 \tag{3.1}$$

We note that Definition 1 says nothing about the security of the secret-commitment scheme. This is discussed next.

3.2 Defining Security

3.2.1 Towards a Definition

The semantics of a secret commitment should ensure that after the Commit phase the Receiver does not know anything about the secret message yet, but the Sender can not change it anymore. The definition of what it means for the Receiver "not to know anything about m", and for the Sender "not to be able to alter m" depends on the assumptions which we make on the computational power of the parties, which we formalize using the notion of complexity classes of algorithms (as was explained in Section 2.3). Below we describe the security guarantees in a somewhat informal manner, leaving the exact definition to Subsections 3.2.2 and 3.2.3.

Security guarantees. Let $\mathcal{C}_S, \mathcal{C}_R$ be two classes of algorithms, and let (I_S, I_R), (C_S, C_R), (D_S, D_R), be protocols which comprise a commitment scheme. We say that this commitment scheme is secure against Sender in $\mathcal{C}_S$ and Receiver in $\mathcal{C}_R$, if they satisfy two requirements, which we term *Secrecy* and *Commitment.*

Secrecy: The secrecy requirement tells us that for any two strings m_1, m_2, no algorithm from the class $\mathcal{C}_R$ (which may be employed by a "cheating Receiver") can distinguish between a commitment to m_1 and a commitment to m_2, except with probability which is negligible in the security parameter.

To make this formal, we fix any two strings m_1, m_2, and consider the experiments in which we first run the initialization phase and when run the Commit protocol either with m_1 or with m_2. The "cheating Prover" is given m_1, m_2 and it tries to guess whether this was a commitment to m_1 or to m_2. The secrecy requirement asserts that for any m_1, m_2 (of length polynomial in the security parameter), the advantage of the Receiver in guessing is only negligible in the security parameter.

Commitment: The commitment guarantee of a commitment scheme tells us that no algorithm from the class $\mathcal{C}_S$ (which may be employed by a "cheating Sender") can first execute the Initialization and Commit phases and then de-commit in two different ways, except with negligible probability.

To formalize this, we consider an experiment in which we first run the Initialization and Commit phases, and then run the De-commit phase when the Sender is given an extra input bit. The goal of the cheating Sender is to cause the Receiver to output two different strings at the end of the De-commit phase, depending on whether this "bit" was 1 or 2. The commitment requirement asserts that this goal should only be achieved with negligible probability.

Auxiliary inputs. As we explained in Chapter 2, when a commitment scheme is used inside a larger protocol, the parties may have auxiliary inputs from this higher level protocol, so we should require that the security guarantees still holds in the presence of such auxiliary inputs.

3.2.2 Committing to One Secret Message

We start by presenting a definition which is good enough for the case where we only use the system parameters which are generated in the Initialization phase to commit to a single secret message. In Subsection 3.2.3 we discuss the case where we use the same system parameters to commit to many secret messages.

Definition 2 *Let $\mathcal{C}_S$,$\mathcal{C}_R$ be two classes of algorithms, and let (I_S, I_R), (C_S, C_R), (D_S, D_R) be a commitment scheme. We say that this commitment scheme is secure against Sender in $\mathcal{C}_S$ and Receiver in $\mathcal{C}_R$, if the following two requirements hold*

Secrecy: *For any two algorithms $I_R^*, C_R^* \in \mathcal{C}_R$ and any polynomial $Q(\cdot)$, there exists a negligible function* $\text{neg}(\cdot)$, *so that any integer $k \in \mathcal{Z}^+$, any two strings $m_1, m_2 \in \{0,1\}^{\leq Q(k)}$ and any auxiliary input $z \in \{0,1\}^{\leq Q(k)}$*

$$\Pr\left[\begin{array}{l} (\rho_s, \rho_r) \leftarrow \ \{I_S(1^k) \leftrightarrow I_R^*(z, 1^k)\} \\ |p_1(z, \rho_s, \rho_r, m_1, m_2) - p_2(z, \rho_s, \rho_r, m_1, m_2))| > \text{neg}(k) \end{array}\right] \leq \text{neg}(k) \quad (3.2)$$

where p_1, p_2, are shorthands for the probabilities that the algorithm C_R^* guesses '1' when interacting with a Sender who is committing to m_1, m_2, respectively.

Namely, for $b \in \{1,2\}$ we denote

$$p_b(z,\rho_s,\rho_r,m_1,m_2) \stackrel{\text{def}}{=} \Pr[\ \{C_S(\rho_s,m_b) \leftrightarrow C_R^*(z,\rho_r,m_1,m_2)\} \ = \ (*,1)\ ]$$

Commitment: *For any three algorithms $I_S^*, C_S^*, D_S^* \in \mathcal{C}_S$ and any polynomial $Q(\cdot)$, there exists a negligible function* neg$(\cdot)$ *so that for any integer $k \in \mathcal{Z}^+$ and any auxiliary input $z \in \{0,1\}^{\leq Q(k)}$*

$$\Pr\left[\begin{array}{l} (\rho_s,\rho_r) \leftarrow \{I_S^*(z,1^k) \leftrightarrow I_R(1^k)\} \\ (\tau_s,\tau_r\ ) \leftarrow \{C_S^*(z,\rho_s) \leftrightarrow C_R(\rho_r)\} \\ (*,\ \sigma_1) \leftarrow \{D_S^*(z,\tau_s,1) \leftrightarrow C_R(\tau_r)\} \\ (*,\ \sigma_2) \leftarrow \{D_S^*(z,\tau_s,2) \leftrightarrow C_R(\tau_r)\} \\ \sigma_1 \neq \perp \ \textit{ and } \sigma_2 \neq \perp \ \textit{ and } \sigma_1 \neq \sigma_2 \end{array}\right] = \text{neg}(k) \qquad (3.3)$$

Non-uniformity. As we mentioned in Section 2.3, considering auxiliary inputs (without specifying how these are generated) has the effect of making the algorithms used by "cheating parties" non-uniform, and so any complexity conjecture which we make in order to prove the security of the scheme must hold in the non-uniform model. Also, since we require that the Secrecy requirement holds for any two strings m_1, m_2 which are given as inputs to the "cheating Receiver", then the algorithms used by this "cheating Receiver" must be considered non-uniform even in the absence of auxiliary inputs.

If we want to stay in the uniform model, we must restrict the auxiliary inputs and "secret messages" to those which can be generated by probabilistic polynomial-time algorithms. Informally, this means that although some strings may help a "cheating party", any PPT algorithm has only a negligible chance of finding these strings. We do not formalize this intuition here. The reader is referred to [16], where similar issues (for encryption schemes and zero-knowledge proofs) are treated formally.

3.2.3 Committing to Many Secret Messages

The above definition only guarantees the secrecy and commitment conditions as long as we only use the system parameters from the Initialization phase to commit to a single secret message. In many cases, however, we want to execute the Initialization phase only once, and then execute many instances of the Commit/De-commit phase *with the same security parameters.*

The reason why Definition 2 is not strong enough to handle this case is that, during the De-commit phase of a protocol, one party may have to reveal some secrets about the system parameters, which then can help the other party to violate the security of the system in a subsequent executions.

To illustrate this point, consider the naive attempt at a commitment scheme using one-way permutation which was described in the Introduction.[1] Assume that we slightly modify this construction by using a trapdoor one-way permutation. (Informally, a trapdoor one-way permutation is a one-way permutation for which there exists a trapdoor information that enables easy inversion. See, e.g., [17, Sec. 2.4.4]). This construction can then be modified as follows. In the Initialization phase, the Sender picks a one way permutation π and the corresponding trapdoor information t_π and send π to the Receiver. In the Commit phase, on secret message m, the sender computes $\pi(m)$ and send it to the Receiver. In the De-commit phase, the Sender sends t_π to the Receiver, who can then invert π and compute m.

Although this construction does not satisfy the stringent Secrecy condition of Definition 2, it does provide some measure of secrecy, since it is assumed that the Receiver cannot compute m from π and $\pi(m)$. However, it is clear that this construction cannot be used to commit to more than one secret message, since after the first De-commit the Receiver knows t_π and can easily invert π. For a more natural example of this phenomena, see the remark following the commitment scheme at the beginning of Section 5.2.

[1]Although that construction does not satisfy the stringent Secrecy condition in Definition 2, it still demonstrates the difficultywith this definition in the case of many commitments. Moreover, the same argument can be made about the construction in [17, Construction 6.21] which does satisfy Definition 2.

One plausible way to augment Definition 2 so that it can handle commitments to many messages, is to require that the parties "will not use any secrets about the system parameters". Formally, this means that the Secrecy and Commitment requirements still hold even when each of the parties is given both system parameters ρ_s, ρ_r as input in the Commit phase. Namely, we have the following definition

Definition 3 *Let $\mathcal{C}_S, \mathcal{C}_R$ be two classes of algorithms, and let (I_S, I_R), (C_S, C_R), (D_S, D_R) be a commitment scheme. We say that this commitment scheme is* strongly secure *against Sender in $\mathcal{C}_S$ and Receiver in $\mathcal{C}_R$, if the following two requirements hold*

Strong Secrecy: *For any two algorithms $I_R^*, C_R^* \in \mathcal{C}_R$ and any polynomial $Q(\cdot)$, there exists a negligible function* neg$(\cdot)$ *so that any integer $k \in \mathcal{Z}^+$, any two strings $m_1, m_2 \in \{0,1\}^{\leq Q(k)}$ and any auxiliary input $z \in \{0,1\}^{\leq Q(k)}$*

$$\Pr\left[\begin{array}{l} (\rho_s, \rho_r) \leftarrow \ \{I_S(1^k) \leftrightarrow I_R^*(z, 1^k)\} \\ |p_1(z, \rho_s, \rho_r, m_1, m_2) - p_2(z, \rho_s, \rho_r, m_1, m_2))| > \text{neg}(k) \end{array}\right] \leq \text{neg}(k) \quad (3.4)$$

where for $b \in \{1, 2\}$ we denote

$$p_b(z, \rho_s, \rho_r, m_1, m_2) \ \stackrel{\text{def}}{=} \ \Pr[\ \{C_S(\rho_s, m_b) \leftrightarrow C_R^*(z, \rho_s, \rho_r, m_1, m_2)\} \ = \ (*, 1)\ ]$$

(The only difference between this and the Secrecy requirement in Definition 2 is that C_R^* is also given ρ_s as input.)

Strong Commitment: *For any three algorithms $I_S^*, C_S^*, D_S^* \in \mathcal{C}_S$ and any polynomial $Q(\cdot)$, there exists a negligible function* neg$(\cdot)$*, so that for any integer $k \in \mathcal{Z}^+$ and any auxiliary input $z \in \{0,1\}^{\leq Q(k)}$*

$$\Pr\left[\begin{array}{l} (\rho_s, \rho_r) \leftarrow \{I_S^*(z, 1^k) \leftrightarrow I_R(1^k)\} \\ (\tau_s, \tau_r\ ) \leftarrow \{C_S^*(z, \rho_s, \rho_r) \leftrightarrow C_R(\rho_r)\} \\ (*,\ \sigma_1) \leftarrow \{D_S^*(z, \tau_s, 1) \leftrightarrow C_R(\tau_r)\} \\ (*,\ \sigma_2) \leftarrow \{D_S^*(z, \tau_s, 2) \leftrightarrow C_R(\tau_r)\} \\ \sigma_1 \neq \perp \ \ \textit{and}\ \sigma_2 \neq \perp \ \ \textit{and}\ \sigma_1 \neq \sigma_2 \end{array}\right] = \text{neg}(k) \qquad (3.5)$$

(Again, the only difference between this and the Secrecy requirement in Definition 2 is that C_S^* is also given p_r as input.)

It is easy to show that a commitment scheme which satisfies these stronger requirements remains secure even when the same system parameters are used to commit for polynomially many secrets.

Indeed, a "cheating Sender" can perfectly simulate the distribution over the transcripts of its conversations with the honest Receiver. Therefore, if the Sender is able to open a commitment in two different ways after committing (and de-committing) for polynomially many other secrets, it is also able to do that without these interactions. Similarly, a "cheating Receiver" can perfectly simulate the distribution over the transcripts of its conversations with the honest Sender, so executing polynomially many Commit/De-commit protocols with the Sender cannot help it get more than a negligible advantage in distinguishing between commitments to other secrets.

We finally note that all the commitment schemes in the literature, as well as the ones which we presented in this work, satisfy this strengthened requirements (under the same complexity conjectures used to prove (regular) security).

3.3 Variants of Secret Commitment

We now turn our attention to the different variants of commitment schemes which were discussed in the introduction, and show how they fit within the framework of the general definition above.

Unconditional vs. computational secrecy. A commitment scheme is said to be *unconditionally secret* if it is secure against an arbitrary Receiver. In terms of Definition 2, this means that the class $\mathcal{C}_R$ is the class of all algorithms. A scheme is said to be *computationally secret* if it is secure against a polynomial-time Receiver, so $\mathcal{C}_R$ in Definition 2 is the class of all PPT algorithms.

Unconditional vs. computational commitment. Similarly, we say that a commitment scheme is *unconditionally committing* if the class $\mathcal{C}_S$ in Definition 2 is the class of all algorithms, and we say that it is *computationally committing* if $\mathcal{C}_S$ is the class of all PPT algorithms.

Non-interactive schemes. A commitment scheme is said to be *non-interactive* if the interaction in both the Commit and the De-commit phases consists of a single string sent from the Sender to the Receiver. In this case we can assume w.l.o.g. that the state of the Receiver at the end of Commit phase is equal to the string which was sent by the Sender in this phase (which is why we termed it the *commit string*). Similarly, we can also assume w.l.o.g. that the string which is sent by the Sender in the De-commit phase is equal to its state after the Commit phase (which is why termed it the *de-commit string*). With respect to initialization, we can distinguish between the following variants:

No Initialization. Syntactically, we say that a scheme requires no initialization if both algorithms I_S, I_R in the Initialization phase are non-interactive (so the system parameters of each party can be computed directly from the security parameter, without the need to talk to the other party).

Initialization with a Public Directory. In this model, the Initialization phase consists of both parties depositing a value into a public directory, and retrieving each other's value before the Commit phase. Using our syntax, we formulate this by requiring that the interaction between the Sender and the Receiver in the Initialization phase consists of (at most) one message in each direction, and moreover, each honest party computes its message without seeing the other party's message.

Interactive Initialization. In this model there is no restriction on the behavior of the parties in the Initialization phase.

Initialization by a Trusted Party. This model does not fit into the framework of Definition 2, since Definition 2 allows a "cheating party" to deviate from its protocol during the Initialization phase, whereas the trusted party model assumes that the initialization is done properly. Definition 2 can be weakened to include schemes that can only be implemented in this model, by stating that the Secrecy and Commitment conditions only hold with respect to the pre-specified initialization algorithms I_S, I_R (instead of for every $I_R^* \in \mathcal{C}_R$ or every $I_S^* \in \mathcal{C}_S$, respectively).

3.4 Some Properties of Secret Commitment

We now proceed to show a few basic properties of commitment schemes.

3.4.1 A Characterization of Unconditional Secrecy

We start with a simple and useful lemma which characterize schemes with unconditional secrecy in terms of the communication in the Commit phase. We show that a commitment scheme enjoys unconditional secrecy if and only if for every two "secret messages" m_1, m_2, there is only a negligible statistical difference between the transcripts of committing to m_1 and to m_2.

Lemma 3.1 *A commitment scheme $(I_S, I_R), (C_S, C_R), (D_S, D_R)$ enjoys strong unconditional secrecy if and only if the following hold.*

For every two algorithms I_R^*, C_R^*, and any polynomial $Q(\cdot)$, there exists a negligible function $\text{neg}(\cdot)$ *so that for every $k \in \mathcal{Z}^+$, every $m_1, m_2 \in \{0,1\}^{\leq Q(k)}$ and every auxiliary input $z \in \{0,1\}^{\leq Q(k)}$*

$$\Pr\left[\begin{array}{l} (\rho_s, \rho_r) \leftarrow \;\; \{I_S(1^k) \leftrightarrow I_R^*(z, 1^k)\} \\ \|TR_1(z, \rho_s, \rho_r, m_1, m_2) - TR_2(z, \rho_s, \rho_r, m_1, m_2))\| > \text{neg}(k) \end{array}\right] \leq \text{neg}(k) \quad (3.6)$$

where TR_1, TR_2, respectively, are shorthands for the distribution over transcripts in the Commit phase when the Sender is committing to m_1, m_2. Namely, for $b \in \{1,2\}$, *we denote*

$$TR_b(z,\rho_s,\rho_r,m_1,m_2) \overset{\text{def}}{=} \textbf{trans}\{\{C_S(\rho_s,m_b) \leftrightarrow C_R^*(z,\rho_s,\rho_r,m_1,m_2)\}$$

Proof: One direction of the lemma holds since once we fix z,ρ_s,ρ_r,m_1,m_2 *and any setting of the random tape of* C_R^*, the output of C_R^* becomes a function of the transcript of the Commit phase. Thus, the statistical difference between the outputs of C_R^* when the Sender is committing to m_1 and its output when the Sender is committing to m_2 *conditioned on this fixed setting of the random tape of* C_R^* cannot be larger than the statistical difference between the transcripts when the Sender is committing to m_1 and when the Sender is committing to m_2, conditioned on the same random tape.

Since this holds for every setting of the random tapes, then it also holds when we average over all possible tapes for C_R^*, which means that for any z,ρ_s,ρ_r,m_1,m_2, we have

$$\begin{aligned}&|p_1(z,\rho_s,\rho_r,m_1,m_2) \;-\; p_2(z,\rho_s,\rho_r,m_1,m_2))|\\ &\le \|TR_1(z,\rho_s,\rho_r,m_1,m_2) \;-\; TR_2(z,\rho_s,\rho_r,m_1,m_2))\|\end{aligned}$$

where p_1, p_2 are defined as in Definition 3.

The other direction of this lemma follows since for given inputs $(z,\rho_s,\rho_r,m_1,m_2)$ an algorithm $\hat{C}_R^*$ (which is not restricted in its running time) can go over all possible random tapes for C_S and C_R^* and compute the list of all transcripts t for which $\Pr[TR_1(z,p_s,p_r,m_1,m_2)=t] > \Pr[TR_2(z,p_s,p_r,m_1,m_2)=t]$.

Then, $\hat{C}_R^*$ can execute the Commit phase with the Sender (using C_R^* on the Receiver's side) thus obtaining a transcript τ. Finally, if τ is on the pre-computed list then $\hat{C}_R^*$ outputs '1', and otherwise it outputs '2'. It follows by definition that for every pair (m_1,m_2), the statistical advantage of $\hat{C}_R^*$ in distinguishing between m_1 and m_2 is exactly $\|TR_1(z,\rho_s,\rho_r,m_1,m_2) - TR_0(z,\rho_s,\rho_r,m_1,m_2))\|$.

■

3.4.2 Computational Assumptions are Necessary

Proposition 3.2 *There is no commitment scheme which achieve both unconditional secrecy and unconditional commitment.*

Proof: Let $(I_S, I_R), (C_S, C_R), (D_S, D_R)$ be a commitment scheme, let m_1, m_2 be any two messages (e.g., m_1 ='0', m_2 ='1'). For $k \in \mathcal{Z}^+$ and $b \in \{1, 2\}$, we denote by $\text{View}_R(k, m_b)$ the distribution over pairs (ρ_r, τ) where ρ_r is the Receiver's system-parameter and τ is the transcript of the Commit phase when the Sender is Committing to m_b. Namely,

$$\text{View}_R(k, m_b) \stackrel{\text{def}}{=} \left\{ (\rho_r, \tau) : \begin{array}{l} (\rho_s, \rho_r) \leftarrow \{I_S(1^k) \leftrightarrow I_R(1^k) \\ \tau \leftarrow \textbf{trans}\{\{C_S(\rho_s, m_b) \leftrightarrow C_R(\rho_r)\} \end{array} \right\}$$

There are two possible cases to consider

1. $\|\text{View}_R(k, m_1) - \text{View}_R(k, m_2)\| > \frac{1}{2}$. In this case there exists an algorithm which distinguishes between these distributions with advantage of at least $\frac{1}{2}$. This algorithm runs the Initialization and Commit phases with all possible random tapes for I_S, I_R, C_S and C_R, for both messages m_1, m_2, and computes a list of all the pairs (ρ_r, τ) which have higher probability in $\text{View}_R(k, m_1)$ than in $\text{View}_R(k, m_2)$. Then it executes the Initialization and Commit phases with the Sender (using I_R, C_R on the Receiver's side) thus obtaining a pair (ρ_r, τ). If this pair is on the pre-computed list then the algorithm outputs '1', and otherwise it outputs '2'. It follows by definition that this algorithm has advantage at least $\frac{1}{2}$ in distinguishing between commitments to m_1 and commitments to m_2.

2. $\|\text{View}_R(k, m_1) - \text{View}_R(k, m_2)\| \leq \frac{1}{2}$. In this case, if we execute the experiment

$$\begin{array}{l} (\rho_s, \rho_r) \leftarrow \{I_S(1^k) \leftrightarrow I_R(1^k) \\ \tau \leftarrow \textbf{trans}\{\{C_S(\rho_s, m_1) \leftrightarrow C_R(\rho_r)\} \end{array}$$

then with probability at least $\frac{1}{2}$, the pair (ρ_r, τ) has non-zero probability according to the distribution $\text{View}_R(k, m_2)$. In particular, this means that (with probability at least $\frac{1}{2}$), there exists ρ'_s such that the transcript τ is consistent with $C_S(\rho'_s, m_2)$.

Given such a transcript τ and enough running-time, a "cheating Sender" can find ρ_s, ρ'_s and random-tapes r, r' so that

(a) The execution of C_S on input (ρ_s, m_1) and random-tape r is consistent with τ.

(b) The execution of C_S on input (ρ'_s, m_2) and random-tape r' is also consistent with τ.

we denote by δ_1, δ_2, respectively, the de-commit strings in the two executions above. Once δ_1, δ_2 are computed, the "cheating Sender" can open the commitment as m_1 by running $D_S(\delta_1)$ and as m_2 by running $D_S(\delta_2)$.

If there exists infinitely many integers k for which the first case happen then the scheme is not unconditionally secret, and if there exists infinitely many integers k for which the second case happen then the scheme is not unconditionally committing. ■

3.4.3 Non-interactive Secret Commitment

We next show that it is impossible to have a non-interactive commitment scheme without initialization which is unconditionally secret. More precisely, we prove the following:

Proposition 3.3 *A non-interactive commitment scheme without initialization which is unconditionally secret, cannot be (even computationally) committing.*

Proof: A non-interactive commitment scheme with no initialization can be described solely in terms of the algorithms C_S and D_R. Without loss of generality, we can assume that the algorithm C_S is given the security parameter 1^k and the string m and it outputs both the commit and de-commit strings, which we denote here, respectively, by γ, δ. The string γ is sent to the Receiver in the Commit phase and the string δ is sent in the De-commit phase. The algorithm D_R is then given $1^k, \gamma, \delta$, and it outputs either a string σ (which is presumably equal to m) or the special symbol $\perp$.

Below we denote the distribution which $C_S(1^k, m)$ induces over the commit strings γ by $COM(1^k, m)$.

Assume that we have algorithms C_S, D_R as above which unconditionally satisfy the Secrecy requirement. Then, by Lemma 3.1, for any polynomial $Q(\cdot)$ there exists a negligible function $\text{neg}(\cdot)$ so that any $k \in \mathcal{Z}^+$ and any two strings $m_1, m_2 \in \{0,1\}^{\leq Q(k)}$, we have

$$\|COM(1^k, m_2) - COM(1^k, m_1)\| \leq \text{neg}(k)$$

Therefore, if we fix any $k \in \mathcal{Z}^+$ and two strings $m_1, m_2 \in \{0,1\}^{\leq Q(k)}$, there exists a value γ for the commit string which has a non-zero probability according to both distributions $COM(1^k, m_2)$ and $COM(1^k, m_1)$. Hence, there exists a random tape r_1 on which C_S with input $(1^k, m_1)$ generates the commit string γ, and another random tape r_2 on which C_S with inputs $(1^k, m_2)$ generates the same commit string. Denote by δ_1 the de-commit string which is generated by C_S with input $(1^k, m_1)$ and random-tape r_1, and by δ_2 the de-commit string which is generated by C_S with input $(1^k, m_2)$ and random-tape r_2

Consider now the auxiliary input $z \stackrel{\text{def}}{=} (\gamma, \delta_1, \delta_2)$, and a "cheating Sender" which gets z as an auxiliary input, sends the commit string γ in the Commit phase, and sends either δ_1 or δ_2 in the De-commit phase. Clearly, this "cheating Sender" can be implemented in polynomial-time, and it violates the Commitment condition (with probability 1). ■

A few comments are in order here.

- The proof of Proposition 3.3 relies heavily on the fact that we require security against non-uniform Senders (namely, that the Sender is given an arbitrary auxiliary input). Indeed, no such proof is known for the uniform case.

- As opposed to unconditionally secret schemes, it is easy to see that under standard complexity conjectures (such as the existence of one-way permutations) one can construct unconditionally committing (and computationally secret) schemes which are non-interactive and have no initialization (e.g, [17, Construction 6.21.]).

Chapter 4

An Algebraic Construction

"Listen to this, Piglet", said Eeyor, "and then you'll know what we're trying to do"

A.A. Milne / The House at Pooh Corner

In this chapter we present a construction which uses the claw-free permutation-pairs of Goldwasser, Micali and Rivest [22], and is a modification of the scheme which is presented in [2]. This construction achieves unconditional secrecy and computational commitment under the factorization conjecture which we describe below.

A main difference between this construction and the GMR-based construction in [2], is in the Initialization phase. The original scheme (as well as most other factoring based schemes) relies on composite numbers of a special form, and it requires an interactive initialization protocol in which these special-form numbers are established. In that scheme we cannot let any of the parties pick these special form composites on its own, since the security of both parties depend on the proper choice.

We present here a new technique which eliminates the need for interactive initialization. Instead, in our scheme we simply let the Receiver choose the composite number and send it to the Sender (so it can be implemented in a non-interactive model with directory). Our scheme is unique in that the form of this composite does not effects the security of the Sender.

This chapter is organized as follows: We start by presenting some number-theoretic background material which is needed in order to understand the constructions. We then briefly discuss the scheme due to Blum which was the first scheme to use this number-theoretic approach. Next we describe a GMR-based scheme similar to [1, 2], which improves on Blum's scheme in terms of the length of the commit-string, and finally we show our scheme which improves on the GMR-scheme by eliminating the expensive initialization step.

4.1 Number-Theoretic Background

We start by briefly presenting some background material from number theory which we use in this constructions. The following are a few basic definition, notations and facts which we need in the sequel. Proofs of the facts below can be found in any elementary textbook on Number Theory (e.g., [28, Ch. I,II]).

1. An integer $p > 1$ is said to be *a prime* if the only positive integers which divide it are 1 and p itself. An integer which is not a prime is said to be *a composite.* We say that two integers are *relatively primes* if their greatest common divisor is one.

2. For any integer $N > 1$, the set of all the positive integers which are smaller than N and are relatively primes with N, together with the operation of multiplication mod N, form an Abelian group. We denote this group by Z_N^*, and the number of elements in this group is denoted by $\varphi(N)$.

3. If p is an odd prime power (i.e., $p = q^e$ for a prime q and a positive integer e) then the group Z_p^* is cyclic. Namely, there exists an element $g \in Z_p^*$ such that $Z_p^* = \{g^i \bmod p \ : \ 0 \leq i < \varphi(p)\}$. Such an element is called *a generator* of Z_p^*.

4. For an integer $N > 1$, an element $y \in Z_N^*$, is said to be a *quadratic residue* mod N is there exists $x \in Z_N^*$ so that $x^2 \equiv y \pmod{N}$. In this case we say that x is a square-root of y mod N. The set of quadratic residues mod N forms a subgroup of Z_N^*, which we denote by QR_N.

5. (The Chinese Remainders Theorem). If p, q are relatively primes and $N = pq$, then for any element $x \in Z_p^*$ and any element $y \in Z_q^*$ there exists a unique element $z \in Z_N^*$

so that $z \equiv x \pmod{p}$ and $z \equiv y \pmod{q}$.

6. If p is a prime and $p \equiv 3 \pmod{4}$, then for every element $x \in Z_P^*$, exactly one of the elements x, $p - x$ is a quadratic residue mod p.

7. If p, q are primes, with $p \equiv 3 \pmod{8}$ and $q \equiv 7 \pmod{8}$, then 2 is a quadratic residue mod q but not mod p.

8. If p, q are primes and $N = pq$, then an element $x \in Z_N^*$ is a quadratic residue mod N if and only if (x mod p) is a quadratic residue mod p and (x mod q) is a quadratic residue mod q.

9. In the sequel, we say that an integer N is a *Williams integer* [40] if it is a product of two primes, one which is congruent to 3 mod 8 and another which is congruent to 7 mod 8. (These integers are a special case of *Blum integers*, which are products of two primes, both congruent to 3 mod 4.)

10. If N is a Williams integer, then every quadratic residue mod N has exactly four square-roots mod N, and exactly one of these square-roots is itself a quadratic residue mod N.

4.1.1 The Factorization Conjecture

The conjecture we need in order to prove the security of our scheme is that factoring Williams integers is infeasible. Formally, for any integer k denote by $PRIMES_3(k)$, $PRIMES_7(k)$, respectively, the uniform distributions over all k-bit primes which are congruent to 3 mod 8, and 7 mod 8. Then we have

The Factorization Conjecture. For any PPT algorithm A and any polynomial $Q(\cdot)$ there exists a negligible function neg$(\cdot)$ so that for any $k \in \mathcal{Z}^+$ and any auxiliary input $z \in \{0,1\}^{\leq Q(k)}$,

$$\Pr\left[\begin{array}{l} p \leftarrow PRIMES_3(k) \\ q \leftarrow PRIMES_7(k) \\ N \leftarrow pq \\ A(z, N) = (p, q) \end{array}\right] \leq \text{neg}(k)$$

(where pq denotes the product of the integers p and q, and all the integers are represented in binary).

4.1.2 The GMR Claw-Free Permutation Pairs

In [22], Goldwasser, Micali and Rivest described the following construction: Let N be a Williams integer, and we start by defining two functions

$$f_{N,0}(x) \stackrel{\text{def}}{=} x^2 \pmod N \quad \text{and} \quad f_{N,1}(x) \stackrel{\text{def}}{=} 4x^2 \pmod N$$

Then, for any string $s = b_1 b_2 \cdots b_n$ we define $f_{N,s}(x) \stackrel{\text{def}}{=} f_{N,b_1}(\cdots f_{N,b_n}(x)\cdots)$. The properties of these functions which we use in the sequel are described next.

Proposition 4.1 **([22])** *If N is a Williams integer, then both function $f_{N,0}(\cdot)$ and $f_{N,1}(\cdot)$ are permutations over the quadratic residues mod N.*

Proof: Fact 10 in Section 4.1 immediately implies that squaring is a permutation over the quadratic residues mod N. Also, since four is a quadratic residue mod N, then multiplication by four is also a permutation over the quadratic residues mod N. Finally, a composition of permutation is itself a permutation. ■

Corollary 4.2 *If N is a Williams integer, then for any string $s \in \{0,1\}^*$, the function $f_{N,s}$ is a permutations over the quadratic residues mod N.*

Proposition 4.3 **([40, 22])** *Under the factorization conjecture, no PPT algorithm can find – given a random Williams integer N – two quadratic residues $x, y \in QR_N$ for which $f_{N,0}(x) = f_{N,1}(y)$. More precisely, for every PPT algorithm A and every polynomial $Q(\cdot)$ there exists a negligible function* neg$(\cdot)$ *so that for any $k \in \mathcal{Z}^+$ and* *any auxiliary input* $z \in \{0,1\}^{\leq Q(k)}$

$$\Pr\left[\begin{array}{l} p \leftarrow PRIMES_3(k) \\ q \leftarrow PRIMES_7(k) \\ N \leftarrow pq \\ (x,y) \leftarrow A(z,N) \\ x,y \in QR_N \textit{ and } f_{N,0}(x) = f_{N,1}(y) \end{array}\right] \leq \text{neg}(k)$$

Proof: Assume that the Proposition statement does not hold for some PPT algorithm A, and we show another PPT algorithm A' which violates the factorization conjecture. Algorithm A' is given the auxiliary input z and the Williams integer $N(=pq)$. It first runs $A(z,N)$, and obtains its output (x,y). It then outputs $p = \gcd(x+2y, N)$ and $q = \gcd(x-2y, N)$.

We now show that when N is a Williams integer, if $x,y \in QR_N$ and $x^2 \equiv 4y^2 \pmod N$, then p,q are the prime factors of N. Since we assumed that the above conditions hold with probability which is not negligible (for some auxiliary input z), then A' indeed violates the factorization conjecture.

Let N be any Williams integer (and denote its prime factors by p,q), and let x,y be any elements in QR_N such that $x^2 \equiv 4y^2 \pmod N$. Therefore, we have

$$x^2 - 4y^2 = (x+2y)(x-2y) \equiv 0 \pmod N \tag{4.1}$$

From Fact 8 in Section 4.1 we know that since x,y are quadratic residues mod N, then they are also quadratic residues mod p and q (when reduced mod p,q respectively). From Facts 6 and 7 in Section 4.1 we know that 2 is not a quadratic residues mod p and $q-2$ is not a quadratic residues mod q, which implies that $(2y \bmod p)$ is not a quadratic residues mod p and $(-2y \bmod q)$ is not a quadratic residues mod q. We therefore conclude that

$$\begin{array}{ll} x \not\equiv 2y \pmod p & \Rightarrow x - 2y \not\equiv 0 \pmod p \\ x \not\equiv -2y \pmod q & \Rightarrow x + 2y \not\equiv 0 \pmod q \end{array}$$

Denote now $u \stackrel{\text{def}}{=} x - 2y$ and $v \stackrel{\text{def}}{=} x + 2y$, and consider Eq. (4.1). Since the right-hand-side of it is divisible by N, the so is the left-hand-side, so we know that uv is divisible by both p and q. On the other hand, we know that u is not divisible by p and v is not divisible by q. Since p, q are primes, it must be the case that u is divisible by q but not p, and that v is divisible by p but not q. Therefore, $\gcd(v, N) = p$ and $\gcd(u, N) = q$. ■

Corollary 4.4 ([22]) *Under the factorization conjecture, no PPT algorithm can find – given a random Williams integer N – two quadratic residues $x, y \in QR_N$ and two strings $s_0, s_1 \in \{0,1\}^*$ so that none of s_0, s_1 is a prefix of the other and $f_{N,s_0}(x) = f_{N,s_1}(y)$.*

More precisely, for every PPT algorithm A and every polynomial $Q(\cdot)$ there exists a negligible function $\text{neg}(\cdot)$ so that for any $k \in \mathcal{Z}^+$ and any auxiliary input $z \in \{0,1\}^{\leq Q(k)}$

$$\Pr\left[\begin{array}{l} p \leftarrow PRIMES_3(k) \\ q \leftarrow PRIMES_7(k) \\ N \leftarrow pq \\ (s_1, s_2, x, y) \leftarrow A(z, N) \\ s_0, s_1 \textit{ are not prefixes of one another} \\ \textit{and } x, y \in QR_N \textit{ and } f_{N,s_0}(x) = f_{N,s_1}(y) \end{array}\right] \leq \text{neg}(k)$$

Proof: Assume that N is a Williams integer and that we have (s_1, s_2, x, y) as above, and we show how to compute $x', y' \in QR_N$ so that $f_{N,0}(x) = f_{N,1}(y)$. Since none of s_0, s_1 is a prefix of the other, there exists a (possibly empty) string s such that we can write s_0 as a concatenation $s_0 = ss'_0$ and s_1 as a concatenation $s_1 = ss'_1$, and the first bit in s'_0, s'_1 is not the same.

We note that since $f_{N,s}(f_{N,s'_0}(x)) = f_{N,s_0}(x) = f_{N,s_1}(y) = f_{N,s}(f_{N,s'_1}(y))$, and since $f_{N,s}(\cdot)$ is a permutation, then $f_{N,s'_0}(x) = f_{N,s'_1}(y)$. Assume now w.l.o.g. that the first bit in s'_0 is 0 and the first bit in s'_1 is 1. We then can write $s'_0 = 0s''_0$ and $s'_1 = 1s''_1$.

Finally, denote $x' \overset{\text{def}}{=} f_{N,s_0''}(x)$ and $y' \overset{\text{def}}{=} f_{N,s_1''}(y)$, then $x', y' \in QR_N$ and we have

$$f_{N,0}(x') = f_{N,0s_0''}(x) = f_{N,s_0'}(x) = f_{N,s_1'}(y) = f_{N,1s_1''}(x) = f_{N,1}(y')$$

■

4.2 The Blum Scheme

The first commitment scheme to use the above algebraic constructions was due to Blum [4]. The Blum scheme uses a quantity, called the *Jacobi symbol*, which can be computed for any element $x \in Z_N^*$. For an element $x \in Z_N^*$, the Jacobi-symbol of x mod N is denoted $(\frac{x}{N})$, and it always equals to either $+1$ or -1. (For a definition of the Jacobi symbol, see, e.g., [28, §II.2].) The Blum scheme uses the following properties of the Jacobi symbol:

a. There is a polynomial time algorithm which, given x, N, computes $(\frac{x}{N})$.

b. If N is a Blum-integer, then every quadratic-residue mod N has two square-roots with Jacobi-symbol $+1$ and two square-roots with Jacobi-symbol -1.

c. If N is a Blum-integer, and if $x, y \in Z_N^*$ are such that $(\frac{x}{N}) = +1$, $(\frac{y}{N}) = -1$, and $x^2 = y^2 \pmod{N}$, then given x, y, N, one can compute the prime factorization of N in polynomial time.

In the Blum scheme, the initialization phase consists of picking a random Blum-integer with k bits (on security parameter 1^k), in such a way that its prime factorization is unknown to the Sender. To commit to single bit $b \in \{0,1\}$, the Sender picks an element $x \in Z_N^*$, computes $y = x^2 \bmod N$, $s = (\frac{x}{N}) \cdot (-1)^b$ and Send y, s to the Receiver. Notice that this means computing one modular-multiplication and one Jacobi-symbol computation, and sending $(k+1)$ bits for every bit in the secret.[1]

To de-commit the bit b, the Sender sends x to the Receiver, who can then compute $(\frac{x}{N})$ and therefore b. Thus, the de-commit string also contains k bits for every bit in

[1]In fact, the Jacobi-symbol computation can be replaced with another modular multiplication using a simple trick.

the message.

We note that the Blum scheme is non-interactive, but has very long commit and de-commit strings, and it requires interactive initialization to choose the Blum-integer. In the next section we describe a scheme which is based on the GMR construction and reduces the size of the commit and de-commit strings, but still requires interactive initialization. Then, in Section 4.4 we show how me modify the GMR-based scheme to eliminate the interactive initialization.

4.3 The GMR-based Scheme

The following is a simple commitment scheme that uses the GMR construction (which is very similar to the scheme in [1, 2]). We assume that the Sender and the Receiver use some prefix free encoding function Enc so that for no two strings $m \neq m'$ is $Enc(m)$ a prefix of $Enc(m')$. An example of an encoding with this property is $Enc(b_1 b_2 \dots b_n) = b_1 0 b_2 0 \dots b_n 1$. Using this simple encoding we have $Enc(m) = 2|m|$. (There are also prefix-free encodings where $Enc(m) = |m| + O(\log |m|)$.)

Initialization: On security parameter 1^k, the Sender and the Receiver "choose at random" a Williams integer N with k bits, in such a way that the Sender does not know the prime factors of N. This Williams integer is then used as the system-parameter for both the Sender and the Receiver. We discuss this phase in more details below.

Commit phase: Given a secret $m \in \{0,1\}^*$, the Sender computes $s = Enc(m)$, picks a random element $x \in Z_N^*$ and sends $y = f_{N,s}(x^2)$ to the Receiver.

De-commit Phase: The Sender sends both m and x to the Receiver. the Receiver computes $s = Enc(m)$ and verifies that $y = f_{N,s}(x^2)$.

It is obvious from the description that this scheme satisfies the Validity requirement in Definition 2. To show that it also satisfies the other two conditions we prove two claims:

Theorem 4 *The scheme above enjoys unconditional Secrecy, provided that the Initialization phase guarantees that the system parameter N is a Williams integer.*

Proof: If N is indeed a Williams integer, then by Corollary 4.4, $f_{N,s}$ is a permutation for every s. Thus, for every y (which is a quadratic residue mod N) and every s, there exists exactly one quadratic residue mod N, denoted x, such that $y = f_{N,s}(x)$. This means that for any s and any $y \in QR_N$

$$\Pr_{x \leftarrow Z_N^*} \left[f_{s,N}(x^2) = y \right] = \frac{1}{|QR_N|}$$

Therefore, the distribution which is induced on the message sent by the Sender in the Commit phase is uniform over QR_N, regardless of s (and hence, also of m), which implies that the Receiver has no advantage in guessing the Sender's secret. ■

Theorem 5 *Under the Factorization Conjecture, the above scheme enjoys computational Commitment, provided that the Initialization phase guarantees that the distribution induced over the system parameter N is the same as the distribution in the Factorization conjecture.*

Proof: This is an immediate consequence of Corollary 4.4. ■

4.3.1 Efficiency of the Scheme

Interaction. The Commit and De-commit phases in the scheme above are non-interactive. The Initialization phase, however, must be interactive to ensure that the system-parameter is a Williams integer. We return to this matter in the next section.

Communication. On security parameter 1^k, the number of bits sent in the Commit phase above is always equal to k, regardless of the length of the secret m, since the system-parameter N is a k-bit integer and Sender sends an element in Z_N^*. In the De-commit phase, the Sender sends the secret m itself and k more bits.

Running-time. In the scheme above, the Sender performs one or two modular multiplications for every bit in s (which has about the same length as m) to compute the commit string. This amount of work can be somewhat reduced using a construction by Damgård [8]. This construction uses larger families of permutations to reduce the number of multiplication. The idea is to use (say) 256 different permutations rather than just two, and to view s as a sequence of bytes, where each byte specifies one permutation. Of course, for every r we can use the same idea to get one or two multiplications per r bits of s using 2^r permutations.

Clearly, we pay for this saving in running-time by having to keep many more bits to describe these larger families of permutations, and by having to choose one of these families in the Initialization phase. Moreover, the security properties of this scheme are somewhat weaker than those of the original scheme. In particular, in the original scheme we could convert an algorithm with probability ϵ of braking the Commitment requirement into one which factors the system parameter N with the same probability. In the new scheme, instead, the success probability of the factorization algorithm is something like $\frac{\epsilon}{2^{r-1}}$.

4.3.2 The Initialization Phase

The main problem with the above scheme is the implementation of the initialization phase. Clearly, it is important to choose the system parameter N in such a way that the Sender will not be able to factor it easily. Notice, however, that it doesn't matter whether the Receiver knows the factorization of N. One idea is therefore to let the Receiver choose N in the appropriate way and send it to the Sender. But if the Sender doesn't know the factorization of N, how can it verify that N it is a Williams integer?

At first glance this may not seem like a real problem. After all, the Sender can choose the starting point x at random, so it may be able to hide the string s from the Receiver even if $f_{N,0}, f_{N,1}$ are not permutations. Unfortunately, this is not the case. Consider for example $N = 5$ and a string of one bit b. It is easy to see that for any element $x \in Z_5^*$ we have $f_{5,0}(x^2) = 1$ and $f_{5,1}(x^2) = 4$. Thus the Receiver can recover the bit from the commitment string.

One way to solve this problem by letting the Receiver choose N and then prove (by means of a zero-knowledge proof) to the Sender that it is of the right form. This zero-knowledge proof, however, must be interactive, and it can be quite expensive in terms of running time and communication. It will therefore be desirable to have a scheme where choosing a "bad N" does not help the Receiver to get any advantage in guessing m from the commit string. We present such a scheme below.

4.4 Our Scheme

The only difference between the following scheme and the previous one is that after computing $y = f_{N,s}(x^2)$, the Sender squares y for k more times (where k is the number of bits in N) and sends the result to the Receiver. Note that using our notation, this is equivalent to computing $f_{N,0^k s}(x^2)$. The new scheme is:

Initialization: On security parameter 1^k, the Receiver picks at random two primes $p \leftarrow PRIMES_3\left(\left\lceil\frac{k}{2}\right\rceil\right), q \leftarrow PRIMES_7\left(\left\lceil\frac{k}{2}\right\rceil\right)$, computes $N \leftarrow pq$ and sends N to the Sender. The Sender verifies that N is of the right length.

Commit phase: Given the system parameter N (of k bits) and a secret m, the Sender computes $s = Enc(m)$, picks a random element $x \in Z_N^*$ and sends $y \leftarrow f_{N,0^k s}(x^2)$ to the Receiver.

De-commit Phase: The Sender sends both m and x to the Receiver. The Receiver computes $s = Enc(m)$ and verifies that $y = f_{N,0^k s}(x^2)$.

It is clear that the new scheme still enjoys computational commitment under the Factorization conjecture. Namely, if the Receiver picks N according to the protocol then it is infeasible for the Sender to find two different secrets with the same commit string (if factoring is hard). The harder part is to show that even if the Receiver tries to "cheat" by picking a "bad" N, it still does not get any advantage in guessing the secret m from the commit string.

4.4.1 Proof of Secrecy

We need to show that the modified scheme satisfies the secrecy requirement for any integer N (even if N is not "of the right form"). To do that we prove the following lemma

Lemma 4.5 *Let N be any integer and denote the number of bits in N by k, and let s_0, s_1 be any two strings. Then, for any element $y \in Z_n^*$ we have*

$$\Pr_{x \leftarrow Z_N^*}\left[f_{N,0^k s_0}(x^2) = y\right] = \Pr_{x \leftarrow Z_N^*}\left[f_{N,0^k s_1}(x^2) = y\right]$$

Below we give an elegant proof for Lemma 4.5 which is in part due to Damgård (private communication).[2] For this proof, we need to review a few facts. The first fact about the function $f_{s,N}(\cdot)$ was first observed by Goldreich:

Proposition 4.6 ([15]) *For any integer N, any string s and any element $x \in Z_N^*$, $f_{s,N}(x) = 2^{2\hat{s}} \cdot x^{2^{|s|}}$, where $\hat{s}$ is the integer whose binary representation is s.*

We now need some facts about the structure of the group Z_N^*. We start with a definition and a few notations:

Definition 6 *Let n be an integer, $n > 1$, and let x be an element in Z_n^*.* The order of x, *denoted $ord(x)$, is the smallest positive integer e so that $x^e = 1 \pmod{n}$. We denote by O_n the subset containing all the elements of odd order in Z_n^*. That is,*

$$O_n \stackrel{\text{def}}{=} \{x \in Z_n^* \ : \ ord(x) \text{ is odd}\}$$

Proposition 4.7 *Let n be an integer, $n > 1$. Then O_n is a subgroup of Z_n^*.*

Proof: O_n is closed under multiplication since for any $x, y \in Z_n^*$, $ord(xy)$ must divide $ord(x) \cdot ord(y)$. Since $ord(x), ord(y)$ are both odd, then so is $ord(x) \cdot ord(y)$ and thus so is $ord(xy)$. ■

[2]A longer proof for the same claim can be found in the preliminary version of [23].

Proposition 4.8 *For any integer $n > 1$, squaring mod n is a permutation over O_n.*

Proof: It is sufficient to show that for every $x \in O_n$ there exists $y \in O_n$ so that $x = y^2 \pmod{N}$. So let $x \in O_n$ and denote the order of x by $2r+1$. Then if we set $y = x^{r+1} \pmod{n}$ we have $y^2 = x^{2r+2} = x \cdot x^{2r+1} = x \pmod{n}$. Finally, by Proposition 4.7, $y = x^{r+1} \in O_n$. ■

Proposition 4.9 *For any integer $n > 1$, any element $x \in Z_n^*$ and any $\ell \geq |n|$, $x^{2^\ell} \in O_n$.*

Proof: Denote $ord(x) = 2^i \cdot r$ where r is an odd integer. Since $ord(x) < \varphi(n) < 2^{|n|}$ then $i < |n| \leq \ell$. Moreover, we have $ord(x^{2^i}) = r$, so $x^{2^i} \in O_n$, and since O_n is closed under squaring (Proposition 4.7), then also $x^{2^\ell} = (x^{2^i})^{2^{\ell-i}} \in O_n$. ■

Proposition 4.10 *Let $n > 1$ be an integer. If we uniformly select an element $x \leftarrow Z_n^*$ and square it $|n|$ times or more mod n, then we get a uniformly distributed element in O_n. Namely, for any $\ell \geq |n|$ and $o \in O_n$*

$$\Pr_{x \leftarrow Z_n^*}\left[x^{2^\ell} = o \pmod{n}\right] = \frac{1}{|O_n|}$$

Proof: Denote the prime factorization of n by $n = q_1 \cdot q_2 \cdots q_m$, where the q_i's are powers of distinct primes ($q_i = p_i^{e_i}$ for some prime p_i and positive integer e_i). Recall that Z_n^* is homomorphic to $Z_{q_1}^* \times Z_{q_2}^* \cdots \times Z_{q_m}^*$, and this homomorphism induces a homomorphism between O_n and $O_{q_1} \times O_{q_2} \cdots \times O_{q_m}$. Thus, it is sufficient to show that for each of the q_i's, uniformly selecting an element $x \leftarrow Z_{q_i}^*$ and squaring it ℓ times yields a uniformly distributed element in O_{q_i}. We distinguish between two "types" of q_i's:

Type 1: q_i is a power of two. In this case the only element of odd order in $Z_{q_i}^*$ is 1, so $|O_{q_i}| = 1$. Indeed, since $\ell \geq |N| \geq |q_i|$, then by Proposition 4.9 we have

$$\Pr_{x \leftarrow Z_{q_i}^*}\left[x^{2^\ell} = 1 \pmod{q_i}\right] = \Pr_{x \leftarrow Z_{q_i}^*}\left[x^{2^\ell} \in O_{q_i} \pmod{q_i}\right] = 1$$

Type 2: q_i is a power of an odd prime. In this case $Z_{q_i}^*$ is a cyclic group. So let g be a generator in this group and we denote $ord(g) = \varphi(q_i) = r \cdot 2^t$ where r is an odd integer and $t \leq |q_i|$. Also, denote $h = g^{2^t} \pmod{q_i}$. Then the odd-order elements in $Z_{q_i}^*$ are $h, h^2, h^3, \cdots, h^r = 1$, so we have $|O_{q_i}| = r$.

Consider now some element $o = h^e \in O_{q_i}$, and we compute the probability that $x^{2^t} = o \pmod{q_i}$ when x is chosen at random in $Z_{q_i}^*$. Picking a random element in $Z_{q_i}^*$ is equivalent to picking an exponent at random $e' \in \{1, 2, \cdots r2^t\}$ and computing $x = g^{e'} \pmod{q_i}$. Moreover, we have

$$x^{2^t} = (g^{e'})^{2^t} = h^{e'} = h^{e' \bmod r} \pmod{q_i}$$

So we have $x^{2^t} = o \pmod{q_i}$ if and only if $e \equiv e' \pmod{r}$. Thus we have

$$\Pr_{x \leftarrow Z_{q_i}^*}\left[x^{2^t} = o \pmod{q_i}\right] = \Pr_{e' \leftarrow \{1,\cdots,r2^t\}}\left[e' \equiv e \pmod{r}\right] = \frac{1}{r} = \frac{1}{|O_{q_i}|}$$

Moreover, since (by Proposition 4.8) squaring is a permutation over O_{q_i} then also for every $\ell \geq |q_i| \geq t$ we have $\Pr_{x \leftarrow Z_{q_i}^*}\left[x^{2^\ell} = o \pmod{q_i}\right] = \frac{1}{|O_{q_i}|}$. ■

Proof: (of Lemma 4.5) Armed with Propositions 4.6-4.10, we can now prove Lemma 4.5. Let s be any string, let N be any integer, and denote $k = |N|$. Consider a random element $x \in Z_N^*$ and denote $z = (f_{s,N}(x^2))^{2^k}$. Then from Proposition 4.6 we have

$$z = \left(f_{s,N}(x^2)\right)^{2^k} = \left(2^{2\hat{s}}(x^2)^{2^{|s|}}\right)^{2^k} = \left(2^{2\hat{s}}\right)^{2^k} \cdot x^{2^{(|s|+k+1)}}$$

Proposition 4.9 implies that $\left(2^{2\hat{s}}\right)^{2^k} \in O_N$, and from Proposition 4.10 we have that $x^{2^{|s|+k+1}}$ is a uniformly distributed element in O_N (which is independent of s). Thus, z is a uniformly distributed element in O_N, regardless of s. This concludes the proof of Lemma 4.5. ■

We therefore conclude that

Theorem 7 *The above commitment scheme enjoys unconditional secrecy, and under the Factorization conjecture is also enjoys computational commitment.*

Chapter 5

A Construction from Message-Digest Functions

In this chapter we present a commitment scheme which uses message-digest functions. This construction achieves unconditional secrecy and computational commitment under the assumption that the Sender can not find collisions in the message-digest function which is used for the scheme. The scheme which we describe in this chapter is essentially the same as scheme due to Damgård, Pedersen and Pfitzmann [12] (which was later re-discovered by us in [24]). We provide a more direct proof of correctness for this scheme than the one in [12]. This proof also let us generalize the construction slightly.

5.1 Background

We start by describing the main tools which are used in this scheme, namely message-digest functions and universal hash functions.

5.1.1 Message Digest Functions

Informally, a message-digest function[1] is a function $f : \{0,1\}^* \rightarrow \{0,1\}^k$ (for some integer k) so that it is infeasible to find two different strings $x \neq y$ so that $f(x) = f(y)$. For practical purposes, the SHA algorithm [14] is often considered to be a message-digest function (for $k = 160$).

Message-digest function were first defined formally by Damgård in [8]. From the formal point of view, for every $k \in \mathcal{Z}^+$ we have a family of functions from $\{0,1\}^*$ to $\{0,1\}^k$, and the infeasibility requirement is formulated with respect to a function which is chosen at random from that family. A construction of message-digest functions can be described by two PPT algorithms: a GENERATE algorithm which given 1^k picks a function $f : \{0,1\}^* \rightarrow \{0,1\}^k$ and outputs its description, and an EVALUATE algorithm which – given a description of f and a string x – outputs $f(x)$. Formally, we have

Definition 8 *A pair of PPT algorithms (G, E) comprise a message-digest scheme if it satisfies the following properties:*

Hashing. *The functions which G outputs on input 1^k map arbitrary long strings into strings of length k. Namely, for every $k \in \mathcal{Z}^+$ and every string $x \in \{0,1\}^*$*

$$\Pr\left[\begin{array}{l} f \leftarrow G(1^k) \\ y \leftarrow E(f,x) \\ |y| = k \end{array}\right] = 1$$

For convenience of notations, we often use the shorthand $f(x)$ instead of $E(f,x)$.

Collision-Intractability. *For any PPT algorithm A and any polynomial $Q(\cdot)$, there exists a negligible function* neg$(\cdot)$ *so that for all $k \in \mathcal{Z}^+$ and all auxiliary inputs*

[1]These function are also called collision-intractable hash functions. However, to avoid confusion with other types of hash functions which we use in this construction, we call them message-digest functions throughout the chapter.

$z \in \{0,1\}^{\leq Q(k)}$,

$$\Pr\left[\begin{array}{l} f \leftarrow G(1^k) \\ (x,y) \leftarrow A(z,f) \\ x \neq y \textit{ and } f(x) = f(y) \end{array}\right] \leq \text{neg}(k)$$

5.1.2 Universal Hashing

Universal hashing was introduced by Carter and Wegman [6] and it plays a very important role in many areas of computer-science.

Definition 9 *Let S and T be two sets, and let H be a family of functions from S to T. We say that H is a* universal family of hash functions *if for every two different elements $s_1 \neq s_2$ in S, and for every two elements t_1, t_2 in T*

$$\Pr_{h \leftarrow H}[h(s_1) = t_1 \textit{ and } h(s_2) = t_2] = \frac{1}{|T|^2}$$

An easy example of such family is the family of all affine transformations between two linear spaces. In particular, let $S = \{0,1\}^l$ and $T = \{0,1\}^n$, then we define

$$H = \{h_{A,b} : \ A \in \{0,1\}^{n \times l}, b \in \{0,1\}^n\}$$

where we define $h_{A,b}(x) \overset{\text{def}}{=} Ax + b$ (all the operations take place in a linear space over $GF(2)$). It is easy to see that H above is a universal family of hash functions from $\{0,1\}^l$ to $\{0,1\}^n$. In this construction, it takes $n(l+1)$ bits to specify a function from H. A more efficient construction is obtained by restricting the matrix A to be a Toeplitz matrix. Namely, we require that A would be fixed on the diagonals, $A_{i,j} = A_{i+1,j+1}$. Again, it is easy to see that this still constitutes a universal family of hash functions, and it takes $2n + l - 1$ bits to describe any function in this family.

The Smoothing Property

The property of Universal hash functions which we need in our construction is called the *Smoothing* property.[2] Informally, for a family F of functions from a set S to a set T, the smoothing property asserts that every subset of S which is much larger than T is mapped almost uniformly to T by all but a small fraction of the functions in F.

To make this formal, let F be a family of functions from S to T, and denote by U_F the distribution over pairs $\langle f, t\rangle$ which is induced by picking $f \leftarrow F$ and $t \leftarrow T$ uniformly at random and independently. Also, for any subset $S' \subseteq S$, denote by $D_{S',F}$ the distribution over pairs $\langle f, f(s)\rangle$ which is induced by picking $f \leftarrow F$ and $s \leftarrow S'$. uniformly at random and independently. That is,

$$U_F \stackrel{\text{def}}{=} \{\langle f,t\rangle \;:\; f \leftarrow F,\ t \leftarrow T\}\ , \qquad D_{S',F} \stackrel{\text{def}}{=} \{\langle f, f(s)\rangle \;:\; f \leftarrow F,\ s \leftarrow S'\}$$

The celebrated Leftover-lemma [25] asserts that if F is a universal family of hash functions, and if $|S'| \gg |T|$ then $D_{S',F}$ is very close to U_F, and more generally, the statistical difference $\|D_{S',F} - U_F\|$ deceases with the ratio $|S'|/|T|$. More precisely

Theorem 10 (The Leftover-lemma [25]) *If H is a universal family of hash functions from S to T, then for every $S' \subseteq S$, the statistical difference between U_F and $D_{S',F}$ is bounded by*

$$\|U_F - D_{S',F}\| \le \sqrt{\frac{|T|}{|S'|}}$$

(Note that when $|S'| \le |T|$ the theorem only implies the trivial bound $\|U_F - D_{S',F}\| \le 1$ which is true for any two distributions.) A self-contained proof of this theorem can be found in [27, Ch. 8].

5.2 A Simplified Construction

In this section we describe a simplified version of the construction. Although this simplified version in fact violates secrecy, in that the Receiver learns the length of the

[2]In some places in the literature it is also called the Extraction property.

secret, it demonstrates all the ideas which we later need for the scheme in Section 5.3.

For the rest of this section, let (G, E) be a message-digest scheme. To commit to a secret of length n with security parameter k, the parties use a universal family of hash functions (e.g., the family of Toeplitz matrices which was discussed above) from $\{0,1\}^L$ to $\{0,1\}^n$, where $L \stackrel{\text{def}}{=} n+2k$. We denote this family by $H_{L,n}$. The commitment scheme is defined as follows (see Fig. 5-1 for an illustration of this scheme).

Initialization. On security parameter 1^k, the Receiver runs $G(1^k)$ (where G is the generation algorithm for the message-digest scheme), and sends the resulting function f to the Sender.

Commit Phase. To commit to a secret $s \in \{0,1\}^n$, the Sender uniformly picks a string $x \leftarrow \{0,1\}^L$, computes $y = f(x)$ and verifies[3] that $|y| = k$. It then uniformly picks a function $h \leftarrow H_{L,n}$ and computes $t = h(x) \oplus s$, where $\oplus$ denotes a bitwise exclusive-or. The Sender sends to the Receiver the commit string $\langle y, h, t \rangle$.

De-commit Phase. The de-commit string which the Sender sends to the Receiver, is x. The Receiver verifies that $y = f(x)$ and computes $s = h(x) \oplus t$.

This scheme clearly satisfies the Validity condition in Definition 1.

Remark. In the scheme above, the Sender can pick $x \leftarrow \{0,1\}^L, h \leftarrow H_{L,n}$ and compute $y = f(x)$ in the Initialization phase, since x, h are independent of the secret. However, moving the choice of x to the Initialization phase means that this scheme can not be used to commit to many secrets (since x is revealed in the De-commit phase, and knowledge of x compromise the secrecy of the scheme). Somewhat surprisingly, it is possible to pick h in the Initialization phase and use it for many commitments without compromising the secrecy of the scheme.

[3]We note that verifying that $|y| = k$ is important in the case that the Receiver tries to "cheat" in the Initialization phase and send a function f which is not generated by $G(1^k)$.

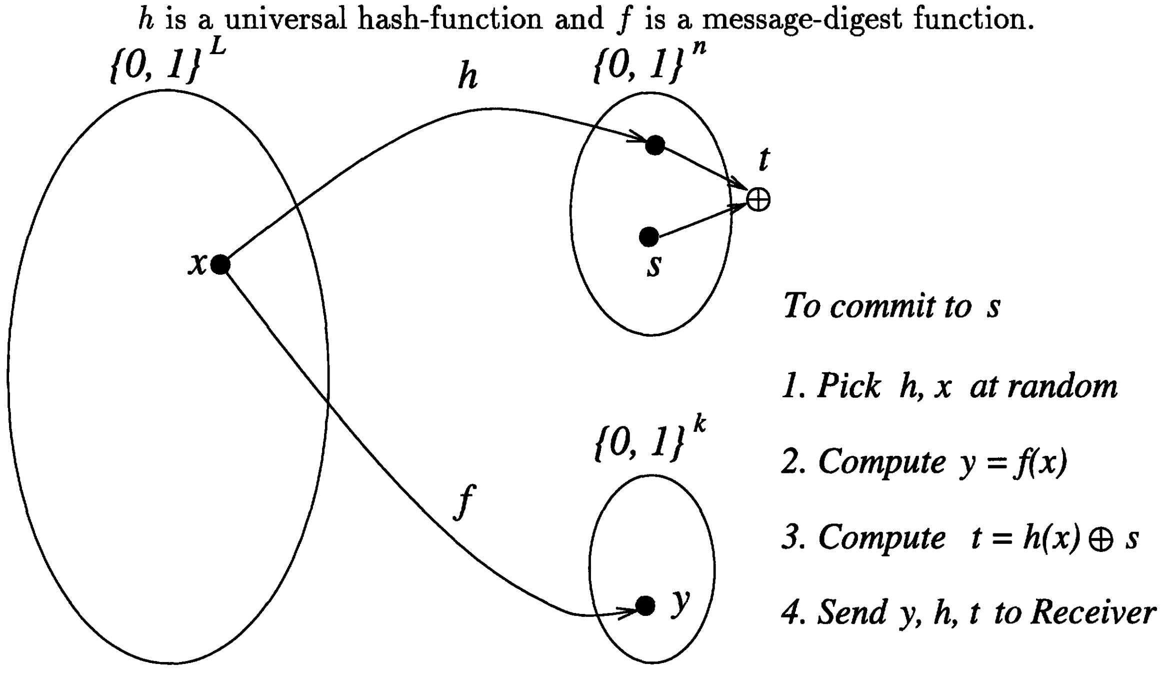

Figure 5-1: A simplified MD-based commitment scheme.

5.2.1 Security of the Scheme

It is obvious that the Commitment condition in Definition 2 is an immediate consequence of the Collision-intractability of the message-digest scheme. The less obvious part is to prove that the Receiver gets almost no statistical advantage in guessing the secret s from the commit string. Technically speaking, this is not true since the commit string reveals the length of the secret. However, we can show that it does not reveal anything else. Specifically, in the following lemma we show that for any two strings s_0, s_1, of the same length, the distributions which are induced on the commit strings for s_0, s_1 are statistically close. (This lemma is later used in Section 5.3 to prove the security of a modification of this simplified scheme).

Notations. In the lemma below we use the following notations. We fix two integers $k, n \in \mathcal{Z}^+$ and set $L = n + 2k$. Then, for any function $f : \{0,1\}^L \rightarrow \{0,1\}^k$ and any string $s \in \{0,1\}^n$ we use the notation $C_f(s)$ to denote the distribution over the commit strings which is induced by the Commit phase above when committing to the secret s using the function f.

Lemma 5.1 *Fix any two integers $k, n \in \mathcal{Z}^+$ and set $L = n + 2k$. Then, for any function $f : \{0,1\}^L \to \{0,1\}^k$ and any two strings $s_0, s_1 \in \{0,1\}^n$*

$$\|C_f(s_1) - C_f(s_0)\| \leq 2^{-\frac{k}{2}+1}$$

Proof idea. The proof of Lemma 5.1 follows from the Leftover-lemma via a sequence of claims which are rather straightforward (if somewhat technical). Very roughly, the idea behind the proof is as follows. Since $L > k$ and f goes from $\{0,1\}^L$ to $\{0,1\}^k$, then "most" strings $y \in \{0,1\}^k$ have "very large" pre-images under f. Therefore, by the Leftover-lemma, if we pick a random element x from any of these pre-images and a random function $h \leftarrow H_{L,n}$, then the distribution over the pairs $\langle h, h(x)\rangle$ is "almost the same" as the uniform distribution over pairs $\langle h, t\rangle$ (where $t \leftarrow \{0,1\}^n$). Since the former distribution is essentially the same[4] as the distribution over commit-strings *conditioned on* y, we conclude that for most y's, we get almost the same distribution, regardless of the secret.

Proof: We start by setting a few notations. Let n, k, L and f be as in the statement of Lemma 5.1, and consider an experiment in which we pick a string $\chi \leftarrow \{0,1\}^L$, a function $\eta \leftarrow H_{L,n}$ and a string $\tau \leftarrow \{0,1\}^n$, compute $\varphi = f(\chi), \sigma = \tau \oplus \eta(\chi)$ and output $\langle \varphi, (\eta, \tau), \sigma\rangle$.

For given strings $y \in \{0,1\}^k$, $s, t \in \{0,1\}^n$ and a function $h \in H_{L,n}$, denote the probability that the above experiment results in outputting $\langle y, (h,t), s\rangle$ by $p[y, (h,t), s]$. Below we also consider some of the marginal probabilities and conditional probabilities in this experiment. Using our notations for probabilistic experi-

[4]The actual distribution, conditioned on y, when committing to a secret s, is on pairs $\langle h, h(x) \oplus s\rangle$, but this is just as close to the uniform distribution.

ments, these are defined as

$$p[y,(h,t),s] \overset{\text{def}}{=} \Pr\left[\begin{array}{l} \chi \leftarrow \{0,1\}^L \\ \eta \leftarrow H_{L,n} \\ \tau \leftarrow \{0,1\}^n \\ \eta = h \text{ and } \tau = t \text{ and } f(\chi) = y \text{ and } \tau \oplus \eta(\chi) = s \end{array}\right]$$

$$p[y] \overset{\text{def}}{=} \Pr\left[\begin{array}{l} \chi \leftarrow \{0,1\}^L \\ f(\chi) = y \end{array}\right] \qquad \left(= \frac{|f^{-1}(y)|}{2^L}\right)$$

$$p[y,(h,t)] \overset{\text{def}}{=} \Pr\left[\begin{array}{l} \chi \leftarrow \{0,1\}^L \\ \eta \leftarrow H_{L,n} \\ \tau \leftarrow \{0,1\}^n \\ \eta = h \text{ and } \tau = t \text{ and } f(\chi) = y \end{array}\right] \qquad \left(= \frac{p[y]}{2^n|H_{L,n}|}\right)$$

$$p[s] \overset{\text{def}}{=} \Pr\left[\begin{array}{l} \chi \leftarrow \{0,1\}^L \\ \eta \leftarrow H_{L,n} \\ \tau \leftarrow \{0,1\}^n \\ \tau \oplus \eta(\chi) = s \end{array}\right] \qquad \left(= 2^{-n}\right)$$

$$p[(h,t),s \mid y] \overset{\text{def}}{=} \Pr\left[\begin{array}{l} \chi \leftarrow f^{-1}(y) \\ \eta \leftarrow H_{L,n} \\ \tau \leftarrow \{0,1\}^n \\ \eta = h \text{ and } \tau = t \text{ and } \tau \oplus \eta(\chi) = s \end{array}\right]$$

$$p[y,(h,t) \mid s] \overset{\text{def}}{=} \Pr\left[\begin{array}{l} \chi \leftarrow \{0,1\}^L \\ \eta \leftarrow H_{L,n} \\ \eta = h \text{ and } f(\chi) = y \\ \quad \text{and } \eta(\chi) = t \oplus s \end{array}\right]$$

(We bind (h,t) together in these notations since we view them as a description of a function $\tilde{h}(x) \overset{\text{def}}{=} h(x) \oplus t$, and also to stress the difference in the roles of t and s.)

It is easy to verify that the following equalities hold for any $y \in \{0,1\}^k$, $s,t \in$

$\{0,1\}^n$ and $h \in H_{L,n}$

$$\begin{aligned}
&\text{(by definition)} & \Pr[C_f(s) = \langle y,h,t\rangle] &= p[y,(h,t)|s] & (5.1)\\
&\text{(by definition)} & \forall \Delta \in \{0,1\}^n,\ p[(h,t),s|y] &= p[(h,t\oplus\Delta),s\oplus\Delta|y] & (5.2)\\
&\text{(Bayes'es theorem)} & p[y,(h,t)|s] = \frac{p[y]\cdot p[(h,t),s|y]}{p[s]} &= 2^n p[y] p[(h,t),s|y] & (5.3)
\end{aligned}$$

Let us denote by P the probability distribution over the triples $\langle y,h,t\rangle$ which is induced by the above experiment. Namely, for any $\langle y,h,t\rangle$ we have $P(\langle y,h,t\rangle) \overset{\text{def}}{=} p[y,(h,t)]$. For any fixed string $s_0 \in \{0,1\}^n$, we bound the statistical difference between the distributions $\|C_f(s_0) - P\|$. We do this in three steps: In the first step we express this statistical difference as a sum of terms, in the second step we show how to bound each of the terms in this sum, and in the third step we combine these bounds to obtain the final result.

Step 1. For any string $s_0 \in \{0,1\}^n$,

$$\begin{aligned}
\|C_f(s_0) - P\| &= \sum_{y,h,t} |\Pr[C_f(s_0) = \langle y,h,t\rangle] - P(\langle y,h,t\rangle)| & (5.4)\\
&\overset{(5.1)}{=} \sum_y \sum_{h,t} |\ p[y,(h,t)|s_0] - p[y,(h,t)]\ |\\
&\overset{(5.3)}{=} \sum_y \sum_{h,t} \left| 2^n \cdot p[y] \cdot p[(h,t),s_0|y] - \frac{p[y]}{2^n\,|H_{K,n}|} \right|\\
&= \sum_y p[y] \underbrace{\left(2^n \sum_{h,t} \left| p[(h,t),s_0|y] - \frac{1}{2^{2n}|H_{K,n}|} \right| \right)}_{\text{denote this by } q[y,s_0]}
\end{aligned}$$

(Above, the numbers over the equality-signs represent the equations which were used to derive these equalities.)

Step 2. For every two strings $y \in \{0,1\}^k, s_0 \in \{0,1\}^n$, we denote the parenthesized part in the above expression by $q[y, s_0]$. Namely,

$$q[y,s_0] \stackrel{\text{def}}{=} 2^n \sum_{h,t} \left| p[(h,t),s_0|y] - \frac{1}{2^{2n}|H_{K,n}|} \right|$$

Next we show how to bound $q[y, s_0]$ using the Leftover-lemma. We start by introducing some more notations. With each $h \in H_{n,L}$ and $t \in \{0,1\}^n$ we identify the function $\tilde{h}_t : \{0,1\}^L \to \{0,1\}^n$ which is defined as $\tilde{h}_t(x) \stackrel{\text{def}}{=} h(x) \oplus t$, and we consider the family $\tilde{H}_{L,n} \stackrel{\text{def}}{=} \{ \tilde{h}_t \ : \ h \in H_{L,n}, \ t \in \{0,1\}^n \}$. Since $H_{L,n}$ is a universal family of hash functions, then so is $\tilde{H}_{L,n}$.

Using the notations of the Leftover-lemma, we denote the uniform distribution over the pairs $\langle \tilde{h}_t, s \rangle$ (where $\tilde{h}_t \leftarrow \tilde{H}_{L,n}, \ s \leftarrow \{0,1\}^n$) by $U_{\tilde{H}_{L,n}}$. Also, for every subset $X \subseteq \{0,1\}^L$ we denote by $D_{X,\tilde{H}_{L,n}}$ the distribution over $\langle \tilde{h}_t, s \rangle$ which is induced by picking at random $\tilde{h}_t \leftarrow \tilde{H}_{L,n}, \ x \leftarrow X$ and setting $s = \tilde{h}_t(x)$. In particular, here we are interested in subsets of the form $X = f^{-1}(y)$ for strings $y \in \{0,1\}^k$. Notice that these are perfectly good subsets of $\{0,1\}^L$, so by the Leftover-lemma, for any $y \in \{0,1\}^k$

$$\|D_{f^{-1}(y),\tilde{H}_{L,n}} - U_{\tilde{H}_{L,n}}\| \leq \sqrt{\frac{2^n}{|f^{-1}(y)|}} \tag{5.5}$$

We note that by definition, the uniform distribution $U_{\tilde{H}_{L,n}}$ assigns a probability of $\frac{1}{2^{2n}|H_{K,n}|}$ to every pair $\langle \tilde{h}_t, s \rangle$, and the distribution $D_{f^{-1}(y),\tilde{H}_{L,n}}$ assigns to $\langle \tilde{h}_t, s \rangle$ a probability of $p[(h,t),s|y]$. Hence, $\|D_{f^{-1}(y),\tilde{H}_{L,n}} - U_{\tilde{H}_{L,n}}\| = \sum_{h,t,s} \left|p[(h,t),s|y] - \frac{1}{2^{2n}|H_{K,n}|}\right|$.

The right hand side of the last expression is "almost what we need", except that it includes a summation over s, whereas we need just the partial sum for a single string s_0. This is where we use Eq. (5.2). For every string $s_0 \in \{0,1\}^n$, we can change the order of summation to get

$$\begin{aligned} \|D_{f^{-1}(y)} - U_{\tilde{H}_{L,n}}\| &= \sum_{h,t,s} \left| p[(h,t),s|y] - \frac{1}{2^{2n}|H_{K,n}|} \right| \\ &= \sum_{\Delta \in \{0,1\}^n} \sum_{h,t} \left| p[(h,t \oplus \Delta), s_0 \oplus \Delta|y] - \frac{1}{2^{2n}|H_{K,n}|} \right| \end{aligned} \tag{5.6}$$

$$\overset{(5.2)}{=} \sum_{\Delta\in\{0,1\}^n} \sum_{h,t} \left| p[(h,t),s_0|y] - \frac{1}{2^{2n}|H_{K,n}|} \right|$$
$$= 2^n \sum_{h,t} \left| p[(h,t),s_0|y] - \frac{1}{2^{2n}|H_{K,n}|} \right| = q[y,s_0]$$

From Eq. (5.5) and (5.6) we conclude that for any $y \in \{0,1\}^k$ and any $s_0 \in \{0,1\}^n$

$$q[y,s_0] \leq \sqrt{\frac{2^n}{|f^{-1}(y)|}} \tag{5.7}$$

Step 3. We now have all the ingredients needed to prove Lemma 5.1. Combining Eq. (5.4) and Eq. (5.7) we have for every $s_0 \in \{0,1\}^n$,

$$\begin{aligned} \|C_f(s_0) - P\| &\overset{(5.4)}{=} \sum_y p[y]q[y,s_0] \overset{(5.7)}{\leq} \sum_y p[y]\sqrt{\frac{2^n}{|f^{-1}(y)|}} \\ &\overset{(*)}{\leq} \sqrt{\sum_y p[y]\frac{2^n}{|f^{-1}(y)|}} = \sqrt{\sum_y \frac{|f^{-1}(y)|}{2^L}\cdot\frac{2^n}{|f^{-1}(y)|}} \\ &= \sqrt{\sum_{y\in\{0,1\}^k} 2^{n-L}} = 2^{(n+k-L)/2} = 2^{-k/2} \end{aligned} \tag{5.8}$$

where the inequality which is marked by $(*)$ holds since the square-root function is concave. Finally, Eq. (5.8) implies that for any two strings $s_0, s_1 \in \{0,1\}^n$,

$$\|C_f(s_0) - C_f(s_1)\| \leq \|C_f(s_0) - P\| + \|C_f(s_1) - P\| \leq 2^{-\frac{k}{2}+1}$$

which concludes the proof of Lemma 5.1. ■

5.2.2 Efficiency of the Scheme

The Commit and De-commit phases of this scheme are non-interactive and the Initialization can be done in the public-directory model. The local computation in the scheme amounts to one application of the message-digest function to a string of length $n + 2k$ and one application of a universal hash function to the same string. If we use the Toeplitz-matrix construction for universal hashing, then the size of the commit string is $|y| + |h| + |t| = 2n + 3k$. (In fact, in this case we can reduce the size of the commit string to $n + 3k$ by combining the descriptions of $h = (A, b)$ and t, so that instead of sending $\langle y, (A, b), t\rangle$ we send $\langle y, A, b \oplus t\rangle$.) The size of the de-commit string is $|x| = n + 2k$.

5.2.3 A Slight Generalization

We note that the only property of the family $H_{L,n}$ which we used above is the smoothness property. It follows that any family of functions with this smoothness property can be used. In fact, for the above scheme we can do with a much weaker smoothness property than what is guaranteed by universal hashing family. Using the notations of the Leftover-lemma, any family which guarantees that for every subset S' we have $||U_F - D_{S',F}|| = O\left(\left(\frac{|T|}{|S'|}\right)\right)^{\epsilon}\right)$ for any $\epsilon > 0$ can be used in our scheme.

5.3 A Secure MD-based Construction

The above construction does not satisfy the Secrecy condition of Definition 2, since the Receiver still learns the size of the secret. Of course, this can trivially be solved by padding each secret to the maximum length (which is some polynomial $Q(k)$ for security parameter 1^k), but this would have a severe effect on the efficiency of the scheme. Also, the length of the commit-string in the above scheme is linear in the length of the secret. (Recall that this is worse than in the scheme commitment schemes in Chapter 4, where the length of the commit-string is only $O(k)$.) In this section we describe how to modify this scheme so that it also hides the length of the secret, and at the same time get an $O(k)$-bit commit string.

The idea is that to commit to the secret s, the Sender first computes the k-bit message-digest of s, $m = f(s)$, and then apply the above scheme to the string m. To de-commit s, the Sender sends both the string s and the de-commit string of the previous scheme. The Receiver computes $m = f(s)$ and checks that m is the "secret" in the first scheme. Since we always run the previous scheme on a string of length k, then the Receiver does not learn anything about the secret s. Also the commit string

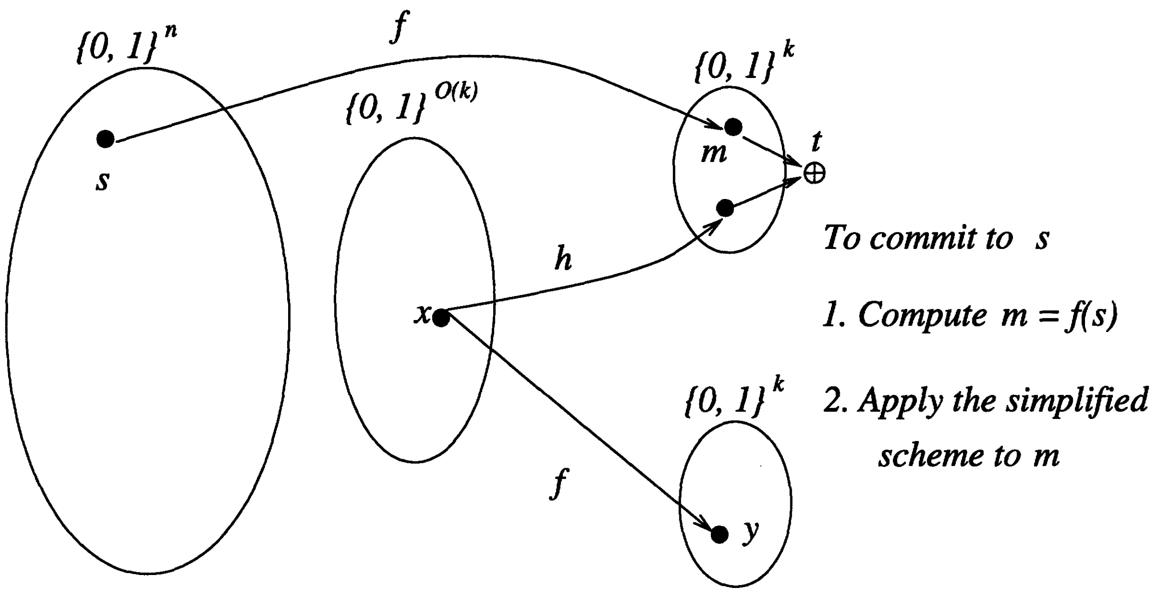

Figure 5-2: A secure MD-based commitment scheme.

is of length $k + 3k = O(k)$, regardless of the length of the secret s. The full scheme, therefore, is as follows (see Fig. 5-2 for an illustration).

Initialization. On security parameter 1^k, the Receiver runs $G(1^k)$ (where G is the generation algorithm for the message-digest scheme), and sends the resulting function f to the Sender.

Commit Phase. To commit to a secret $s \in \{0,1\}^*$, the Sender first computes $m = f(s)$ and checks that m is a k-bit string. Then it uniformly picks a string $x \leftarrow \{0,1\}^L$ (where $L = 3k$), computes $y = f(x)$ and verifies that $|y| = k$, uniformly picks a function $h \leftarrow H_{L,k}$ and computes $t = h(x) \oplus m$. The Sender sends to the Receiver the commit string $\langle y, h, t \rangle$.

De-commit Phase. The Sender sends to the Receiver the de-commit string $\langle s, x \rangle$. The Receiver verifies that $y = f(x)$ and that $f(s) = h(x) \oplus t$.

We note that we can save a little in the length of the commit-string (at the price of adding some local computation), by applying f to the commit string before sending it to the Receiver. Namely, instead of sending $\langle y, h, t \rangle$ in the Commit phase, the Sender can send $f(y, h, t)$ (which is a k-bit string) in the Commit phase and send $\langle y, h, t \rangle$

themselves in the De-commit phase. This way, the length of the commit string is only k (rather than $4k$).

Theorem 11 *The modified MD-based commitment scheme enjoys unconditional secrecy and computational commitment, provided that the algorithms (G, E) which are used in it constitute a message-digest scheme.*

Proof: The Validity condition is obvious. For the Secrecy condition, recall that for any string s, the induced distribution over the commit strings (using message-digest function f) is $C_f(f(s))$ (where $C_f(\cdot)$ is the distribution induced by the construction in Section 5.2.) By Lemma 5.1 we have for any two strings $s_0, s_1 \in \{0,1\}^{\leq Q(k)}$, $\|C_f(f(s_0)) - C_f(f(s_1))\| \leq 2^{-\frac{k}{2}+1}$

The Commitment condition follows since for every two different de-commit strings $\langle s, x\rangle, \langle s', x'\rangle$ which correspond to the same commit string $\langle y, h, t\rangle$ either $x \neq x'$ and $f(x) = f(x') = y$, or $x = x'$ in which case $s \neq s'$ and $f(s) = f(s') = h(x)$. In either case, breaking the Commitment condition requires breaking the message-digest scheme. ■

Bibliography

[1] G. Brassard and C. Crèpeau. Nontransitive Transfer of Confidence: A Perfect Zero-Knowledge Interactive Protocol for SAT and Beyond. In *Proc. 27th IEEE Symp. on Foundations of Comp. Science*, IEEE, 1986. pages 188–195.

[2] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 37(2):156–189, 1988.

[3] G. Bleumer, B. Pfitzmann and M. Waidner. A Remark on a Signature Scheme where Forgery can be Proved. In I.B. Damgård, editor, *Proc. of Eurocrypt'90*, Lecture Notes in Computer Science, volume 473, Springer-Verlag, 1990. Pages 441–445.

[4] M. Blum. Coin flipping by telephone. In *Proc. IEEE Spring COMPCOM*, IEEE, 1982. Pages 133–137.

[5] M. Blum and S. Micali. Coin flipping into a well. Unpublished, 1981.

[6] L. Carter and M. Wegman. Universal Hash Functions. J. of Computer and System Science 18, 1979, pp. 143-154.

[7] D. Chaum, E. van Heijst and B. Pfitzmann. Cryptographically Strong Undeniable Signatures, Unconditionally Secure for the Signer. In J. Feigenbaum, editor, *Proc. Crypto '91*, Lecture Notes in Computer Science, volume 576, Springer-Verlag, 1992. Pages 470–484.

[8] I.B. Damgård. Collision free hash functions and public key signature schemes. In David Chaum and Wyn L. Price, editors, *Proceedings of EUROCRYPT 87.* Lecture Notes in Computer Science No. 304. Springer-Verlag, 1988. Pages 203–216.

[9] I.B. Damgård. On the existence of a bit commitment schemes and zero-knowledge proofs. In G. Brassard, editor, *Proceedings CRYPTO 89.* Lecture Notes in Computer Science No. 435. Springer-Verlag, 1990. Pages 17–29.

[10] I.B. Damgård, Practical and Provably Secure Release of a Secret and Exchange of Signatures. T. Helleseth, editor, *Proc. EuroCrypt '93*, Lecture Notes in Computer Science, volume 765, Springer-Verlag, 1994. pages 200–217.

[11] W. Diffie and M. E. Hellman, New Directions in Cryptography. *IEEE Trans. Inform. Theory*, IT-22, 1976, pages 644-654.

[12] I.B. Damgård, T.P. Pedersen, and B. Pfitzmann. On the existence of statistically hiding bit commitment schemes and fail-stop signatures. In Douglas R. Stinson, editor, *Proceedings CRYPTO 93.* Lecture Notes in Computer Science No. 773. Springer, 1994. Pages 250–265.

[13] H. Dobbertin, A. Bosselaers, and B. Preneel. RIPEMD-160: A strengthened version of RIPEMD. Third Workshop on Fast Software Encryption, 1996.

[14] Federal Information Processing Standards, Publication 180. Specifications for a Secure Hash Standard (SHS).

[15] O. Goldreich. Two Remarks Concerning the Goldwasser-Micali-Rivest Signature Scheme. In A.M.Odlyzko, editor, *Proceedings CRYPTO 86*, Lecture Notes in Computer Science No. 263. Springer-Verlag, 1987. Pages 104–110.

[16] O. Goldreich. A Uniform Complexity Treatment of Encryption and Zero-Knowledge. *Journal of Cryptology*, Vol. 6, No. 1 (1993), Pages 21–53.

[17] O. Goldreich. Foundations of Cryptography – Fragments of a Book (1995)

[18] O. Goldreich and A. Kahan. How to Construct Constant-Round Zero-Knowledge Proofs Systems for NP. *Journal of Cryptology*, Vol. 9, No. 2, 1996.

[19] 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*, vol. 38, no. 1, 1991, pp. 691-729.

[20] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. In *SIAM Journal on Computing*, 18(1) : 186–208, 1989.

[22] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2) : 281–308, 1988.

[23] S. Halevi, Efficient commitment with bounded sender and unbounded receiver. To appaer in *Journal of Cryptology.* Preliminiary version appeared in *Proc. Crypto '95.* Lecture Notes in Computer Science, volume 963, Springer-Verlag, 1995. Pages 84–96.

[24] S. Halevi and S. Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In N. Koblitz, editor, *Proc. Crypto '96.* Lecture Notes in Computer Science, volume 1109, Springer-Verlag, 1996. pages 201–215.

[25] J. Håstad, R. Impagliazzo, L.A. Levin and M. Luby. Construction of Pseudo-random Generator from any One-Way Function. Manuscript, 1993. (To appear in *SICOMP.*) See preliminary versions by Impagliazzo et. al. in *21st STOC* and J. Håstad in *22nd STOC.*

[26] R. Impagliazzo and M. Yung. Direct minimum-knowledge computations. In C. Pomerance, editor, *Proc. Crypto '87*, Lecture Notes in Computer Science, volume 293, Springer-Verlag, 1988. Pages 40–51.

[27] M. Luby. Pseudorandomness and Cryptographic Applications Princeton University Press, 1996.

[28] N. Koblitz. A Course in Number Theory and Cryptography. *Graduate Texts in Mathematics* 114, Springer-Verlag, 1987.

[29] M. Naor. Bit commitment using pseudo-randomness. In G. Brassard, editor, *Proceedings CRYPTO 89*, Lecture Notes in Computer Science No. 435. Springer-Verlag, 1990. Pages 128–137.

[30] M. Naor and M. Yung. Universal one-way hash functions and their cryptographic applications. In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989. Pages 33-43.

[31] M. Naor, R. Ostrovsky, R. Venkatesan, and M. Yung. Perfect zero-knowledge arguments for *NP* can be based on general complexity assumptions. In Ernest F. Brickell, editor, *Proceedings CRYPTO 92*, Lecture Notes in Computer Science No. 740. Springer-Verlag, 1992. Pages 196–214.

[32] C.H. Papadimitriou. Computational Complexity Addison-Wesley Publishing Company, 1994.

[33] T.P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In J. Feigenbaum, editor, *Proc. Crypto '91*, Lecture Notes in Computer Science, volume 576, Springer-Verlag, 1992. pages 129–140.

[34] B. Pfitzmann. Digital Signature Schemes: General Framework and Fail-Stop Signatures. Lecture Notes in Computer Science, vol. 1100, Springer-Verlag, Heidelberg, August 1996.

[35] B. Pfitzmann and M. Waidner. Fail-Stop Signatures and their Applications. Securicom 91, Paris, 1991.

[36] J. Pieprzyk and B. Sadeghiyan. Design of Hashing Algorithms. Lecture Notes in Computer Science, vol. 756, Springer-Verlag, Berlin-Heidelberg, 1993.

[37] R.L. Rivest, A. Shamir and L. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, Vol. 21, 1978, pages 120-126.

[38] J. Rompel. One Way Functions are Necessary and Sufficient for Secure Signatures. In *Proceedings of the 22st Annual ACM Symposium on Theory of Computing*, 1990, Pages 387-394.

[39] P.W. Shor. Algorithms for Quantum Computation: Discrete Logarithms and Factoring. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, IEEE, 1994. pages 124-134.

[40] H.C. Williams, A Modification of the RSA Public-Key Encryption Procedure. In *IEEE Trans. on Info. Theory*, Volume IT-26, No. 6, 1980. Pages 726-729.

32 33-37