Adaptive Security in the Threshold Setting: From Cryptosystems to Signatures

by

Christopher Jason Peikert

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering

and

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

Author . . . .

Department of Electrical Engineering and Computer Science

May 23, 2001

Certified by . . . .

Ronald L. Rivest

Viterbi Professor of Computer Science and Engineering

Thesis Supervisor

Certified by . . . .

Anna Lysyanskaya

Co-supervisor

Thesis Supervisor

Accepted by . . . .

Arthur C. Smith

Chairman, Department Committee on Graduate Theses

Adaptive Security in the Threshold Setting: From Cryptosystems to Signatures

by

Christopher Jason Peikert

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2001, in partial fulfillment of the
requirements for the degrees of
Bachelor of Science in Computer Science and Engineering
and
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Threshold cryptosystems and signatures schemes provide ways to distribute trust throughout a group and increase the availability of cryptographic systems. A standard approach in designing these protocols is to base them upon existing single-party systems having the desired properties.

Two recent signature schemes [13, 18] have been developed which are provably secure using only some standard number-theoretic hardness assumptions. Both schemes rely upon inversion of a prime number modulo a secret value. We provide a multiparty modular inversion protocol that is secure against an adaptive adversary, thereby enabling threshold signature schemes with stronger security properties than any previous result.

As a tool, we also develop an adaptively-secure, erasure-free threshold version of the Paillier cryptosystem. Because of its homomorphic properties, this cryptosystem is of independent interest and is useful in the context of secure elections, lotteries, and general multiparty computation.

Thesis Supervisor: Ronald L. Rivest
Title: Viterbi Professor of Computer Science and Engineering

Thesis Supervisor: Anna Lysyanskaya
Title: Co-supervisor

Acknowledgments

I would first like to give my endless thanks to Anna Lysyanskaya, whose expert guidance and advice made this thesis possible. From the valuable critiques of my initial drafts, to the countless hours of brainstorming, her contributions to this work cannot be understated. The shape and form of this thesis are in large part due to her; however, any errors which remain are solely my responsibility.

I would also like to recognize Professor Ronald Rivest for generously offering his valuable time to serve in an advisory role. My interest in cryptography is in large part due to my many positive experiences with him as an instructor and leader in the field.

Finally I would like to thank my parents, Frances and Michael, for their unconditional love, support, and guidance – and my sister Stephanie, whom I treasure far more than she knows.

Contents

Chapter 1

Introduction

In any real-world scenario requiring security, an essential question is whom to trust. If millions of dollars or human lives are at risk, it is unacceptable to place all of your trust in just one person: what if he becomes corrupt, or is blackmailed, or loses the ability to perform his job? Likewise, in an environment involving electronic transactions, it is unwise to place critical information on just one server: what if it is broken into, or crashes? We would like to be able to distribute trust among a group of entities, so that any majority of them can perform an operation, but any malicious minority of them cannot do harm. How can this requirement be stated formally, and are there technical ways to distribute trust in a secure way? These questions are addressed by *secret-sharing* schemes, *threshold cryptosystems*, and *threshold signature protocols*.

Secret-sharing protocols were discovered independently in 1979 by Shamir [28] and Blakley [2]. They enable a group of n parties to collectively share a secret value so that any group of more than t parties can combine their individual *shares* to discover the secret, but any group of t or fewer conspiring parties cannot obtain any information about the secret. A natural use of secret-sharing protocols is to protect cryptographic keys: if a secret key is distributed across several systems, then an adversary must compromise more than a certain *threshold* t of them before learning anything about the key.

When decrypting or signing a value, the parties must perform some computation using the shared secret. Instead of explicitly re-assembling the private key (which would create a single point of failure), the parties communicate according to a protocol which lets each party compute the result, without letting the participants learn anything about each others' shares. Threshold protocols must also handle the real-world possibility that some of the parties may have been compromised, and are controlled by an adversary. This adversary may attempt to learn something about the secret key, or try to cause the protocol to fail or produce incorrect results. The goal, then, is to design *robust* protocols, which succeed in the presence of malicious participants, while retaining the secrecy of the keys. Desmedt and Frankel [14] were the first to demonstrate a threshold decryption protocol having these properties, based on the ElGamal cryptosystem.

This idea is also applied in several *threshold signatures schemes* for classical public-key signature systems such as RSA [27] and DSS [24]. Unfortunately, the formal security of these systems relies upon a heuristic tool known as the *random oracle model*, which is not equivalent to real security in certain schemes [8]. Newer signature systems [13, 18] do away with the random oracle model, and rely only upon some reasonable number-theoretic assumptions. Designing efficient threshold versions of these systems is therefore important for very high-security applications and from a theoretical point of view.

A recent protocol due to Catalano *et al.* [9] describes how to efficiently invert a given prime, modulo a shared secret. The most common use for this protocol is when the shared secret is the Euler totient function ϕ of a public RSA modulus N. Computing inverses modulo $\phi(N)$ is a crucial step in the new signature protocols described above [13, 18]. Therefore, an efficient threshold protocol for inversion over a shared secret is necessary for threshold versions of those schemes. However, the protocol in Catalano *et al.* requires large share sizes and is difficult to describe, analyze, and implement. In addition, the protocol is only secure in the *static adversary* model, which makes somewhat unrealistic assumptions about the power of the adversary. These problems can be overcome by using techniques from the literature on secure multiparty computation.

Secure multiparty computation (MPC) was introduced by Yao in 1982 [30]. It allows a group of parties, each of whom has a private input, to compute a function on the inputs such that every party learns the value of the function, but nothing else (besides what can be inferred from the value of the function). Ideally, each input would be sent over a secret channel to a trusted third party, who would compute the result and send it to all the participants. The goal of MPC is to aims to eliminate the need for a trusted third party, while retaining the security of the ideal scenario, even when some of the participants may be malicious. Threshold signatures are a natural application of MPC: shares of the private key qualify as inputs, some of the parties may be malicious, and each honest party should end up with a valid signature at the end of the protocol.

Cramer *et al.* [12] have constructed an efficient MPC protocol based upon any threshold encryption scheme having certain homomorphic properties (one such encryption scheme is Paillier's cryptosystem [25]). The work is especially valuable for its unique and simple secure multiplication protocol. The communication requirements for each participant are reasonable in the *broadcast* model, making this MPC protocol feasible for general-purpose secure computation. Finally, the protocol is *optimally-resilient* against the adversary, i.e. security is maintained even if the adversary corrupts up to half of the participants.

A deficiency in all of these threshold signature and MPC protocols is their assumption of the *static* adversarial model. In this model, the adversary controls a fixed set of participants, which are chosen before the protocol begins. The more realistic *adaptive* adversarial model assumes that the adversary can choose new parties to corrupt at any time, based on its view over the history of the protocol. The adversary still remains limited, however, by the threshold number of parties it may corrupt.

Constructing protocols that are provably secure against an adaptive adversary is a difficult task, because the adversary's corruption strategy is unknown and may depend upon public values as well as the internal states of the other corrupted parties. Frankel *et al.* [17] and Canetti *et al.* [7] have developed techniques for designing adaptively-secure threshold cryptosystems, which were improved upon by Jarecki and Lysyanskaya [22].

This thesis will first define the important notions and tools needed to develop secure protocols. It will then describe a simple and adaptively-secure modular inversion protocol built upon a homomorphic threshold cryptosystem.

Then, using techniques from the threshold Paillier cryptosystem presented in Fouque *et al.* [16], we will develop a threshold variant that is secure against an adaptive adversary. With this primitive and a few modifications to existing tools, we will have an efficient, adaptively-secure threshold modular inversion protocol, which enables adaptively-secure threshold versions of the Cramer-Shoup [13] and Gennaro-Halevi-Rabin [18] signature schemes. Threshold signature schemes having security properties of this strength are a novel result.

Chapter 2

Secure Multiparty Computation (MPC)

In this chapter we summarize the intuition behind secure multiparty computation in an adversarial setting, and present a slightly modified subset of the formal model due to Canetti [6]. Other similar definitions have been given by Goldwasser and Levin [19], Micali and Rogaway [23], and Beaver [1].

The goal of secure multiparty computation is to emulate an *ideal process* that works as follows: all parties send their inputs to an incorruptible *trusted party* over a secure channel. The trusted party computes the output(s) of the function and sends them back to the appropriate parties, again over a secure channel. The adversary, who controls a set of parties, has very limited power: it can only learn the public inputs and outputs of all the parties, and (at its option) modify the inputs and outputs of the corrupted parties.

Intuitively, a real-life protocol securely evaluates a function if executing the protocol amounts to *emulating* the ideal process for that function. This emulation occurs if, for *any* adversary in the real-life model, there exists an adversary in the ideal world that induces a *similar output*. What is meant by the phrase "similar output" will be formalized in this chapter.

This basic idea is a natural generalization of the *simulation approach* used to define secure encryption [20] and zero-knowledge proof systems [21], but the formulation is more detailed. It intuitively guarantees two things: first, that anything an adversary could learn in real-life could also be learned by some adversary in the ideal model; and second, that the honest parties have the same outputs in real-life as in the ideal model. Because the adversary's view is so restricted in the ideal model, the first condition ensures *secrecy* in the real-life model. Additionally, because the adversary has limited influence over the honest parties' outputs in the ideal model, the second condition ensures the *correctness* of the real-life protocol.

2.1 Preliminary Definitions

We now proceed to some of the definitions that will enable us to formalize the intuition described above. By 1^k we mean the unary representation of a nonnegative integer k.

Definition 2.1 (Distribution Ensemble) *A* distribution ensemble

$$X = \{X(1^k, a)\}_{k \in \mathbb{N}, a \in D}$$

is an infinite sequence of probability distributions, where a distribution $X(1^k, a)$ is associated with each value of $k \in \mathbb{N}$ and $a \in D$ for some domain D. (Typically, $D = \{0,1\}^*$.)

In this thesis, the distribution ensembles correspond to outputs of computations in which the parameter k is taken to be the *security parameter*, and a corresponds to a tuple of inputs. All complexity characteristics will be measured in terms of the security parameter.

Definition 2.2 (Computational Indistinguishability [20], [6]) *Given $\delta : \mathbb{N} \to [0,1]$, we say that two distribution ensembles X and Y have* computational distance at most δ *if for every algorithm $\mathcal{D}$ that is probabilistic polynomial-time in its first input, for all sufficiently large k, all a, and all auxiliary information $w \in \{0,1\}^*$, we have*

$$|\Pr[\mathcal{D}(1^k, a, w, x) = 1] - \Pr[\mathcal{D}(1^k, a, w, y) = 1]| < \delta(k),$$

where x, y are chosen from distribution $X(1^k, a)$ and $Y(1^k, a)$ respectively, and the probabilities are taken over the choices of x, y, and the random choices of $\mathcal{D}$.

If ensembles X and Y have computational distance at most k^{-c} for all $c > 0$ then we say that X and Y are computationally indistinguishable *and write $X \overset{c}{\approx} Y$.*

For simplicity of exposition, we may say that two *random variables* are computationally indistinguishable when their corresponding ensembles (which will be clear from the context) are computationally indistinguishable.

Note that the distinguisher $\mathcal{D}$ is given access to an arbitrary auxiliary string w, which is fixed *before* the random choices of x and y are made. In our case, this w corresponds to the knowledge gathered by the adversary during previous protocol executions (this is an important technical condition needed to prove that secure protocols can be composed).

We now define *multiparty functions*, which are a formalization of the functions to be evaluated by the parties. This definition differs slightly from Canetti's, in that his employs l inputs and outputs, while we have one secret and one public input/output for each party for a total of $2l$ inputs and outputs.

Definition 2.3 (Multiparty Function) *An l-*party function *is a probabilistic function $f : 1^* \times (\{0,1\}^*)^{2l} \times \{0,1\}^* \to (\{0,1\}^*)^{2l}$, where the length of the first input is the security parameter and the last input is the supply of random bits. The function must be computable in time polynomial in the security parameter.*

Intuitively, we interpret l-party functions as follows: each party P_i has access to a public input x_i^p and a secret input x_i^s, and computes a public output y_i^p and a secret output y_i^s. The *input vector* $\vec{x}$ is the $2l$-tuple $(x_1^s, x_1^p, \ldots, x_l^s, x_l^p)$, which is the second input to the function f. The *output vector* $\vec{y} = f(1^k, \vec{x}, r)$ is a $2l$-tuple defined similarly.

We now turn to the task of defining security for a multiparty function. First, we describe the real-life model of computation. Then, we formalize the ideal process, and finally, we formally define the meaning of *emulation* of an ideal process in the real-life model.

2.2 The Real-Life Model

An *l-party protocol* π is a collection of l interactive probabilistic algorithms (implemented as Turing machines), where by P_i we denote the ith algorithm. Each party starts with a public input $x_i^p \in \{0,1\}^*$, a secret input $x_i^s \in \{0,1\}^*$, a random input $r_i \in \{0,1\}^*$, and the security parameter k. The parties communicate only over a *broadcast channel* (there are no channels, private or otherwise, between individual parties)[1].

A *real-life static adversary* $\mathcal{A}$ is another interactive, probabilistic polynomial-time Turing machine controlling the behavior of the corrupted parties. Its inputs are the identities of the corrupted parties and their respective inputs, an auxiliary input, a random input r_0, and the security parameter k. (The auxiliary input, again, represents knowledge gained by the adversary during previous interactions with the other parties.) We say that an adversary is *t-limited* if it controls at most t parties[2].

Computation proceeds in *rounds*, where each round proceeds as follows: first, the uncorrupted parties generate their messages for the round, according to their respective algorithms. These messages become known to the adversary, who then generates messages on behalf of the corrupted parties. Finally, all the messages for the round are simultaneously placed on the broadcast channel for all parties to see. This captures the essence of a synchronous broadcast network with *rushing*, i.e. one which allows the adversary to learn the uncorrupted parties' messages before generating its own.

When all the rounds have completed (as determined by the protocol), all parties locally generate their outputs. The uncorrupted parties broadcast their public outputs, while the corrupted parties output a special symbol $\perp$ to indicate that they are corrupted (this condition merely simplifies the definition of security, and may be as-

[1]Other reasonable communication models, such as *secure channels* and *point-to-point*, are often assumed, but we will not consider them in this thesis.

[2]We say that the adversary is *static* because the corrupt parties are fixed before the protocol begins. This is in contrast to an *adaptive* adversary, who has the power to corrupt parties of its choosing *during* the computation, based on its view up to that point. Adaptive adversaries will be discussed in a cryptographic setting in a later chapter.

sumed without loss of generality). In addition, the adversary outputs some arbitrary function of its *view* of the computation. The view consists of the adversary's auxiliary and random inputs, followed by the corrupted parties' input values and random inputs, all public inputs and outputs, and all the messages sent during the protocol.

Let $\mathrm{ADVR}_{\pi,\mathcal{A}}(1^k, \vec{x}, z, \vec{r})$ denote the output of the real-life adversary with auxiliary input z when interacting with parties running protocol π on inputs $\vec{x}$ and random inputs $\vec{r} = (r_0, \ldots, r_l)$, with security parameter k. While $\vec{x}$ and $\vec{r}$ are parameters to ADVR, we stress that they are not explicitly given to $\mathcal{A}$; instead, $\mathcal{A}$ receives implicit information about them based on the behavior of the uncorrupted parties. Let $\mathrm{EXEC}_{\pi,\mathcal{A}}(1^k, \vec{x}, z, \vec{r})_i$ denote the outputs of party P_i from this execution. Let

$$\begin{aligned} \mathrm{EXEC}_{\pi,\mathcal{A}}(1^k, \vec{x}, z, \vec{r}) &= \mathrm{ADVR}_{\pi,\mathcal{A}}(1^k, \vec{x}, z, \vec{r}), \\ &\quad\ \mathrm{EXEC}_{\pi,\mathcal{A}}(1^k, \vec{x}, z, \vec{r})_1, \ldots \\ &\quad\ \mathrm{EXEC}_{\pi,\mathcal{A}}(1^k, \vec{x}, z, \vec{r})_l. \end{aligned}$$

Let $\mathrm{EXEC}_{\pi,\mathcal{A}}(1^k, \vec{x}, z)$ denote the distribution of $\mathrm{EXEC}_{\pi,\mathcal{A}}(1^k, \vec{x}, z, \vec{r})$ where $\vec{r}$ is uniformly chosen, and let $\mathrm{EXEC}_{\pi,\mathcal{A}}$ be the ensemble $\{\mathrm{EXEC}_{\pi,\mathcal{A}}(1^k, \vec{x}, z)\}_{k\in\mathbb{N},(\vec{x},z)\in\{0,1\}^*}$.

2.3 The Ideal Process

The ideal process is given by an l-party function to be evaluated (recall Definition 2.3). The adversary $\mathcal{S}$ is an interactive probabilistic polynomial-time Turing machine describing the behavior of the corrupted parties. The adversary's inputs are the identities of the corrupted parties, a random input r, an auxiliary input z, and the security parameter k. In contrast to the real-life model, there is also an incorruptible *trusted party* $\mathcal{T}$ who knows k.

The ideal process on an input vector $\vec{x}$ proceeds as follows: first, the adversary sees the inputs of all the corrupted parties, and may change them arbitrarily; say that the vector of all parties' (possibly modified) inputs is $\vec{m}$. Next, each party P_i gives its (possibly modified) input values to the trusted party $\mathcal{T}$. Then, $\mathcal{T}$ chooses random r_f, and hands each P_i the pair $(y_i^s, y_i^p) = (f(1^k, \vec{m}, r_f)_{2i-1}, f(1^k, \vec{m}, r_f)_{2i})$ (note that the parties do not learn r_f, except for the information provided by the resulting values of y_i^s, y_i^p). Finally, each uncorrupted party P_i outputs (y_i^s, y_i^p), and each corrupted party outputs the special symbol $\perp$. In addition, the adversary outputs some function of its *view* of the computation. This view consists of the adversary's random input, the corrupted parties' inputs, all the public inputs and outputs, and the resulting secret outputs y_i^s for corrupted parties P_i.

Let $\mathrm{ADVR}_{f,\mathcal{S}}(1^k, \vec{x}, z, \vec{r})$, where $\vec{r} = (r_f, r)$, denote the output of ideal-adversary $\mathcal{S}$ on security parameter k, random input r, and auxiliary input z, when interacting in an ideal process on (original) input $\vec{x}$, in which the trusted party has random input r_f. Let $\mathrm{IDEAL}_{f,\mathcal{S}}(1^k, \vec{x}, z, \vec{r})_i$ denote the outputs of party P_i in the ideal process. Then let the vector

$$\begin{aligned} \mathrm{IDEAL}_{f,\mathcal{S}}(1^k, \vec{x}, z, \vec{r}) \quad = \quad & \mathrm{ADVR}_{f,\mathcal{S}}(1^k, \vec{x}, z, \vec{r}), \\ & \mathrm{IDEAL}_{f,\mathcal{S}}(1^k, \vec{x}, z, \vec{r})_1, \ldots \\ & \mathrm{IDEAL}_{f,\mathcal{S}}(1^k, \vec{x}, z, \vec{r})_l \end{aligned}$$

denote the outputs of the parties on input $\vec{x}$, adversary $\mathcal{S}$, and random inputs $\vec{r}$. Let $\mathrm{IDEAL}_{f,\mathcal{S}}(1^k, \vec{x}, z)$ denote the distribution of $\mathrm{IDEAL}_{f,\mathcal{S}}(1^k, \vec{x}, z, \vec{r})$ when $\vec{r}$ is uniformly distributed, and let $\mathrm{IDEAL}_{f,\mathcal{S}}$ denote the ensemble $\{\mathrm{IDEAL}_{f,\mathcal{S}}(1^k, \vec{x}, z)\}_{k \in \mathbb{N}, (\vec{x}, z) \in \{0,1\}^*}$.

2.4 A Definition of Security

With definitions of the two models in hand, we can now proceed to a definition of secure computation. The definition should capture the notion that a real-life protocol π *emulates* the ideal process for calculating a function f. This means that, for any (t-limited) real-life adversary $\mathcal{A}$, there should exist an ideal-process adversary $\mathcal{S}$ such that the output of the protocol with adversary $\mathcal{A}$ is indistinguishable from the output of the ideal process with adversary $\mathcal{S}$. Said another way, a real-life adversary cannot learn anything more about the computation than an ideal-world adversary could learn, nor can it affect the outputs of the uncorrupted players any more than an ideal-world adversary could. A precise definition follows.

Definition 2.4 (Security of a Multiparty Protocol [6]) *Let π be a protocol for l parties, and let f be an l-party function. We say that π t-*securely *evaluates f if, for any t-limited real-life adversary $\mathcal{A}$, there exists an ideal-process adversary $\mathcal{S}$ such that*

$$IDEAL_{f,\mathcal{S}} \overset{c}{\approx} EXEC_{\pi,\mathcal{A}}.$$

2.5 Modular Composition

A natural way of solving problems is to break them into smaller tasks, and then combine the results into an overall solution. We would like to be able to do a similar thing with protocols: design small sub-protocols which are themselves secure, and combine them into a larger protocol which we would also like to be secure. Under a proper (and reasonable) formalization, one can prove that such a composition is indeed secure. The formalization will be geared toward *nonconcurrent* composition, where at most one sub-protocol is executed in any communication round. Security under concurrent composition is non-trivial to prove, and seems to require more assumptions about the adversary [15].

We first describe the model for executing a protocol with the assistance of a trusted third party for evaluating l-party functions $f_1, \ldots, f_m$. This model is called the $(f_1, \ldots, f_m)$-*hybrid model*, and is obtained as follows. We start with the real-life model of Section 2.2, augmented with a trusted party $\mathcal{T}$ for calculating $f_1, \ldots, f_m$. During certain rounds specified by the protocol, one of the functions $f_1, \ldots, f_m$ is evaluated by $\mathcal{T}$ using the ideal-world process for that function (see Section 2.3). A fresh random string is used in each ideal function call.

Let $\text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}(1^k, \vec{x}, z)$ denote the random variable describing the output of the computation in the $(f_1, \ldots, f_m)$-hybrid model with protocol π, adversary $\mathcal{A}$, security parameter k, inputs $\vec{x}$, and auxiliary input z. As above, let $\text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}$ denote the distribution ensemble $\{\text{EXEC}^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}(1^k, \vec{x}, z)\}_{k\in\mathbb{N},(\vec{x},z)\in\{0,1\}^*}$.

Modifying π to use a real-life protocol ρ_i for an l-party function f_i, instead of an ideal-world evaluation of the function, is done in a natural way. At the beginning of the round, each party saves its internal state. The call to the trusted party is replaced by an invocation of the protocol ρ. Once that protocol is complete, each party restores its state and resumes the protocol π, where the party's local outputs are used in place of the value that would have been received from the trusted party. We stress that no uncorrupted party resumes execution of π until all of the uncorrupted parties have completed ρ (this is necessary to avoid violating the nonconcurrency condition).

Let $\pi^{\rho_1,\ldots,\rho_m}$ denote protocol π, originally designed for the $(f_1,\ldots,f_m)$-hybrid model, where each ideal evaluation call to f_i is replaced by a call to sub-protocol ρ_i.

We now rigorously define security for protocols which evaluate functions in the hybrid model, then state a theorem due to Canetti which captures the intuition that if secure protocols are sequentially composed, then the resulting protocol will remain secure.

Definition 2.5 (Security in the Hybrid Model [6]) *Let $f_1,\ldots,f_m,g$ be l-party functions and let π be a protocol for l parties in the $(f_1,\ldots,f_m)$-hybrid model. We say that π t-*securely evaluates g in the $(f_1,\ldots,f_m)$-hybrid model *if, for any t-limited adversary $\mathcal{A}$ in the $(f_1,\ldots,f_m)$-hybrid model, there exists an ideal-process adversary $\mathcal{S}$ such that*

$$IDEAL_{g,\mathcal{S}} \stackrel{c}{\approx} EXEC^{f_1,\ldots,f_m}_{\pi,\mathcal{A}}.$$

Theorem 2.1 (Modular Composition of Secure Protocols [6]) *Let $t < l$, and let $f_1,\ldots,f_m,g$ be l-party functions. Let π be an l-party protocol that t-securely evaluates g in the $(f_1,\ldots,f_m)$-hybrid model where no more than one ideal evaluation call is made at each round, and let $\rho_1,\ldots,\rho_m$ be l-party protocols such that ρ_i t-securely evaluates f_i. Then the protocol $\pi^{\rho_1,\ldots,\rho_m}$ t-securely evaluates g.*

Chapter 3

Σ-Protocols

In this chapter we define a special kind of zero-knowledge proof called a *Σ-protocol* [12]. First we define Σ-protocols in a two-party setting, then we demonstrate how to implement them in a multiparty setting using *trapdoor commitments*.

3.1 Two-Party Σ-Protocols

Assume there is a binary relation R consisting of pairs (x, w), where x is interpreted as a public instance of some problem, and w as a solution (or witness) to the instance. We then have a 3-round proof of knowledge for R that proceeds as follows: both the prover and verifier receive the public value x, and the prover alone receives w, such that $(x, w) \in R$. Conversations are of the form (a, r, z): the prover sends its first message a, the verifier chooses a *challenge* r at random, the prover responds with z, and the verifier either accepts or rejects. There is a security parameter k, such that $|x| = k$ and the lengths of a, r, and z are linear in k.[1]

Let the prover P and verifier V be probabilistic polynomial-time interactive Turing machines. Let $PV(x, w)$ denote the distribution of conversations between P and V on (x, w) (including whether the verifier accepts or not), and let $PV(x, w)[r]$ denote the distribution of conversations between P and V on (x, w) in which r occurs as the

[1]In general, the lengths of a, r, and z may be polynomial in k, but the linear bound simplifies the analysis when extending Σ-protocols to a multiparty setting.

challenge.

A a prover-verifier pair (P, V) is a Σ-protocol if the following holds:

- The protocol is *complete*: if $(x, w) \in R$, then $PV(x, w)$ is an accepting conversation with probability 1.

- The protocol is *special honest-verifier zero-knowledge*: there exists a probabilistic polynomial-time *simulator* S which, on input x and r, generates a conversation of the form (a, r, z), such that $S(x, r) = PV(x, w)[r]$. That is, the output of S on r is identically-distributed with the conversations between P and V, given that r occurs as the challenge.

- With high probability, the prover *knows* w. More precisely, there exists a polynomial-time *knowledge extractor* which, on inputs x and any pair of accepting conversations (a, r, z) and (a, r', z') where $r \neq r'$, always outputs w such that $(x, w) \in R$.[2]

3.2 Trapdoor Commitments

A *trapdoor commitment scheme* is much like a regular commitment scheme: a party P commits to a value by running some probabilistic algorithm on the value. The commitment gives no information about the committed value. At some later stage, P *opens* the commitment by revealing the committed value and the random coins used by the commitment algorithm. P must not be able to find a different value (and corresponding random string) that would yield the same commitment.

Trapdoor commitment schemes have one additional property: there exists a *trapdoor value* which allows P to construct commitments that he can open arbitrarily, such that this cheating is not detectable. Here we define these notions formally:

[2]This is not the most general definition of a proof knowledge. For instance, we could allow the knowledge extractor to be probabilistic, and give it access to a polynomial number of accepting conversations. However, all of the Σ-protocols we will encounter in this thesis meet the stricter definition, which simplifies many of the proofs of security.

Definition 3.1 (Trapdoor Commitment Scheme) *A pair of algorithms* (G, C) *is called a* trapdoor commitment scheme *if the following properties hold:*

1. *(Key generation) G is a probabilistic polynomial-time algorithm on a security parameter k, outputting a* public key *pk and a* trapdoor *t. We write $(pk, t) \leftarrow G(1^k)$.*

2. *(Hiding) For $(pk, t) \leftarrow G(1^k)$, all s, s', and for uniformly-chosen r, r', the distributions (over r and r', respectively) $C(s, r, pk)$ and $C(s', r', pk)$ are computationally indistinguishable.*

3. *(Binding) For any probabilistic circuit family $\{A_k\} \in \mathbf{P}/poly$, any constant $c > 0$, and for all sufficiently large k,*

$$\begin{aligned}&\Pr[\ (pk, t) \leftarrow G(1^k);\ (s, r, s', r') \leftarrow A_k(pk) : \\ &\quad (s \neq s')\ \wedge\ C(s, r, pk) = C(s', r', pk)] \leq 1/k^c\end{aligned}$$

4. *(Trapdoor property) There is a probabilistic polynomial-time algorithm whose output c, given $(pk, t) \leftarrow G(1^k)$, is indistinguishable from $C(s, r, pk)$ over uniform r (by the hiding property, s is irrelevant to the distribution). Furthermore, there is another efficient algorithm which, on input pk, t, c and any s, outputs r such that $c = C(s, r, pk)$.*

3.3 Multiparty Σ-Protocols

The goal of a multiparty Σ-protocol is for many parties to make claims of knowledge such that all parties will be convinced. If all players are honest-but-curious, a naive way of achieving this goal is to make each prover participate in a separate (two-party) Σ-protocol with each of the other players. However, this approach incurs significant communication overhead, and it is not private against an active adversary, since Σ-protocols are only honest-verifier zero-knowledge.

Cramer *et al.* [12] describe how to generate a single challenge that is acceptable to all honest provers. In one case (when $l \leq 16k$), it is simply sufficient to let each party generate a $\lceil 2k/l \rceil$-bit random string, and concatenate all the strings to create an m-bit challenge, where $2k \leq m \leq 16k$. Since half of the parties must be honest, at least k of the challenge bits were chosen randomly, which (for the purposes of proving soundness) is exactly equivalent to performing the Σ-protocol with a random challenge of length k. If $l > 16k$, a preprocessing phase returns as public output a random subset of $4k$ parties. Except with exponentially small probability in k, this subset will contain at least k honest parties, so each party in the subset chooses a single bit a random, and the bits are concatenated to form the challenge.

The complete description of a multiparty Σ-protocol is as follows: in a preprocessing phase, a public key k_i for a trapdoor commitment scheme is generated for each P_i, and is distributed to all the parties by a key-distribution protocol which hides the trapdoor values. In a single proof phase, some subset P' of parties contains the parties who are to prove knowledge.

1. Each $P_i \in P'$ computes a_i, the first message of the two-party Σ-protocol. It then broadcasts a commitment $c_i = C(a_i, r_i, k_i)$, where r_i is chosen randomly by P_i.

2. A challenge r is generated by several parties, as described above. This single challenge will be used by all the provers.

3. Each $P_i \in P'$ computes the answer z_i to the challenge r, and broadcasts a_i, r_i, z_i.

4. Every party can check every proof by verifying that $c_i = C(a_i, r_i, k_i)$ and that (a_i, r, z_i) is an accepting conversation in the two-party Σ-protocol.

The first message from P_i is trapdoor-committed in order to prove the security of the protocol: the simulator imitates the setup phase by invoking G for every party and keeping the commitment trapdoors for the honest parties. The simulator then generates cheating commitments for the honest parties (since it does not have the necessary witnesses), and opens them appropriately after seeing the common challenge

r and invoking the two-party Σ-protocol simulator. By the properties of trapdoor commitments and the two-party simulator, the adversary gets an appropriate view.

A formal proof of security for this protocol is given in Cramer *et al.* [12], but repeating it in detail would take us too far afield. However, the proof hinges upon a general simulator and knowledge-extractor *subroutine*, $\mathcal{E}_\Sigma$, which will be useful in our own proofs of security. $\mathcal{E}_\Sigma(\mathcal{A}, x_{P'}, t_H, k_P)$ generates output $(\mathcal{A}', P'', w_{P'' \cap C})$, with the following specifications:

- $\mathcal{A}$ is the start-state of the adversary, $x_{P'} = \{x_i\}_{i \in P'}$ are the instances to which the parties P' must prove knowledge of witnesses, $t_H = \{t_i\}_{i \in H}$ are the commitment trapdoors of the honest parties H, and $k_P = \{k_i\}_{i \in P}$ are the public commitment keys for all the parties.

- $\mathcal{A}'$ is the state of the adversary after running the multiparty Σ-protocol, and $P'' \subseteq P'$ is the set of parties completing their proofs correctly.

- $\mathcal{S}_\Sigma$ runs in expected polynomial time, and the $(\mathcal{A}', P'')$ part of its output is identically distributed to the output of a real execution of the Σ-protocol, given start state $\mathcal{A}$ of the adversary.

- Except with negligible probability, $w_{P'' \cap C}$ is a set of valid witnesses to the instances x_i belonging to the corrupted parties who completed their proofs correctly.

Briefly, the algorithm of $\mathcal{E}_\Sigma$ uses the trapdoors of the honest parties to create commitments that can be opened arbitrarily, and shows them to the adversary on behalf of the honest parties. These commitments are used to create accepting Σ-protocol conversations for the honest parties. Additionally, for every corrupt party that completes a correct proof, $\mathcal{E}_\Sigma$ rewinds the adversary and chooses a new challenge (by using fresh random bits on behalf of the honest parties). Each corrupt party then generates a new proof, and by using the knowledge extractor of the two-part Σ-protocol, this allows $\mathcal{E}_\Sigma$ to compute the desired witnesses.

Chapter 4

Homomorphic Threshold Encryption

Cramer *et al.* [12] show how to perform secure multiparty computation of any circuit using homomorphic threshold encryption as a primitive. Here we give rigorous definitions for a semantically secure, homomorphic threshold cryptosystem secure against a static t-limited adversary.

4.1 Threshold Cryptosystems

Here we define threshold encryption schemes and their security properties.

Definition 4.1 (Threshold Cryptosystem) *A* statically-secure threshold cryptosystem *for parties* $P = \{P_1, \dots, P_l\}$ *with* threshold $t < l$ *and security parameter* k *is a 5-tuple* $(K, \mathrm{KG}, M, E, \textsc{Decrypt})$ *having the following properties:*

1. *(Key space) The* key space $K = \{K_{k,l}\}_{k,l \in \mathbb{N}}$ *is a family of finite sets of the form* $(pk, sk_1, \dots, sk_l)$*. We call* pk *the* public key *and* sk_i *the* private key share *of party* P_i*. For* $C \subseteq P$ *we denote the family* $\{sk_i\}_{i \in C}$ *by* sk_C*.*

2. *(Key generation) The exists a* t*-secure* key generation l*-party protocol* KG *which, on input* 1^k*, computes, in probabilistic polynomial time, common out-*

put pk and secret output sk_i for party P_i, where $(pk, sk_1, \ldots, sk_l) \in K_k$. We write $(pk, sk_1, \ldots, sk_l) \leftarrow \mathrm{KG}(1^k)$ to represent this process.

3. *(Message sampling) There exists some probabilistic polynomial-time algorithm which, on input pk, outputs a uniformly random element from a* message space *M_{pk}. We write $m \leftarrow M_{pk}$ to describe this process.*

4. *(Encryption) There exists a probabilistic polynomial-time algorithm E which, on input pk and $m \in M_{pk}$, outputs an encryption $\overline{m} = E_{pk}(m)[r]$ of m. Here r is a uniformly random string used as the random input, and $E_{pk}(m)[r]$ denotes the encryption algorithm run on inputs pk and m, with random tape containing r.*

5. *(Decryption) There exists a t-secure protocol* Decrypt *which, on common public input $(\overline{M}, pk)$ and secret input sk_i for each uncorrupted party P_i, where sk_i is the secret key share of the public key pk (as generated by* KG*) and $\overline{M}$ is a (polynomial-size in k) set of encrypted messages $M \subseteq M_{pk}$, returns M as a common public output.*[1]

6. *(Threshold semantic security) For all probabilistic circuit families $\{S_k\}$ (called the* message sampler*) and $\{D_k\}$ (called the* distinguisher*), all constants $c > 0$, all sufficiently large k, and all $C \subseteq P$ such that $|C| \leq t$,*

$$\begin{aligned}\Pr[\ & (pk, sk_1, \ldots, sk_l) \leftarrow \mathrm{KG}(1^k);\\ & (m_0, m_1, s) \leftarrow S_k(pk, sk_C);\\ & i \xleftarrow{R} \{0,1\};\ e \leftarrow E(pk, m_i);\\ & b \leftarrow D_k(s, e):\\ & b = i] < 1/2 + 1/k^c\end{aligned}$$

[1] *The input must contain a* set *of ciphertexts because we require that the decryption algorithm be secure under parallel composition; however, the MPC model does not necessarily preserve security in this setting.*

4.2 Homomorphic Properties

We also need the cryptosystem to have the following homomorphic properties:

1. (Message ring) For all public keys pk, the message space M_{pk} is a ring in which we can compute efficiently using the public key only. We denote the ring $(M_{pk}, \cdot_{pk}, +_{pk}, 0_{pk}, 1_{pk})$. We require that the identity elements 0_{pk} and 1_{pk} be efficiently computable from the public key.

2. ($+_{pk}$-homomorphic) There exists a polynomial-time algorithm which, given public key pk and encryptions $\overline{m}_1 \in E_{pk}(m_1)$ and $\overline{m}_2 \in E_{pk}(m_2)$, outputs a uniquely-determined encryption $\overline{m} \in E_{pk}(m_1 +_{pk} m_2)$. We write $\overline{m} = \overline{m}_1 \boxplus \overline{m}_2$. Likewise, there exists a polynomial-time algorithm for performing subtraction: $\overline{m} = \overline{m}_1 \boxminus \overline{m}_2$.

3. (Multiplication of a ciphertext by a ring element) There exists a probabilistic polynomial-time algorithm which, on input pk, $m_1 \in M_{pk}$ and $\overline{m}_2 \in E_{pk}(m_2)$, outputs a random encryption $\overline{m} \leftarrow E_{pk}(m_1 \cdot_{pk} m_2)$. We assume that we can multiply a ring element from both the left and right. We write $\overline{m} \leftarrow m_1 \boxdot \overline{m}_2 \in E_{pk}(m_1 \cdot_{pk} m_2)$ and $\overline{m} \leftarrow \overline{m}_1 \boxdot m_2 \in E_{pk}(m_1 \cdot_{pk} m_2)$. Let $(m_1 \boxdot \overline{m}_2)[r]$ denote the unique encryption produced by using r as the random coins in the multiplication-by-ring-element algorithm.

4. (Addition of a ciphertext and a ring element) There exists a probabilistic polynomial-time algorithm which, on input pk, $m_1 \in M_{pk}$ and $\overline{m}_2 \in E_{pk}(m_2)$, outputs a uniquely-determined encryption $\overline{m} \in E_{pk}(m_1 +_{pk} m_2)$. We write $\overline{m} = m_1 \boxplus \overline{m}_2$.

5. (Blindable) There exists a probabilistic polynomial-time algorithm B which, on input pk and $\overline{m} \in E_{pk}(m)$, outputs an encryption $\overline{m}' \in E_{pk}(m)$ such that $\overline{m}' = E_{pk}(m)[r]$, where r is chosen uniformly at random.

6. (Check of ciphertextness) By C_{pk} we denote the set of possible encryptions of any message, under the public key pk. Given $y \in \{0,1\}^*$ and pk, it is easy to

check whether $y \in C_{pk}$.

7. (Proof of plaintext knowledge) Let $L_1 = \{(pk, y) : pk \text{ is a public key} \wedge y \in C_{pk}\}$. There exists a Σ-protocol for proving the relation R_{POPK} over $L_1 \times (\{0,1\}^*)^2$ given by $R_{POPK} = \{((pk, y), (x, r)) \;:\; x \in M_{pk} \wedge y = E_{pk}(x)[r]\}$. Let $\mathcal{E}_{POPK}$ be the simulator for this Σ-protocol, which is just a special case of $\mathcal{E}_\Sigma$ described in Chapter 3.

8. (Proof of correct multiplication) Let $L_2 = \{(pk, x, y, z) : pk \text{ is a public key} \wedge x, y, z \in C_{pk}\}$. There exists a Σ-protocol for proving the relation R_{POCM} over $L_2 \times (\{0,1\}^*)^3$ given by $R_{POCM} = \{((pk, x, y, z), (d, r_1, r_2)) \;:\; y = E_{pk}(d)[r_1] \wedge z = (d \boxdot x)[r_2]\}$.

We call any such scheme meeting these additional requirements a *homomorphic threshold cryptosystem.*

From these properties, it is clear how to perform addition of two ciphertexts: use the $+_{pk}$ algorithm, following by an optional blinding step. The remaining operation to be supported is secure multiplication of ciphertexts. That is, given $\overline{a}$ and $\overline{b}$, determine a ciphertext $\overline{c}$ such that $c = a \cdot_{pk} b$, without leaking any information about a, b, or c. Cramer *et al.* [12] give the MULT protocol for secure multiplication, which we reproduce here:

1. Each party P_i chooses a random $d_i \in M_{pk}$, and broadcasts an encryption $\overline{d_i}$. All parties prove (via a multiparty Σ-protocol) knowledge of their respective values d_i.

2. Let $d = d_1 +_{pk} \cdots +_{pk} d_n$. All parties now compute $\overline{a} \boxplus \overline{d_1} \boxplus \cdots \boxplus \overline{d_n}$, an encryption of $a +_{pk} d$. The ciphertext is threshold decrypted, so all players know $a +_{pk} d$.

3. Party P_1 sets $a_1 = (a +_{pk} d) -_{pk} d_1$; all other parties P_i set $a_i = -d_i$. Note that every player can compute an encryption of each a_i, and that $a = a_1 +_{pk} \cdots +_{pk} a_n$.

4. Each P_i broadcasts an encryption $\overline{a_i \cdot_{pk} b}$, and proves in zero-knowledge that the multiplication is correct.

5. Let C be the set of players for which the previous step succeeded, and let F be the complement of C. The parties first decrypt the ciphertext $\boxplus_{i \in F} \overline{a_i}$, giving them the value $a_F = \sum_{i \in F} a_i$. All parties then compute an encryption $\overline{a_F \cdot_{pk} b}$. From this, and $\{\overline{a_i b} \mid i \in C\}$, all parties compute an encryption $(\boxplus_{i \in C} \overline{a_i \cdot_{pk} b}) \boxplus \overline{a_F \cdot_{pk} b}$, which is an encryption of $a \cdot_{pk} b$.

Intuitively, this protocol is secure if the encryption is secure, because other than the outputs, only random values and values already known to the adversary are ever decrypted. Cramer *et al.* give a formal proof of security [12].

Chapter 5

Adaptively-Secure Inversion Modulo a Shared Secret

Here we describe a protocol for computing a prime's inverse with respect to a shared secret modulus. This task is the fundamental step in the Cramer-Shoup [13] and Gennaro-Halevi-Rabin [18] signature schemes. These schemes are provably secure based only upon some reasonable number-theoretic assumptions, without a random oracle. Therefore, it is valuable to construct threshold versions of these schemes.

Catalano *et al.* [9] present two versions of a modular inversion protocol which are secure against a static adversary. The first protocol is private but not robust, while the second adds robustness at the cost of more complexity. Here we give an adaptively-secure protocol, based on their simpler version.

We assume the existence of a homomorphic threshold cryptosystem as described in Chapter 4. We denote an encryption of a message x as $\overline{x}$ when the public key is clear from the context. We also assume a trapdoor commitment scheme as described in Chapter 3.

In all of our protocols, any deviation that is detectable by all honest parties causes the misbehaving party to be excluded from all protocols for the life of the system. Upon detecting a dishonest party, the others restart *only the current protocol* from the beginning. Intuitively, this general strategy is necessary to prevent an adversary from gaining some level of control over the protocol by failing to open its commitments after witnessing the honest parties' behavior. This rule will apply in each round of every protocol, even when not stated explicitly.

A note about the round-efficiency of this rule: the number of rounds of a single protocol execution is bounded only by a constant multiple of the threshold t (since one corrupt party may force a restart every time). However, the adversary can force a total of only $O(t)$ extra rounds to be executed over all invocations, which is a negligible amortized cost over the life of the system. (This assumes that all protocols are constant-round when no malicious parties are present, which will be the case.)

5.1 The Adaptive Adversary Model

We now introduce a stronger adversarial model that is in many ways more realistic than the static adversary model. We refrain from giving a precise formulation of the model in terms of multiparty computation because the definitions are actually *stricter* than necessary for our threshold cryptographic protocols. However, the reader may see Canetti [6] for details. Frankel *et al.* [17] provide definitions of the model and methodologies for proving adaptive security in many cryptographic settings (including ours). We summarize their results here.

Recall that a t-limited static adversary must choose at most t parties to corrupt before any protocols are executed. The adversary sees all messages sent to the corrupt parties and may control them arbitrarily, but it may *not* corrupt any other parties after the setup phase. In many settings, this is an unreasonable restriction. For example, in real life, a malicious entity may choose which servers to break into after seeing the broadcast messages of some protocol. An adversary in the static model is prohibited from doing such a thing.

In contrast, a t-limited *adaptive* adversary may choose to corrupt any party at any point over the lifetime of the system, as long as it does not corrupt more than t parties in total. The choices may be based on everything the adversary has seen up to that point (all broadcast messages and the internal states of all other corrupted parties). When an adaptive adversary corrupts a party, it is given the entire computation history of that party and takes control of its actions for the life of the system. Note that this prohibits the honest parties from making *erasures* of their internal states at any time. In certain contexts, erasures are a reasonable (and sometimes necessary) primitive. However, our protocols will not require them.

It is easy to see that an adaptive adversary is at least as strong as a static one: an adaptive adversary may mimic a static adversary by simply corrupting the same choice of parties before the system is initialized. Conversely, there are protocols that are secure in the static model, but are provably insecure in the adaptive model.

As expected, security of a protocol is defined in the adaptive model using the simulation paradigm. For any adaptive adversary $\mathcal{A}$, there must exist a simulator $\mathcal{S}$ which interacts with $\mathcal{A}$ to provide a view which is computationally indistinguishable from the adversary's view of the real protocol. The main difficulty in designing secure protocols in the adaptive model is in being able to "fake" the messages of the honest parties such that there are consistent internal states that can be supplied to the adversary when it chooses to corrupt new parties. In fact, we will design the protocols such that the simulator can supply consistent states on behalf of all honest parties except one, which we call the "single inconsistent party," and denote P_S. We stress than the inconsistent party is chosen at random by the global simulator, and remains the same throughout all simulator subroutines.

We will design simulators that supply a suitable view to the adversary *provided* P_S *is not corrupted*, or said another way, the adversary's view will be indistinguishable from a real invocation *up to the point at which* P_S *is corrupted* (if ever). This condition is sufficient for proving security: if the adversary asks to corrupt P_S, then we abort *all running simulations* (since our simulators may call upon each other as subroutines), reset the adversary to its initial state, randomly choose a new P_S, and restart the global simulator. Because the adversary can corrupt at most half of the parties, and its view is indistinguishable from an interaction with honest parties, the probability of the adversary corrupting P_S in a single run is at most negligibly better than $1/2$. Therefore the expected number of runs is constant, and the global simulator runs in expected polynomial time.

5.1.1 Adaptively-secure multiparty Σ-protocols

In Chapter 3 we introduced the notion of Σ-protocols, which are essentially honest-verifier zero-knowledge proofs of knowledge, with the added convenient restriction that any two accepting conversations are sufficient to compute the witness. Cramer *et al.* [12] describe how to efficiently perform multiparty Σ-protocols by generating a shared challenge string to be answered by each prover, but the analysis is only valid in the static adversary model. Specifically, when $l > 16k$, the scheme relies upon a trusted oracle to choose a random $4k$-party subset after the adversary has chosen its corrupted parties. These $4k$ parties each contribute one bit to each random challenge, so that with high probability at least k of the bits are chosen honestly. In the adaptive adversary model, however, this is insufficient because the adversary may just choose to corrupt the $4k$ chosen parties, thereby destroying the integrity of the challenges.

To resolve this issue, we simply let each party contribute a random $\lceil 2k/l \rceil$-bit string, and concatenate them. This produces an m-bit challenge, where $2k \leq m \leq 16k$, so at least k of the challenge bits are chosen honestly, as desired. When $l \leq 16k$, only $O(k)$ bits are broadcast. When $l > 16k$, the cost is $O(l)$ broadcast bits, which is still reasonable.

With this modification, the generic multiparty Σ-protocol scheme remains secure in the adaptive adversary model, and the generic simulator/witness extractor $\mathcal{E}_\Sigma$ can be used as-is. It is also important to note that the Mult protocol from Cramer *et al.* [12] can be implemented securely in the adaptive adversary model, with minor changes.

5.2 A Preliminary Subprotocol

First we assume existence of a secure protocol Mad (meaning "multiply, add, decrypt") which has the following specification:

- public inputs $w, \overline{x}, \overline{y}, \overline{z}$ to all parties
- public output $F = wx + yz$ for all parties.

Given a suitable homomorphic threshold cryptosystem, MAD can be implemented using the secure MULT and DECRYPT protocols as follows:

1. Each party publishes a trapdoor-commitment to a random string r_i for use in the multiplication-by-ring-element algorithm.

2. The parties open their commitments, and compute r as the exclusive-or of all properly-decommitted strings.

3. Each party runs the multiplication-by-ring-element algorithm on inputs w and $\overline{x}$ with random string r, yielding a common random ciphertext $\overline{wx}$.

4. The parties enter the MULT protocol on $\overline{y}, \overline{z}$, yielding common random ciphertext $\overline{yz}$.

5. Each party uses the deterministic addition-of-ciphertexts algorithm to compute a common input $\overline{wx + yz}$ to the DECRYPT protocol, yielding common output $F = wx + yx$, as desired.

The simulator $\mathcal{S}_{Mad}$ for this protocol is straightforward. Its input is $w, \overline{x}, \overline{y}, \overline{z}$ as well as the public output $F = wx + yz$. It also receives the identity of the inconsistent party P_S and its trapdoor value t_S. The simulator operates as follows:

1. On behalf of honest parties, commit to random strings. For P_S, use the trapdoor t_S to create a cheating commitment. Receive commitments from the corrupted parties.

2. Decommit honestly on behalf of the honest parties, and decommit to an arbitrary value on behalf of P_S. Receive decommitments from the corrupted parties.

3. Rewind the adversary to before the commitments are opened. Choose a random string r and let $r_S = r \oplus r_C \oplus r_H$, where r_C is the exclusive-or of the corrupted parties' (now revealed) r_is, and r_H is the exclusive-or of the honest parties' (excluding P_S) r_is. Decommit honestly on behalf of the honest parties,

and decommit as r_S on behalf of P_S by using the trapdoor t_S. Receive decommitments from the corrupted parties; the values must be the same as the earlier decommitments except with non-negligible probability by the binding property.

4. All parties take the exclusive-or of all decommitted values, yielding the truly-random value r. Using r as the random input to the multiplication-by-ring-element algorithm, they all compute a common random ciphertext $\overline{wx}$.

5. Compute a random ciphertext $\overline{yz}$ by encrypting $wx + yz$ (which is given as input), subtracting the value $\overline{wx}$ from the previous step, and blinding. Run $\mathcal{S}_{Mult}$ on inputs $\overline{y}, \overline{z}, \overline{yz}$ to simulate the multiplication protocol.

6. Run $\mathcal{S}_{Decrypt}$ on $\overline{wx} \boxplus \overline{yz}$ (from the last two steps) and $F = wx + yz$ to simulate the decryption protocol.

It is clear that $\mathcal{S}_{Mad}$ gives the adversary a suitable view of the protocol, if P_S is never corrupted. The simulator always acts honestly on behalf of all other honest parties, so there is always a consistent internal state that can be shown to the adversary upon corruption. In the simulation, the value of r is truly random, while in the real protocol, r is indistinguishable from random due to the hiding property. By the trapdoor property, the commitment from P_S is indistinguishable from an honest commitment. Finally, the security of the MULT and DECRYPT protocols ensures that their respective simulators generate suitable views.

5.3 Two Preliminary Proofs

In the inversion protocol, each party provides a ciphertext and must prove that it is an encryption of zero. For the Paillier cryptosystem, this is merely a proof of nth residuosity modulo n^2. Such a proof and is virtually identical to a zero-knowledge proof of quadratic residuosity mod n as given by, for example, Goldwasser *et al.* [21].

In addition, each party must publish a ciphertext and prove that the corresponding plaintext lies within a specified range. Boudot [3] describes such a proof for committed values, and a proof of equality between a committed value and a ciphertext in the Paillier cryptosystem can be constructed using standard techniques (see, for example, Camenisch and Lysyanskaya [4]).

5.4 The Inversion Protocol

The INVERT protocol has the following specification:

- common public input $(pk, e, N, \overline{\phi}, \{k_i\})$. Here e is a prime to be inverted modulo the secret ϕ, N is an upper bound on the value of ϕ, and $\{k_i\}$ is the set of all public trapdoor commitment keys.
- secret input sk_i, the ith secret key share, to party P_i.
- common public output P', the set of parties completing the protocol correctly, and $\overline{d_i}$ for each $P_i \in P'$ where d_i is described below.
- secret output d_i from party $P_i \in P'$. The $\{d_i\}$ constitute an additive sharing of the inverse, i.e. $\sum_{P_i \in P'} d_i = e^{-1} \bmod \phi$.

The protocol proceeds as follows:

1. Each P_i publishes a random encryption $\overline{0_i}$ of zero, and proves that it is valid (see Section 5.3). All parties internally compute $\overline{\phi_B} = (\boxplus \overline{0_i}) \boxplus \overline{\phi}$ where the summation is taken over all parties who gave valid proofs.
2. Each P_i chooses random λ_i from the range $[0 \ldots N^2]$, and random r_i from the range $[0 \ldots N^3]$, and encrypts them to get $\overline{\lambda_i}$ and $\overline{r_i}$, respectively.
3. Each P_i (in an order that may be chosen by the adversary) commits to his ciphertexts $\overline{\lambda_i}$ and $\overline{r_i}$.
4. Each P_i (again, in an adversary-chosen order) decommits by broadcasting $\overline{\lambda_i}$ and $\overline{r_i}$, and the random strings used to generate the commitments. If a party decommits improperly or fails to decommit, it is excluded from this and all

future invocations of the protocol. The protocol is then restarted from the beginning.

5. Each party proves that its λ_i and r_i values are within the proper intervals, respectively: each party first publishes commitments to both values, then proves that the committed values are the same as their respective plaintexts, and finally proves that the committed values are within range. See Section 5.3 for details.

6. Each party proves knowledge of its plaintexts using a multiparty Σ-protocol. Let P' be the set of parties for which *both* proofs were correct, the let $\lambda = \sum_{i \in P'} \lambda_i$, $R = \sum_{i \in P'} r_i$, and $F = Re + \lambda\phi$. Again, any party which gives an incorrect proof, or fails to give a proof, is excluded from this and all future invocations, and the protocol is restarted.

7. The parties run the MAD protocol on e, $\overline{R}$, $\overline{\lambda}$, and $\overline{\phi_B}$, where $\overline{R} = \boxplus_{i \in P'} \overline{r_i}$, $\overline{\lambda} = \boxplus_{i \in P'} \overline{\lambda_i}$ by addition of ciphertexts. This protocol securely computes the value $F = Re + \lambda\phi$ as the common output.

8. Each party determines whether $(e, F) = 1$. Because e is prime, $(e, F) \neq 1$ only if e divides λ, which happens with probability about $1/e$ because at least one λ_i is chosen at random. If $(e, F) \neq 1$, the parties repeat the protocol from the first step. Otherwise, all parties compute a, b such that

$$\begin{aligned} aF + be = 1 \quad &\iff \quad aRe + a\lambda\phi + be = 1 \\ &\iff \quad (aR + b) \equiv e^{-1} \bmod \phi. \end{aligned}$$

P_i's share is $d_i = ar_i$ for $i > 1$, and $d_1 = ar_1 + b$ for $i = 1$. Note that any party can use the homomorphic properties of the cryptosystem to compute an encryption $\overline{d_i}$ for any $i \in P'$, because the values of a and b are known to all parties, as well as encryptions $\overline{r_i}$ for all $i \in P'$.

5.5 A Proof of Security

Theorem 5.1 (Security of inversion protocol) *For $t < l/2$,* INVERT *is an adaptively t-secure multiparty protocol for computing $e^{-1} \bmod \phi$.*

Proof: We prove security of the INVERT protocol using the MAD simulator $\mathcal{S}_{Mad}$. We will also assume a secure key-generation protocol for the homomorphic cryptosystem. We describe the construction of such a protocol for a threshold Paillier cryptosystem in Chapter 6. Lastly, we will assume a secure key-generation protocol for the trapdoor commitment scheme.

Let k_P be the public commitment keys for all the parties. Let P_S be the inconsistent party and t_S be its trapdoor value determined by the simulator for the key-generation protocol. Given $\mathcal{A}$, we will construct a simulator subroutine $\mathcal{S}_{Invert}$ which, on input $(\mathcal{A}, pk, e, N, \overline{\phi}, k_P, P_S, t_S)$, outputs $(\mathcal{A}', d_H)$, where $\mathcal{A}'$ is the state of the adversary after the protocol, and d_H are the additive shares belonging to the honest parties (these two items completely determine the output vector for the protocol).

$\mathcal{S}_{Invert}$ operates as follows:

1. For each honest P_i except P_S, honestly publish and prove validity of a random encryption of zero. For P_S, publish a blinding of $\overline{N} \boxminus \overline{\phi}$ and use $\mathcal{E}_{\Sigma}$ and the trapdoor value t_S to give a false proof of validity. At this stage, $\phi_B = N$, and all of the parties hold an encryption of N instead of an encryption of ϕ.

2. Through the decommitment phase, behave honestly. That is, choose random λ_i and r_i for each honest party, commit to their ciphertexts, and decommit honestly.

3. During the round in which the parties prove plaintext knowledge, call the subroutine $\mathcal{E}_{POPK}$ on the corrupted parties' ciphertexts $\overline{\lambda_i}$ and $\overline{r_i}$. (Note that $\mathcal{E}_{POPK}$ does not need any trapdoor values, because the simulator actually knows the relevant plaintexts for the honest parties.) As described in Chapter 3, with high probability $\mathcal{E}_{POPK}$ returns the set of corrupted parties which completed their

proofs of plaintext knowledge successfully, and the plaintexts λ_i, r_i for those same corrupted parties.

4. Set $R' = \sum_{i \in P'} r_i$ and $\lambda' = \sum_{i \in P'} \lambda_i$, where P' is the set of all parties remaining. Run $\mathcal{S}_{Mad}$ on $e, \boxplus_{i \in P'} \overline{r_i}, \boxplus_{i \in P'} \overline{\lambda_i}, \overline{\phi_B} = \overline{N}$, and $F' = R'e + \lambda' N$.

5. Proceed exactly according to the protocol, repeating if $(e, F') \neq 1$.

It is clear that the simulator runs in expected polynomial time. It remains to be shown that the output of the simulator is computationally indistinguishable from the output of a real run of the protocol. Let us assume for now that this is not the case, and that there is an adversary $\mathcal{A}$ which can distinguish between a real-life execution of INVERT and an interaction with $\mathcal{S}_{Invert}$ with non-negligible advantage. We will provide a reduction that uses $\mathcal{A}$ to break the semantic security of the cryptosystem, thus establishing a contradiction. The reduction will employ a *hybrid simulator* interacting with $\mathcal{A}$.

5.5.1 The hybrid simulator

Consider the simulator $\mathcal{S}_{Hybrid}$ which receives the public key pk of the homomorphic cryptosystem, the public commitment keys k_P, the identity of the inconsistent party P_S, and its commitment trapdoor value t_S. In addition, it is supplied with $N, e, \phi, \overline{\phi}$, a ciphertext $\overline{b}$ where b is either 0 or 1, and an auxiliary input representing the state of the adversary $\mathcal{A}$. These inputs are supplied as the auxiliary output of the message-sampling algorithm in a semantic security experiment, where the goal is to determine with non-negligible advantage whether $b = 0$ or $b = 1$. We now describe the interaction of this hybrid simulator with the adversary. (As always, any deviation from the protocol by a corrupt party causes its exclusion, and a restart.)

1. For each $P_i \neq P_S$, publish a random encryption of zero and proves its validity. For P_S, publish $(N - \phi) \boxdot \overline{b}$ and give a false proof of validity using $\mathcal{E}_\Sigma$ and the trapdoor t_S. Let $\overline{\phi_B}$ be as in the INVERT protocol.

2. For all honest $P_i \neq P_S$, honestly choose λ_i and r_i and commits to their ciphertexts. For P_S, choose λ_S and r_S from the proper ranges, but use the commitment trapdoor t_S to create cheating commitments. Receive all commitments from the corrupt parties.

3. For all honest $P_i \neq P_S$, decommit the ciphertexts honestly. For P_S, open the cheating commitments as $\overline{\lambda_S}$ and $\overline{r_S}$.

4. Honestly prove plaintext knowledge for all honest parties, and use $\mathcal{E}_{POPK}$ to extract the corrupt parties' correctly-proven λ_i and r_i values. Let λ_C and R_C be the respective sums of these variables for the corrupt parties, and let λ_H and R_H be the respective sums of the honest parties' λ_is and r_is.

5. Rewind the adversary to the point immediately preceding the round in which the parties publish their ciphertexts. Solve for λ'_H and R'_H such that $F = (\lambda_C + \lambda_H)\phi + (R_C + R_H)e = (\lambda_C + \lambda'_H)N + (R_C + R'_H)e$. We shall prove that such λ'_H and R'_H are easy to compute, and are similarly distributed with λ_H and R_H (respectively).

6. For all honest parties $P_i \neq P_S$, again honestly decommit to their ciphertexts $\overline{\lambda_i}$, $\overline{r_i}$. For P_S, open the cheating commitments as blinded ciphertexts $\overline{\lambda'_S}$ and $\overline{r'_S}$, where $\overline{\lambda'_S} = \overline{\lambda_S} \boxplus (\lambda'_H - \lambda_H) \boxdot \overline{b}$, and $\overline{r'_S} = \overline{r_S} \boxplus (R'_H - R_H) \boxdot \overline{b}$. In the same round, the corrupt parties must decommit to the same values as they did before rewinding (doing otherwise would constitute a break of the commitment scheme, and the reduction is straightforward). Therefore λ_C and R_C remain unchanged.

7. Honestly prove plaintext knowledge on behalf of all honest parties $P_i \neq P_S$, and use $\mathcal{E}_{POPK}$ and the commitment trapdoor t_S to provide fake proofs of plaintext knowledge for $\overline{\lambda'_S}$ and $\overline{r'_S}$. Also receive proofs of plaintext knowledge from the corrupt parties.

8. Run $\mathcal{S}_{Mad}$ on e, the homomorphic sums of the ciphertexts, the secret input ciphertext, and the value F. Finish the protocol honestly.

We now prove the correctness of the reduction. Certainly if the adversary corrupts any party besides P_S, the hybrid can supply a valid internal state because it is acting honestly on behalf of that party. We now show that if $b = 0$, the output of $\mathcal{S}_{Hybrid}$ is indistinguishable from a real run of the INVERT protocol. Similarly, if $b = 1$, the output is indistinguishable from the output of $\mathcal{S}_{Invert}$. Therefore an adversary that can detect a simulation of INVERT can be used to break the semantic security of the underlying cryptosystem.

First, assume that $b = 0$. Then it is easy to verify that the hybrid acts honestly on behalf of all the uncorrupted parties, and in the first round P_S indeed publishes a random encryption of zero, so $\phi_B = \phi$. The only deviation from the real protocol occurs in the creation of cheating commitments for P_S and in the proofs of plaintext knowledge, but these commitments are computationally indistinguishable from honest commitments by assumption. Because $\lambda'_S = \lambda_S$ and $r'_S = r_S$, the behavior of P_S is indistinguishable from an honest party's in the real protocol.

Now assume that $b = 1$. Then P_S publishes a random encryption of $N - \phi$ as in the simulation, and $\phi_B = N$. Note that all λ_i, r_i belonging to honest parties are chosen uniformly except for λ'_S and r'_S. But as we will show, the distributions of those variables are statistically indistinguishable from the respective uniform distributions. So in fact the behavior of P_S in the hybrid is indistinguishable from its behavior under $\mathcal{S}_{Invert}$.

It only remains to be proven that λ_S, r_S are similarly-distributed with λ'_S, r'_S (respectively), which we do here. We assume for simplicity that $N - \phi = O(\sqrt{N})$, as is the case when $\phi = \phi(N)$ and N is the product of two large primes of approximately equal size. First we state the following lemma:

Lemma 5.1 ([9]) *Let x, y be two integers such that $(x, y) = 1$ and A, B two integers such that $A < B$, $x, y < A$, and $B > Ax$. Then every integer z in the closed interval $[xy - x - y + 1, Ax + By - xy + x + y - 1]$ can be written as $ax + by$ where $a \in [0, A]$ and $b \in [0, B]$. Furthermore, there exists a polynomial-time algorithm that on input x, y, and z, outputs such a and b.*

Let us denote λ as $\lambda_C + \lambda_H$, λ' as $\lambda_C + \lambda'_H$, R as $R_C + R_H$, and R' as $R_C + R'_H$. We apply this lemma twice, first with $x = \phi$, $y = e$, and again with $x = N$, $y = e$ to conclude that any integer F in the interval $[\delta, \Delta]$ can be written both as $\lambda\phi + Re$, and as $\lambda' N + R'e$, where $\lambda, \lambda' \in [0, nN^2]$ and $R, R' \in [0, nN^3]$. Here $\delta = Ne - e + 1$, and $\Delta = n(N^2\phi + N^3 e) - \phi e + \phi + e - 1$.

Now, given any fixed λ_C, R_C (the sums of the adversaries' chosen values in the protocol) and any λ_H (respectively, R_H) distributed as the sum of at least $n/2$ honestly-chosen uniform values from $[0, N^2]$ (respectively, $[0, N^3]$), it is easy to see by Chernoff bounds that the probability that F falls outside the range $[\delta, \Delta]$ is negligible since both bounds fall far away from the mean of F.

Now suppose $F \in [\delta, \Delta]$ and λ_C, R_C are fixed as in the protocol. Given a pair λ_H, R_H such that $F = (\lambda_C + \lambda_H)\phi + (R_C + R_H)e$, we present an efficient mapping that produces λ'_H, R'_H such that $F = (\lambda_C + \lambda'_H)N + (R_C + R'_H)e$. That is,

$$\lambda\phi - \lambda' N = (R' - R)e$$

Since $(N, e) = 1$, for any given λ there exists a unique and efficiently-computable $\lambda' \in [\lambda, \lambda + e - 1]$ such that $\lambda\phi - \lambda' N$ is a multiple of e. This determines the value $\lambda'_H - \lambda_H + \lambda_S = \lambda'_S$ (one of the values published by the first honest party in the hybrid simulator), and from that we can solve for $R' - R + r_S = r'_S$ (the other published value).

We need only show that λ_S, λ'_S and r_S, r'_S are close enough in a statistical sense, i.e. that their differences are small relative to the sizes of the intervals from which they are drawn. Indeed,

$$\frac{|\lambda' - \lambda|}{N^2} \leq \frac{e}{N^2} \leq \frac{1}{N}$$

and

$$|r_1 - r'_1| = \frac{|\lambda\phi - \lambda' N|}{e} = \left|\frac{(\lambda - \lambda')\phi}{e} + \frac{\lambda'(\phi - N)}{e}\right| \leq \left|\phi - \frac{nN^2\sqrt{N}}{e}\right| \leq nN^2\sqrt{N}$$

Thus

$$\frac{|r_1 - r_1'|}{N^3} \le \frac{n}{\sqrt{N}}$$

which again is negligible. This completes the proof.

Chapter 6

An Adaptively-Secure Threshold Paillier Cryptosystem

Paillier [25] introduced a probabilistic cryptosystem which is provably secure under a novel number-theoretic hardness assumption related to *composite-degree residuosity classes* modulo n^2, where n is an RSA modulus. Its most valuable feature is additive homomorphism, that is, the product of two ciphertexts is an encryption of the sum of the plaintexts. Besides the application to MPC and threshold cryptography studied in this thesis, the homomorphism property is also useful for secure elections and lottery schemes.

In this chapter, we describe an adaptively-secure threshold variant of the Paillier cryptosystem, which is provably secure under a novel number-theoretic hardness assumption. We will again be working in the adaptive adversary model, described in Section 5.1.

6.1 Preliminaries

We introduce the following notation: for any $n \in \mathbb{N}$, $\lambda(n)$ denotes Carmichael's lambda function, defined as the largest order of the elements of $\mathbb{Z}_n^*$. It is known that if the prime factorization of an odd integer n is $\prod_{i=1}^{k} q_i^{f_i}$, then $\lambda(n) = \mathrm{lcm}_{i=1\ldots k}(q_i^{f_i-1}(q_i - 1))$.

6.1.1 Shamir threshold secret sharing

Shamir [28] proposed a protocol to share a secret element s in a field F among l parties in such a way that any $t+1$ parties can efficiently recover s, but any group of up to t of them cannot gain any information about s. This scheme can also be made to work over the integers, with similar restrictions on what a group of t parties can learn about the secret.

The protocol is based on the Lagrange polynomial interpolation formula that allows computation of $P(X)$, for any X, if P is a polynomial of degree t and if $t+1$ values $P(x_i)$ are known for distinct elements $x_1, \ldots, x_{t+1}$:

$$P(X) = \sum_{i=1}^{t+1} \prod_{j=1, j \neq i}^{t+1} \frac{X - x_j}{x_i - x_j} \cdot P(x_i)$$

In order to share a secret s, choose a random polynomial P of degree t such that $P(0) = s$ and each party receives a unique point $(x_i, P(x_i))$ with $x_i \neq 0$. The above formula shows how, given $t+1$ points, to discover $P(0) = s$.

When working over the integers, it is useful to define a variant of the Lagrange interpolation coefficients. If there are l parties in all, let $\Delta = l!$, and define

$$\mu_{i,j}^S = \Delta \cdot \frac{\prod_{j' \in S \setminus \{j\}} (i - j')}{\prod_{j' \in S \setminus \{j\}} (j - j')} \in \mathbb{Z}.$$

Therefore for any set S of $t+1$ distinct points at which the value of P is known, we have

$$\Delta P(i) = \sum_{j \in S} \mu_{i,j}^S P(j).$$

6.1.2 Proving equality of discrete logs

Our cryptosystem will require a zero-knowledge protocol to prove equality of discrete logs in a cyclic group of unknown order (in this case, the cyclic group of squares mod n^2). Specifically, if g is a generator and h is an element of the group, and we are given elements G and H, we want a protocol to prove that the discrete log of G base

g equals the discrete log of H base h. Furthermore, we want this protocol to be a proof of knowledge; that is, the verifier should be convinced that the prover knows the common discrete log.

Chaum and Pedersen [10] were first to provide a non-interactive zero-knowledge proof of discrete log equality that is secure in the random oracle model. Camenisch and Michels [5] gave a knowledge extractor for this protocol which requires only two accepting conversations. It is straightforward to convert the non-interactive version into a Σ-protocol which is secure without the random oracle model assumption: simply replace the hash step with a random challenge sent by the verifier.

6.2 The Paillier Cryptosystem

The Paillier cryptosystem [25] is based on *composite-degree residuosity classes*, and has the desired homomorphic properties. It is based upon the Carmichael lambda function in $\mathbb{Z}_{n^2}^*$ and two useful facts regarding it: for all $w \in \mathbb{Z}_{n^2}^*$,

$$w^{\lambda(n)} = 1 \bmod n, \quad \text{and} \quad w^{n\lambda(n)} = 1 \bmod n^2.$$

Here we recall the cryptosystem.

Key generation. Let $n = pq$ where p, q are primes. Let $g = (1+n)^a b^n \bmod n^2$ for random $a, b \in \mathbb{Z}_n^*$. The public key is (n, g) and the secret key is $\lambda(n)$.

Encryption. To encrypt a message $M \in \mathbb{Z}_n$, randomly choose $x \in \mathbb{Z}_n^*$ and compute the ciphertext $c = g^M x^n \bmod n^2$.

Decryption. To decrypt c, compute $M = \frac{L(c^{\lambda(n)} \bmod n^2)}{L(g^{\lambda(n)} \bmod n^2)} \bmod n$ where the domain of L is the set $S_n = \{u < n^2 \;:\; u = 1 \bmod n\}$ and $L(u) = \frac{u-1}{n}$.

The security of the scheme is based upon the composite residuosity class problem, which is exactly the problem of decrypting a ciphertext. Semantic security can be proven based on the hardness of detecting nth residues mod n^2.

Fouque *et al.* [16] present a threshold version of the Paillier cryptosystem, using techniques developed by Shoup [29] for threshold RSA signatures. The version presented in Fouque *et al.* is provably secure only in the static adversary model, assuming the semantic security of the non-threshold version.

6.3 An Adaptively-Secure Threshold Version

Here we present the novel result of a threshold Paillier cryptosystem which is secure in the adaptive-adversary model, based upon the security of Paillier's cryptosystem and the existence of trapdoor commitment schemes. This cryptosystem is inspired by the statically-secure threshold version presented in Fouque *et al.*

6.3.1 Description of the protocols

Recall $\Delta = l!$, where l is the number of parties.

Key generation. We first describe key generation in terms of an l-party function (recall Definition 2.3) on input k, the security parameter. This function is evaluated by a trusted party, who distributes the proper values to the parties.

Choose an integer n, the product of two strong primes p, q of length k such that $p = 2p' + 1$ and $q = 2q' + 1$, and $\gcd(n, \phi(n)) = 1$. Set $\lambda = 2p'q' = \lambda(n)$. Choose random $(a, b) \leftarrow \mathbb{Z}_n^* \times \mathbb{Z}_n^*$, and let $g = (1+n)^a b^n \bmod n^2$. The secret key is the value $\beta\lambda$ for a random $\beta \leftarrow \mathbb{Z}_n^*$, which is shared additively as follows: for all parties P_i but one, choose random $s_i \leftarrow \mathbb{Z}_{n\lambda}$, and choose the last s_i such that $\sum_{i \in P} s_i = \beta\lambda \bmod n\lambda$. The public key is the triple (g, n, θ), where $\theta = a\beta\lambda \bmod n$. To compute public verification keys, choose a random public square v from $\mathbb{Z}_{n^2}^*$, and let $v_i = v^{s_i} \bmod n^2$. In addition, compute polynomial backups for each s_i as follows: let $a_{i,0} = \Delta s_i$, and choose random $a_{i,j} \leftarrow [-\Delta^2 n^3/2, \ldots, \Delta^2 n^3/2]$, then define a polynomial *over the integers* $f_i(X) = \sum_{j=0}^{t} a_{i,j} X^j$ (so that $f_i(0) = \Delta s_i$). To each party P_j, give the values $f_i(j)$ for all i. Finally, compute public commitments for these backup shares using any perfectly-hiding commitment scheme, such as Pedersen's [26]. Let the public value $w_{i,j}$ be a commitment to $f_i(j)$ under public key k_j and random string $r_{i,j}$, and give $r_{i,j}$ to party P_j.

A result of Cramer *et al.* [11] states that for any l-party function, there is an adaptively-secure protocol which evaluates it. Therefore there is a simulator which, given all the outputs of the function (excluding any values belonging only to P_S), interacts with the adversary and gives it a suitable view of the key generation protocol. In Section 6.3.3 we describe how to compute suitably-distributed inputs to this simulator.

It is worth noting that the key generation protocol provided by Cramer *et al.* may be very inefficient, but it is only executed once to initialize the threshold cryptosystem.

Encryption. To encrypt a message $M \in \mathbb{Z}_n$, pick random $x \leftarrow \mathbb{Z}_n^*$ and compute the ciphertext $c = g^M x^n \bmod n^2$.

Computing decryption shares. Player P_i computes his decryption share $c_i = c^{s_i} \bmod n^2$, and proves via a Σ-protocol that c_i^2 (in base c^2) and v_i (in base v) have the same discrete log s_i in $\mathbb{Z}_{n^2}^*$.

Combining shares. If any party P_i refuses to publish his c_i, or gives an invalid proof, then the other parties reconstruct his secret share s_i as follows. All parties P_j publish their backup shares $f_i(j)$ and random string $r_{i,j}$, and all parties verify that $w_{i,j}$ is a valid commitment. Because there are at least $t+1$ honest parties, each party may pick some $t+1$ honestly-published values $f_i(j)$, and by interpolation, discover $s_i = f_i(0)/\Delta$ and compute $c_i = c^{s_i} \bmod n^2$.

Now each party has a correct value $c_i = c^{s_i} \bmod n^2$, for all i. The message can be computed by each party as follows:

$$\frac{L\left(\prod_{i \in P} c_i\right)}{\theta} = \frac{L\left(c^{\sum_{i \in P} s_i \bmod n\lambda}\right)}{\theta} = \frac{L(c^{\beta\lambda})}{\theta} = \frac{L(g^{\beta\lambda M})}{\theta} = \frac{a\beta\lambda M}{\theta} = M \bmod n$$

since the value $\theta = a\beta\lambda \bmod n$ is part of the public key.

6.3.2 Simulating decryption

Suppose we are given an adversary $\mathcal{A}$ which participates in our threshold cryptosystem. Then we will construct a machine $\mathcal{S}$ which will interact with $\mathcal{A}$ during the decryption protocol, and act on behalf of the honest parties. We want the simulator to provide a view to the adversary that is computationally indistinguishable from a real run. We now describe the operation of the simulator, within the adaptive adversary model, and later prove that it provides an appropriate view to the adversary.

The input to $\mathcal{S}$ is a tuple $(s_P, v_P, \{a_{i,j}\}, \{w_{i,j}\}, g, n, \theta, v, c, M, P_S, k_P, t_S, \mathcal{A})$, defined as follows:

- s_P are simulated secret key shares for all the parties,
- v_P are simulated verification keys for all the parties,
- $\{a_{i,j}\}$ are simulated coefficients defining the polynomials f_i as in the protocol,
- $\{w_{i,j}\}$ are simulated trapdoor commitments for all the parties,
- $\{r_{i,j}\}$ are random strings used to generate the corresponding commitments,
- (g, n, θ) is a simulated public key for the threshold cryptosystem,
- v is a simulated generator of the cyclic subgroup of squares $\bmod n^2$,
- c is the ciphertext to be decrypted,
- M is the decryption of c,
- P_S is the identity of the single inconsistent party,
- k_P are the trapdoor commitment public keys for all parties,
- t_S is the commitment trapdoor for P_S,
- $\mathcal{A}$ is the state of the adversary before the protocol execution.

(In the next section, we describe how these simulated values can be generated from only a public key from the single-server Paillier cryptosystem.)

The simulator acts honestly on behalf of all uncorrupted parties P_i (excluding P_S) by publishing $c_i = c^{s_i} \bmod n^2$ and proving correctness of the decryption shares. On behalf of P_S, the simulator publishes $c_S = (1 + M\theta n) \prod_{i \neq S} c^{-s_i} \bmod n^2$ and provides a false proof of correctness using t_S. If any corrupted party fails to provide a correct decryption share, the simulator honestly interpolates that party's secret share as in the decryption protocol, and proceeds normally. The simulator then honestly computes the plaintext by multiplying the published shares, yielding $(1 + M\theta n) \bmod n^2$, applying L, and dividing by θ to get common output M.

The view of the adversary under the simulation is statistically indistinguishable from a real run of the protocol, provided that all public inputs are suitably simulated. If the adversary corrupts any party P_j (other than P_S), that party's behavior over every invocation of the protocol is consistent with the secret s_j revealed to the adversary. In addition, the adversary is entitled to see $f_i(j)$ and $r_{i,j}$, for all i. When $j \neq S$, the values are consistent with anything else the adversary has seen. For $i = S$, we prove below that with high probability, any set of at most t values $f_S(j)$ is distributed similarly regardless of the value being shared, and therefore the simulated values $f_i(j)$ are statistically indistinguishable from those in a real run.

The simulator only deviates from the real protocol in the behavior of the single inconsistent party P_S. If the adversary corrupts P_S at any point, then the simulator rewinds the simulation to the very beginning (before the key generation phase). Up to the point that P_S is corrupted, the adversary's view is indistinguishable from a real run. Because P_S is a randomly-chosen party, and the number of parties that the adversary may corrupt is at most $t < l/2$, the probability that the adversary chooses to corrupt P_S at any point during the simulation is less than $1/2$. Therefore the expected number of runs of the simulation is less than two, and the simulator runs in expected polynomial time.

6.3.3 Simulating key generation

We now show that the outputs of the key generation function can be simulated (up to statistical closeness), given a public key (g', n) and the identity of the single inconsistent party P_S. (It is sufficient to simulate every value produced by the key generation function, except the secret share s_S belonging to P_S. This is because the entire simulation is aborted if the adversary ever attempts to corrupt P_S, so we need not simulate its private data.) When these values are given to the simulator for the key-generation protocol, it generates a suitable view for the adversary.

Choose random $(x, y, \theta) \leftarrow \mathbb{Z}_n^* \times \mathbb{Z}_n^*$ and let $g = (g')^x y^n \bmod n^2$. Choose random $\alpha \leftarrow \mathbb{Z}_n^*$, and let $v = g^{2\alpha}$. Then for each player, choose random $s_i \leftarrow [0, \ldots, \lfloor n/2 \rfloor - 1]$, and create verification shares $v_i = v^{s_i} \bmod n^2$ for all parties but P_S. For P_S, set

$$v_S = (1 + 2\alpha\theta n) v^{-\sum_{i \neq S} s_i} \bmod n^2.$$

Finally, create commitments $w_{i,j}$ honestly (from polynomials with free terms Δs_i and random coefficients) for all i and j, and random $r_{i,j}$.

First, note that the statistical difference between the uniform distributions on $\mathbb{Z}_{n\lambda}$ and $[0, \ldots, \lfloor n/2 \rfloor - 1]$ is $O(n^{-1/2})$, so any set of at most $l-1$ secret keys s_i is statistically indistinguishable between a real and simulated run. Both g and θ are uniformly chosen from their respective domains, and are identically-distributed with their respective values in the real protocol. In addition, v is a random element of Q_{n^2}, the cyclic group of squares mod n^2. Because $|Q_{n^2}| = pqp'q'$, and $\phi(pqp'q') = (p-1)(q-1)(p'-1)(q'-1)$, v is a generator of Q_{n^2} with high probability, and is identically-distributed with its value in the real protocol.

Note that any set of at most $l - 1$ simulated verification keys v_i is statistically close to a real set. However, in the real protocol with a fixed v, the values of $l - 1$ verification shares induce a distribution upon the last (because the values of $l - 1$ secret shares s_i induce a distribution upon the last). That is, it is necessary and sufficient that $\prod_{i \in P} v_i = v^{\beta\lambda} \bmod n^2$ for some uniformly-chosen β from $\mathbb{Z}_n^*$. In the simulation, we choose $\prod_{i \in P} v_i = v^{\beta\lambda} = (1 + 2\alpha\theta n) \bmod n^2$ without knowing λ but just by randomly choosing θ, which induces a uniform distribution upon β as desired.

Finally, we note that the simulated set $\{w_{i,j}\}$ is identically-distributed to its counterpart in the real protocol, by the perfect-hiding of the commitment scheme.

It remains to be shown that the simulated values $f_i(j)$ for all i and for the adversary's chosen j are indistinguishable from those in a real run. It is clear that the $f_i(j)$ are identically distributed for $i \neq S$, because the simulator behaves honestly. It is also obvious that the points of different polynomials are independent. We therefore show that with high probability, the values $f_S(j)$ seen by the adversary are consistent with a polynomial having free term $\Delta\hat{s}_S$ and coefficients from the proper range, for any value of $\hat{s}_S$.

Let $f_S(X)$ be the polynomial used in the simulation, that is, $f_S(X) = \Delta s_S + \sum_{i=1}^{t} a_{S,j} X^j$ where the $a_{S,j}$ are randomly chosen. Say that the adversary has corrupted a set of parties C, with $|C| \le t$. We wish to find a polynomial $\hat{f}_S(X)$ such that $\hat{f}_S(0) = \Delta\hat{s}_S$ for an arbitrary $\hat{s}_S$, and $\hat{f}_S(i) = f_S(i)$ for $i \in C$. Consider a polynomial $h(X)$ such that $h(0) = \Delta(\hat{s}_S - s_S)$, and $h(i) = 0$ for $i \in C$. Then we have $\hat{f}(X) \equiv f(X) + h(X)$. By interpolation,

$$h(X) = \sum_{i \in C} h(i) \cdot \prod_{j \neq i, j \in \{0\} \cup C} \frac{z - j}{i - j} = \Delta(\hat{s}_S - s_S) \prod_{j \in C} \frac{z - j}{-j}$$

and by expanding the product, the coefficient of X^i in $h(X)$ is:

$$\Delta(\hat{s}_S - s_S) \sum_{B \subseteq C, |B| = i} \frac{\prod_{j \in B}(-j)}{\prod_{j \in C}(-j)} \in \mathbb{Z}$$

which is bounded in absolute value by

$$\sum_{B \subseteq C, |B| = i} \Delta(\hat{s}_S - s_S) \le \Delta(\hat{s}_S - s_S) \binom{t}{i} \le \frac{\Delta(\hat{s}_S - s_S) t!}{i!(t-i)!} \le \Delta(\hat{s}_S - s_S) t! \le \Delta^2 n^2 / 2$$

since $\hat{s}_S, s_S \in \{0, \ldots n^2/2\}$.

Now the coefficients of $\hat{f}(X)$ are outside of the desired range only if any of the coefficients of $f(X)$ are outside of $[-\Delta^2(n^3 - n^2)/2, \ldots, \Delta^2(n^3 - n^2)/2]$. By the union bound, this happens with probability at most t/n, which is negligible. In addition, there is a bijection between the coefficients of f and the coefficients of $\hat{f}$ when $s_S, \hat{s}_S$, and C are fixed. Therefore the distribution of the coefficients of $\hat{f}$ is statistically close to uniform, as desired.

6.4 A Reduction from the Original Cryptosystem

With these simulations in hand, the reduction from one-server semantic security to threshold semantic security is straightforward. Assume there is an adversary that can break the security of the threshold cryptosystem. Given a public key (g', n) for the single-server Paillier cryptosystem, we first simulate the key generation protocol and any decryptions as described above. (Recall that the public key of the threshold cryptosystem is $(g = (g')^x y^n \bmod n^2, n, \theta)$ for some uniformly-chosen x, y, θ.) The adversary then outputs two messages m_0, m_1, which we send to an oracle, who responds with a random encryption c of m_b for some random bit b. We compute $\chi = c^x \bmod n^2$ (where x is the value chosen by the key generation simulator) and give it to the adversary. By assumption, the adversary can distinguish with non-negligible advantage whether χ is an encryption of m_0 or m_1 under (g, n, θ). This is equivalent to whether c is an encryption of m_0 or m_1 under (g', n), hence we have broken the semantic security of the original cryptosystem. This completes the reduction.

Bibliography

[1] D. Beaver. Secure multi-party protocols and zero-knowledge proof systems tolerating a faulty minority, 1991.

[2] G. R. Blakley. Safeguarding cryptographic keys. In Richard E. Merwin, Jacqueline T. Zanca, and Merlin. Smith, editors, *1979 National Computer Conference: June 4–7, 1979, New York, New York*, volume 48 of *AFIPS Conference proceedings*, pages 313–317, Montvale, NJ, USA, 1979. AFIPS Press.

[3] F. Boudot. Efficient proofs that a committed number lies in an interval, 2000.

[4] Jan Camenisch and Anna Lysyanskaya. An identity escrow scheme with appointed verifiers. In *Proc. CRYPTO 2001, to appear*, 2001.

[5] Jan Camenisch and Markus Michels. A group signature scheme based on an RSA-variant. Technical Report RS-98-27, BRICS, November 1998.

[6] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, Winter 2000.

[7] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In Michael Wiener, editor, *Advances in Cryptology – CRYPTO ' 99*, Lecture Notes in Computer Science, pages 98–115. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany, August 1999.

[8] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited, March 1998. 1st (draft) version.

[9] Dario Catalano, Rosario Gennaro, and Shai Halevi. Computing inverses over a shared secret modulus. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 2000.

[10] D. Chaum and T. Pedersen. Wallet databases with observers.

[11] Cramer, Damgard, Dziembowski, Hirt, and Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1999.

[12] Ronald Cramer, Ivan Damgaard, and Jesper B. Nielsen. Multi-party computations from threshold homomorphic encryption. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 2001.

[13] Ronald Cramer and Victor Shoup. Signature schemes based on the strong RSA assumption. December 1998.

[14] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Giles Brassard, editor, *Advances in Cryptology – CRYPTO ' 89*, Lecture Notes in Computer Science, pages 307–315, Santa Barbara, CA, USA, 1990. International Association for Cryptologic Research, Springer-Verlag, Berlin Germany.

[15] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *ACM Symposium on Theory of Computing*, pages 409–418, 1998.

[16] P. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries, 2000.

[17] Y. Frankel, M. Yung, and P. MacKenzie. Adaptively-secure distributed public-key systems. *Lecture Notes in Computer Science*, 1643:4–??, 1999.

[18] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In *EUROCRYPT: Advances in Cryptology: Proceedings of EUROCRYPT*, 1999.

[19] S. Goldwasser and L. Levin. Fair computation of general functions in presence of immoral majority. In A. J. Menezes and S. A. Vanstone, editors, *Proc. CRYPTO 90*, pages 77–93. Springer-Verlag, 1991.

[20] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2), April 1984.

[21] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proc. 17th ACM Symp. on Theory of Computing*, pages 291–304, Providence, 1985. ACM.

[22] Stanislaw Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptography: Introducing concurrency, removing erasures (extended abstract). In Bart Preneel, editor, *Advances in Cryptology – EUROCRYPT ' 2000*, Lecture Notes in Computer Science, pages 221–242, Brugge, Belgium, May 2000. Springer-Verlag, Berlin Germany.

[23] S. Micali and P. Rogaway. Secure computation. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 392–404. Springer, 1992.

[24] NIST. The digital signature standard, proposed by NIST. *CACM: Communications of the ACM*, 35, 1992.

[25] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. *Lecture Notes in Computer Science*, 1592:223–??, 1999.

[26] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 129–140. Springer, 1992. Lecture Notes in Computer Science No. 576.

[27] R. L. Rivest, A. Shamir, and L. Adelman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978. The basics of trap-door functions and the famous RSA public key cryptosystem are presented in this paper.

[28] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, 1979.

[29] Victor Shoup. Practical threshold signatures. In *Theory and Application of Cryptographic Techniques*, pages 207–220, 2000.

[30] Andy Yao. Protocols for secure computation. In IEEE, editor, *23rd annual Symposium on Foundations of Computer Science, November 3–5, 1982, Chicago, IL*, pages 160–164, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1982. IEEE Computer Society Press.