Efficient Threshold Cryptosystems

by

Stanisław Jarecki

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

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2001

Author .....
Department of Electrical Engineering and Computer Science
April 20, 2001

Certified by .....
Shafi Goldwasser
Professor of Computer Science
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Departmental Committee on Graduate Students

Efficient Threshold Cryptosystems

by

Stanisław Jarecki

Submitted to the Department of Electrical Engineering and Computer Science
on April 20, 2001, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

A threshold signature or decryption scheme is a distributed implementation of a cryptosystem, in which the secret key is secret-shared among a group of servers. These servers can then sign or decrypt messages by following a distributed protocol. The goal of a threshold scheme is to protect the secret key in a highly fault-tolerant way. Namely, the key remains secret, and correct signatures or decryptions are always computed, even if the adversary corrupts less than a fixed threshold of the participating servers.

We show that threshold schemes can be constructed by putting together several simple distributed protocols that implement arithmetic operations, like multiplication or exponentiation, in a threshold setting. We exemplify this approach with two discrete-log based threshold schemes, a threshold DSS signature scheme and a threshold Cramer-Shoup cryptosystem. Our methodology leads to threshold schemes which are more efficient than those implied by general secure multi-party computation protocols. Our schemes take a constant number of communication rounds, and the computation cost per server grows by a factor linear in the number of the participating servers compared to the cost of the underlying secret-key operation.

We consider three adversarial models of increasing strength. We first present distributed protocols for constructing threshold cryptosystems secure in the *static* adversarial model, where the players are corrupted before the protocol starts. Then, under the assumption that the servers can reliably erase their local data, we show how to modify these protocols to extend the security of threshold schemes to an *adaptive* adversarial model, where the adversary is allowed to choose which servers to corrupt during the protocol execution. Finally we show how to remove the reliable erasure assumption. All our schemes withstand optimal thresholds of a minority of malicious faults in a realistic partially-synchronous insecure-channels communication model with broadcast.

Our work introduces several techniques that can be of interest to other research on secure multi-party protocols, e.g. the *inconsistent player* simulation technique which we use to construct efficient schemes secure in the adaptive model, and the novel primitive of a *simultaneously secure* encryption which provides an efficient implementation of private channels in an adaptive and erasure-free model for a wide class of multi-party protocols.

We include extensions of the above results to: (1) RSA-based threshold cryptosystems; and (2) stronger adversarial models than a threshold adversary, namely to proactive and creeping adversaries, who, under certain assumptions regarding the speed and detectability of corruptions, are allowed to compromise all or almost all of the participating servers.

Thesis Supervisor: Shafi Goldwasser
Title: Professor of Computer Science

Acknowledgments

I would like to thank my advisor, Shafi Goldwasser, for supporting my studies and my research, for her advice and guidance throughout, for supervising this thesis, and in particular for asking me to write this material in a more formal way than I initially intended. Even though it took longer to write it, in this way my work has become more sound and intellectually satisfying, and hopefully also more useful to the research community. Secondly, I want to thank Hugo Krawczyk, who co-supervised this thesis, with whom I had a pleasure of working on several research projects which have contributed to it, and who have always given me judicious advice and much encouragement.

I want to thank all other scientists with whom I have worked on research projects in cryptography. My thesis is based on results of many of these collaborations. I was lucky to be learning from them and to share with them the camaraderie of doing a good work together. Thus my gratitude, respect, and warm greetings go to Ran Canetti, Rosario Gennaro, Amir Herzberg, Markus Jakobsson, Anna Lysyanskaya, Andrew Odlyzko, Tal Rabin, and Moti Yung.

It is also my great pleasure to thank all other researchers, both professors and students, whose friendship and good advice have given me support during my studies, and whose expertise and excitement about cryptography have been an inspiration and a source of ideas to me. Thus I would like to thank Yevgeniy Dodis, Yael Gertner, Shai Halevi, Ari Juels, Tal Malkin, Silvio Micali, Daniele Micciancio, Zulfikar Ramzan, Leo Reyzin, Ron Rivest, Adam Smith, and Salil Vadhan. I am afraid that this short list unfairly omits some people, whom I hereby ask for forgiveness.

I want to gratefully acknowledge the help I received from Jim Horning, the director of the Strategic Technology and Research Lab at InterTrust Technologies, and from the whole STARLab team, for allowing me to spend the first four months at STARLab on completing the write-up of this thesis. Similarly, I want to thank the Institute of Foundations of Computer Science in Warsaw (IPI PAN), and in particular Dr. Marian Srebrny, who hosted me for a month while I was working on this write-up in Poland.

I want to thank all my dear Boston friends, who cheered me on and shared with me the joys and challenges of a graduate student life at MIT.

Finally, my biggest gratitude goes to my dear mother, to my loving sister Dorota, and to Stephanie Malloy, who gave me an invaluable support during my last year at MIT. I dedicate this work to them.

Contents

List of Figures

Chapter 1

Introduction

1.1 Threshold Cryptosystems: Exposition

In a standard public-key cryptosystem the secret-key operation is performed by a single server that stores the secret key. We will call such implementations of a secret-key operation a *centralized* cryptosystem. Since public-key cryptosystems are used to guarantee some level of protection against dishonest behavior of some entities, such dishonest entities will try to attack this single server in an attempt to steal or to destroy the secret key, and thus destroy the protection established by this cryptosystem. The idea of a *threshold* cryptosystem is to remove this single point of failure by distributing the secret-key operation among a group of servers. The goal of a threshold cryptosystem is to design this distributed implementation of the secret-key operation in such a way that the cryptosystem remains secure in the presence of adversaries that can break into, spy on, disconnect, crash, or in any other way corrupt some of the participating servers, as long as the number of such corrupted servers is smaller than certain threshold.

In this work we consider two types of public-key cryptosystems: encryption schemes and signature schemes. In the first the secret key is used to decrypt, in the latter to sign. Both operations can be fault-tolerantly distributed, the first with a threshold signature scheme, the latter with a threshold decryption scheme. We will refer to either one as simply a *threshold scheme.*

A threshold signature scheme replaces the single server, who is the sole holder of the secret key and who performs the signature operation, with a group of servers, often referred to as *players* in a threshold scheme. This group of servers together "holds" the secret key in such a manner that they can, as a group, produce signatures even if some of these servers are controlled by an adversary and diverge from the prescribed protocol in an arbitrarily malicious way. There are two security objectives that threshold signature schemes must achieve: i) A threshold scheme must be *robust*, i.e. a correct signature must be efficiently produced as long as the subset of the corrupted servers stays below a certain threshold. ii) Furthermore, under the same condition, the underlying signature scheme must remain *unforgeable*, i.e. the adversary who corrupts less than a specified threshold of servers and participates in the signature protocol, invoked on the messages that the adversary adaptively chooses, remains unable to forge signatures.

Similarly for the encryption schemes, a secret key can be held in a distributed way by a

group of servers which perform a decryption operation via a threshold decryption protocol. The adversary who corrupts less than a certain threshold of the servers can neither disrupt the remaining servers from producing correct decryptions nor can he break the semantic security of the underlying encryption scheme.

In a (t, n)-threshold scheme, n servers maintain the private key and participate in the private key operation. The adversary needs to break at least $t + 1$ of the servers to be able to forge signatures (respectively, decrypt messages), or to stop the system from efficiently producing valid signatures (respectively, decryptions). Thus the threshold schemes essentially provide fault-tolerant protection of *secrecy* and of *availability* of the secret key, in the presence of computer break-ins or any other local computer faults

Attempts at Simple Solutions. We can elucidate the functionality of threshold schemes by examining a few straightforward attempts at achieving their desirable properties.

Attempt 1: For example, a replication of the secret key among a group of servers would protect its availability in the presence of computer break-ins, i.e. the production of correct decryptions or signatures would be assured as long as at least one of the servers remained uncorrupted, but such replication would obviously weaken the protection of the secrecy of the key.

Attempt 2: A better attempt would be to use standard secret sharing to distribute the secret key among n players, and whenever the secret-key operation needs to be performed, to reconstruct this key at some server, perform the reconstruction, and erase the key, so that only the shares of the key persist. This solution would still not remove a single point of failure. The adversary learns the secret key if he corrupts this one server at the time the secret-key operation is performed.

Attempt 3: A much closer approximation to the functionality of an $(\lfloor \frac{n-1}{2} \rfloor, n)$-threshold *signature* scheme could proceed as follows. Let's use any regular signature scheme as a black-box, and let each of the n servers create its own signing key and public key pair (let's call them "partial keys"), and let the set of all the n public keys be the public key of the new scheme. We define a valid signature of the new scheme as a collection of valid signatures under more than $\lfloor \frac{n-1}{2} \rfloor$ of these partial keys. If the underlying signature scheme is secure, such scheme would be both robust and unforgeable as long as the adversary does not corrupt more than the majority of the servers. One can also propose a similarly fault-tolerant encryption scheme. Every server has its own instance of a standard encryption scheme, and a ciphertext is formed by first encrypting a message under some symmetric key, then secret-sharing this key and encrypting each share under a public key of each server. The ciphertext is a concatenation of all these ciphertexts.

A clear drawback of the above proposals is that the size of the public key and of each signature (resp., ciphertext), as well as the time to verify each signature (resp., encrypt a message), grows by the factor of n. This would be too inefficient for many applications, e.g. in wireless communication.

Moreover, this attempt at approximating the functionality of threshold schemes does not provide *transparency* to the system users. If someone wanted to use the above methods to add fault-tolerance to an existing standard centralized cryptosystem, they would have to change the public key of the cryptosystem, and also change the signature verification (resp., message encryption) software of every system user, since the public-key operations of the new schemes are non-standard. Furthermore, in the case of a signature scheme, they

would have to either make all the previous signatures invalid or they would have to re-sign them. Such global operation would be both costly and difficult to carry out securely. In contrast, a threshold scheme can add fault tolerance to existing cryptosystems in a way which is transparent for the system users. The signature (resp., decryption) output by the group of servers will be the same as in the standard centralized scheme. In any system that utilizes a signature (resp., decryption) service one can replace the server that implements the secret-key operation with a threshold scheme, and no other element of the system needs to change.

A Complex Solution: Secure Multi-Party Computation. Threshold schemes are actually implied by the general results on *secure multi-party computation* (MPC), which provide protocols for computing any function, including signature and decryption operations of any public-key cryptosystem, in a threshold setting.[1] Since the introduction of secure multi-party computation by [Yao82, GMW87], there appeared many MPC solutions, e.g. in the works of [BGW88, CCD88, RB89, BMR90, Bea91, MR91, BH92, BOCG91, CFGN96, CDD+99, Can00]. These solutions were increasingly more efficient, and provably secure in increasingly stronger computational, communicational and adversarial models.

The generality of the above solutions implies certain inefficiencies when such protocols are used to implement threshold cryptosystems for schemes like DSS, Cramer-Shoup, or RSA. Namely, all the above solutions are based on protocols that compute a single arithmetic or Boolean gate in a threshold setting. Such protocols are then composed to compute *any* arithmetic or binary circuits in a threshold setting. Thus the complexity of the resulting threshold protocols depends on the complexity of the circuit representation of the computed function. If we look at two particular efficiency measures of a distributed protocol, namely the number of communication rounds and the computation cost of each participating player, then all the above general solutions except [BMR90] imply threshold DSS, Cramer-Shoup, and RSA schemes with non-constant number of rounds, while [BMR90] implies schemes in which every player performs a very high number of cryptographic operations. Such costs are prohibitive for the applications of threshold schemes that we consider.[2]

1.2 Contribution of This Thesis

This thesis shows how efficient threshold schemes can be built from distributed protocols that fault-tolerantly implement simple arithmetic operations on secret-shared data. We exemplify this methodology by constructing a threshold DSS signature scheme, which involves adding, multiplying, exponentiating, and inverting secret-shared data. We also show that such techniques can be applied to other discrete-log based schemes, like a Cramer-Shoup cryptosystem, or to schemes working over composite moduli, like RSA.

Our techniques lead to efficient threshold schemes. In a realistic partially-synchronous insecure-channels point-to-point communication model with broadcast,[3] each execution of

[1] See Appendix D for an explanation of how threshold schemes can be implemented with general MPC protocols.

[2] See Appendix D for a more detailed discussion of the costs of implementing threshold cryptosystems with general MPC protocols.

[3] See Chapter 2 for a discussion of the communication model.

our threshold key generation, signature generation, or decryption protocol takes a small constant number of communication rounds (which can often be further reduced if some work is performed off-line, for example to only one round in the case of threshold DSS), and bears the following amortized costs per each participating player. Each player broadcasts $O(nk)$ bits and privately sends to other servers $O(nk)$ bits, where n is the number of participating players, and k is the security parameter. The local computational cost of each player is $O(nk)$ modular k-bit multiplications if $k = \Omega(n^2 \log n)$.[4] Furthermore, unlike in the case of the general MPC protocols, the cost of implementing private channels for our threshold schemes in the adaptive erasure-free model is comparable to the cost of a conventional public-key encryption scheme.[5]

The methodology and the results presented in this thesis summarize the contributions to threshold cryptography that were developed in a series of works written by Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, Tal Rabin, and Ran Canetti [GJKR96b, GJKR96a, GJKR99, CGJ+99], and in a recent work written by Stanisław Jarecki and Anna Lysyanskaya [JL00]. We also present the extensions to proactive security created by Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, Moti Yung, and Markus Jakobsson [HJKY95, HJJ+97].

1.2.1 Our Approach: Protocol Building Blocks for Arithmetic Operations

Our main contribution is a presentation of distributed protocols from which efficient secure threshold cryptosystems can be built. Since algorithms of many cryptosystems can be broken down to a series of *simple arithmetic operations*, our methodology is to construct distributed protocols that implement such arithmetic operations in a threshold setting, and then to put together such building-block protocols to construct threshold key generation, signing, or decryption protocols for various signature schemes or cryptosystems. In particular, we show how to use our building-block protocols to construct two discrete-log based schemes: a complete threshold DSS signature scheme [NIS91] (i.e. threshold DSS key generation and threshold DSS signature generation), and a complete threshold Cramer-Shoup cryptosystem [CS98] (i.e. threshold Cramer-Shoup key generation and threshold Cramer-Shoup decryption). This should be contrasted with the general secure multi-party computation methodology, which is to break down the circuit that computes a cryptographic operation into gates, and give a protocol for computing *each gate* in a threshold setting. This "arithmetic-operation-based" rather than "circuit-based" approach to finding efficient protocols for useful threshold schemes is often referred to as *threshold cryptography*. It started with the work of [Des88, Boy89, CH89, DF89, Fra89] who looked for ways to fault-tolerantly compute an exponentiation operation without recoursing to general secure computation protocols. (For an early survey on threshold cryptography see also [Des94].)

The departure from the per-gate approach of the general multi-party computation protocols is of major consequence for the techniques and the efficiency of threshold cryptosystems. In particular, it is worth noting that the building block subprotocols we provide are *not* secure multi-party computation protocols for any circuit. Yet, as we show in this thesis, these subprotocols can be composed into provably secure threshold schemes.

[4] The exact number of modular multiplications is $O(nk + n^2 \min(n \log n, k))$.

[5] See Section 2.2 for an explanation of the adaptive and erasure-free adversarial models.

We can explain our methodology on the example of a version of an ElGamal signature scheme. Let p be a prime and g a generator of the multiplicative group $\mathbb{Z}_p{}^*$. The ElGamal key generation is composed of two steps: (1) picking a secret key $x \in [1,p]$ at random; and (2) computing the public key $y = g^x \bmod p$. The generation of a signature on message $m \in [1,p]$ is then a series of the following steps: (3) picking a random temporary secret $k \in [1,p]$; (4) computing a public value $r = g^{1/k} \bmod p$; and (5) computing public value $s = k(m + xr) \bmod (p-1)$ and outputting (r,s) as the signature on m.[6]

The *threshold* ElGamal key generation protocol that can be constructed following our methodology would be a composition of the following distributed protocols: (A) a protocol to pick a *shared* secret x at random in $[1,p]$; and (B) a protocol to exponentiate modulo p a known value g to a shared secret x. A threshold ElGamal signature generation protocol needs three more building block protocols: (C) a protocol to exponentiate modulo p a known value g to an *inverse* of a shared secret k; (D) a protocol to create a sharing of a secret $(m + xr) \bmod (p-1)$ from public values m and r and from the sharing of secret x; and (E) a protocol to multiply two shared secrets, k and $(m + xr)$, modulo $p - 1$. Each of these protocols must guarantee that the shared secret values are indeed secret, and that the correct outputs are computed, even if t of the n participating servers are corrupted and behave in an arbitrarily malicious way.

1.2.2 Organization of the Thesis: Adversarial Models Considered

All our schemes are secure in a realistic partially-synchronous insecure-channels communication model. They all tolerate adversaries who can corrupt up to a minority of the participating players, which is the maximum number of faults a threshold scheme can tolerate,[7] and who can make the corrupted players behave in an arbitrarily malicious way.[8]

This thesis is divided into three major parts, each of which present separate techniques to handle the following three *variants* of the general adversarial model described above, in the order of increasing adversarial power and increasing technical difficulty of designing protocols that are provably secure against them.[9] In each part we present protocols that securely implement basic arithmetic operations in a given variant of the threshold setting. Furthermore, we exemplify the usefulness of these building-block protocols by putting them together into a threshold DSS signature scheme and proving its security in the given setting.

- In Chapter 4 we provide techniques for protocols that are secure if the malicious adversary is modeled as a *static* adversary, which means that the subset of t players the adversary can corrupt must be chosen before the execution of the protocol. This chapter is based on material previously published in [GJKR96b], [GJKR99], [CGJ+99] and [JL00].

[6] This version of ElGamal is a similar to DSS. Compare Section 4.3.1.

[7] It is easy to why threshold schemes cannot handle more than a minority of corrupted players. If $n \leq 2t$ and t corrupted players refused to participate, for example in the threshold signing protocol, then the remaining t players could not perform the signature operation on their own. If they could, the t-threshold adversary who corrupted *those* t players would learn how to sign messages on his own. Thus if $n \leq 2t$ then a threshold scheme would be either not robust or not unforgeable in the presence of a t-threshold adversary.

[8] See Chapter 2 for a detailed discussion of our adversarial setting.

[9] In Section 2.2 we describe these three variants of the threshold adversarial model in detail.

- In Chapter 5 we provide additional techniques to extend the security of protocols presented in Chapter 4 to a strictly stronger model where the malicious adversary is modeled as an *adaptive* adversary, i.e. where he is allowed to choose which servers to corrupt during the protocol execution. These techniques, however, enable efficient threshold schemes under an additional assumption of the ability of servers to reliably erase their local data. This chapter is based on material previously published in [CGJ+99] and [JL00].

- Finally, in Chapter 6 we show how to remove the assumption of reliable erasure of local data. Thus we provide threshold schemes that are secure in *adaptive erasure-free* adversarial model. This chapter is based on material previously published in [JL00].

Furthermore, we provide some important extensions of the above results:

- In Appendix A we show that the building-block protocols presented in Chapters 4-6 can be put together to create a threshold Cramer-Shoup cryptosystem. Here we use the material previously published in [JL00].

- In Appendix B we show that our distributed protocols can be used, under the assumption of reliable data erasure, to construct schemes that are secure against strictly stronger attacks than those modeled by a threshold adversary. Namely, instead of a threshold adversary, who can corrupt no more than t servers throughout the system operation, we can tolerate adversaries, who, under certain assumptions on the speed of corruptions and their detectability by the remaining servers, are allowed to compromise all or almost all of the participating servers. We provide two formalizations of such increased adversarial strength: i) A *mobile* adversary, who can choose a different subset of t corrupted servers each day (or any other pre-defined time period); and ii) A *creeping* adversary, who can permanently corrupt all players except of two, but his corruptions are detected fast enough for the remaining players to run certain protective "system-healing" protocols. In this appendix we build on the material previously published in [HJKY95] and [HJJ+97].

- In Appendix C we describe how our techniques can be applied to cryptosystems working over composite modulus, and thus generate efficient adaptive and erasure-free threshold RSA signature (or decryption) schemes. We note however that the resulting RSA schemes rely on a single trusted party during the key generation phase. This appendix is based on material published in [CGJ+99] and [JL00].

For reader's orientation we quickly overview the remaining chapters of this thesis:

- In Chapter 2 we explain our communication and adversarial model in detail.

- Chapter 3 contains some preliminary definitions and important notational conventions we adopt in the technical part of this thesis.

- Chapter 7 describes certain initialization protocols, i.e. distributed generation of an instance of a discrete-log problem or an instance of a Pedersen commitment scheme, which enable the threshold protocols discussed in Chapters 4-6.

1.2.3 Contributions to Research on Secure Protocols

We consider the most important contribution of this thesis to be our framework for expressing security properties of threshold protocols in a way which allows for modular composition of such protocols into larger threshold schemes. We show that a secrecy property of a protocol $\mathcal{P}_{op}$ which implements some arithmetic operation op in the threshold setting can be phrased in terms of an existence of a simulator $\mathsf{SIM}_{\mathcal{P}_{op}}$ which simulates an execution of $\mathcal{P}_{op}$ *and faciliates a simulation of any subsequent protocol that executes on the outputs of* $\mathcal{P}_{op}$. We show furthermore that once such properties are established for some protocols $\mathcal{P}_{op_1}$ and $\mathcal{P}_{op_2}$ then they imply a corresponding property about a *sequential composition* of these two protocols, which shows that such composition properly implements a composition of operations op_1 and op_2 in a threshold setting.

Apart of validating in this way the "arithmetic building-block" approach to secure multi-party computation, our work provides several other specific contributions to research on secure multi-party protocols:

- We show that composition of simple distributed protocols into larger protocols must be done carefully. For example, we show that a simple and widely known distributed key generation protocol for discrete–log based schemes, first proposed in [Ped91b], is insecure. We also show how to replace it by a similarly simple but secure protocol. This is the subject of Section 4.2 in Chapter 4.

- We present the *single inconsistent player* simulation technique of [CGJ+99, FMY99b], which enables efficient adaptively-secure protocols for a threshold computation of an exponentiation function, and thus enables efficient adaptively-secure threshold cryptosystems. This technique is likely to be applicable in the design and analysis of other adaptively-secure protocols. We explain it in Chapter 5.

- We present the first efficient implementation of private channels in the adaptive and erasure-free model, for a wide class of multi-party protocols which includes our threshold DSS, Cramer-Shoup, and RSA schemes. We achieve this by identifying a novel class of *simultaneously secure* public-key encryption schemes, a notion which is stronger than semantic security but weaker than the non-committing encryption proposed by [CFGN96] to implement private channels in the adaptive and erasure-free model for *general* multi-party protocols. In contrast to the best currently known non-committing encryption implementations, we provide an implementation of a simultaneously secure encryption whose efficiency is comparable to a regular encryption scheme. This novel encryption primitive was introduced by [JL00], and we explain it here in Chapter 6.

1.3 Applications of Threshold Schemes

1.3.1 Fault Tolerance

Whenever the security of a system depends on the inability of a malicious adversary to learn or destroy some secret key, there is a need for a threshold scheme to maintain such key in a fault-tolerant way.

A good example of a system whose security relies on the secrecy and availability of certain root secret key is a Key Certification Authority. All security properties of a public-key infrastructure depend on the authenticity of KCA's signatures on users' public-key credentials. Consequently, a KCA is the primary target for attacks on a public-key infrastructure. Whoever breaks into the KCA server and learns its secret key can impersonate any user. Another example of a central signature key that gives high rewards to someone who learns it, is a secret key of a bank in an e-cash (or any electronic commerce) scheme. Moreover, one needs to protect not only the secrecy but also the availability of such sensitive keys. If a key is lost, a new (public,secret) key pair must be picked, the new public key must be securely redistributed, and all the existing signatures need to be re-signed. This operation would not only be costly, but it would also lead to further security threats.

On the other hand, it is difficult and costly to make a single computer truly attack-resistant. First of all, computers often fail accidentally, because of software faults or power outages, or they "fail" in a planned way, e.g. when they need to be shut down for maintenance. Moreover, if a computer provides a sensitive service, e.g. a signature service played by a KCA, it is a potential target of attacks. The adversary might try to learn the secret signature key by infiltrating the signing server with a virus, or he might be interested in disabling the service from performing its function, and plant a virus which would change the secret key of the signing server or otherwise modify its signing algorithm. Even if the signing server was exceptionally secure against virus intrusions, an adversary could still break into it by a physical intrusion or even by bribing the personnel that maintains this server.

Sensitive secret-key decryption services could also be subject to such attacks. Borrowing an example from [CG99], we can imagine that mail to any member of some organization is encrypted under a single public key of that organization. If the corresponding decryption key is stored on a single server, that server would constitute a single point of failure in the system and would be a subject to similar attacks as those described above. Therefore, the decryption key can be held instead in a distributed way by a group of servers, and each message can be decrypted via a threshold decryption scheme. (Note that in this application the output of threshold decryption should be held only by the final addressee of the message.) In such way, the secrecy and the availability of the secret decryption key of this organization can be protected in a highly fault tolerant way. The adversary would have to break into more than a certain threshold of the servers in order to read the encrypted messages or to destroy the secret key of the organization. There are other applications that can use a threshold decryption scheme to gain fault-tolerance against computer break-ins and break-downs, for example the optimistic exchange of secrets [Mic99] and secure data storage [GGJR97].

Implementation Considerations. The threshold approach relies on the assumption that the difficulty of corrupting servers grows with the number of corruptions. This is not always the case, for example if the administration of the servers makes it trivial to corrupt all of them after corrupting the first one. However, we can increase the resistance of a threshold scheme against such single points of failure by a careful administration of the servers. Indeed, a threshold scheme should never be the only security element of a fault-tolerant system. The servers that participate in a threshold scheme should be additionally protected with intrusion-detection software, firewalls, back-ups, and physical access control

mechanisms.

One such protective measure which could be viewed not as an enhancement but an alternative to threshold schemes is *key delegation*. We explain this method as follows. In some applications the authority of the root signature key, e.g. the signature key of a KCA, can be delegated to a temporary signature key which for some time period produces signatures on behalf of the root key. The root key must previously sign a certificate that includes the verification part of the temporary key and the time period in which its signatures are valid. The signatures of the temporary key must be accompanied by this certificate.

Compared to threshold schemes, key delegation has some advantages. Namely, compared to the off-line storage of the root signature key in the key delegation mechanism, the on-line threshold distribution of this key introduces some vulnerability to the system by complicating the interface of a server that stores the root key. With the key delegation method this interface can be tightly controlled, for example by elaborate human-access control procedures [Ve00], because the root key needs to be used only once a certain time period, e.g. once a day, to sign the certificate of the temporary signature key. In contrast, distribution of the secret key via a threshold scheme requires that the participating servers be connected via a network, and that they respond to on-line requests for signatures from untrusted users, which opens the signature service up to more attacks. On the other hand, key delegation has also some disadvantages compared to threshold schemes. Namely, it has several single points of failure. For example, an adversary that can break into an on-line server can disable the on-line temporary key, or learn it and forge signatures for the time limited by its certificate.

However, one can combine the benefits of the two methods, because each single point of failure in the key delegation method can be made more secure via threshold cryptosystems. The tolerance to on-line attacks can be increased by distributing the temporary signature key among a group of Internet servers and implementing the signing operation via a threshold signature scheme. Fault-tolerance can also be added to the storage of the root key by implementing it with a threshold scheme over a local area network built in some protected "vault". Moreover, threshold cryptography can help maintain the key that controls physical access to this vault. We can imagine that the door to the vault is equipped with an electronic lock and that its key is distributed in a threshold way onto hand-held devices connected with a wireless network, each device held by one employee of the company that maintains the signature service. To open the door, these devices must then participate in a threshold signature protocol to authenticate themselves to the door lock and authorize an opening of the vault.

1.3.2 Division of Trust

In addition to achieving fault-tolerance by distributed implementation of traditionally centralized services, threshold cryptosystems can also be used for "naturally distributed" applications, in which a group or an organization decides that some function should be performed on its behalf, only if more than some threshold of the group members agree to it. Such arrangement in the organization reflects the apportioning of trust within the group. Thus we might have an organization where no single member is free to decide to perform an operation, but a group of more than a certain threshold of members is entrusted to do so.

A threshold scheme can also be used to alleviate the need of trust in systems that use a trusted third party, but where there is no single entity that all (or even most) system users would trust to implement this third party in an honest and disinterested way. Instead of a single authority, the trusted third party can be implemented by a group of authorities who would maintain the required secret-key service via a threshold scheme. Mistrustful users would be assured that the service is not abused, provided that they believed that no more than a threshold of this group of authorities is dishonest. Threshold decryption schemes are used in this way in multi-authority election schemes [CF85] and in key escrow [Mic92].

1.3.3 Importance of Threshold Schemes for Standard Systems

We described above the applications of threshold signatures (KCA, bank's key, organization key) and threshold decryption (gateway decryption, secure databases, multi-authority election, key escrow). Most of these applications could utilize any (threshold) signature or public-key encryption scheme. However, if possible, it is preferable to base them on standard schemes like DSS signature [NIS91] and RSA signature and encryption [RSA78], because the security properties of these two schemes are well established and accepted. Furthermore, these two schemes are already used by Key Certification Authorities in Public-Key Infrastructures, and this creates a specific need for upgrading *their* secret-key operations with threshold schemes, i.e. with threshold protocols for DSS or RSA signature. The recent cryptosystem of Cramer-Shoup [CS98] is potentially also quite important, because its efficiency is comparable (it is worse by a small constant factor) to that of RSA (it is similar to the efficiency of ElGamal), but it is proven secure against adaptive chosen message attack, the strongest standard notion of security for encryption schemes. Consequently, in this thesis we chose to exemplify our methodology with a construction of threshold schemes for these specific public-key systems. We construct a threshold DSS signature scheme and a threshold Cramer-Shoup cryptosystem, and we explain how to use our building-block protocols in a threshold RSA.[10]

1.4 History of Research on Threshold Cryptosystems

As we discussed in Sections 1.1 and 1.2, threshold schemes are implied by general secure multi-party computation results.[11] However, the protocols such general results imply for specific threshold schemes like DSS or RSA are too computationally expensive to be practical. In response to this drawback, the work of [Des88, Boy89, CH89, DF89, Fra89] started the field of *threshold cryptography*, namely a design of efficient protocols for threshold computation of *specific arithmetic operations*, (and compositions of such operations), which eventually led to efficient and practically feasible protocols for threshold DSS, RSA, and other important cryptosystems, like Cramer-Shoup.

The progress in research on threshold cryptography has been marked by threshold schemes that are either more efficient or secure against increasingly more powerful ad-

[10] For the history of research on threshold schemes for DSS, RSA, and Cramer-Shoup cryptosystems, see Section 1.4.

[11] See also Appendix D.

versaries, and thus offering higher fault-tolerance for protecting the secret-key operation. We summarize the history of this research below. The discussion of different adversarial models considered for threshold cryptography is a subject of Chapter 2. We refer the reader to that chapter for an explanation of the terms used in the following research summary.

The following works mark the progress of **Threshold DSS**:

- The DSS signature scheme is a modification of the ElGamal signature [ElG85b]. The first threshold cryptosystem for an ElGamal-like signature scheme is due to [Har94]. The protocol of [Har94] is secure against $t < n/2$ passive faults.

- The first solution for shared DSS was given by [CMI93]. It is secure in a static adversary model and it tolerates $t < n/3$ faults among n players. However, this solution implicitly relies on a non-standard DDH-like assumption.

- Shared DSS was independently studied by [Lan95], whose solution tolerates only $t < \sqrt{n}$ passive, i.e. gossip-only faults.[12]

- [GJKR96b] provided a static $n/3$-threshold DSS protocol whose security is based solely on the hardness of forgery of regular DSS signatures. Furthermore, this solution allows much better on-line efficiency of signature generation. The static threshold DSS scheme we present in Section 4.3 is based on this result.

- Threshold DSS schemes provided in [CMI93] and [GJKR96b] can both be proactivized, i.e. made secure against mobile adversaries, using the proactive secret-sharing techniques of [HJKY95, HJJ+97]. We present these techniques in Section B.1.

- A simplified protocol for multiplication of shared secrets was introduced in [GRR98] (presented here in Section 4.4.2). Together with the techniques of [GJKR96b], this multiplication protocol yields a threshold DSS scheme with optimal $n/2$-threshold resilience (presented here in Section 4.4.3).

  Moreover, [GRR98] introduced a "fast-track" verifiable secret-sharing protocol which enables a factor of n increase in the computation efficiency of threshold schemes for the common case of execution without active faults.

- However, solutions of [CMI93] and [GJKR96b], as well as the proactivization techniques of [HJKY95, HJJ+97], utilize as a subprotocol a distributed key generation protocol of [Ped91b], which contains a subtle bug that invalidates the security proofs of all these subsequent protocols. Fortunately, this bug was identified and fixed by [GJKR99]. We explain why the protocol of [Ped91b] is insufficient in Section 4.2.3 and present the fix in Sections 4.2.4-4.2.5.

[12] Langford presents some additional schemes but of more limited applicability, a 2-out-of-n scheme that withstands up to one faulty party, and a general t-out-of-n scheme that uses pre-computed tables of one-time shares and that requires a higher level of trust for the generation of these tables. See [Lan95] for details.

- A further improvement was made by [CGJ+99, FMY99b, FMY99a], who modified the statically-secure protocols to achieve an optimal-threshold proactive shared DSS secure against an *adaptive* adversary in the *erasure-enabled* computational model. The adaptive threshold schemes of [CGJ+99] is a subject of Chapter 5. The extension of these threshold schemes to the mobile adversarial model is discussed in Section B.1.

- These adaptively secure results were then improved by [JL00] who removed the need to recourse to reliable erasure in these protocol. These improvements are presented in Chapter 6.

- The adaptively secure results were also recently improved by [Lys99] in a different direction. The participating players continue to use erasures but the resulting protocols can be proven secure under concurrent composition. In other words, the players that implement a threshold DSS scheme of [Lys99] can be involved in multiple concurrent executions of the threshold DSS signature generation protocol, each instance signing a different message.

Research on **Threshold RSA** signature/decryption generation also progressed gradually:

- In two independent works [Boy89, Fra89] gave a simple protocol for shared RSA signature generation which uses straightforward additive secret-sharing instead of Shamir secret-sharing [Sha79]. This protocol is secure only against spying faults.

- [DF91] gave a heuristic solution to threshold RSA that is secure against a $n/2$-threshold of halting faults.

- [FD92, DDFY94] gave a provably secure RSA scheme in the same context.

- Independent results of [FGY96, GJKR96a] introduced robustness to these protocols, thus providing full $n/2$-threshold RSA secure in a static adversary model.

- The first proactive RSA secure in the static model was given by [FGMY97b], but that solution did not tolerate the optimal threshold of faults.

- [FGMY97a, Rab98, Sho00] gave increasingly simpler and more efficient solutions for optimal-threshold RSA secure in the static model. Furthermore, [FGMY97a, Rab98] provide also proactive versions of their RSA protocols.

- Adaptively-secure, optimal-threshold proactive RSA are given by [CGJ+99, FMY99b, FMY99a]. Protocols of [CGJ+99] are based on the statically-secure proactive threshold RSA of [Rab98] while protocols of [FMY99b, FMY99a] extend the static proactive results of [FGMY97a].

- [JL00] improved the above adaptive threshold RSA results, by improving their efficiency and eliminating the need to recourse to local data erasure. This solution to threshold RSA is described in Appendix C.

We point out, however, that all the above solutions to threshold RSA require additional trust assumptions during the *initialization* of the system of n players that will implement the shared RSA signature/decryption protocol (see Sec. 2.2). There is an independent line of research that aims at reducing these trust assumptions, i.e. searches for *secure* protocols that generate the shared RSA key (i.e. protocols that are resilient and protecting the secrecy of the generated shared RSA key, in the presence of an adversary that corrupts a threshold of the players), whatever form of the shared RSA key is demanded by the above-listed threshold RSA signature/decryption protocols. See [BF97, FMY99c, Gil99].

The research on threshold **Cramer-Shoup** [CS98] appeared as follows:

- The first threshold Cramer-Shoup cryptosystem was a statically-secure protocol given in [CG99]. This protocol could be separated into off-line pre-computation stage and a non-interactive on-line protocol.
- In [JL00], the threshold Cramer-Shoup protocol of [CG99] was modified to achieve security in the adaptive erasure-free model. However, the [JL00] protocol was no longer non-interactive. The adaptively-secure threshold Cramer-Shoup protocol of [JL00] is presented here in Appendix A, and the erasure-free techniques of [JL00] are presented in Chapter 6.

Chapter 2

Modeling Security of Threshold Schemes

In this chapter we describe the computation and communication model used by the threshold signature and decryption schemes we propose. We also explain several ways in which we model faults and other adversarial behavior. We then formalize the notion of security for a threshold signature scheme (and a threshold cryptosystem) relative to a specific adversarial model. In this thesis we consider three main different models, each corresponding to fewer constraints on the adversarial behavior, and hence to an increasing strength of the attack against a threshold scheme. The stronger the adversarial model relative to which a scheme is proven secure the more fault-tolerant the scheme is.

2.1 Standard Model

Computation and Communication Model. Our computation model consists of a set of n dedicated *players* $\{P_1, \ldots, P_n\}$ and an *adversary*. We model each of those $n+1$ parties as a probabilistic polynomial-time (PPT) interactive Turing Machine (TM) [GMR89]. The n players model the servers that implement the distributed signature (or decryption) service. We assume that the number of servers n is polynomially related to the security parameter k. Each player P_i starts its operation having as input this security parameter encoded in binary. We stress that we do not model the *user* of a threshold scheme with a separate PPT TM. Instead, we assume that the adversary is the sole user of the threshold scheme. This reflects an assumption that the user of a threshold scheme can be dishonest.

We assume that all $n+1$ parties are connected by a complete network of insecure point-to-point channels, to which we will often refer to as *links*. However, in the presentation of our protocols we will assume that the parties are connected by a complete network of *authenticated* point-to-point channels, because authentication on these channels can be implemented using standard signature schemes, under an appropriate computational assumption.[1] In addition, we will also assume that the parties have access to a dedicated and authenticated

[1] See [BCK98] on implementing authenticated channels for a network of players. Note that to establish the authenticated channels we also need to assume initial secure distribution of verification keys.

broadcast channel; by dedicated we mean that if some party X broadcasts a message, it is received by every other party and recognized as coming from X. Such broadcast can be implemented over authenticated point-to-point channels with an authenticated Byzantine agreement protocol [DS83].

We assume that the communication channels provide *(partially) synchronous* message delivery. By partially synchronous communication model we mean that the messages sent on either a point-to-point or the broadcast channel are received by their recipients within some fixed time bound. A failure of a communication channel to deliver a message within this time bound can be treated as a failure of the sending party.[2] While messages arrive in this partially synchronous manner, the protocol as a whole proceeds in synchronized rounds of communication, i.e. the honest players start a given round of a protocol at the same time. To guarantee such round synchronization, and for simplicity of discussion, we take the standard assumption for synchronous networks that the players are equipped with synchronized clocks. Note that in the authenticated channels setting synchronized clocks can be achieved in the presence of a minority of malicious faults [LMS85, HSSD84].

Interface with the User. The user of the fault-tolerant service that the n players implement can communicate with this service in two ways. The communication can proceed through a proxy, i.e. a special player designated as a *gateway* between the n players and the user. This gateway forwards user's messages to the n players and decides on the messages sent back to the user by taking the majority of the final outputs originated by the n players. Such gateway guarantees full transparency of the distributed signature or decryption service to the user, i.e. the user communicates with it as if it was a standard centralized signature or decryption scheme. Notice that if a gateway is corrupted, the most it can do is to stage a denial of service attack. We can easily protect our system from such attack by having any of the participating players serve as a gateway. If it refuses to work properly then another player takes over the role of a gateway server. Such replication of a gateway functionality does not compromise security.

The second possibility is that the user communicates with each of the n servers directly. This option requires a (minimal) modification to the user's code when upgrading a system that uses a standard centralized implementation of some signature or decryption service. However, it also reduces communication between the servers.

In both cases we make a simplifying assumption that the honest players agree on which input to perform the threshold protocol, i.e. for example in the case of a threshold signature scheme we assume that the uncorrupted servers agree whether to sign any given message. For example they might have the same policy for deciding which message to sign, or they might agree via some voting protocol.

Modeling Faults with a "Threshold" Adversary. To provide truly fault tolerant secret-key services (signature and decryption), we assume that some threshold of t out of n of the participating players can exhibit faults. We model such local computer faults

[2]Treating a denial-of-service attack on the link as an attack on a sender might in some adversarial settings unnecessarily decrease the resilience of the distributed system. An alternative approach can be taken following the work of [CHH97]. Namely, if each sender disperses each of its messages to all other serves who then relay it to the receiver, then message delivery is assured even if the adversary stages a denial-of-service attack on a minority of links coming out of every server. This solution increases the number of communication rounds by the factor of two and the number of messages sent by the factor of $2n$.

in the most general way. We assume the worst case that all faults are orchestrated by a single malicious adversary who schedules which computers to corrupt and decides on the actions of each corrupted server, for example an exposition of its secrets, or any other any modification of its prescribed behavior. Thus we will use the terms "fault" and "adversary" interchangeably. Furthermore, we only consider adversaries whose actions can be encoded as an execution of some efficient *adversarial algorithm*. In other words, we model an adversary as another interactive probabilistic polynomial-time machine (PPT TM). In particular, his computational resources might thus be constrained based on some complexity assumptions. We note that this is a necessary constraint. Since we aim to create threshold schemes for standard public-key systems, we must assume those systems secure, which always implies computational limitations for the adversary.

Specifically, all our schemes require the discrete-logarithm assumption, and whatever assumption necessary to implement authenticated channels. To implement private channels, our static and adaptive erasure-enabled schemes require whatever assumption necessary to implement semantically-secure encryption (either public-key or symmetric-key), while our adaptive erasure-free schemes require a Decisional Diffie-Hellman (DDH) assumption. (See Section 2.2 below for explanation of the variants of our model, and for a separate paragraph on implementation of private channels.) Furthermore, each threshold scheme requires the same assumption that enables the standard public-key scheme, e.g. threshold DSS requires an assumption of unforgeability of DSS, threshold Cramer-Shoup requires existence of a universal family of one-way hash functions (UOWHF) [NY89], etc.

We assume a t-threshold adversary who can corrupt up to t among the n players that implement the threshold scheme. (In Section 2.2 we discuss a crucial restriction on when the adversary is allowed to make a decision upon which t servers to corrupt.) In this thesis we consider thresholds t which are a constant fraction of the number of players n. The default threshold we consider is an optimal threshold of minority of faults, i.e. $t = \lfloor \frac{n-1}{2} \rfloor$. We will also discuss, for the purposes of exposition, protocols resistant to smaller thresholds, specifically $t = \lfloor \frac{n-1}{4} \rfloor$. For simplicity of notation, we will denote these models as "$n/2$-threshold" or "$n/4$-threshold". With the exception of a mobile adversarial model (see Section 2.2 below), a corrupted player is assumed to be corrupted for the rest of the computation.

The point to point links between players are insecure, and hence the adversary can listen to all messages sent on every channel. However, as we discuss in Section 2.2 below, under appropriate assumptions we efficiently implement private channels in each adversarial model we consider. Since we thus always implement both link authentication and link privacy, in the description of our protocols we will assume that the point-to-point links are *secure*, i.e. both authenticated and private.

When the adversary corrupts a party P, he learns P's *current internal state*. Furthermore, from this point on, all messages directed to P become known to the adversary. The adversary can also send messages in the name of P. In every communication round the adversary decides upon the order in which players (both corrupted and uncorrupted) should be *activated* to perform their protocol for that round. When the adversary activates an *uncorrupted* party P, this party receives the messages sent to it in the previous round and it generates its messages for this round. Once all the uncorrupted parties were activated, the adversary generates (according to his own protocol) the messages to be sent by the

corrupted servers, and the next round of computation can begin.

The adversarial ability to schedule the order in which players are activated reflects a partially synchronous communication model where all messages can still be delivered relatively fast, in which case, in every round of communication, the malicious adversary can wait for the messages of the uncorrupted players to arrive, then decide on his computation and communication for that round, and still get his messages delivered to the honest parties on time. Therefore we can in fact assume the worst case that the adversary speaks last in every communication round, as described by the model above. In the cryptographic protocols literature this is also known as a *rushing* adversary.

In the formal model of security of threshold schemes we assume that the adversary controls also the user of the scheme. It is a standard way of modeling an attack on a cryptographic system to assume that the user that interacts with the system tries to spoil it somehow.

2.2 Variants of the Model

Restrictions on Maliciousness of Faults. The above description is the default adversarial model we adopt in this paper. The ability of the adversary to control the behavior of the corrupted players is referred to as a (fully) *malicious* adversary model. Protocols which are resistant in the presence of such malicious adversary are often called *robust*. In the secure multi-party computation and threshold cryptography literature, as well as in the literature on non-cryptographic distributed algorithms, two crucial restrictions on this adversarial power are often considered. Thus we can consider an *eavesdropping*, or "*gossip-only*", adversary who learns all the information stored and received by the corrupted players and sent on the broadcast channel, but can neither divert the corrupted players from their prescribed protocol nor send any messages on the broadcast channel. Such restriction models only network viruses or dishonest server operators who manage to spy on certain players, or maybe spy on only certain parts of these players' memory, but cannot modify players' behavior. Thus such faults are sometimes referred to as "*passive*" faults, while the malicious faults are called "*active*". A weaker restriction is a *halting* adversary model, where the adversary has the eavesdropping power plus the ability to stop any of the corrupted servers at any time in the protocol. This model extends the previous one by including "stopping" faults, caused for example by flooding a server with messages and effectively disconnecting it from the network, by a virus attack that crashes the operating system, by a power failure, etc. We stress however that our protocols are by default resistant against the worst-case scenario of a fully malicious adversary.

Scheduling of Faults. Another major distinction between security models for distributed protocols is whether the attacker is *static* or *adaptive* (the latter is also sometimes called "dynamic"). In both cases the threshold adversary is allowed to corrupt up to t parties. However, in the case of an adaptive adversary the decision of which parties to corrupt can be made at any time during the run of the protocol and, in particular, it can be based on the information gathered by the attacker during this run. In contrast, in the static case the attacker is restricted to choosing its victims independently of the information it learns during the protocol. Therefore, in the static model the subset of corrupted parties can be seen as chosen and fixed by the attacker before the start of the protocol's run. Formally, in

the adaptive case the description of the interaction between the adversary and the protocol in Section 2.1 must be modified as follows. Unless the adversary has corrupted t players already, in every communication round he decides not only on the order in which he activates the uncorrupted players, but also on whether to corrupt some of them.

Static adversary models well the adversarial behavior that is caused by *inherently dishonest* human operation. This is a meaningful threat in applications which distribute trust among different entities. For example, if Key Escrow is administered by several organizational entities, only some fraction of which we hope is dishonest, then it is natural to assume that those entities are dishonest inherently rather than dynamically become so. In other words, in this case the identities of the corrupt parties are chosen independently from the protocol execution, and hence this type of faults is sufficiently modeled by the static adversary.

On the other hand, the static fault model imposes an artificial restriction if the faults are not due to inherent dishonesty of participants but due to a coordinated attack by an adversary who searches for vulnerabilities of the participating players. Such adversary can attack both the hardware that carries out the computation on behalf of a player (e.g. by searching for a hole in the operating system of a server, by cryptoanalyzing its communication keys, or by disconnecting a server from the network) and the people that maintain the computing facility (e.g. by bribing them). Each such attempt represents the adversary's decision to apportion some resources of time and money to attacking some player, and hence there is no reason to assume that the adversary cannot make these decisions dynamically, on the basis of the information he gathers about the protocol execution so far.

It is known that protocols secure in the static model can be insecure in the adaptive one unless the number of participating players n is only logarithmic in the security parameter k [CFGN96, CDD+99, Can00]. Therefore, since the adaptive adversary model appears to better capture threats facing fault-tolerant distributed applications, threshold schemes that are provably adaptively secure are preferable to static ones. Even though the bulk of published works on distributed cryptography assumes a static adversary, it is due to the difficulties encountered when trying to design and prove protocols resistant to adaptive adversaries, rather than due to a belief that static adversary is a sufficient model of fault scheduling. In fact, in this thesis we also start the presentation of our threshold protocols in Chapter 4, which are provably secure only under the simplifying restriction of the static adversarial model.[3] Such presentation allows us to separate the technical difficulties that are common to both models and to better explain the solutions to those difficulties. It also allows us to isolate in Chapter 5 a different set of techniques which enable efficient protocols that are resistant in the adaptive model.

Assumption of Reliable Data Erasure. The adaptively-secure threshold protocols we present in Chapter 5 rely on the ability of the participating players to erase some of the local data that they produce during the protocol. For brevity we will refer to this assumption as to an *erasure-enabled* computation model. This is a drawback because secure erasure of

[3]We note, however, that our statically secure protocols of Chapter 4 suffer no *known* vulnerabilities in the adaptive model. Their "only" drawback is our inability to prove them secure in the adaptive model for n larger than $O(\log k)$. Again, given the results of [CFGN96, CDD+99, Can00] mentioned above, if $n = \omega(\log k)$ then a threshold scheme which is provably secure in the adaptive model is preferable.

data is difficult to guarantee in practice. On the hardware level, it is difficult to permanently erase information from physical storage devices. On the system maintenance level, the need to permanently erase date complicates standard computer system bookkeeping and backup procedures. Finally, perhaps the most difficult problems arise on the operating system level, since in order to securely erase the data, one needs to erase it from all the caches and from the part of the hard drive that was used for page swaps, etc.

In Chapter 6, we show that in the adaptively secure threshold setting it is possible to get rid of the need of secure data erasure altogether. Namely, we provide threshold protocols that stay adaptively secure, as those of Chapter 5, but can be executed without recoursing to data erasure. The new protocols are also not significantly more computationally intensive compared to those of Chapter 5, and hence they seem to provide the most efficient way to handle the difficulty of implementing reliable erasure of local data on standard hardware and operating systems.

We refer to a computational model which does not provide a reliable procedure to erase local data as an *erasure-free* model. Formally, the interaction between the adversary and the protocol described in Section 2.1 is the same in both models except that the current internal state revealed to the adversary upon a corruption of a player contains a whole computational history of that player in the erasure-free model, while in the erasure-enabled model it contains only the local data that has not been previously erased.

Implementing Private Channels. We assume a realistic communication model of insecure channels. On the other hand the threshold protocols we propose require private and authenticated communication between the participating players. We mentioned that authentication can be implemented with standard signatures. However, the task of explicitly providing means to establish link *secrecy* can be non-trivial, depending on the adversarial model considered. We explain this difficulty in detail in Section 6.1 while here we only state its conclusions.

Efficient implementation of private channels in the static adversarial setting is easy. Given appropriate complexity assumptions we achieve private communication via semantically secure encryption. In adaptive adversarial setting there is an inexpensive technique of [BH92] which implements private channels under the assumption of availability of reliable data erasure. Thus in both Chapter 4, where we present protocols secure in the static adversarial setting, and in Chapter 5, where we consider an adaptive but erasure-enabled model, we can assume that the point-to-point channels are private, and thus present our protocols as executing over such channels. However, no practical and general implementation of secure channels is so far known in the adversarial model which is both adaptive and erasure-free. We come close to such general implementation in Chapter 6, where we construct a novel primitive of a *simultaneously secure* encryption scheme. Simultaneously secure encryption implements private channels in the adaptive erasure-free model for a wide class of distributed protocols which includes our threshold schemes. We explain this issue further in Section 6.1.[4]

Relaxation of the Adversarial Model: Trusted Party during the Initialization. The threshold techniques we propose imply adaptive erasure-free protocols for threshold RSA signature or decryption generation. However, they do not imply secure protocols for

[4]Implementing private channels is also non-trivial in the proactive model discussed below.

threshold generation of RSA secret key. Therefore our RSA function-application protocols need to be preceded with a key generation protocol that assumes a *trusted dealer*. Positing an existence of a single trusted party during the initialization of the system is a relaxation of the adversarial model, but it can be justifiable in some scenarios.

Beyond a Threshold Adversary: Mobile and Creeping Adversarial Models. In Appendix B we show that in an erasure-enabled computational model our building-blocks protocols can be used to create schemes that withstand stronger attacks than those modeled by a threshold adversary. While a threshold adversary can corrupt no more than t servers throughout the system operation, we can also consider the following two strictly stronger models: 1) A *mobile* adversary, introduced in the work of [OY91], who can choose a different subset of t corrupted servers *every day* (or any other fixed time-period); and 2) A *creeping* adversary, who can corrupt all players except two, but his corruptions must be detected fast enough so that the remaining players can trigger certain protective protocols.

We call schemes that are secure in either of these two adversarial settings "*proactive*". When proactive systems were introduced in [OY91, CH94, HJKY95], this term was used to designate resistance solely to the mobile adversary, and this is also how we use this term in the research review in Section 1.4. Here we extend the notion of proactive security and proactivization to security in the creeping adversarial model.

Proactive schemes provide better security especially for long-lived systems, where the adversary might have enough time to eventually corrupt almost all participating players. However, as long as these attacks are detectable and do not happen all at once – which is modeled, albeit differently, by both the mobile and the creeping adversarial model – the currently uncorrupted players can "proactively" perform various self-healing protocols that neutralize the adversary's advantage gained from corruptions staged so far.

The self-healing protocols we consider are based on re-randomization of the secret-sharing of a secret key, which makes the information learned by the adversary so far useless in his consecutive corruptions. Additionally, for protection against a creeping adversary, when some player is corrupted and this corruption is detected by the remaining players, these players must be able to reduce the degree of the secret-sharing polynomial during such re-randomization. Similarly, when a new player is added, the players must be able to increase the degree of the secret-sharing polynomial. In fact, all these techniques should be implemented and used together, so that the resulting scheme is secure against combinations of mobile and creeping adversarial behaviors.

2.3 Efficiency Criteria

Threshold schemes can differ by the level of security they offer, namely by the adversarial model in which they can be proven secure. Thus we talk about an "static" or "adaptive" threshold schemes, meaning schemes which are secure respectively in static or adaptive adversarial models. However, threshold schemes can also greatly differ in efficiency. We list here some important efficiency criteria for threshold schemes:

- **Computation and Communication Complexity**: We measure the amount of local computation and of bits sent by each participating player. We count the amount of private and public communication separately. If the interface between a user a

distributed secret-key service does not include a gateway, we also measure the computation and communication cost carried by the user. Because the most expensive arithmetic operation in the threshold schemes we design is a modular exponentiation, the amount of local computation in our schemes is driven by the number of "long" exponentiations, i.e. modular exponentiations where the bit-length of the exponent is about its maximum.

- **Communication Rounds**: We assume that our protocols proceed in synchronized communication rounds, during which the participating servers send messages to one another. The communication rounds are synchronized via synchronized clocks. All our protocols take a constant number of such rounds. However, the rounds have to be long enough to tolerate standard delays in message delivery provided by the underlying network.[5] Therefore the number of such rounds should be minimized.

- **Non-Interactive Schemes**: Because the synchronized communication rounds are expensive, it is important to search for non-interactive threshold schemes where the participating servers do not need to communicate with one another at all. Instead, given that the user communicates with the secret-key service without a gateway (see the "Interface with the User" paragraph in Section 2.1), then after receiving the user's input each server communicates only with that user. At the end of such communication, the user locally reconstructs the (signature or decryption) output of the threshold scheme. This communication might take several rounds, but the rounds do not need to be synchronized. We call such threshold schemes *non-interactive*.[6] Non-interactive threshold schemes are possible, often in the optimistic execution mode (see below), often only in the on-line part of the protocol (see below), and often at the price of extending the public key of the cryptosystem by some verification information that allows the user to filter out incorrect messages the user might receive from corrupted servers. Examples of such schemes are the threshold DSS of [GJKR96b] presented here in Chapter 4, many threshold RSA solutions, e.g. [GJKR96a, Sho00], and the threshold Cramer-Shoup of [CG99].

- **Efficient Optimistic Execution**: Threshold schemes should be designed so that they can be executed more efficiently in an "optimistic" manner, i.e. assuming that there are no active faults. This idea was proposed in [GRR98], who showed how to execute a Verifiable Secret-Sharing protocol (see Section 4.2.3) in a "fast-track" mode. If the faults do occur, the full-fledged version of the protocol is executed, and the proper verification checks identify and eliminate the faults. This optimistic execution could be for example applied to the non-interactive threshold signature protocols. The user might first try to reconstruct the signature from the partial outputs received from

[5] If the bounds on message delivery are set too short then the fault-tolerance of a scheme decreases, because too many network delays are going to be interpreted as player faults. This negative effect of network delays can be somewhat decreased by the techniques of [CHH97], see footnote 2, page 24.

[6] In some threshold cryptography publications the term "non-interactive threshold schemes" is applied only to a more restrictive scenario in which the above communication between the user and the servers takes only one round.

any large-enough group of servers, and only if it is incorrect the user should request that the servers execute the truly fault-tolerant protocol.

- **Off-line/On-line Division of Computation**: In many applications of secret-key services one would like to minimize latency of function computation. This is possible in many threshold schemes, including the DSS and Cramer-Shoup schemes we provide, if the players pre-compute part of the protocol during spare time, and then complete the rest quickly when the input comes. We call the pre-computation stage "off-line", and the latter stage an "on-line" computation. For both our DSS and Cramer-Shoup threshold schemes, the on-line part of the protocol takes just one round of communication, and thus it can be non-interactive in the sense discussed above.

- **Use of Broadcast**: The broadcast communication mechanism is implemented with the (authenticated) Byzantine-Agreement protocol of [DS83], which is expensive. It can take up to $t + 1$ rounds of (synchronized) communication in the presence of t faults. Worse, the source of a fault might not be completely identified. For example, if player A claims that player B did not send any message to A, only A and B know which one is cheating, and the faulty player cannot be uniquely identified by all the honest players. Therefore, the adversary can theoretically inject such faults during every single broadcast. (In practice, the system maintenance will be alerted that *either* of the two players is faulty and should be examined until the source of fault is found and dealt with.) Even in the default fault-free execution, each broadcast still takes two communication rounds where every player signs and resends the broadcast message to every other player. Unfortunately, all our threshold schemes which are *not non-interactive*, i.e. in which the participating players communicate with one another, use broadcast. This concern should motivate a further research on efficient threshold protocols that are either non-interactive or do not use broadcast.

- **Elimination (and Amortization) of Active Faults**: A crucial efficiency criterion is whether active faults that prolong the computation (for example by switching the execution from running in the efficient optimistic mode) can be identified by all the honest players, and in particular unequivocally identified by the human personnel who maintains the servers. If that's the case then if a threshold protocol is executed often, then the amortized cost of each execution is the cost of an optimistic execution where the active faults are not present.

2.4 Formal Definitions of Security of Threshold Schemes

We provide formal definitions of secure Threshold Signature Schemes. These definitions are relative to an adversarial model described in Section 2.1 and 2.2, where we specified the execution of a distributed protocol in the three adversarial models we consider in this thesis: static, adaptive erasure-enabled, and adaptive erasure-free.[7]

[7]Our model of a network attack follows the work of [MR91, Can00] which defined such attack formally in the context of general multi-party computation.

Our definition of unforgeable threshold signature schemes is an extension of the standard definition of secure signature scheme of [GMR88] to the threshold setting where the functionality of the private key holder is distributed among multiple servers. Thus we extend the capabilities of the signature-scheme adversary of [GMR88], namely the adaptive chosen message attack, with the capabilities of a threshold adversary which attacks these multiple servers, learns their private states and possibly diverts them from the prescribed protocol. We present our definitions in a communication model where an authenticated broadcast channel is available.

First we recall the [GMR88] definition of chosen message attack (CMA) security of a standard signature scheme and then we describe an equivalent definition for security of a *threshold* signature scheme.

Definition 1 (CMA Security of a Signature Scheme) *A secure signature scheme* SS *is a triple of efficient algorithms (*Key-Gen, Sig-Gen, Ver*), where* Key-Gen *is a key generation algorithm which on input a security parameter* 1^k *outputs a pair* (s, v) *where* s *is a secret key and* v *is a public key. The signing algorithm* Sig-Gen *on input a message* M *and the secret key* s *outputs signature* u *s.t. the verification algorithm* Ver *on input* (u, v, M) *accepts the signature as correct.*

Consider the following interaction between an adversary algorithm i.e. a family of interactive probabilistic polynomial-time algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$*, and an interactive algorithm* $\mathcal{O}_{SS}$*, called* signature oracle*, which implements the* SS *signature scheme. First, on common input* 1^k*,* $\mathcal{O}_{SS}$ *runs the* Key-Gen *algorithm to generate* (s, v) *and sends* v *to* $\mathcal{A}_k$*. Then* $\mathcal{A}_k$ *can send messages* $M_1, M_2, ...$ *of its choice to* $\mathcal{O}_{SS}$*, which on each received message* M_i *returns to* $\mathcal{A}_k$ *signature* $u_i = \mathsf{Sig\text{-}Gen}(s, M_i)$*. At some point* $\mathcal{A}_k$ *can also output a candidate (message,signature) pair* M, s*. We say that* $\mathcal{A}_k$ forges *if* $M \neq M_i$ *for all* i *and* $\mathsf{Ver}(v, M, s) = 1$*.*

We say that SS *is chosen-message attack (CMA) secure signature scheme if the probability that* A_k *forges is a negligible function of the security parameter* k*.*

An n-server threshold signature scheme (TSS) τ is a triple (Key-Gen, Sig-Gen, Ver), a distributed key generation protocol, a distributed signature generation protocol, and a verification algorithm (see a detailed description below). We require (Definition 2 below) that τ is unforgeable, i.e. that the threshold adversary that attacks the n servers and participates in the protocols they execute should be unable to come up with a valid signature on a message that was not signed by the servers, even after invoking the servers to sign any sequence of adaptively chosen messages. We also require (Definition 3 below) that the signature scheme τ is *robust*, i.e. that if the servers execute a key generation and then a signing protocol then a valid public key and valid signatures are always generated. We formalize the three components (Key-Gen, Sig-Gen, Ver) of a TSS as follows:[8]

• Key-Gen, a distributed key generation protocol, is a probabilistic polynomial-time protocol run by the n servers. Each server has a public input a security parameter k, random input

[8]In the description of Sig-Gen the adversary is the only user of the signature service implemented by the TSS. Such formalization allows for concise definitions of both unforgeability and robustness of a TSS, where we assume that the adversary invokes the signature protocol either (or both) to learn to forge signatures, or to try to disrupt the servers from successfully executing the protocol.

r_i,[9] and generates a private output s_i, a share of the signature key, and a public output v, the public key. We establish a convention that a public output of a protocol is defined to be the majority of what our protocols denote as "public output" of the players. Our protocols will maintain an invariant that, under some hardness assumptions ensuring robustness, all uncorrupted players have the same "public output". Since the uncorrupted players are a majority, the public output can be defined by what the majority outputs. To simplify the interaction between the servers and the user (modeled here as an adversary), we assume that the public output is sent out by the players on the broadcast channel.

• Sig-Gen, a distributed signature generation protocol, is a probabilistic polynomial-time protocol run by the n servers. This protocol is invoked by an adversary when he sends a message of a form '`[sign,`M`]`' on the broadcast channel. Each server has public inputs M and v, a private input s_i, and a random input r_i.[10] (Presumably, values s_i and v are private and public outputs of a run of the key generation protocol.) This protocol has only public output, a signature u on message M, which we will denote by '`[signed,`M`,`u`]`'.

• Ver, a signature verification algorithm, is a polynomial-time algorithm which can be run by any party. It takes an input v and a pair (M, u). (Presumably, v is the public output of the key generation protocol, M is some message, and u is a signature.) The output is `pass/fail`.

Adversarial Run of a Threshold Signature Scheme. We define the security of a threshold scheme in a non-uniform model of computation. We model a t-threshold adversary as a family of interactive probabilistic polynomial-time Turing Machines (PPT TM) $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$, where $\mathcal{A}_k$ is the algorithm the adversary follows if the public input is a security parameter k.[11]

An *adversarial run* of a TSS τ = (Key-Gen, Sig-Gen, Ver) with a t-threshold adversary $\mathcal{A}$ consists of an execution of protocol Key-Gen with the participation of $\mathcal{A}$, on public input a security parameter k encoded in unary, and on a vector $\vec{r}$ of random inputs for the servers and the adversary. We assume that the number of servers n is at most polynomial in k. $\mathcal{A}$'s participation in an execution of a distributed protocol depends on the adversarial model (static,adaptive erasure-enabled, or adaptive erasure-free). We describe it formally in the "Adversarial View" paragraph below. Let v and $s_1...s_n$ denote, respectively, the public and private outputs of the servers from this execution of Key-Gen. Next, each time the adversary submits a message '`[sign,`M`]`' on the broadcast channel, the servers run protocol Sig-Gen, with the participation of $\mathcal{A}$, on public inputs M, v, private inputs s_i of the servers, and a vector of random inputs $\vec{r}$ for the servers and the adversary. We stress that $\mathcal{A}$ can invoke

[9] For the sake of conciseness, in the descriptions of the distributed protocols we give in this thesis, we do not explicitly mention the random inputs of the players, although the players do indeed require randomness in all the protocols we discuss.

[10] The servers and the adversary use a different part of their random strings r_i in vector $\vec{r}$ as their source of randomness for the execution of Key-Gen and for each instance of Sig-Gen.

[11] We adopt a convention to denote the adversary that participates (i.e. attacks) any threshold scheme or its subprotocol (e.g. protocols Key-Gen and Sig-Gen), by the same symbol "$\mathcal{A}$". We can assume that the messages of the honest players in any protocol are extended by a tag which specifies which protocol they are executing. Hence a single adversarial algorithm $\mathcal{A}$ can represent a family of different adversarial algorithms for attacking different distributed algorithms. E.g., a single interactive PPT TM algorithm $\mathcal{A}$ can specify the adversary's behavior during an execution of Key-Gen and the adversary's behavior during an execution of Sig-Gen.

many copies of protocol Sig-Gen. However, all the invocations of Sig-Gen are assumed to be sequential, i.e. a new invocation does not start until the previous one is completed. The output of this adversarial run of the threshold signature scheme τ, denoted $\mathsf{out}_{\tau,\mathcal{A}}(k,\vec{r})$, is the public output v of Key-Gen, followed by the concatenation of all the messages of the type '[sign,M]' sent on the broadcast channel by the adversary, and the messages of the type '[signed,M,u]', sent on the broadcast channel by the parties and the adversary. Let $\mathsf{out}_{\tau,\mathcal{A}}(k)$ denote the distribution of $\mathsf{out}_{\tau,\mathcal{A}}(k,\vec{r})$ where $\vec{r}$ is uniformly chosen from its domain. Now we can formalize the notions of unforgeability and robustness of a TSS.

(In the definition below we use a concept of a negligible function. We call function $f(k)$ *negligible* if it is asymptotically smaller than an inverse of any polynomial, i.e. if for every polynomial $p(k)$, there exists k_0 s.t. $f(k) < 1/p(k)$ for all $k \geq k_0$.)

Definition 2 (Unforgeability of TSS)
A TSS τ = (Key-Gen, Sig-Gen, Ver) *is unforgeable in certain adversarial model if for any adversary $\mathcal{A}$ in that model, the random variable* $\mathsf{out}_{\tau,\mathcal{A}}(k)$ *satisfies the following property, except for probability negligible in k: If* $\mathsf{out}_{\tau,\mathcal{A}}(k)$ *contains a public key v and a message* '[signed,M,u]' *such that* $\mathsf{Ver}(v,M,u)$ = pass, *then it contains also a message* '[sign,M]'.
Informally: If the output includes a message '[signed,M,u]' with a valid signature u on a message M, then this message must have been produced by the distributed signature protocol invoked on message M by a message '[sign,M]' that also appears in the output. In other words, the adversary $\mathcal{A}$ cannot produce signatures otherwise but by asking the signing servers to do it.

Definition 3 (Robustness of TSS)
A TSS τ = (Key-Gen, Sig-Gen, Ver) *is robust in certain adversarial model if for any adversary $\mathcal{A}$ in this model, the random variable* $\mathsf{out}_{\tau,\mathcal{A}}(k)$ *satisfies the following property, except for probability negligible in k: 1)* $\mathsf{out}_{\tau,\mathcal{A}}(k)$ *starts with a public key v; and 2) If* $\mathsf{out}_{\tau,\mathcal{A}}(k)$ *contains a message* '[sign,M]' *then it also contains a message* '[signed,M,u]' *s.t.* $\mathsf{Ver}(v,M,u)$ = pass.
Informally: If the signing servers are asked to sign some message M then the adversary cannot disable them from producing a valid signature on that message.

Definition 4 (Security of TSS)
A TSS τ *is secure in certain adversarial model if it is both robust and unforgeable in that adversarial model.*

Remark 1: On the influence of other protocols on the servers. Our definition of security postulates that, upon corrupting a server, the adversary sees the current internal state of the server; however, unlike in the definition of general multi-party protocols, only the state of the current execution of the TSS is seen. In particular, the adversary does not see internal data that pertains to other protocols that may be in execution at that server. This assumption can be justified in the context of threshold protocols if the n players that participate in the protocol are dedicated to the TSS task and do not run any additional protocols.

Remark 2: Other definitions of security of TSS. A somewhat different approach to defining security of TSS is taken in [CHH97]. There, an ideal distributed signature service is formulated; A TSS is secure if it "emulates" the ideal process, in some well defined way. That formalization provides a natural way for defining security even when no broadcast channels (nor any authenticated communication channels) exist. In contrast, our definition and protocols are presented in a convenient communication model where the parties are provided with a broadcast channel. Note that if there is no broadcast channel then defining security of TSS becomes considerably more tricky. In particular, it is no longer clear that all servers invoke each copy of Sig-Gen with the same input message.

Remark 3: On robustness in practice. Note that in both definitions it does not matter whether messages '`[sign,`M`]`' and '`[signed,`M`,`u`]`' are adjacent in the output of the adversarial run of a threshold signature scheme. It follows in particular that the above definition of robustness is very liberal, since it considers a scheme robust just as long as every request for a signature is eventually satisfied. In practice we want threshold schemes which process the requests at some reasonable rate.

The Adversarial View. A central tool in the proofs of security of our threshold schemes is the concept of an *adversarial view* of an execution of a threshold scheme. The adversarial view is the computational history of the interactive probabilistic polynomial-time Turing Machine (PPT TM) $\mathcal{A}$ during an adversarial run of a threshold scheme. This history is fully determined by the content of $\mathcal{A}$'s input tape, its random tape, and all the messages it receives on its interactive tapes.

We can formally model the interaction between the adversary and the players in the adversarial run described above as a computation of the interactive PPT TM $\mathcal{A}$ and another interactive PPT TM $\mathcal{P}$ which follows the prescribed protocols Key-Gen and Sig-Gen on behalf of the currently uncorrupted players. In this computation, $\mathcal{A}$ is connected to $\mathcal{P}$ by the following interactive tapes, whose number depends on the current number d (where $d \leq t$) of the corrupted players: (1) $n - d$ read-only tapes that correspond to the authenticated broadcast channels of the uncorrupted players; (2) d write-only tapes that correspond to the authenticated broadcast channels of the corrupted players; (3) $d(n - d)$ read-write tapes that correspond to the authenticated point-to-point links between each corrupted and each uncorrupted player; (4) A read-write tape on which $\mathcal{A}$ writes the identity of a player it decides to corrupt (a static adversary starts the computation by writing an identity of some t players on this tape), and as long as $d \leq t$, $\mathcal{A}$ immediately receives from $\mathcal{P}$ the current computational history of the requested player (in the adaptive erasure-enabled model $\mathcal{A}$ receives only the *current* computational state of the requested player, i.e. the currently unerased local data of that player); (5) A write-only tape on which the adversary can broadcast a public input '`[sign,`M`]`' which models a request sent by the user of the threshold signature scheme to sign message M; (6) A write-only tape on which the adversary writes an identity of an uncorrupted player it wants to *activate*, i.e. an identity of player the adversary wants to trigger to perform its prescribed protocol of a current round. When $\mathcal{P}$ receives such request for some uncorrupted party P_i, it delivers to P_i the messages sent to it in the previous round and it generates its messages for this round. In particular, $\mathcal{P}$ delivers to $\mathcal{A}$ the messages broadcast by P_i or the messages it sends to the corrupted players. Once all the uncorrupted parties have been activated, $\mathcal{A}$ can send to $\mathcal{P}$ some messages on behalf

of the corrupted servers, and the next round of computation can begin.

We will denote the view of an adversary $\mathcal{A}$ in its run against some protocol $\mathcal{P}$ on inputs $in_1, in_2, ...$ (as specified by this protocol) as $\mathsf{View}_{\mathcal{P},\mathcal{A}}(in_1, in_2, ...)$. Such view is a random variable whose distribution is induced by the uniform choice of vector $\vec{r}$ of random inputs of the adversary and the servers. We will often extend this notation and consider an adversarial view of a random execution of a protocol on inputs $in_1, in_2, ...$ subject to a constraint that some of the *public outputs* of this execution attain particular values $out_1, out_2, ...$ We will denote an adversarial view of such execution as $\mathsf{View}_{\mathcal{P},\mathcal{A}}(in_1, in_2, ...; out_1, out_2, ...)$. The distribution of this random variable is induced by the distribution of the vector $\vec{r}$ of random inputs of the adversary and the servers chosen uniformly among those vectors which cause the execution of protocol $\mathcal{P}$ in the presence of adversary $\mathcal{A}$ on inputs $in_1, in_2, ...$ to output values $out_1, out_2, ...$ as public outputs.

In the security proofs of our protocols, we will consider either an adversarial view of an execution of a protocol, or an adversarial view of a *simulation* of this protocol. The latter is defined as a computational history of the same adversary PPT TM $\mathcal{A}$, but whose interactive tapes are connected not to the actual network of players that follow the prescribed protocol (i.e. to the above PPT TM $\mathcal{P}$), but to a special-purpose PPT TM SIM called a *simulator*. We explain the usefulness of such simulators for our security proofs in Section 2.5 below. We denote the adversarial view of a simulation, where the adversary $\mathcal{A}$ interacts with the simulator SIM, as $\mathsf{View}_{SIM,\mathcal{A}}(in_1, in_2, ..., in_{SIM})$, where inputs in_1, in_2, ..., in_{SIM} are specified in the description of the simulation process (it must be specified there whether the inputs are public or private, and whether they are inputs to the adversary or to the simulator). We will also consider specific executions of the simulation process, subject to the constraint that the public outputs of this simulation attain particular values $out_1, out_2, ...$ We denote the adversary's view of such simulation as $\mathsf{View}_{SIM,\mathcal{A}}(in_1, in_2, ..., in_{SIM}; out_1, out_2, ...)$. The public output of a simulation is defined similarly to the public output of any protocol, i.e. as a majority of outputs tagged as "public output" by the players controlled by the adversary and by the virtual players controlled by the simulator. These values will often be either broadcast by each player (including the virtual players in the simulation), or they will be computable from such broadcasts. In other words, they will be computable from the information sent on the public channels.

Defining Secure Threshold Cryptosystems. We can define the (chosen ciphertext) security of a threshold cryptosystem in the same manner as we defined the (adaptive chosen message) security of a threshold signature scheme above. Namely, we can extend the [NY90] definition of the chosen ciphertext security of a standard cryptosystem to the threshold setting. Such definition of a security of a threshold cryptosystem was given in [SG98].

2.5 Methodology for Proving Security of Threshold Schemes

We discuss the general technique of proving security of a threshold signature or decryption scheme by reducing it to the security of the underlying public-key scheme. For simplicity and concreteness, our presentation focuses mainly on threshold signature schemes. However, most of the issues we raise are applicable to other threshold functions and cryptosystems.

Proof by Reduction and the Role of Simulators and Extractors. We use the usual "reductionist" approach in proofs of security of threshold signature schemes. To prove unforgeability, we show that given a t-threshold adversary $\mathcal{A}$ that forges signatures when interacting with a threshold signature scheme TSS, we can construct a forger $\mathcal{F}$ that forges signatures of the underlying centralized signature scheme SS. (We assume here that TSS is a threshold realization of some standard centralized signature scheme SS.) Thus we *reduce* the claim that the threshold signature scheme TSS is unforgeable to the assumption that the centralized signature scheme SS is unforgeable.

Recall that a (chosen-message attack) unforgeability of a centralized signature scheme SS is formalized via an interaction between some forger $\mathcal{F}$ (PPT TM) and an oracle $\mathcal{O}_{SS}$ which implements the SS scheme [GMR88]. The interaction proceeds as follows. On common input a security parameter k, the oracle $\mathcal{O}_{SS}$ runs the key generation algorithm specified by SS, gives the resulting public key v to $\mathcal{F}$ and keeps the secret key s to itself. Then $\mathcal{F}$ is allowed to ask for signatures on messages $m_1, m_2, ...$ of its choice, and for each m_i it sends to $\mathcal{O}_{SS}$, the oracle runs the signing algorithm specified by SS on m_i and s, and sends back the resulting signature u_i to $\mathcal{F}$. We say that scheme SS is *(chosen-message attack) unforgeable* if for every PPT TM $\mathcal{F}$ the probability that in the above interaction $\mathcal{F}$ outputs at some point a valid signature u under key v on some message m, where m is not equal to any message m_i that $\mathcal{O}_{SS}$ signed for $\mathcal{F}$, is a negligible function of the security parameter k.

A key ingredient in the reduction of unforgeability of TSS to unforgeability of SS is a *simulation of the adversary's view* in its run against the threshold scheme.[12] The forger $\mathcal{F}$ plays the role of a *simulator* of an adversarial view of the threshold scheme. $\mathcal{F}$ builds a virtual distributed environment composed of n virtual servers and interacts with the adversary $\mathcal{A}$ on their behalf. Note that $\mathcal{A}$ is a PPT interactive algorithm and thus it can be used not only in an actual attack against a network of real players, as described in the "Adversarial View" paragraph in Section 2.4 above, but against a network of virtual players who all follow instructions of the forger $\mathcal{F}$ (we will call these players "simulated" by $\mathcal{F}$). Adversary $\mathcal{A}$ expects first to participate in the distributed key generation and then in a series of executions of the distributed signature protocol invoked on messages of his choice.[13] Therefore $\mathcal{F}$ has to first simulate to $\mathcal{A}$ an execution of the distributed key generation protocol. However, to later translate $\mathcal{A}$'s forgery against TSS into a forgery against SS, forger $\mathcal{F}$ has to simulate to $\mathcal{A}$ an execution of the distributed key generation that results in the public key v chosen by the $\mathcal{O}_{SS}$ oracle. Then $\mathcal{A}$ will successively invoke the distributed signature protocol on messages $m_1, m_2, ...$ of his choice, and $\mathcal{F}$, having access to the oracle $\mathcal{O}_{SS}$, will obtain a signature u_i on each such message m_i under the public key v, and it will *simulate* to A its view of an execution of the distributed signature protocol which on input m_i outputs u_i. Eventually, if $\mathcal{A}$ outputs a forgery in this simulated, virtual environment, $\mathcal{F}$ can output it as a forgery against SS.[14]

[12]The technique of simulation of adversary's view in a protocol was introduced by [GMR85].

[13]See the interaction between the adversary and the network of players implementing TSS described in the "Adversarial Run of a Threshold Signature Scheme" paragraph in Section 2.4.

[14]Such construction of the forger $\mathcal{F}$ is exhibited for example in the proof of unforgeability of the threshold DSS signature scheme in Theorem 3, page 106. In the proof of Theorem 3 the forger $\mathcal{F}$ discussed above is denoted SIM, because, as we explained above, the forger *simulates* the execution of the threshold scheme to the threshold adversary $\mathcal{A}$.

The crucial element in the construction of this forger is a demonstration that the adversary's view of the simulated interaction is indistinguishable from its view of a real interaction with parties running the threshold scheme.[15] Thus the technical core of proofs of unforgeability of our threshold schemes is a construction of appropriate simulator algorithms. Such simulators must be able to generate views that are indistinguishable from the adversary's view of a corresponding protocol under specific input/output conditions. Namely, the run of the distributed key generation must be simulated to output a *given* public key chosen by the oracle that implements the underlying centralized scheme. Similarly, the distributed signature scheme must be simulated to output a particular signature given by the same oracle. We refer to this as the problem of *hitting* a particular value in the simulation.

A corresponding standard technique for proving *robustness* of a threshold scheme is to exhibit a *knowledge extractor* $\mathcal{E}$, which, on input an instance of some hard problem (e.g. the discrete logarithm problem), interacts with the adversary algorithm $\mathcal{A}$ by playing the part of the honest players in the protocol, and in case the adversary $\mathcal{A}$ succeeds in inducing the honest players into producing an invalid output, it extracts from the adversary's behavior a solution to the hard problem which was its input. By exhibiting such extractor, we *reduce* the robustness of our threshold protocol to some standard hardness assumption.

"Building Block" Subprotocols and their Simulators and Extractors. The threshold schemes we propose are constructed from "building block" subprotocols that perform simple arithmetic operations on shared data, for example create a random shared number, compute its inverse, multiply two shared numbers, etc. (See the discussion and the example of our methodology in Section 1.2.1.) Therefore our approach to exhibiting a simulator for a threshold signature scheme, and thus to proving its unforgeability, is to exhibit a simulator for every building block subprotocol this scheme is composed of, and to show that a simulator of the threshold scheme can be built by composing the simulators of the building block subprotocols.

A simulator of each building block subprotocol exhibits a *secrecy* property of this subprotocol, which states that the adversary learns nothing from the protocol beyond the public inputs and outputs of this protocol, or in other words, that the adversary learns as much by participating in the threshold implementation of some arithmetic operation as he would learn from observing this operation as a black-box. To exhibit such secrecy a simulator must produce the adversarial view of the protocol given as inputs the protocol's *public inputs and outputs*.

However, many simulators of our building-block subprotocols take more as an input than the public inputs and outputs of that subprotocol. This happens if an execution of some protocol always follows an execution of some other protocol. We will explain this by taking as an example the simulator SIM_{Exp} of a threshold exponentiation protocol Exp (Figure 4-9). To understand why this simulator takes more than a public output $A = m^a$ of the Exp protocol we must consider that protocol Exp can be executed only subsequently to some previous threshold protocol which establishes a secret-sharing of an exponent a. Let us consider a distributed key generation protocol DKG (Figure 4-10). It consists of two subprotocols, a "joint" verifiable secret-sharing protocol RVSS (Figure 4-6) in which, on the public input a Pedersen commitment instance, the players generate a sharing of a

[15] The notion of computational indistinguishability was introduced in [Yao82] and [GM82].

random number, a secret key x, and protocol Exp which computes the public key $y = g^x$ from that sharing. These protocols are all discussed in detail in Chapter 4. Here we want to point out that the RVSS protocol outputs an ensemble of some public data, some private data output by each player, and some data output by the adversary. We call this set of data a *joint secret-sharing* of the created value x, and we denote it as RVSS-data$[x]$. In the subsequent exponentiation protocol Exp, the inputs that each participating party takes to execute Exp are the outputs this party computed in the preceding protocol RVSS, i.e. its data in RVSS-data$[x]$. For the sake of this discussion let us treat as the only public output of Exp (and DKG) the publicly computed value $y = g^x$.[16]

Since the public output of DKG is a random value y in G_q, the input to simulator SIM_{DKG} of DKG just such random value y in G_q. The aim of SIM_{DKG} is to present to the adversary a view that is computationally indistinguishable from its view of a random run of DKG which outputs this y. (By convention, we denote such "target" inputs to the simulators with a star, and so in Figure 4-10 the input of SIM_{DKG} is denoted as y^*.) To simulate the DKG protocol, SIM_{DKG} first simulates the RVSS protocol. However, this simulation consists simply of executing this protocol on behalf of the uncorrupted players (see Figure 4-10). In particular, SIM_{DKG} gets the public outputs and the private outputs of the uncorrupted players of such execution of RVSS. SIM_{DKG} then passes these outputs, together with the target value y to the simulator SIM_{Exp} which simulates the second step of DKG, i.e. the Exp protocol. This is why a simulator of a building-block subprotocol like Exp can receive more inputs than the public inputs and outputs of this subprotocol. The same logic applies to almost all the building-block protocols we discuss in this thesis, e.g. a threshold multiplication protocol Mult or a threshold inverse-computation protocol Reciprocal. Such protocols are executed always within some larger protocol $\mathcal{P}$ which first creates secret-sharings of some values, e.g. secret-sharings of values to be multiplied by Mult or inverted by Reciprocal. Consequently, simulators of such protocols receive as inputs, from a simulator $\mathsf{SIM}_{\mathcal{P}}$ of $\mathcal{P}$, the data that corresponds to the private data of the uncorrupted players in such secret-sharings.

To prove *robustness* of a given building block protocol we either build an extractor for that protocol, or we show how robustness follows from the robustness of subprotocols from which the protocol in question is composed. The purpose of exhibiting an extractor algorithm for some protocol is to show that if some adversary $\mathcal{A}$ has a non-negligible probability of disrupting this protocol, then there exists an efficient algorithm which on input an instance of some hard computational problem can efficiently extract an answer to this problem from an interaction with the adversary $\mathcal{A}$.

[16]The reader might notice that an execution of Exp on input RVSS-data$[x]$ is not guaranteed to output g^x. However, for the sake of simplifying the present discussion, let's assume that this output is always computed.

Chapter 3

Preliminaries

We provide some preliminary definitions and we explain the notational conventions we use throughout the technical part of the thesis.

3.1 Definitions

In this work we concentrate on threshold schemes based on the discrete–logarithm problem.[1] We first define a Discrete–Log Instance:

Definition 5 (Discrete–Log Instance) *A discrete–log instance is a triple (p, q, g) such that p and q are primes, q divides $p - 1$, and g is a generator of a multiplicative subgroup of $\mathbb{Z}_p^*$ of order q, which we denote as G_q.*

We restrict ourselves to working in a prime-order subgroup G_q of $\mathbb{Z}_p^*$ instead of in $\mathbb{Z}_p^*$ directly, because prime order of the group simplifies both the threshold protocols and the proofs of their security. We note that most discrete-log based cryptosystems are either designed to work in this subgroup or can be modified to do so.

An instance of a discrete logarithm problem must be generated before any of our discrete-log based protocols is executed. It can be generated by any participating server, but since that server might be already corrupted from the start, we require it to use a *verifiable* discrete–log instance generation procedure:

Definition 6 (Verifiable Discrete–Log Instance Generator IG)
The verifiable discrete–log instance generator IG is a pair (G, V) of a probabilistic polynomial-time algorithm G, and a polynomial-time algorithm V, which satisfy the following two properties:

1. *$G(1^k)$ outputs a triple (p, q, g) and a string* proof *s.t. $V(1^k, p, q, g,$* proof*$)$ accepts*
2. *$V(1^k, p, q, g,$* proof*$)$ accepts if and only if (p, g, g) is a discrete–log instance and $|q| = k$*

[1] The DLog-based threshold schemes we present can be generalized to groups based on elliptic curves.

We note that such verifiable discrete–log instance generation procedures exist, for example the algorithm of [Bac85].

The robustness of the distributed protocols we present in this chapter relies on the assumption of hardness of computing discrete logarithms on DLog instances that are generated in our protocols. We first define a notion of a *negligible function*, namely a function which is asymptotically smaller than any polynomial:

Definition 7 (Negligible Function) *We call function $f(k)$* negligible *if for every polynomial $p(k)$, there exists k_0 s.t. $f(k) < 1/p(k)$ for all $k \geq k_0$.*

Definition 8 (Discrete–Log Instance Family) *We call an infinite sequence of strings $DL = (s_1, s_2, ...)$ a* discrete–log instance family *if there is a polynomial $p(z)$ s.t. each string s_k encodes a discrete–log instance (p, q, g) s.t. $|q| = k$ and $|p| \leq p(k)$. We call the triple (p, q, g) encoded by s_k a* discrete–log instance of security parameter k in DL.

Definition 9 (Discrete–Log Intractability Assumption) *For every discrete–log instance family DL, and every non-uniform family of probabilistic polynomial-time algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$, the probability $\mathsf{Prob}[\mathcal{A}_k(p, q, g, g^x \bmod p) = x]$, is a negligible function of k, where (p, q, g) is a discrete–log instance of security parameter k in DL, and the probability is taken over random bits of $\mathcal{A}$ and the choice of x distributed uniformly in $\mathbb{Z}_q$.*

We note that all known algorithms for computing discrete logarithms in $\mathbb{Z}_p{}^*$ run in time which is asymptotically larger than any polynomial in the length of the largest prime factor of $p - 1$ (see e.g. [Kob94]). In particular, if (p, q, g) is output by some verifiable discrete–log instance generator on input a security parameter k (even if its random choices are determined by the adversary), then $p - 1$ is divisible by prime q of length $|q| = k$. Therefore we can assume that the probability that any probabilistic algorithm running in time polynomial in k computes the discrete logarithm in group G_q is negligible in k.

We recall the definition of *statistical difference* between two random variables. This notion is used in the analysis of our distributed protocols. Recall that a support of a random variable is a set of values the variable can attain.

Definition 10 (Statistical Difference) *If X and Y are two random variables with finite support, then the* statistical difference *$\delta(X, Y)$ is defined as*

$$\delta(X, Y) = \frac{1}{2} \sum_{\alpha} |\mathsf{Prob}[X = \alpha] - \mathsf{Prob}[Y = \alpha]|$$

Variable α in the above sum ranges over the union of the supports of the two random variables.

Pedersen's Trapdoor Commitment Scheme. An important tool used throughout the threshold schemes we discuss in this thesis is a trapdoor commitment scheme due to Pedersen [Ped91a]. We briefly summarize this scheme here below.

A commitment scheme is a protocol between two parties S and R, and the scheme of Pedersen works as follows. On instance (p, q, g, h), where (p, q, g) is a discrete–log instance

of an appropriate security parameter, and h is a random element in $\mathbb{Z}_q$, party S can commit itself to some string $x \in \mathbb{Z}_q$ by picking a random $\hat{x}$ in $\mathbb{Z}_q$ and sending to R value $C = g^x h^{\hat{x}} \bmod p$. Party R learns no information about x because for every $g \neq h$ in G_q, for every $x \in \mathbb{Z}_q$ there exists a unique $\hat{x}$ s.t. $C = g^x h^{\hat{x}} \bmod p$. After this "commit stage", S can decommit and reveal x by sending $(x, \hat{x})$ to R. Therefore this scheme protects the secrecy of the commitment information in the information theoretic sense.

The second essential property of a commitment scheme is the assurance it gives to party R that S, once committed to some value, cannot decommit itself in two different ways. Pedersen's scheme guarantees this property in under the discrete-log assumption in the following sense. Note that if $g^x h^{\hat{x}} = g^{x'} h^{\hat{x}'}$ then $\log_g h = (x - x')/(y' - y)$, and therefore if discrete–log is intractable then for all efficient algorithms S*, for any discrete–log instance (p, q, g) of security parameter k, and for h a randomly chosen element in G_q, there is at most negligible probability (in k) that S* chooses $C \in G_q$ from which it can successfully decommit itself in two different ways, i.e. as $x, \hat{x}$ or $x', \hat{x}'$ where $x \neq x'$.

The commitment scheme given by Pedersen has an additional property that together with the public instance (p, q, g, h) there can be generated a "trapdoor" value which allows an efficient algorithm to successfully decommit itself to *any* string x. In the case of the scheme by Pedersen, this trapdoor value is $\sigma = \log_g h$. Note that knowing σ allows S to open $C = g^x h^{\hat{x}} = g^{x+\sigma\hat{x}}$ to any $x' \in \mathbb{Z}_q$ by sending to R a pair $(x', \hat{x}')$ where $x' + \sigma\hat{x}' = x + \sigma\hat{x}$.

All three properties of Pedersen's commitment scheme are crucial for security of Pedersen's Verifiable Secret Sharing protocol (see Section 4.2.4), which is a basic building block of threshold protocols we discuss in this thesis.

Representation. In our protocols and security proofs we will use the notion of a representation of one number in $\mathbb{Z}_p{}^*$ as a product of exponentiations of some other numbers in $\mathbb{Z}_p{}^*$. Let $g_0 \in G_q$ and $g_1,\ldots,g_n$ be n distinct elements in G_q. We say that $\alpha_1,\ldots,\alpha_n \in (\mathbb{Z}_q)^n$ is a *representation of* g_0 *in bases* $g_1,\ldots,g_n$, if $g_0 = g_1^{\alpha_1} \cdots g_n^{\alpha_n} \bmod p$. Note that every representation of g_0 in $g_1,\ldots,g_n$ gives a linear equation $\alpha_1 x_1 + \cdots + a_n x_n = 1 \bmod q$ for variables $x_i = \log_{g_0} g_i \bmod p$.

3.2 Notational Conventions

We bring together the notational conventions we use in the technical part of this thesis.

Notation for Modular Operations. From now on all exponentiation and discrete logarithm operations will be implicitly carried out modulo p, and hence we will write $y = g^x$ and $x = \log_g y$. All mathematical operations in this section are always carried either modulo p or q, and most of the time we abbreviate the notation and skip the specification of which of the two modulus a given operation needs. The rule of thumb is that operations on secret data, e.g. secrets, shares, private random values, are carried out modulo q, and operations on public data, e.g. commitments or public verification values, are carried out modulo p.

Conventions for Protocols and Simulators. All the threshold protocols discussed in this thesis take as their public input a discrete–log instance (p, q, g). The sole exception is the discrete–log instance generation protocol DL-IG. Furthermore, all the threshold protocols we discuss need as a public input a second element h of G_q. Values (p, q, g, h) together form an instance of a Pedersen commitment scheme [Ped91a]. Again, the only protocol

we discuss which does not take h as an input is the h-generation protocol h-IG. These initialization protocols are discussed in Chapter 7. We refer to a composition of DL-IG and h-IG as to a Pedersen commitment instance generation protocol Ped-IG. In many threshold protocols we present, the commitment instance (p, q, g, h) is not explicitly mentioned as a public input to a protocol because it is subsumed by other inputs, e.g. by a secret-sharing data-structure RVSS-data, defined in Figure 4-7, which contains values (p, q, g, h) as public data.

Sometimes we accompany some distributed protocol $\mathcal{P}$ we propose by a description of a *simulator* $\mathsf{SIM}_{\mathcal{P}}$ which is needed in an analysis of the security of this protocol. The process of simulation is a computation of two interactive algorithms, the simulator SIM and the adversary $\mathcal{A}$, where the simulator controls the uncorrupted players, and the adversary controls the corrupted players. Therefore a description of a simulation process is similar to a description of the protocol itself. (See the description of the adversary as an interactive algorithm in Section 2.4 and the discussion of the simulation process in Section 2.5.)

In the descriptions of the simulation processes we denote the set of corrupted players as *Bad* and the set of uncorrupted players as *Good*, where $Bad \cup Good = \{P_1, ..., P_n\}$. In Chapter 4 we assume that the adversary is static, and therefore in the proofs of security in Chapter 4 we can assume that the adversary corrupts the maximum allowed threshold of players before the protocol starts. A static adversary must decide on the identity of the corrupted players before the protocol starts. Even if the adversary physically corrupts them only at some later point in the protocol, we can still model it as an immediate corruption and just give the adversary the extra knowledge for free. In particular, in the descriptions of the simulators in that chapter we will assume that $|Bad| = t$.

This changes in Chapters 5 and 6 where we consider an adaptive adversary who does not have to choose the identities of players in set *Bad* at the beginning of the protocol execution. In these two chapters *Bad* and *Good* will denote, unless otherwise stated, sets of the *currently* corrupted and uncorrupted players.

Concise Dictionary of Terms

For a quick reference we provide a list of terms which we commonly use when discussing threshold protocols:

- *protocol*: Any efficient (i.e. probabilistic polynomial time) protocol, i.e. a (PPT) interactive algorithm which specifies the actions of the n participating players.

- *simulator*, *extractor*: Interactive Probabilistic Polynomial Time Turing Machines (PPT TM), i.e. efficient interactive algorithms. The notion of a simulator and an extractor is discussed further in Section 2.5.

- *adversary*: A non-uniform family of interactive PPT TMs, denoted $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$. We model the adversary as following the instructions of algorithm $\mathcal{A}_k$ when the public input to the distributed protocol which the adversary attacks is a security parameter k. Our way of modeling an adversary is described formally in Section 2.4, with more discussion in Sections 2.1-2.2.

- *adversarial history* or *adversary's view* (of either a protocol or a simulation): A computational history of the adversary, during his interaction with the servers that implement the protocol or with a simulator algorithm. This interaction is explained in Section 2.4. Note that the lenght of the adversarial history is upper-bounded by some fixed polynomial in the security parameter k, because an adversary is a probabilistic polynomial-time algorithm.

- *adversary's input and output*: During either the execution or the simulation of any protocol, the adversary's input always includes an adversarial history from executions of any previous protocols. The adversary's output consists of his computational history at the time the protocol terminates.

- *public output*: In most protocols every uncorrupted player tags some of its outputs as "public". A public output is defined as a majority of such values marked by all the participating players. Our protocols maintain it as an invariant that all honest players mark the same value as a "public output", and therefore, since the uncorrupted players are always a majority, a public output is well-defined. (This invariant holds often only under an assumption of a discrete logarithm intractability, and often except for probability which is at most negligible in the security parameter.)

- *DLog instance of security parameter k*: A discrete-log instance (p, q, g) of parameter k in some Discrete-Log Instance Family DL. In particular, the bit-length of prime q is equal to the security parameter k, while the length of prime p is bound by some fixed small polynomial in k (see Definition 6 above).

- *negligible function*: A function of a security parameter k which is asymptotically smaller than an inverse of any polynomial. I.e., $f(k)$ is negligible if for any polynomial $p(k)$, there exists k_0 s.t. for all $k \geq k_0$, $f(k) < 1/p(k)$.

Chapter 4

Static Threshold Cryptosystems

4.1 Introduction

In this chapter we present building blocks of discrete-log based threshold schemes secure in a static adversarial model. Since in the static model the adversary picks the threshold of players it wants to corrupt before the protocol starts, we will denote by *Bad* the set of those corrupted players and we will assume the worst case that $|Bad| = t$.

The chapter is organized as follows. In Section 4.2 we construct a protocol for threshold key generation for discrete-log based schemes. In Section 4.3 we show a threshold DSS signature generation protocol which is resilient against $n/4$ threshold of faults. These two protocols together form threshold DSS1 signature scheme TSS which is as secure in the presence of a $n/4$-threshold adversary as a standard DSS in the presence of an adversary who cannot corrupt the central server but can stage a chosen message attack. Then in Section 4.4 we show how to improve the resistance of this threshold scheme to withhold a $n/2$ threshold of faults. We include the $n/4$-threshold protocols because they are simpler than the $n/2$-threshold ones. In particular they rely on a much simpler shared multiplication subprotocol which tolerates however only $n/4$-threshold of faults. We then show that the resilience of the DSS scheme becomes optimal if this building-block protocol is replaced with a more involved but optimally-resilient shared multiplication protocol.

We note that all the protocols discussed in this chapter rely on public input an instance of a Pedersen commitment scheme. A distributed initialization protocol which securely generates such an instance is described in Chapter 7.

This chapter is based on material published in [GJKR99] (Section 4.2), [GJKR96b] (Section 4.3), and [CGJ+99] (Section 4.4).

4.2 Distributed Key Generation for DLog-based Schemes

4.2.1 Introduction

We first present a protocol for secure Distributed Key Generation for DLog-based schemes, for example for threshold DSS. We abbreviate a protocol with this functionality as DKG. Distributed key generation is a main component of threshold cryptosystems. It allows a set of n servers to jointly generate a pair of public and private keys according to the distribution

defined by the underlying cryptosystem without having to ever compute, reconstruct, or store the secret key in any single location, and without assuming any trusted party (dealer). While the public key is output in the clear, the private key is maintained as a (virtual) secret shared via a threshold scheme. In particular, no attacker can learn anything about the key as long as it does not break into a specified number, $t+1$, of servers. This shared private key can be later used by a threshold cryptosystem, e.g., to compute signatures as in the threshold DSS signature protocol of Section 4.3, without ever being reconstructed in a single location. For discrete–log based schemes, distributed key generation amounts to generating a secret-sharing of a random, uniformly distributed value $x \in \mathbb{Z}_q$, and making public the value $y = g^x$. We refer to such a protocol as DKG.

A DKG protocol must be robust, i.e. it must be able to run in the presence of a malicious adversary who corrupts a fraction (or threshold) of the players and forces them to follow an arbitrary protocol of his choice. Informally, we say that a DKG protocol is secure if the output of the non-corrupted parties is correct (i.e. the shares held by the good players define a unique uniformly distributed value x and the public value y satisfies $y = g^x$), and the adversary learns no information about the chosen secret x beyond, of course, what is learned from the public value y.

Solutions to the shared generation of private keys for discrete-log based threshold cryptosystems [DF89] have been known and used for a long time. Indeed, the first DKG scheme was proposed by Pedersen in [Ped91b]. It then appeared, with various modifications, in several papers on threshold cryptography, e.g., [CMI93, Har94, LHL94, GJKR96b, HJJ+97, PK96, SG98], and distributed cryptographic applications that rely on it, e.g., [CGS97]. Moreover, a secure DKG protocol is an important building block in other distributed protocols for tasks different than the generation of keys. One example is the generation of randomizers in discrete-log based signature schemes (for example the r value in a (r, s) DSS signature as in Section 4.3). Another example is the generation of the refreshing polynomial in proactive threshold schemes discussed in Appendix B.

The basic idea in Pedersen's DKG protocol [Ped91b] (as well as in the subsequent variants) is to have n parallel executions of Feldman's Verifiable Secret Sharing (VSS) protocol [Fel87] (presented below in Figure 4-1) in which each player P_i acts as a dealer of a random secret x_i that he picks. The secret value x is taken to be the sum of the properly shared x_i's. Since Feldman's VSS has the additional property of revealing $y_i = g^{x_i}$, the public value y is the product of the y_i's that correspond to those properly shared x_i's.

In this paper we show that, in spite of its use in many protocols, Pedersen's DKG cannot guarantee the correctness of the output distribution in the presence of an adversary. Specifically, we show a strategy for an adversary to manipulate the distribution of the resulting secret x to something quite different from the uniform distribution. This flaw stresses a well-known basic principle for the design of cryptographic protocols, namely, that secure components can turn insecure when composed to generate new protocols. We note that this ability of the attacker to bias the output distribution represents a flaw in several aspects of the protocol's security. It clearly violates the basic correctness requirement about the output distribution of the protocol; but it also weakens the secrecy property of the solution. Indeed, the attacker acquires in this way some a-priori knowledge on the secret which does not exist when the secret is chosen truly at random. Moreover, these attacks translate into flaws in the attempted proofs of these protocols; specifically, they show that

simulation arguments used to prove the secrecy of these protocols must fail.

In contrast to the above, we present a protocol that enjoys a full proof of security. We first present the formal requirements for a secure solution of the DKG problem, then present a particular DKG protocol and rigorously prove that it satisfies the security requirements. In particular, we show that the output distribution of private and public keys is as required, and prove the secrecy requirement from the protocol via a full simulation argument. Our solution is based on ideas similar to Pedersen's DKG (in particular, it also uses Feldman's VSS as a main component), but we are careful about designing an initial *commitment phase* where each player commits to its initial choice z_i in a way that prevents the attacker from later biasing the output distribution of the protocol. For this commitment phase we use another protocol of Pedersen, i.e., Pedersen's VSS (Verifiable Secret Sharing) protocol as presented in [Ped91a]. Our solution preserves most of the efficiency and simplicity of the original DKG solution of [Ped91b], in particular it has comparable computational complexity and the same optimal threshold of $t < n/2$.

Organization: In Section 4.2.2 we define a secure DKG protocol. In Section 4.2.3 we describe previously proposed solutions to the DKG problem, and we show where they fail. Section 4.2.4 presents an important subprotocol used by this solution, namely a Verifiable Secret Sharing [VSS] by Pedersen, while Section 4.2.5 presents the secure DKG protocol and its full analysis.

4.2.2 Requirements of a Secure DKG Protocol

In this section we define the minimal requirements for a secure distributed key generation protocol. As we mentioned above, distributed generation of keys in a discrete–log based scheme amounts to generating a discrete-log instance (p, q, g), a secret-sharing of a random, uniformly distributed value $x \in \mathbb{Z}_q$, and making public the value $y = g^x$. The protocol can also generate some other public and private outputs which allow subsequent efficient and robust reconstruction (or efficient and robust *use*, as in the threshold signature or decryption schemes) of the shared secret x.

As described in Section 2.4, we model the adversary as a non-uniform family of interactive PPT TM's $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$, and we model an execution of the DKG protocol *in the presence of an adversary* as an interaction of two interactive PPT algorithms, the adversarial algorithm $\mathcal{A}$ and an algorithm which implements the protocol DKG on behalf of the uncorrupted players. In the definition below we use the notions of an adversarial view defined in Section 2.4. See also Section 2.5 for a discussion of the notion of an adversarial view during a simulation of a protocol. Recall also that function $f(k)$ is called *negligible* if for every polynomial $p(k)$, there exists k_0 s.t. $f(k) < 1/p(k)$ for all $k \geq k_0$.

Definition 11 *A DKG protocol is a probabilistic polynomial-time protocol run by n servers on public input a discrete-log instance (p, q, g). A DKG protocol is executed in the presence of an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$, We denote the private outputs of the n servers as $s_1, ..., s_n$.*

We call a DKG protocol DKG *t-*secure *in a given adversarial model (static, adaptive erasure-enabled, adaptive erasure-free) if it satisfies the following correctness and secrecy requirements:*

Correctness: *For every t-threshold adversary $\mathcal{A}$ in a given adversarial model, and for every discrete-log instance family DL, protocol* DKG *satisfies the following requirements:*

(C1) Except for probability at most negligible in k, a random execution of DKG *on input a discrete-log instance (p, q, g) of security parameter k in DL is* sharing-successful, *which we define as an instance of* DKG *where the outputs of the uncorrupted players satisfy the following property:*

The outputs s_i of the uncorrupted servers contain values $\alpha_i \in \mathbb{Z}_q$ which form a Shamir secret-sharing over group $\mathbb{Z}_q$. Namely, if there exists a unique t-degree polynomial f over $\mathbb{Z}_q$ such that $f(i) = \alpha_i \bmod q$ for each i, and thus all subsets of $t+1$ shares of the honest players define the same unique shared secret key $x = f(0) \bmod q$.

(C1') There is an efficient protocol Rec *such that if an execution of* DKG *is sharing-successful and it is followed by* Rec *on its outputs, then, except for the probability negligible in k, the public output of* Rec *is the unique value x defined by property (C1).*

(C2) For every (p, q, g) in DL the distribution of values x defined by a random sharing-successful execution of DKG *is uniform in $\mathbb{Z}_q$. (In other words, this distribution of x is defined by a run of* DKG *where the vector of random inputs of the adversary and the players is chosen uniformly among those that lead to sharing-successful executions of* DKG.*)*

(C3) Except for probability at most negligible in k, a random execution of DKG *on input a discrete-log instance (p, q, g) of security parameter k in DL is* successful, *which we define as a sharing-successful instance of* DKG *where the outputs of the uncorrupted players satisfy the following two additional properties:*

- *Outputs s_i of the uncorrupted players contain the same value of a public key $y \in G_q$. Thus in this case y can be called a* public output *of* DKG *(see Remark 1 below).*
- *$y = g^x$ where x is defined by the sharing-successful property of this* DKG *instance.*

Secrecy: *There exists a simulator* SIM, *such that for every t-threshold adversary $\mathcal{A}$ in a given adversarial model, for every discrete-log instance (p, q, g) of security parameter k, and for every $y \in G_q$, the following two distributions are identical:*

- $\mathsf{View}_{DKG,\mathcal{A}}((p, q, g); y)$, *an adversarial view of a random execution of protocol* DKG *which on public input (p, q, g) produces the public output y.*
- $\mathsf{View}_{SIM,\mathcal{A}}((p, q, g), y; y)$, *an adversarial view of a random simulation of* DKG *(i.e. an interaction of* SIM *and $\mathcal{A}$) which on public input (p, q, g) and on* SIM*'s private input y produces public output y.*

Less Formally: There exists a simulator which on input y simulates to the adversary his view of a random instance of protocol DKG that outputs such y.
Informally: The adversary learns nothing about the secret-shared value x generated in the protocol except for what is implied by the public output $y = g^x \bmod p$.

Remark 1. Recall from Section 2.4 that by convention we define a public output of a protocol as a majority of values marked as "public outputs" by the participating servers. Therefore if DKG is a secure DKG protocol, we can call the value of the public key y defined by the correctness condition (C3) a public output of DKG.

Remark 2. Correctness conditions (C1), (C2), and (C3), form a minimal set of requirements needed in all known applications of such a protocol. In many applications a stronger version of (C1) is desirable, namely condition (C1'), which reflects two additional aspects: (1) It requires the existence of an *efficient procedure* to build the secret x out of any $t+1$ shares of honest players; and (2) it requires this procedure to be *robust*, i.e. the reconstruction of x should be possible also in the presence of malicious parties that try to foil the computation. We will show that property (C1') is useful not only in applications that require explicit reconstruction of the secret, but also in applications (such as threshold cryptosystems) that use the secret x in a distributed manner (without ever reconstructing it) to compute some cryptographic function, e.g. a signature.

Remark 3. Alternatively, we can define the Distributed Key Generation protocol as starting on input a security parameter k alone, and generating first a discrete-log instance (p, q, g) s.t. $|q| = k$. This is an approach we take in the definition of a threshold signature scheme (Definition 4, Section 2.4), where a key generation protocol Key-Gen is assumed to be running on public input 1^k only. In the Definition 11 above we consider a DKG protocol as taking (p, q, g) as inputs because this definitional choice allows us to express the secrecy property of the Distributed Key Generation protocol *on its own*. We note that this definition seems unsatisfactory in the sense that we do not know how to apply it to prove that a threshold signature scheme, which utilizes a DKG protocol secure in the sense of this definition, is secure in the sense of Definition 4. A more robust definition of security of the DKG protocol would be preferable.

4.2.3 The Insecurity of a Common DKG Protocol

Feldman's Verifiable Secret Sharing. The simple but insecure DKG protocol originally proposed in [Ped91b] relies on a Verifiable Secret Sharing [VSS] protocols due to Feldman [Fel87]. Verifiable Secret Sharing (VSS) is a protocol between a dealer who wants to share a secret, and a group of n players which receive shares of this secret and can later reconstruct it. (In our uses of VSS protocols, the dealer is also one of the recipients).

In Shamir's original secret-sharing protocol [Sha79], to share a secret $x \in \mathbb{Z}_q$, the dealer chooses at random a polynomial $f(z)$ over Z_q of degree t, such that $f(0) = x$. It then secretly transmits to each player P_i a share $\alpha_i = f(i) \bmod q$. It is clear that t or less players have no information about the secret while $t+1$ can easily reconstruct it by polynomial interpolation.

It is well known however that in the presence of a malicious adversary, Shamir's secret sharing protocol is not secure. Indeed it is possible for a dealer to share values which do not lie on a polynomial of degree t. Also dishonest players may contribute incorrect shares at reconstruction time. A *verifiable secret sharing (VSS)* protocol is intended to prevent this possibility.

Feldman's verifiable secret sharing protocol [Fel87] extends Shamir's secret sharing

method in a way that allows the recipients of shares to verify that the shares they receive from the dealer are consistent (i.e., that any subset of $t+1$ shares determines the same unique secret), and to filter out the incorrect shares submitted by the dishonest players at reconstruction time. The protocol can tolerate up to $t < n/2$ malicious faults *including the dealer*. We present this protocol, denoted FelVSS, in Figure 4-1.

Protocol FelVSS, $(2t+1 \leq n)$

Public Input:	DL-instance (p, q, g)
Secret Input of Dealer P:	x the secret value in Z_q to be shared
Secret Output of Player P_i:	$\alpha_i = f(i)$ the share of x
	(or null if P_i disqualified the dealer P)
	[If $P_i = P$, also polynomial $f(z)$]
Public Output:	verification function $F(z) = g^{f(z)}$

(a) P chooses a random t-degree polynomial $f(z) = c_0 + c_1 z + ... + c_t z^t$ in Z_q, where $c_0 = f(0) = x$. P sends shares $\alpha_j = f(j)$ to each $P_j \neq P$ and broadcasts values $C_k = g^{c_k}$ for $k = 0, ..., t$, which defines a verification function $F(z) = \prod_{k=0}^{t} (C_k)^{z^k}$.

(b) Each P_j checks if the share received satisfies a verification equation $g^{\alpha_j} = F(j)$. If the check fails, P_j broadcasts a *complaint* against P.

(c) If P receives P_j's complaint then P broadcasts value α_j s.t. $g^{\alpha_j} = F(j)$. Each player P_j decides to disqualify the dealer P if and only if

- There were more than t complaints broadcast in Step (b), or
- P does not respond with α_j that satisfies the verification equation.

Reconstruction: Each P_i broadcasts share α_i s.t. $g^{\alpha_i} = F(i)$. Any $t+1$ shares that pass this verification are interpolated into a t-degree polynomial $f(z)$, and the secret x is recovered as $f(0)$.

Figure 4-1: Feldman's Verifiable Secret Sharing

Feldman's VSS is resistant to malicious faults. If there are less than $t < n/2$ faulty players, if the honest players do not disqualify the dealer then the reconstruction procedure is guaranteed to produce a unique secret x. Secondly, if the dealer is honest then the honest players always accept its sharing, and the reconstructed value a is equal to the dealer's input.

Notice however that the value x is shared with only computational secrecy, e.g., information $F(0) = g^{c_0} = g^x$ is leaked. However, it can be shown that an adversary that learns t or less shares cannot obtain any information on a beyond what can be derived from g^x. The proof of this fact uses a simulation argument which we sketch here. Given any t (or fewer) shares (known to the adversary) and $g^x = F(0)$, one can generate the distribution of the other public information in the protocol, i.e. values $C_1, \ldots, C_t$, as follows. Assume the known shares are $\alpha_1, \ldots, \alpha_t$. Thus we know $g^{f(i)} = g^{\alpha_i}$, $i = 1, \ldots, t$, as well as $g^{f(0)} = g^x$. This allows us to compute C_k for $k = 1 \ldots, t$ using the equation $C_k = g^{c_k} = \prod_{i=0}^{t} (g^{f(i)})^{\lambda_{ki}}$

where λ_{ki} are coefficients such that $c_k = \sum_{i=0}^{t} \lambda_{ki} f(i)$.[1]

Note that FelVSS allows a *trusted* dealer to share a randomly chosen secret key x among n parties in a way that produces a public key $y = g^x = F(0)$ as a public output of the protocol. In other words, FelVSS satisfies the security requirements for a DKG protocol as long as the dealer is not corrupted by the adversary.[2]

The Insecure DKG Protocol: Joint-Feldman. Pedersen [Ped91b] proposed the first solution to the Distributed Key Generation problem, i.e. the first DKG protocol. The proposal specifies the run of n parallel executions of FelVSS as follows. Each player P_i selects a random secret $x_i \in \mathbb{Z}_q$ and shares it among the n players using FelVSS. This defines the set *Qual* of players who were not disqualified as dealers, i.e. the set of players whose shared secrets can be robustly reconstructed. The random secret x is set to be the sum of the properly shared secrets, and each player can compute his share of x by locally summing up the shares he received. The value y can be computed as the product of the public values $y_i = g^{x_i}$ generated by each of the FelVSS protocols. Similarly, the verification function $F(z)$ necessary for robust reconstruction of x in FelVSS, can be computed from the verification values generated in each FelVSS.

In Figure 4-2 we present a simplified version of the protocol proposed in [Ped91b], which we call Joint-Feldman. By concentrating on the core of the protocol we are able to emphasize the central weakness in its design.[3]

An Attack Against Joint-Feldman. We show how an adversary can influence the distribution of the public output y of Joint-Feldman to a non-uniform distribution.

It can be seen, from the above description of the protocol that the determining factor for what the value x will be, is the definition of the set *Qual*. The attack utilizes the fact that the decision whether a player is in *Qual* or not, even given the fully synchronous communication model, occurs after the adversary has seen the values y_i of all players. The values y_i are made public in Step a and the disqualification of players occurs in Steps b-c. Using this timing discrepancy, the attacker can affect the distribution of the pair (x, y), for example by forcing the last bit of y to equal 0.

More specifically the attack works as follows. Assume the adversary wants to bias the distribution towards keys y whose last bit is 0. It assumes two faulty players, say P_1 and P_2. In Step a, P_1 gives players $P_3, ..., P_{t+2}$ shares which are inconsistent with his broadcast values, i.e. they do not pass the test of Step b. The rest of the players receive consistent shares. Thus, in Step b there will be t complaints against P_1, yet t complaints are not sufficient for disqualification. Now, at the end of Step a the adversary computes $\alpha = \prod_{i=1}^{n} y_i$ and $\beta = \prod_{i=2}^{n} y_i$. If α ends with 0 then P_1 will do nothing and continue the protocol as written. If α ends with 1 then the adversary forces the disqualification of P_1 in Step c. This is achieved by asking P_2 to also broadcast a complaint against P_1, which brings the number of complaints to $t + 1$. This action sets the public value y to β which

[1] If $[f(0), \ldots, f(t)]^T = A \cdot [c_0, \ldots, c_t]^T$, where A is a $(t+1)$ by $(t+1)$ Vandermonde matrix with i^k in row $i = 0, \ldots, t$ and column $k = 0, \ldots, t$, then $[c_0, \ldots, c_t]^T = A^{-1} \cdot [f(0), \ldots, f(t)]^T$ and λ_{ki}'s are entries of A^{-1}.

[2] More precisely, if the adversary can corrupt the dealer then the protocol does not achieve the secrecy property and the correctness property (C3).

[3] Many variants of this protocol have appeared in the literature and they each exhibit the same essential flaw. We review these variants and argue their flaws in Appendix E.

Protocol Joint-Feldman, $(2t + 1 \leq n)$

Public Input:	DL-instance (p, q, g)
Public Output:	set $Qual \in \{P_1, ..., P_n\}$ of qualified players
	verification function $F_x(z)$,
	including value $y = F_x(0) = g^x$,
	verification functions $F_{x_i}(z)$ for $P_i \in Qual$
Secret Output of Player P_i:	$f_{x_i}(z)$ the polynomial P_i secret-shared
	α_{ji} the share of P_j's polynomial for $P_j \in Qual$
	α_i the share of generated key x

Steps (a-c): Each P_i chooses its input x_i at random in Z_q, and performs the FelVSS protocol on that input as a dealer. All these instances of FelVSS proceed in parallel. We denote the polynomial used by P_i to secret-share x_i as f_{x_i}, its shares as α_{ij} for $j = 1, ..., n$, and its public verification function as $F_{x_i}(z)$. We denote $y_i = g^{x_i} = F_{x_i}(0)$.

At the end of Step (c) each player forms a set of non-disqualified players $Qual$ and computes its final secret-share as $\alpha_i = \sum_{P_j \in Qual} \alpha_{ji}$. The verification function $F_x(z)$ is defined as $\prod_{P_j \in Qual} F_{x_j}(z)$. The public value y is equal to $F_x(0)$. The secret shared value x itself is not computed by any party, but it is equal to $x = \sum_{P_j \in Qual} x_i$.

Figure 4-2: An *insecure* solution for a Distributed Key Generation Protocol

ends with 0 with probability 1/2. Thus effectively the attacker has forced strings ending with 0 to appear with probability 3/4 rather than 1/2.

Why the Simulation Fails. An attempt to prove this protocol secure would use a simulation argument. Following is an explanation of why such a simulator would fail. Consider a simulator SIM which receives the value y and needs to produce an adversarial view that looks like an execution of the Joint-Feldman protocol that ends with y as a public output. Assume that the adversary controls t players $P_1, ..., P_t$. In Step a, the simulator has to broadcast as part of the Feldman verification data the values $y_i = F_{x_i}(0)$ for each uncorrupted player P_i, and hence SIM has to commit itself to the values $x_i = \log_g y_i$ shared by each uncorrupted player P_i. Therefore, if SIM could predict in Step a the set Q of faulty players which will belong to set $Qual$ in Step c, then it would broadcast the values y_i, $P_i \in Good$, so that $(\prod_{P_i \in Q} y_i) \cdot (\prod_{P_i \in Good} y_i) = y \bmod p$. However, the attack described in the paragraph above can be easily extended to a strategy that allows the adversary to decide in Steps b-c on the set Q. Moreover, the adversary can behave in Step a in such a way that in Steps b-c it can decide to make Q *any* subset of $\{P_1, ..., P_t\}$. Therefore, from the point of view of SIM, there are 2^t possibilities for Q, and so it seems that SIM has no effective strategy to simulate this computation in polynomial time.

Essential Problem: No Perfectly Secret Commitment on the Inputs. Essentially, we are dealing here with the same pitfall that characterizes the distributed coin-flipping problem. A problem of distributed coin-flip, introduced by Blum for the case of two parties in [Blu82], asks whether a group of players, some of whom may be dishonest, can compute as a common output an unbiased random bit. A possible solution to both the distributed

coin-flipping and the distributed key generation problem is to create a random number by letting each player contribute its own random number and then summing up the results. However, if the dishonest player can first see the contribution of the honest players and base his contribution on that knowledge, then the dishonest player will be able skew the result. The Joint-Feldman protocol allows the dishonest players to do just that because it is not perfectly secret and *not* non-interactive in the presence of faults. Namely, Joint-Feldman protocol has two bad properties: 1) The g^{x_i} information about each x_i is leaked in Step (a); and 2) Faulty players do not truly submit their x_i contributions to x in Step a of the protocol because each contribution of a faulty player can still be withdrawn during the complaint resolution procedure of Steps b-c. The two properties combined allow the faulty players to decide their contributions based on some knowledge of the contributions of the honest players even in the perfectly synchronous communication model.

4.2.4 Joint Sharing of a Random Secret and Distributed Coin-Flip

Once we identified the essential problem of the Joint-Feldman protocol, it is easy to fix it by following the paradigm of the original solution of [CGMA85] to the problem of distributed coin-flipping. This solution has also become a paradigm for general multi-party secure computation. Namely, the contributions x_i must be committed to in a way that is both perfectly secret and which guarantees that the contributions cannot be withdrawn. Perfectly secret verifiable secret sharing scheme implements a commitment which satisfies both of these two properties. The particular perfectly secret VSS which is secure against $n/2$-threshold of faults and enables efficient extraction of key $y = g^x$ from the shared values x_i is a VSS by Pedersen [Ped91a], which we denote PedVSS and present here in Figure 4-3.

In this section we first describe the properties of this protocol, and then we show that a protocol RVSS (Figure 4-6), which consists of parallel execution of PedVSS by each player on a random input x_i, generates a sharing of a random secret x defined as a sum of the x_i's shared by the non-disqualified dealers. Because the generated shared secret is random and unbiased, we will refer to the RVSS protocol followed by reconstruction of x as to a Distributed Coin-Flip protocol, even though x is not a random bit but a number uniformly distributed in group $\mathbb{Z}_q$. The RVSS protocol is secure under parallel composition, which makes it easy to use in threshold schemes, where it is often useful to generate many random shared numbers at the same time. In Section 4.2.5 we will show how $y = g^x$ can be quickly extracted from the data generated by RVSS, and we argue why the resulting protocol, denoted DKG, is a secure Distributed Key Generation protocol for DLog-based threshold schemes.

Pedersen's Verifiable Secret Sharing

We present the Pedersen's VSS protocol, denoted PedVSS, in Figure 4-3. The PedVSS protocol requires, in addition to the parameters (p, q, g) which are inherent to the DKG problem, a second generator h of group G_q. Values (p, q, g, h) together form an instance of Pedersen's perfectly secret *Trap-door Commitment* which is used heavily in PedVSS. This element h does not need to be chosen uniformly in G_q, but it must be created so that under the discrete logarithm intractability assumption the adversary cannot find $\log_g h$, the discrete logarithm of h relative to the base g. In Chapter 7 we describe a distributed

protocol Ped-IG which picks an instance of a Pedersen commitment in a way which achieves the above property.

We call the ensemble of outputs of the PedVSS protocol, i.e. the public output and the private outputs of all the parties, a *secret-sharing*, and we denote it by PedVSS-data, or PedVSS-data$[a]$, PedVSS-data$[b]$, etc, where "a" and "b" (and the corresponding Greek letters "α", "β",...) are labels with which we designate all the data in the PedVSS-data ensemble.[4] We call such ensemble a *correct* secret-sharing if the outputs satisfy the correctness properties described in Figure 4-4. We call an execution of PedVSS which outputs a correct secret-sharing *successful*. In other words, this is an execution in which the adversary didn't manage to *cheat*, i.e. to prevent the outputs of PedVSS from forming a correct secret-sharing. By $\mathcal{PVSS}\text{-}\mathcal{DATA}_{(p,q,g,h)}$ we denote a set of all correct secret-sharings PedVSS-data which contain a Pedersen commitment instance (p, q, g, h) as its public data.

In Lemmas 1 and 2 below we show two important robustness properties of the PedVSS protocol. First, under the discrete-log assumption, a random execution of PedVSS is successful, i.e. its outputs PedVSS-data form a correct secret-sharing, except for probability negligible in the security parameter k. In other words, if discrete logarithm is hard then a probabilistic polynomial-time adversary has only a negligible probability of cheating in the PedVSS protocol. Secondly, under the discrete-log assumption, the unique secret-shared value x defined by a correct secret-sharing PedVSS-data$[x]$ can be reconstructed by the subsequent PedVSS-REC protocol, again except for negligible probability.

In Lemma 3 we express the secrecy property of PedVSS. This property says that PedVSS hides the shared secret x in information theoretic sense, i.e. that there is no difference between the adversarial view of an execution of PedVSS which shares some given secret x from an execution which shares some other secret x^*. In Lemma 3 we phrase this property in a way that is useful for building more complicated distributed protocols using PedVSS as a tool. Namely, if an execution of PedVSS in which the dealer P shares some value x^* is followed by some distributed protocol $\mathcal{P}$ which takes the created secret-sharing PedVSS-data$[x^*]$ as an input, for example the reconstruction protocol PedVSS-REC which reconstructs the shared secret x^*, then an adversarial view of a series of executions of PedVSS on P's input x^* followed by PedVSS-REC, is identically distributed to an execution of PedVSS on some other input $x \in \mathbb{Z}_q$ of P followed by a *simulation* of the PedVSS-REC, where the value x^* is given to the simulator as a *target* to be reconstructed.[5]

In the threshold schemes we propose in this thesis, the secret-sharing PedVSS-data created by protocol PedVSS is used in a variety of other protocols, each of which has its corresponding simulation process. We thus need to phrase the above property of PedVSS without referring to any particular protocol $\mathcal{P}$. We do this by isolating a common element in the simulation processes of threshold protocols that use a secret-sharing PedVSS-data as their input. Namely, for each such protocol $\mathcal{P}$ we will argue that its simulator can perform

[4] An exception from this convention is the notation which we use for sharing of a secret denoted as x. We denote the sharing of x as PedVSS-data$[x]$, and we label all the data associated with this sharing with variable name x. However, for the lack of an appropriate Greek letter we label the *polynomial shares* associated with this sharing with α. See, for example, Figure 4-3.

[5] Note that an adversarial view in this statement is a random variable whose distribution is determined, in the first case by the uniform distribution of random inputs of the adversary and the uncorrupted players, and in the second case by the uniform distribution of random inputs of the adversary and the simulator.

protocol PedVSS on some input x, which creates a sharing PedVSS-data$[x]$, followed by a simulation of $\mathcal{P}$ in which the simulator, on input some target value x^* replaces PedVSS-data$[x]$ with an appropriately chosen secret-sharing PedVSS-data$[x^*]$ which shares x^* instead, and then simply performs the protocol $\mathcal{P}$ on this new secret-sharing. We isolate this process of replacement of data in the form of procedure $\mathcal{T}_{PedVSS}$ presented in Figure 4-5, which takes as inputs a given secret-sharing PedVSS-data$[x]$ and outputs its replacement PedVSS-data$[x^*]$. Note that the data which is visible to the adversary, i.e. the public data and the private data of the players controlled by the adversary, must remain the same in PedVSS-data$[x]$ and PedVSS-data$[x^*]$, so this "replacement" is always only a modification of the private data of the players controlled by the simulator.

The essence of the proof of Lemma 3 is that for any uncorrupted dealer P and every x and x^*, the distribution of PedVSS-data$[x^*]$ output by $\mathcal{T}_{PedVSS}$(PedVSS-data$[x], x^*$), where PedVSS-data$[x]$ is output by a random execution of PedVSS on P's input x, is the same as the distribution of PedVSS-data$[x^*]$ output by a random execution of PedVSS on P's input x^*.

This implies, and we state it in Lemma 3, that if we consider the following simulation of the (PedVSS;$\mathcal{P}$) sequence of protocols (for any $\mathcal{P}, x, x^*$), in which the simulator, on input x^*, the "target output of $\mathcal{P}$", first performs PedVSS on the uncorrupted dealer's input x, then modifies the private data of the uncorrupted players by replacing the resulting output PedVSS-data$[x]$ with PedVSS-data$[x^*]$ = $\mathcal{T}_{PedVSS}$(PedVSS-data$[x], \sigma, x^*$), and then performs $\mathcal{P}$ on PedVSS-data$[x^*]$, then the adversarial view of such simulation is the same (i.e. it is identically distributed) as the adversarial view of PedVSS performed on input x^* followed by $\mathcal{P}$. This complicated statement can be simply summed up as "sequence of protocols (PedVSS;$\mathcal{P}$) is simulatable". Note that the above simulation is possible only if the simulator is able to perform the $\mathcal{T}_{PedVSS}$ procedure, i.e. if the simulator knows the Pedersen commitment trapdoor value $\sigma = \log_g h$ (see Figure 4-5).

The proofs of both the secrecy and the robustness properties of PedVSS are based on the proofs that appear in [Ped91a]. Our contribution is the formalization these properties are given here, which enables us to argue that the PedVSS protocol can be combined with others into larger distributed protocols.

Definition 12 *We call an execution of protocol* PedVSS successful *if and only if its outputs* PedVSS-data$[a]$ *form a* correct *secret-sharing, which means that they satisfy the following properties:*

1. *All honest players make the same decision as to whether to disqualify the dealer.*

2. *If the dealer is not disqualified then each honest player P_i holds shares $\alpha_i, \hat{\alpha}_i$, s.t. these shares interpolate to unique t-degree polynomials $f_x(z), f_{\hat{x}}(z)$*

3. *If the dealer is not disqualified then all honest players hold the same verification function $F_x(z)$, such that $F_x(z) = g^{f_x(z)} h^{f_{\hat{x}}(z)}$.*

4. *If P is uncorrupted then the above polynomials $f_x, f_{\hat{x}}$ are chosen by P in Step a, and the secret-shared value $f_x(0)$ is equal to P's input x.*

Protocol PedVSS, $(2t+1 \leq n)$

Threshold Parameter:	t, the degree of the generated polynomials
Public Input:	Pedersen commitment instance (p, q, g, h)
Secret Input of Dealer P:	x the secret value in Z_q to be shared
Secret Output of Player P_i:	α_i, the polynomial share of x
	$\hat{\alpha}_i$, its associated randomness
	(or null if P_i disqualified the dealer P)
	[If $P_i = P$, also polynomials $f_x(z), f_{\hat{x}}(z)$]
Public Output:	verification function $F_x(z) = g^{f_x(z)} h^{f_{\hat{x}}(z)}$ (or null)

(a) P chooses two random polynomials $f_x(z), f_{\hat{x}}(z)$ over Z_q of degree t:

$$f_x(z) = c_0 + c_1 z + \ldots + c_t z^t \qquad f_{\hat{x}}(z) = \hat{c}_0 + \hat{c}_1 z + \ldots + \hat{c}_t z^t$$

where $c_0 = f_x(0) = x$. P sends shares $\alpha_j = f_x(j), \hat{\alpha}_j = f_{\hat{x}}(j)$ to each $P_j \neq P$ and broadcasts values $C_k = g^{c_k} h^{\hat{c}_k}$ for $k = 0, \ldots, t$, which defines a verification function $F_x(z) = \prod_{k=0}^{t} (C_k)^{z^k}$.

(b) Each P_j checks if the shares he received satisfy *Pedersen's verification equation*:

$$g^{\alpha_j} h^{\hat{\alpha}_j} = F_x(j) \tag{4.1}$$

If the check fails, P_j broadcasts a *complaint* against P.

(c) If P receives P_j's complaint then P broadcasts $(\alpha_j, \hat{\alpha}_j)$ s.t. $g^{\alpha_j} h^{\hat{\alpha}_j} = F_x(j)$. Each P_j disqualifies the dealer P and sets public output to null if and only if

- P received more than t complaints in Step c, or
- P does not respond with $(\alpha_j, \hat{\alpha}_j)$ that satisfy the verification equation.

Reconstruction Protocol PedVSS-REC:

Each P_i broadcasts values $(\alpha_i, \hat{\alpha}_i)$ s.t. $g^{\alpha_i} h^{\hat{\alpha}_i} = F_x(i)$. If all shares α_i that pass this verification interpolate into some t-degree polynomial $f_x(z)$, then the public output is $x = f_x(0)$. Otherwise, the public output is null.

Figure 4-3: (PedVSS, PedVSS-REC): Pedersen's Verifiable Secret Sharing

Lemma 1 (Robustness of PedVSS**)**
Consider an execution of (1) protocol Ped-IG *(Figure 7-1) which on public input* 1^k *outputs a Pedersen commitment* (p, q, g, h)*, (2) any other protocol* $\mathcal{P}$*, and (3) protocol* PedVSS *on public input* (p, q, g, h) *with outputs denoted* PedVSS-data$[x]$.

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/2$*-threshold adversary* $\mathcal{A}$*, the above instance of* PedVSS *is* successful *except for probability at most negligible in the security parameter* k.

Proof: First note that property (1) and (4) are trivially satisfied. Second, if the dealer P is uncorrupted then properties (2) and (3) follow. Third, note that if property (2) holds then

Correct **Secret-Sharing** PedVSS-data$[a]$ **of value** a

We call the data ensemble which satisfies the properties listed below a *correct* secret-sharing PedVSS-data$[a]$ of a. A set of such correct secret-sharings for a given Pedersen commitment instance (p, q, g, h) is denoted as $\mathcal{PVSS\text{-}DATA}_{(p,q,g,h)}$.

Threshold Parameter:	t, the degree of the secret-sharing polynomials
Identity of the dealer:	P, one of the players $P_1, ..., P_n$
Public Data:	Pedersen commitment instance (p, q, g, h) bit Q ($Q = 0$ iff dealer P is disqualified) verification function $F_a : \mathbb{Z}_q \rightarrow G_q$, s.t. $F_a(z) = g^{f_a(z)} h^{f_{\hat{a}}(z)}$ for some unique t-degree polynomials $f_a, f_{\hat{a}}$ s.t. $f_a(0) = a$ (or null if $Q = 0$)
Private Data of each $P_i \in$ *Good*:	shares $(\alpha_i, \hat{\alpha}_i)$ s.t. for all $P_i \in$ *Good*, $\alpha_i = f_a(i)$ and $\hat{\alpha}_i = f_{\hat{a}}(i)$ (or null if $Q = 0$)
Private Data of P **(only if honest)**:	"secret-sharing" polynomials $f_a, f_{\hat{a}}$
Private Data of the Adversary:	the adversary's *computational history*

Figure 4-4: Properties of a *correct* secret-sharing PedVSS-data

property (3) holds too. First note that all the uncorrupted players indeed hold the same verification function $F_x(z)$. Since $F_x(i) = g^{\alpha_i} h^{\hat{\alpha}_i}$ for every $P_i \in$ *Good*, then since $|Good| > t$ and since by property (2) values $\alpha_i, \hat{\alpha}_i$ for $P_i \in$ *Good* lie on t-degree $f_x(z)$ and $f_{\hat{x}}(z)$, then (3) follows. Therefore it remains to show that, under the discrete-log assumption, if the dealer is corrupted then property (2) still holds, except for probability which is at most negligible in the security parameter.

In other words, the lemma follows if we show how to violate the discrete logarithm intractability assumption if there exists an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$ who participates in an execution of Ped-IG followed by some other protocol $\mathcal{P}$ and then an execution of PedVSS, such that with probability not negligible in k, $\mathcal{A}$ "breaks" PedVSS, where by "breaking" PedVSS we mean that $\mathcal{A}$ causes the outputs of PedVSS to violate property (2). (We note that the fact that the Ped-IG and PedVSS protocols can be interleaved by any other protocol $\mathcal{P}$ does not influence the proof.)

We reduce such adversary to computation of discrete logarithm in a very simple way. Extractor $\mathcal{E}$ first follows the DL-IG part of Ped-IG to create a DLog instance (p, q, g), and then to compute $\log_g \tilde{g}$ on input a DLog challenge $\tilde{g}$ in G_q, it uses the simulator $\mathsf{SIM}^{(2)}_{h-IG}$ for the h-IG part of the Ped-IG protocol (see Figure 7-1) to generate Pedersen commitment value h for which it knows a representation a, b in bases $g, \tilde{g}$. In other words $\mathcal{E}$ simulates h-IG so that the simulated protocol generates a public output $h = g^a \tilde{g}^b$. We will show that if $\mathcal{A}$ manages to cheat in the subsequent execution of PedVSS, then $\mathcal{E}$ gets two different representations of some number, namely a Pedersen verification value $F_x(i)$ generated in this PedVSS, in bases g, h, which lets $\mathcal{E}$ compute $\log_g h$, and hence lets $\mathcal{E}$ compute also $\log_g \tilde{g} = a + b \log_g \tilde{g}$. Below we fill out the remaining technical details.

PedVSS-data[x] $\rightarrow$ PedVSS-data[x^*] **replacement procedure** $\mathcal{T}_{PedVSS}$

Input: (*Implicit:* identity of the dealer P)
public and private data of uncorrupted players in PedVSS-data[x]
target value x^*
Pedersen's trapdoor $\sigma = \log_g h$
Output: (private data of uncorrupted players in) secret-sharing PedVSS-data[x^*]

Let $Bad, Good$ be the identities of the corrupted and uncorrupted players ($|Bad| = t$).
Assume that the dealer P in PedVSS-data[x] is not corrupted.
Let $f_x, f_{\hat{x}}$ be the outputs of the (uncorrupted) dealer P in PedVSS-data[x].
$\mathcal{T}_{PedVSS}$ computes t-degree polynomials $f_x^*, f_{\hat{x}}^*$ s.t.:

$$f_x^*(i) = f_x(i) \text{ for } P_i \in Bad \text{ and } f_x^*(0) = x^*$$
$$f_x^*(z) + \sigma f_{\hat{x}}^*(z) = f_x(z) + \sigma f_{\hat{x}}(z) \text{ (for all } z)$$

$\mathcal{T}_{PedVSS}$ forms PedVSS-data[x^*] by replacing the private data of each player P_i in $Good$ with $\alpha_i^* = f_x^*(i)$ and $\hat{\alpha}_i^* = f_{\hat{x}}^*(i)$, and the data of dealer P with polynomials $f_x^*, f_{\hat{x}}^*$.

Figure 4-5: $\mathcal{T}_{PedVSS}$: Auxiliary procedure for simulation of PedVSS

Assume that there exits an adversary $\mathcal{A}$ that breaks property (2) with not negligible probability, i.e. that there exists polynomial $p_{\mathcal{A}}(z)$ s.t. for all k_0 there exists $k \geq k_0$ s.t. $\mathcal{A}$ breaks property (2) with probability at least $1/p_{\mathcal{A}}(k)$. Note that for every k there is a vector of random coins $\vec{r}_k$ utilized by the servers and the adversary during the DL-IG part of the Ped-IG protocol (see Figure 7-1), such that the DLog instance (p, g, q) produced by this DL-IG maximizes $\mathcal{A}$'s probability that statement (2) does not hold. Let the discrete-log instance family DL be the family of these instances, and let $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2, ...)$ be the family of algorithms s.t. each $\mathcal{A}'_k$ first *simulates* the run of DL-IG to $\mathcal{A}_k$, where each party is fed random inputs specified by the above vector $\vec{r}_k$, and then follows the algorithm of $\mathcal{A}_k$ during the execution of h-IG (the second part of Ped-IG) and the subsequent execution of $\mathcal{P}$ and then PedVSS, on the public input the above instance (p, q, g). If $\mathcal{A}$ makes statement (2) untrue with at least certain probability, than so does $\mathcal{A}'$.

We construct a PPT TM algorithm family $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2, ...)$, called "extractor", such that the probability Prob$[\mathcal{E}_k(p, q, g, g^x \bmod p) = x]$ is not a negligible function of k, where (p, q, g) is a discrete–log instance of security parameter k in DL, and x is uniformly distributed in $\mathbb{Z}_q$. The existence of such extractor breaks the discrete-log intractability assumption. We describe the algorithm $\mathcal{E}$ as follows. On input $(p, q, g, \tilde{g})$ where $\tilde{g}$ is picked at random in G_q and (p, q, g) is a DLog instance of security parameter k in DL, algorithm $\mathcal{E}$ simulates to $\mathcal{A}'$ the execution of h-IG by running the simulator $\mathsf{SIM}^{(2)}_{h-IG}$ of Figure 7-3 on input $(p, q, g, \tilde{g})$. Under the discrete-log assumption, by Lemma 36, page 177, $\mathsf{SIM}^{(2)}_{h-IG}$ runs in polynomial time and there is only a negligible statistical difference between the view of $\mathcal{A}'$ in this interaction and the view of $\mathcal{A}'$ in an execution of the h-IG protocol on input (p, q, g). In particular, this difference is smaller than $1/(2p_{\mathcal{A}}(k))$ for all large-enough k, and hence for all k_0 there exists $k \geq k_0$ s.t. the probability that $\mathcal{A}_k$ makes statement (2) untrue in the

subsequent execution of PedVSS is at least $1/(2p_{\mathcal{A}}(k))$. Furthermore, by the same lemma, there exists k_{SIM} s.t. for all $k \geq k_{SIM}$, $\mathcal{E}$ receives from $\mathsf{SIM}^{(2)}_{h-IG}$ values $a, b \in \mathbb{Z}_q$ such that $g^a \tilde{g}^b = h$ where h is output in this (simulated) run of h-IG, except for probability $1/(4p_{\mathcal{A}}(k))$. Therefore, for every k_0 there exists $k \geq k_0$ s.t. with probability at least $1/(4p_{\mathcal{A}}(k))$, $\mathcal{A}$ breaks property (2) in the subsequent PedVSS *and* $\mathcal{E}$ knows a, b s.t. $g^a \tilde{g}^b = h$.

Now, $\mathcal{E}$ follows the protocol $\mathcal{P}$ and then the protocol PedVSS on input (p, q, g, h) in the presence of $\mathcal{A}'$ on behalf of the uncorrupted players. We show that if $\mathcal{A}'$ makes (2) untrue in this interaction then $\mathcal{E}$ can compute $\sigma = \log_g h = a + b \log_g \tilde{g}$, and thus compute $\log_g \tilde{g}$. First note that set *Good* must have at least $t+1$ players. If $|Good| = t+1$ then (2) is trivially true. If $|Good| \geq t+2$ and (2) is not true, i.e. not all values α_i, $P_i \in Good$ lie on some t-degree polynomial, then we compute $\sigma = \log_g h$ as follows. Note that since P is not disqualified, shares $(\alpha_i, \hat{\alpha}_i)$, $P_i \in Good$ must satisfy Pedersen's verification equation 4.1, and hence values $\alpha_i + \sigma\hat{\alpha}_i$ lie on some t-degree polynomial $p(z) = \sum_{k=0,\ldots,t} d_k z^k$ in $\mathbb{Z}_q$. If not all values α_i, $P_i \in Good$ lie on some t-degree polynomial then also not all values $\hat{\alpha}_i = (p(i) - \alpha_i)/\sigma$ lie on some t-degree polynomial. And therefore we can recover σ by solving a system of $t+2$ linear equations of the form $-\sigma\hat{\alpha}_i + p(i) = -\sigma\hat{\alpha}_i + d_0 + d_1 i + d_2 i^2 + \ldots + d_t i^t = \alpha_i$ with unknowns $\sigma, d_0, d_1, \ldots, d_t$. This system can be solved because if some $t+2$ values $\hat{\alpha}_i$ do not lie on a t-degree polynomial, they define $t+2$ linearly independent vectors $[-\hat{\alpha}_i, 1, i, i^2, \ldots, i^t]$.

Therefore, by our assumption on $\mathcal{E}$, it follows that for all $k' \geq k_{\mathsf{SIM}}$, there exists $k \geq k'$ s.t. $\mathcal{E}$ outputs $\log_g \tilde{g}$ with probability at least $1/(4p_{\mathcal{A}}(k))$, which means that the probability that $\mathcal{E}$ computes discrete-log is not a negligible function of k, and hence completes the proof of the lemma. □

Lemma 2 (Robustness of PedVSS-REC)
Consider an execution of the following sequence of protocols: (1) protocol Ped-IG *(Figure 7-1) which on public input 1^k outputs a Pedersen commitment instance (p, q, g, h), (2) some protocol $\mathcal{P}$ which on input (p, q, g, h) produces outputs which contain some secret-sharing* PedVSS-data$[x]$*, and (3) protocol* PedVSS-REC *on input* PedVSS-data$[x]$.

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/2$-threshold adversary $\mathcal{A}$, except for probability negligible in the security parameter k, if the output of $\mathcal{P}$ contains some PedVSS-data$[x] \in \mathcal{PVSS\text{-}DATA}_{(p,q,g,h)}$ *where the dealer was not disqualified, then* PedVSS-REC *produces as a public output value $x = f_x(0)$ where f_x is the unique t-degree polynomial defined by (the outputs of the uncorrupted players in)* PedVSS-data$[x]$.

Proof: The lemma follows if we show that if there is an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \ldots)$ who participates in the above executions, such that PedVSS-data$[x] \in \mathcal{PVSS\text{-}DATA}_{(p,q,g,h)}$ and the dealer is not disqualified, but with a probability higher than negligible in k, $\mathcal{A}$ "breaks" PedVSS-REC, then we can break the discrete logarithm intractability assumption. Here by "breaking" PedVSS-REC we mean that $\mathcal{A}$ can force the outputs of PedVSS-REC *not* to produce f_x. To do that, some corrupted player P_i would have to reveal some $\alpha_i, \hat{\alpha}_i$ s.t. $g^{\alpha_i} h^{\hat{\alpha}_i} = F_x(i)$ but $\alpha_i \neq f_x(i)$, where f_x is defined by (the data of the uncorrupted players in) PedVSS-data$[x]$.

The proof below is a trivial extension of the proof of Lemma 1. We show that $\mathcal{E}$ knows all values $f_x(i)$ and $f_{\hat{x}}(i)$ of the secret-sharing defined by PedVSS-data$[x]$, and therefore if

$\mathcal{A}$ publishes $\alpha_i, \hat{\alpha}_i$ as above, $\mathcal{E}$ gets two different representations of $F_x(i)$ in bases g, h, and hence can compute $\log_g h$ and break the discrete-log assumption. We give the details below.

As in the proof of Lemma 1 we show that if there exists an adversary $\mathcal{A}$ which breaks PedVSS-REC in the above sense, then there exists a PPT TM algorithm family $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2, ...)$ called "extractor" and a discrete-log instance family DL such that the probability $\mathsf{Prob}[\mathcal{E}_k(p, q, g, g^x \bmod p) = x]$ is higher than negligible, where (p, q, g) is a discrete–log instance of security parameter k in DL, and x is uniformly distributed in $\mathbb{Z}_q$. Assume that $\mathcal{A}$ breaks PedVSS-REC with higher than negligible probability. As in the proof of Lemma 1, we argue that for every k there is a vector of random coins $\vec{r}_k$ utilized by the servers and the adversary during the DL-IG part of the Ped-IG protocol (see Figure 7-1), such that the DLog instance (p, g, q) produced by this DL-IG maximizes $\mathcal{A}$'s probability of breaking PedVSS-REC. Let the discrete-log instance family DL be the family of these instances, and let $\mathcal{A}' = (\mathcal{A}'_1, \mathcal{A}'_2, ...)$ be the family of algorithms s.t. each $\mathcal{A}'_k$ first *simulates* the run of DL-IG to $\mathcal{A}_k$, where each party is fed random inputs specified by the above vector $\vec{r}_k$, and then follows the algorithm of $\mathcal{A}_k$ during the execution of h-IG (the second part of Ped-IG) and the subsequent execution of $\mathcal{P}$ and then PedVSS-REC on input PedVSS-data$[x]$ contained in the outputs of $\mathcal{P}$. If $\mathcal{A}$ breaks that PedVSS-REC with higher than negligible probability than so does $\mathcal{A}'$.

We describe the algorithm $\mathcal{E}$ as follows. For each k, let (p, q, g) be the DLog instance of security parameter k in DL specified above. As in the proof of Lemma 1, on input $(p, q, g, \tilde{g})$ where $\tilde{g}$ is picked at random in G_q, algorithm $\mathcal{E}$ simulates to $\mathcal{A}'$ the execution of h-IG by running the simulator $\mathsf{SIM}^{(2)}_{h-IG}$ of Figure 7-3 on input $(p, q, g, \tilde{g})$. Under the discrete-log intractability assumption, by Lemma 36, page 177, there is only a negligible statistical difference between the view of $\mathcal{A}'$ in this interaction and the view of $\mathcal{A}'$ in an execution of the h-IG protocol on input (p, q, g). Therefore, the probability that later on $\mathcal{A}$ breaks PedVSS-REC remains higher than negligible. Furthermore, $\mathcal{E}$ receives from $\mathsf{SIM}^{(2)}_{h-IG}$, except for negligible probability, values $a, b \in \mathbb{Z}_q$ such that $g^a \tilde{g}^b = h$ where h is output in this (simulated) run of h-IG.

Then $\mathcal{E}$ follows the protocols $\mathcal{P}$ and PedVSS-REC, in the presence of $\mathcal{A}'$, on behalf of the uncorrupted players. We show that if $\mathcal{A}'$ breaks PedVSS-REC then $\mathcal{E}$ can compute $\sigma = \log_g h = a + b \log_g \tilde{g}$, and thus compute $\log_g \tilde{g}$. Since we assume that the outputs of $\mathcal{P}$ contain some PedVSS-data$[x] \in \mathcal{PVSS}\text{-}\mathcal{DATA}_{(p,q,g,h)}$, then $\mathcal{E}$ can interpolate the t-degree polynomials $f_x, f_{\hat{x}}$ s.t. $\alpha_i = f_x(i)$ and $\hat{\alpha}_i = f_{\hat{x}}(i)$ for $P_i \in \mathit{Good}$. Let $p(z) = f_x(z) + \sigma f_{\hat{x}}(z)$. Note that we have $g^{f_x(z)} h^{f_{\hat{x}}(z)} = g^{p(z)} = F_x(z)$. Now, if for some $P_i \in \mathit{Bad}$, P_i broadcasts $(\alpha_i, \hat{\alpha}_i)$ that satisfy Eq. (4.1) and s.t. $\alpha_i \neq f_x(i)$ then $\mathcal{E}$ gains an instance of two different representations of $F_x(i)$ in bases g, h, i.e. $\mathcal{E}$ holds two pairs $(a, a') \neq (b, b')$ s.t. $g^a h^{a'} = g^b h^{b'}$. But then $g^{a-b} = h^{b'-a'}$ and hence $\log_g h = (a - b)/(b' - a')$. □

Lemma 3 (Polynomial Secrecy of PedVSS)
There exists a simulator SIM *s.t. for every $n/2$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah*, for any distributed protocol $\mathcal{P}$, for every discrete–log instance (p, q, g), for every $h \in G_q$, for every two elements x, x^* in $\mathbb{Z}_q$, for every uncorrupted player P playing the role of the dealer, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*

  - *a run of protocol* PedVSS *with dealer* P*, on public input* (p, q, g, h)*,* $\mathcal{A}$*'s input* ah*, and* P*'s input* x^**, with outputs denoted* PedVSS-data$[x^*]$
  - *a run of* $\mathcal{P}$ *on input* PedVSS-data$[x^*]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *whose private inputs are* (σ, x^*)*, where* $\sigma = \log_g h$
  - *a run of protocol* PedVSS *with dealer* P*, on public input* (p, q, g, h)*,* $\mathcal{A}$*'s input* ah*, and* P*'s input* x*, with outputs denoted* PedVSS-data$[x]$
  - *a replacement of the private data in* PedVSS-data$[x]$ *of the simulated players with the data specified by* PedVSS-data$[x^*] = \mathcal{T}_{PedVSS}$(PedVSS-data$[x], x^*, \sigma$) *and then a run of protocol* $\mathcal{P}$ *on input* PedVSS-data$[x^*]$

Less Formally: For any x and x^*, if the dealer P is uncorrupted then there is no difference between the adversarial view of an execution of secret-sharing PedVSS in which P shares x^*, from a view of PedVSS in which P shares x.

Proof: Let pc stand for an instance (p, q, g, h) of Pedersen commitment. Let $f_x, f_{\hat{x}}$ be any t-degree polynomials s.t. $f_x(0) = x$. Consider a run of PedVSS in which P uses polynomials $f_x, f_{\hat{x}}$ and the random input of $\mathcal{A}$ is $r_{\mathcal{A}}$. Note that once we fix $f_x, f_{\hat{x}}, r_{\mathcal{A}}$ then everything else in the run of this protocol is determined. Denote the outputs of such run as PedVSS-data$_{\mathcal{A},P}$((ah, $r_{\mathcal{A}}$), pc; $f_x, f_{\hat{x}}$). Note that for any t-degree polynomials $f_x^*(z), f_{\hat{x}}^*(z)$ s.t. $f_x^*(i) = f_x(i)$ for $P_i \in Bad$ and $f_x(z) + \sigma f_{\hat{x}}(z) = f_x^*(z) + \sigma f_{\hat{x}}^*(z)$ where $\sigma = \log_g h$, the adversary's output in PedVSS-data$_{\mathcal{A},P}$((ah, $r_{\mathcal{A}}$), pc; $f_x^*, f_{\hat{x}}^*$) is the same as in PedVSS-data$_{\mathcal{A},P}$((ah, $r_{\mathcal{A}}$), pc; $f_x, f_{\hat{x}}$).

It follows that for every $f_x, f_{\hat{x}}, r_{\mathcal{A}}$ the adversarial views in the following two executions are the same:

1. a run of PedVSS on $f_x^*, f_{\hat{x}}^*, r_{\mathcal{A}}$ followed by a run of $\mathcal{P}$ on the resulting PedVSS-data$[x^*]$, where $f_x^*, f_{\hat{x}}^*$ are t-degree polynomials s.t. (1) $f_x^*(0) = x^*$; (2) $f_x^*(i) = f_x(i)$ for $P_i \in Bad$; and (3) $f_x(z) + \sigma f_{\hat{x}}(z) = f_x^*(z) + \sigma f_{\hat{x}}^*(z)$ for all z

2. a run of PedVSS on $f_x, f_{\hat{x}}, r_{\mathcal{A}}$ which outputs PedVSS-data$[x]$ followed by a run of $\mathcal{P}$ on input PedVSS-data$[x^*]$ (and $\mathcal{A}$'s randomness $r_{\mathcal{A}}$), where PedVSS-data$[x^*]$ is returned by $\mathcal{T}_{PedVSS}$(PedVSS-data$[x], x^*, \sigma$)

If we fix x, x^*, and $r_{\mathcal{A}}$, and range the polynomials $f_x, f_{\hat{x}}$ among all t-degree polynomials s.t. $f_x(0) = x$, then we see that the distributions of the adversary view in the following two cases are equal, for every $x, x^*, r_{\mathcal{A}}$:

1. PedVSS on $f_x^*, f_{\hat{x}}^*, r_{\mathcal{A}}$ followed by $\mathcal{P}$ on the resulting PedVSS-data,

   where $f_x^*, f_{\hat{x}}^*$ are random t-degree polynomials s.t. (0) $f_x^*(0) = x^*$; (1) $f_x^*(i) = f_x(i)$ on $P_i \in Bad$; (2) $f_x^*(z) + \sigma f_{\hat{x}}^*(z) = f_x(z) + \sigma f_{\hat{x}}(z)$

   where $f_x, f_{\hat{x}}$ are random t-degree polynomials s.t. $f_x(0) = x$

2. PedVSS on $f_x, f_{\hat{x}}, r_{\mathcal{A}}$ with outputs denoted PedVSS-data$[x]$, followed by $\mathcal{P}$ on inputs PedVSS-data$[x^*]$ output by $\mathcal{T}_{PedVSS}$(PedVSS-data$[x], x^*, \sigma$)

   where $f_x, f_{\hat{x}}$ are random t-degree polynomials s.t. $f_x(0) = x$

The second item above describes the same distribution as the second item in the claim of the lemma we are proving. Furthermore, the first item above describes the same distribution as the first item in the claim of the lemma, because (1) since f_x is a random polynomial s.t. $f_x(0) = x$, then f_x^* is a random polynomial s.t. $f_x(0) = x^*$; and (2) there is a one-to-one mapping between a choice of $f_{\hat{x}}$ and a choice of $f_{\hat{x}}^*$, and thus since $f_{\hat{x}}$ is a random polynomial then so is $f_{\hat{x}}^*$.

Finally, since this argument holds for every $r_{\mathcal{A}}$, the lemma follows. □

The Coin-Flip Protocol RVSS: Parallel Execution of PedVSS

We can extend the secrecy property of PedVSS expressed in Lemma 3 above to the case of *parallel* execution of polynomially-many instances of PedVSS, where each of the n players acts as a dealer in some number of PedVSS protocols. An extension of Lemma 3 for such case would state that for any two sets of inputs V and V^* which determine the values secret-shared by the uncorrupted players, there is no difference between the adversarial view of such parallel PedVSS instances in which the uncorrupted players share values specified by set V, and the adversarial view of such parallel PedVSS instances in which the uncorrupted players share values specified by set V^*. The proof of Lemma 3 above can be easily extended to such case.

It follows that if all the players run, in parallel, n instances of PedVSS protocol in which each player shares a *randomly* chosen input, then each of the secrets shared by the uncorrupted players remains hidden from the adversary in the information theoretic sense. Therefore we can now fix the two problems that make the Joint-Feldman protocol an insecure DKG protocol, by executing first a "Random Verifiable Secret Sharing" protocol RVSS which consists of such n parallel executions of PedVSS by each player on a randomly chosen input, and then "summing up" the created secret-sharings PedVSS-data$[x_1]$, ..., PedVSS-data$[x_n]$ into a "joint secret-sharing" RVSS-data$[x]$ where $x = x_1 + ... + x_n$. We will show later on that the public key $y = g^x$ can be efficiently extracted from such joint secret-sharing RVSS-data$[x]$ via a threshold exponentiation protocol Exp. Indeed, the RVSS is a building block of all threshold protocols we discuss in this thesis, like the threshold multiplication protocol Mult, threshold inverse computation Reciprocal, and in effect in any threshold signature scheme or threshold cryptosystem built according to our methodology, like the threshold DSS scheme DSS-TSig presented in Section 4.3.5.

We present the RVSS protocol in Figure 4-6. It fixes the problems of Joint-Feldman because the adversary learns nothing about the shared secrets x_i, $P_i \in \mathit{Good}$, from the secret-sharings of the uncorrupted players, and hence the secrets x_i, $P_i \in \mathit{Bad} \cap \mathit{Qual}$ that the corrupted players decide to share, are distributed independently from the secrets shared by the uncorrupted players. Hence the overall shared secret

$$x = \sum_{\mathcal{P}_i \in Qual} x_i = \sum_{P_i \in Good} x_i \quad + \sum_{P_i \in Bad \cap Qual} x_i$$

is distributed uniformly in $\mathbb{Z}_q$ as well. Because the shared secret x generated by RVSS is uniformly distributed in $\mathbb{Z}_q$, we often call protocol RVSS followed by the secret-sharing reconstruction protocol RVSS-REC a "Distributed Coin-Flip".

Furthermore, as we state formally in Lemma 6 below, not only the shared value x, but

the whole secret-sharing polynomial $f_x(z)$ looks to the adversary as a random polynomial chosen subject to the constraint that it agrees with the shares $\alpha_i = f_x(i)$ that the adversary holds. We will use this "polynomial secrecy" property to argue the security of protocols that build on top of RVSS. We state this property in the same manner as we stated the secrecy property of PedVSS above, namely using the "replacement" procedure $\mathcal{T}_{RVSS}$, Figure 4-8, which modifies the outputs of the simulated players in a way that allows future simulations of some protocol $\mathcal{P}$ that proceeds RVSS (see Lemma 6 below).

Protocol RVSS creates a joint secret-sharing RVSS-data, $(2t+1 \leq n)$

[**Note:** RVSS is a parallel execution of n instances of PedVSS on random inputs, whose outputs are then "added up". Moreover, RVSS-REC and PedVSS-REC are identical.]

Threshold Parameter:	t, the degree of the generated polynomials
Public Input:	Pedersen commitment instance (p, q, g, h)
Public Output:	set $Qual \in \{P_1, ..., P_n\}$ of qualified players
	verification function $F_x(z)$
	verification functions $F_{x_i}(z)$ for $P_i \in Qual$
Secret Output of Player P_i:	$f_{x_i}(z), f_{\hat{x}_i}(z)$, secret-sharing polynomials of P_i
	$x_i = f_{x_i}(0), \hat{x}_i = f_{\hat{x}_i}(0)$, *additive* shares of P_i
	$\alpha_{ji}, \hat{\alpha}_{ji}$ shares sent by P_j to P_i for $P_j \in Qual$
	$\alpha_i, \hat{\alpha}_i$, *polynomial* shares of P_i

(a)-(c) Each P_i chooses its input x_i at random in Z_q, and performs the PedVSS protocol (with threshold parameter t) on that input as a dealer (Steps (a)-(c), Figure 4-3). All these instances of PedVSS proceed in parallel. We denote the two polynomials used by P_i to secret-share x_i as $f_{x_i}(z), f_{\hat{x}_i}(z)$, the shares P_i sends out as $\alpha_{ij} = f_{x_i}(j), \hat{\alpha}_{ij} = f_{\hat{x}_i}(j)$, and its public verification function as $F_{x_i}(z) = g^{f_{x_i}(z)} h^{f_{\hat{x}_i}(z)}$.

(d) After Step (c) each player forms a set of non-disqualified players *Qual* and computes its final secret-shares as $\alpha_i = \sum_{P_j \in Qual} \alpha_{ji}$ and $\hat{\alpha}_i = \sum_{P_j \in Qual} \hat{\alpha}_{ji}$. The verification function $F(z)$ is defined as $\prod_{P_j \in Qual} F_{x_j}(z)$.

(The secret shared value x itself is not computed by any party, but it is equal to $x = \sum_{P_j \in Qual} x_i$. Note, however, that it is well-defined only if the execution of RVSS was *successful*, see Lemma 4.)

Reconstruction Protocol RVSS-REC:

Each P_i broadcasts values $(\alpha_i, \hat{\alpha}_i)$ s.t. $g^{\alpha_i} h^{\hat{\alpha}_i} = F_x(i)$. If all shares α_i that pass this verification interpolate into some t-degree polynomial $f_x(z)$, then the public output is $x = f_x(0)$. Otherwise, the public output is null.

Figure 4-6: (RVSS,RVSS-REC): Distributed Coin-Flip

We call an ensemble of outputs generated by RVSS, i.e. the public output and the private outputs of all the parties involved, a *joint secret-sharing*, and we denote it as RVSS-data.

We call such joint secret-sharing *correct* if it satisfies the correctness conditions listed in Figure 4-7. We call an execution of RVSS which outputs a correct joint secret-sharing *successful*. In other words, this is an execution in which the adversary didn't manage to *cheat*, i.e. to prevent the outputs of RVSS from forming a correct joint secret-sharing. By $\mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$ we denote a set of all correct joint secret-sharings RVSS-data which contain a Pedersen commitment instance (p, q, g, h) as its public data. (This is a very similar terminology to the one used for PedVSS.)

Since in further protocols we use RVSS multiple times, we will index a particular instantiation of such joint secret-sharing, with Latin letters a, b, etc, as in RVSS-data$[a]$, RVSS-data$[b]$, etc (when referring to polynomial shares we will use the corresponding Greek letters α, β, etc).[6]

We also write RVSS-data$[a, b]$ for a data structure which contains two instances of a joint secret-sharing, RVSS-data$[a]$ and RVSS-data$[b]$. Sometimes we will consider joint secret-sharings where the threshold parameter t' is different than t. We will denote such distributed data-ensembles as RVSS-data$_{t'}[a]$, RVSS-data$_{t'}[b]$, etc.

The RVSS protocol, since it is nothing more than n parallel instances of PedVSS, inherits the robustness properties of PedVSS. Namely, in Lemma 4 we show that under the discrete-log assumption, a random execution of RVSS is successful, i.e. it produces a correct joint secret-sharing, except for negligible probability. In other words, the adversary has a negligible chance of cheating in the RVSS protocol. Similarly, in Lemma 5 we show that under the discrete-log assumption, the unique polynomial f_x defined by a correct joint secret-sharing RVSS-data$[x]$ can be reconstructed by the subsequent RVSS-REC protocol, except again for negligible probability.

Definition 13 *We call an execution of protocol* RVSS successful *if and only if its outputs* RVSS-data$[a]$ *form a* correct *joint secret-sharing, which means that they satisfy the following properties:*

1. *All honest players have the same view of the set of qualified players Qual. Furthermore, all the uncorrupted players Good belong to Qual.*

2. *For each $P_j \in Qual$, each honest player P_i holds shares $\alpha_{ji}, \hat{\alpha}_{ji}$ s.t. these shares interpolate to unique t-degree polynomials $f_{x_j}(z), f_{\hat{x}_j}(z)$*

3. *For each $P_j \in Qual$, all honest players hold the same verification function $F_{x_j}(z) = g^{f_{x_j}(z)} h^{f_{\hat{x}_j}(z)}$*

4. *All honest players hold shares $\alpha_i = \sum_{P_j \in Qual} \alpha_{ji}$ and $\hat{\alpha}_i = \sum_{P_j \in Qual} \hat{\alpha}_{ji}$, which interpolate to unique t-degree polynomials $f_x(z), f_{\hat{x}}(z)$, where $f_x(z) = \sum_{P_j \in Qual} f_{x_j}(z)$ and $f_{\hat{x}}(z) = \sum_{P_j \in Qual} f_{\hat{x}_j}(z)$.*

[6] Similarly as for PedVSS-data (see footnote 4, page 56), we make an exception from this convention for joint sharing of a secret denoted as x. We denote such joint sharing as RVSS-data$[x]$, and we label all the data associated with this sharing with variable name x. However, we label the *polynomial shares* associated with this sharing with α_i (and $\alpha_{ij}, \hat{\alpha}_{ij}$, etc). See, for example, Figure 4-6.

Correct **Joint Secret-Sharing** RVSS-data$[a]$ **of value** a

[**Note:** RVSS-data$[a]$ is essentially a collection of secret-sharings PedVSS-data$[a_1, ..., a_n]$.]

We call the data ensemble which satisfies the properties listed below a *correct* joint secret-sharing RVSS-data$[a]$ of a. A set of such correct secret-sharings for a given Pedersen commitment instance (p, q, g, h) is denoted as $\mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$.

Threshold Parameter:	t, the degree of the secret-sharing polynomials
Public Data:	Pedersen commitment instance (p, q, g, h)
	set *Qual* of qualified players
	verification functions $F_{a_i} : \mathbb{Z}_q \rightarrow G_q$ for $P_i \in Qual$ s.t. $F_{a_i}(z) = g^{f_{a_i}(z)} h^{f_{\hat{a}_i}(z)}$ for some unique t-degree polynomials $f_{a_i}, f_{\hat{a}_i}$ ($F_{a_i}(z)$ is null if $P_i \notin Qual$)
	verification function $F_a(z) = \prod_{P_i \in Qual} F_{a_i}(z)$
Private Data of each $P_i \in Good$:	"secret-sharing" polynomials $f_{a_i}, f_{\hat{a}_i}$
	secret-shared values $a_i = f_{a_i}(0)$, $\hat{a}_i = f_{\hat{a}_i}(0)$ called "additive shares" of a
	shares $\alpha_{ji} = f_{a_j}(i)$, $\hat{\alpha}_{ji} = f_{\hat{a}_j}(i)$ for $P_j \in Qual$ (or null if $P_j \notin Qual$)
	shares $\alpha_i = \sum_{P_j \in Qual} \alpha_{ji}$, $\hat{\alpha}_i = \sum_{P_j \in Qual} \hat{\alpha}_{ji}$ called "polynomial shares" of a
Implicit Secret Data:	"secret-sharing" polynomials $f_a, f_{\hat{a}}$, where $f_a(z) = \sum_{P_i \in Qual} f_{a_i}(z)$ and $f_{\hat{a}}(z) = \sum_{P_i \in Qual} f_{\hat{a}_i}(z)$
	generated shared secret $a = f_a(0)$
Private Data of $\mathcal{A}$:	the adversary's *computational history*

Figure 4-7: Properties of a *correct* joint secret-sharing RVSS-data

5. *All honest players hold the same verification function* F_x *such that* $F_x(z) = g^{f_x(z)} h^{f_{\hat{x}}(z)} = \prod_{P_j \in Qual} F_{x_j}(z)$.

Lemma 4 (Robustness of RVSS**)**
Consider an execution of (1) protocol Ped-IG *(Figure 7-1) which on public input* 1^k *outputs a Pedersen commitment* (p, q, g, h)*, (2) any other protocol* $\mathcal{P}$*, and (3) protocol* RVSS *on public input* (p, q, g, h) *with outputs denoted* RVSS-data$[x]$

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/2$*-threshold adversary* $\mathcal{A}$*, the above sequence of protocols satisfies the following two properties:*

1. *Except for probability negligible in the security parameter, the above instance of* RVSS *is* successful.

2. *The unique secret* x *defined by* RVSS-data$[x]$ *generated by a random* successful *instance of above* RVSS *is uniformly distributed in* $\mathbb{Z}_q$.

Proof: Protocol RVSS consists of n parallel executions of PedVSS, each with a different player as a dealer. By the same argument as used in Lemma 1 we can show that if the adversary has a higher than negligible probability of causing *any* of these executions not to output a correct secret-sharing RVSS-data$[x_i]$ then we can compute discrete logarithms. Therefore, under the discrete-log assumption, except for negligible probability, the players hold a correct secret-sharing RVSS-data$[x_i]$ for every P_i. Furthermore, for each $P_i \in Good$, P_i is not disqualified, and therefore $Good \subseteq Qual$. Thus property (1) of the Lemma follows.

As for property (2), by the secrecy property of PedVSS (Lemma 3), at the end of Step (c), the adversary learns no information about values x_i shared by the uncorrupted players $P_i \in Good$. Therefore, values $\{x_i\}_{P_i \in Good}$, each of which was chosen with uniform distribution by some honest player, are distributed independently of values $\{x_i\}_{P_i \in (Bad \cap Qual)}$, and therefore value $x = \sum_{P_i \in Qual} x_i = \sum_{P_i \in Good} x_i + \sum_{P_i \in (Bad \cap Qual)} x_i$ is uniform in $\mathbb{Z}_q$. □

Lemma 5 (Robustness of RVSS-REC)
Consider an execution of the following sequence of protocols: (1) protocol Ped-IG *(Figure 7-1) which on public input 1^k outputs a Pedersen commitment instance (p, q, g, h), (2) some protocol $\mathcal{P}$ which on input (p, q, g, h) produces outputs which contain some joint secret-sharing* RVSS-data$[x]$*, and (3) protocol* RVSS-REC *on input* RVSS-data$[x]$.

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/2$-threshold adversary $\mathcal{A}$, except for probability negligible in the security parameter k, if the output of $\mathcal{P}$ contains some RVSS-data$[x] \in \mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$*, then* RVSS-REC *produces as a public output value $x = f_x(0)$ where f_x is the unique t-degree polynomial defined by* RVSS-data$[x]$.

Proof: Note that $F_x(z) = \prod_{P_i \in Qual} F_{x_i}(z) = g^{\sum_{P_i \in Qual} f_{x_i}(z)} h^{\sum_{P_i \in Qual} f_{\hat{x}_i}(z)} = g^{f_x(z)} h^{f_{\hat{x}}(z)}$. If a dishonest player can with higher than negligible probability publish $\alpha_i^*, \hat{\alpha}_i^*$ such that $\alpha_i^* \neq f_x(i)$, but which satisfies the Pedersen verification equation $F(i) = g^{\alpha_i^*} h^{\hat{\alpha}_i^*}$, then we can break the discrete-logarithm intractability assumption by the exact same argument as in the proof of robustness of reconstruction PedVSS-REC of a single PedVSS, i.e. Lemma 2.

Here we sketch this reduction. (The full picture can be filled in following the proof of Lemma 2.) An efficient extractor $\mathcal{E}$ controls the uncorrupted players and performs RVSS and RVSS-REC on their behalf. If RVSS is successful, i.e. if its output RVSS-data$[x]$ is a correct joint secret-sharing, then $\mathcal{E}$ can interpolate polynomials $f_x, f_{\hat{x}}$. In particular it holds that $F_x(i) = g^{f_x(i)} h^{f_{\hat{x}}(i)}$. If a dishonest players publishes $\alpha_i^*, \hat{\alpha}_i^*$ such that $\alpha_i^* \neq f_x(i)$, but s.t. $F_x(i) = g^{\alpha_i^*} h^{\hat{\alpha}_i^*}$, then $\mathcal{E}$ gets two representations of $F_x(i)$ in bases g, h, and thus can compute $\log_g h$. Since $\mathcal{E}$ precedes the execution of (RVSS;RVSS-REC) with a simulation of h-IG during which $\mathcal{E}$ embeds an instance $g, \tilde{g}$ of a discrete log problem in the generated value h so that $h = g^a \tilde{g}^b$ and $\mathcal{E}$ knows values a, b, learning $\log_g h$ allows $\mathcal{E}$ to learn $\log_g \tilde{g}$.)
□

Lemma 6 (Polynomial Secrecy of RVSS)
There exists a simulator SIM *s.t. for every $n/2$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah*, for any distributed protocol $\mathcal{P}$, for every discrete-log instance (p, q, g), for every $h \in G_q$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*

  - *a run of* RVSS, *on public input* (p, q, g, h), $\mathcal{A}$*'s input* ah, *with outputs denoted* RVSS-data$[x]$
  - *a run of* $\mathcal{P}$ *on input* RVSS-data$[x]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input* $\sigma = \log_g h$
  - *a run of* RVSS, *on public input* (p, q, g, h), $\mathcal{A}$*'s input* ah, *with outputs denoted* RVSS-data$[x]$
  - *a replacement of the private data in* RVSS-data$[x]$ *of the simulated players with the data specified by* RVSS-data$[x^*] = \mathcal{T}_{RVSS}$(RVSS-data$[x], \sigma$) *and then a run of* $\mathcal{P}$ *on input* RVSS-data$[x^*]$

Less Formally: A (static) adversarial view of an execution of the (RVSS,$\mathcal{P}$) sequence is *the same* as an adversarial view of an execution of RVSS, a *modification* of the private data of the uncorrupted players according to procedure $\mathcal{T}_{RVSS}$ (which, if RVSS-data$[x]$ is correct, replaces it with a sharing RVSS-data$[x^*]$ of a random x^* in $\mathbb{Z}_q$), and then an execution of $\mathcal{P}$.
Informally: A static secure-channels $n/2$-threshold adversary learns nothing about polynomial $f_x(z)$ shared in RVSS protocol except of the shares $f_x(i)$, $P_i \in Bad$, it receives.

Proof: The lemma follows straightforwardly from Lemma 3, because the modification of the joint secret-sharing performed by $\mathcal{T}_{RVSS}$ is really only a modification of one of the secret-sharings in the same way as would be done by $\mathcal{T}_{PedVSS}$. This can easily be seen by inspection, but the formal argument follows.

Let P_S be the uncorrupted player whose outputs are substituted in procedure $\mathcal{T}_{RVSS}$. Note that Lemma 3 can be extended (and it is easy to see that the proof of Lemma 3 holds for such extension) to a case when in parallel with protocol PedVSS performed by dealer P_S, there are other instances of the PedVSS protocol executed, each with a different player as a dealer. The extension of Lemma 3 to this case states that an adversarial view of the following two execution sequences is the same (for any $x, x^* \in \mathbb{Z}_q$ and any protocol $\mathcal{P}$):

1. an execution of PedVSS on P_S's input x^* in parallel with other instances of PedVSS where other players are dealers, followed by an execution of $\mathcal{P}$ on the resulting output

2. an execution of PedVSS on P_S's input x in parallel with the other instances of PedVSS as above, followed by a modification of the private data of the uncorrupted players in PedVSS-data$[x]$ created by the PedVSS where P_S was the dealer, namely the replacement of this data with the corresponding data given in PedVSS-data$[x^*]$ = $\mathcal{T}_{PedVSS}$(PedVSS-data$[x], x^*, \sigma$), and then followed by an execution of $\mathcal{P}$ on the resulting output.

(Note that the fact that the other protocols executed in parallel with the PedVSS where P_S is a dealer are also instances of PedVSS, does not matter in the above argument.)

Note furthermore, that if the above statement holds for any x, x^* in $\mathbb{Z}_q$, then it holds also when x and x^* are random values uniformly distributed in $\mathbb{Z}_q$. In other words, we modify the above slightly to get the following equality of (distributions of) adversarial views in:

RVSS-data$[x] \rightarrow$ RVSS-data$[x^*]$ **replacement procedure** $\mathcal{T}_{RVSS}$

Input: public and private data of uncorrupted players in RVSS-data$[x]$
Pedersen's trapdoor $\sigma = \log_g h$
(Optional:) "target output" element x^* in $\mathbb{Z}_q$
Output: (private data of uncorrupted players in) secret-sharing RVSS-data$[x^*]$

[**Note:** $\mathcal{T}_{RVSS}$ modifies RVSS-data$[x]$ by replacing one of its "component secret-sharings", PedVSS-data$[x_S]$, for some $P_S \in Good$ via procedure $\mathcal{T}_{PedVSS}$ of Figure 4-5.]

Let $Bad, Good$ be the identities of the corrupted and uncorrupted players ($|Bad| = t$). Let P_S be any uncorrupted player. Take polynomials $f_{x_S}, f_{\hat{x}_S}$ contained in P_S's outputs in RVSS-data$[x]$, i.e. the polynomials this player dealt to create RVSS-data$[x]$.
$\mathcal{T}_{RVSS}$ picks random t-degree polynomials $f^*_{x_S}, f^*_{\hat{x}_S}$ s.t.:

$$\begin{aligned} f^*_{x_S}(i) &= f_{x_S}(i) \text{ for } P_i \in Bad \\ f^*_{x_S}(z) + \sigma f^*_{\hat{x}_S}(z) &= f_{x_S}(z) + \sigma f_{\hat{x}_S}(z) \text{ (for all } z) \end{aligned}$$

and then forms RVSS-data$[x^*]$ from RVSS-data$[x]$ by:
- replacing the private data $f_{x_S}, f_{\hat{x}_S}$ of P_S with $f^*_{x_S}, f^*_{\hat{x}_S}$
- replacing the private data $x_S, \hat{x}_S$ of P_S with $x^*_S = f^*_{x_S}(0)$ and $\hat{x}^*_S = f^*_{\hat{x}_S}(0)$
- replacing the private data $\alpha_{Si}, \hat{\alpha}_{Si}$ of each P_i in $Good$ with $\alpha^*_{Si} = f^*_{x_S}(i)$ and $\hat{\alpha}^*_{Si} = f^*_{\hat{x}_S}(i)$
- replacing the private data $\alpha_i, \hat{\alpha}_i$ of these players with values $\alpha^*_i = \sum_{P_j \in Qual \setminus \{P_S\}} \alpha_{ji} + \alpha^*_{Si}$ and $\hat{\alpha}^*_i = \sum_{P_j \in Qual \setminus \{P_S\}} \hat{\alpha}_{ji} + \hat{\alpha}^*_{Si}$

Optional: If the optional "target output" element x^* is provided, $\mathcal{T}_{RVSS}$ picks the above polynomials $f^*_{x_S}, f^*_{\hat{x}_S}$ subject to the additional constraint that

$$f^*_{x_S}(0) = x^* - \sum_{P_i \in Qual \setminus \{P_S\}} f_{x_i}(0)$$

where for each $P_i \in Bad \cap Qual$, f_{x_i} is a t-degree polynomial which $\mathcal{T}_{RVSS}$ interpolates from any $t+1$ shares α_{ij}, $P_j \in Good$.
(Note that if x^* is chosen at random in $\mathbb{Z}_q$ then this additional constraint does not change the distribution of outputs of $\mathcal{T}_{RVSS}$.)

Figure 4-8: $\mathcal{T}_{RVSS}$: Auxiliary procedure for simulation of RVSS

1. an execution of PedVSS on *random* input of P_S's in parallel with instances of PedVSS performed by other players on random inputs, followed by an execution of $\mathcal{P}$ on the resulting output

2. an execution of PedVSS on random input of P_S in parallel with the other instances of PedVSS as above, followed by a replacement of the private data of the uncorrupted players in PedVSS-data dealt by P_S with the corresponding data in PedVSS-data* = $\mathcal{T}_{PedVSS}$(PedVSS-data, x^*, σ) where x^* is chosen at random $\mathbb{Z}_q$, and then followed by an execution of $\mathcal{P}$ on the resulting output.

Notice that in both cases above we consider an execution of PedVSS in which player P_S shares a random input in parallel with n-1 executions of PedVSS in which each other player shares a random input. This is exactly Steps (a)-(c) of RVSS. Therefore, for any distributed protocol $\mathcal{P}'$ which takes a joint secret sharing as an input, if in the above statement we take $\mathcal{P}$ to be a sequence consisting of (1) Step (d) of RVSS (see Figure 4-6); and (2) an execution of protocol $\mathcal{P}'$ on the resulting joint secret-sharing, then the following equality of the adversarial views is implied:

1. an execution of RVSS followed by an execution of $\mathcal{P}'$
2. (a) an execution of Steps (a)-(c) of RVSS
   (b) replacement of private data of the uncorrupted players in PedVSS-data dealt by P_S with the corresponding data in PedVSS-data$^* = \mathcal{T}_{PedVSS}$(PedVSS-data, x^*, σ) where x^* is chosen at random in $\mathbb{Z}_q$
   (c) an execution of Step (d) of RVSS
   (d) an execution of $\mathcal{P}'$

Finally, note that if the adversary is static, then the adversary does not see a difference if the actions of the uncorrupted players correspond to steps (b) and (c) above, or if they correspond to an execution of these steps in a reverse order, i.e. if the uncorrupted players perform first Step (d) of RVSS and then the appropriate data in the resulting joint secret-sharing RVSS-data is replaced according to procedure $\mathcal{T}_{RVSS}$. Thus the lemma follows. □

Parallel Composition. Note that a similar secrecy property can be stated about the *parallel composition* of polynomially-many instances of RVSS. Namely, if several instances of RVSS are executed in parallel, and they create multiple secret-sharings RVSS-data[a_1], RVSS-data[a_2], ..., RVSS-data[a_k], then the adversary's view of any subsequent protocol $\mathcal{P}$ is the same if the private data of the uncorrupted players in these joint secret-sharings is modified according to the $\mathcal{T}_{RVSS}$ replacement procedure. The reason why the above Lemma 6 can be extended in this way is that the secrecy property of PedVSS can be extended, as we discussed before, to parallel execution of polynomially-many instances of PedVSS.

The robustness property extends straightforwardly to the case of parallel execution of multiple instances of RVSS.

4.2.5 Threshold Exponentiation Exp and Secure DKG Protocol

The secure Distributed Key Generation protocol we propose is composed of the following steps: (1) on input a DLog instance (p, q, g), the players jointly generate the Pedersen commitment value $h \in G_q$ via the h-IG protocol described in Chapter 7, (2) the players perform RVSS (Figure 4-6) to generate a secret-sharing RVSS-data[x] of a uniformly distributed secret x; and (3) the players perform a threshold exponentiation protocol on input RVSS-data[x] to extract the public key $y = g^x$ as a public output. The last step is achieved via protocol Exp, which we describe in this section.

For notational convenience, we denote Steps (2) and (3) above as a DKG protocol, shown in Figure 4-10. When we want a secure DKG protocol in the sense of Definition 11,

page 49, we consider protocol h-IG+DKG, i.e. an execution of h-IG, Figure 7-3, followed by an execution of DKG. Similarly, if we want a key generation protocol in the sense of definition of a threshold signature scheme (Definition 4, page 34), then we consider protocol Ped-IG+DKG, i.e. an execution of Ped-IG, Figure 7-1, a protocol that picks both the discrete-log instance (p, q, g) and the additional element h in G_q, followed by an execution of DKG. (See Remark 3, page 51.)

We note that once a public value h in G_q is determined, protocol DKG enjoys the same flavor and simplicity as the insecure DKG protocol Joint-Feldman discussed in Section 4.2.3. I.e., each player shares a random value and the random secret is generated by summing up these values. The computation costs incurred per player in DKG are also comparable to those in Joint-Feldman. All the long modular exponentiations needed during the extraction phase have already been computed during the RVSS phase, thus Exp is basically "for free" from a computational point of view.

Furthermore, we note that even though DKG, unlike Joint-Feldman, needs to be preceded by the h-generating protocol h-IG, the cost of this additional protocol may be amortized in many applications, because once h is initially chosen it can be reused by all subsequent executions of the DKG protocol.

Threshold Exponentiation protocol Exp

We first present a protocol that generalizes the public-key extraction needed by the distributed key generation protocol DKG discussed above, namely a Threshold Exponentiation protocol Exp, presented in Figure 4-9. This protocol allows the players to exponentiate any element $m \in G_q$ to a secret value a shared with RVSS-data$[a]$, and produce a public output $A = m^a$. In DKG, protocol Exp is executed on the sharing of the secret key RVSS-data$[x]$ and on the element g, to output the public key $y = g^x$.

Apart from generalizing the public-key-extraction in DKG, protocol Exp serves as a basis for the distributed Cramer-Shoup key generation protocol and the threshold RSA signature generation protocol (see Appendices A and C). This protocol can be also generalized to computing values expressible as $m_1^{a_1} \cdot m_2^{a_2} \cdot ... \cdot m_k^{a_k}$ where $m_1, m_2, ..., m_k$ are public inputs and each a_i is a secret shared with a different joint secret-sharing RVSS-data$[a_i]$. Examples of such generalization of protocol Exp can be seen in the distributed key generation and the threshold decryption procedure of the threshold Cramer-Shoup cryptosystem given in Appendix A. Another use of the exponentiation protocol Exp is a generation of a new random element g' in G_q by first generating a secret-sharing RVSS-data$[a]$ of a random secret $a \in \mathbb{Z}_q$ via a distributed coin-flip protocol RVSS, and then executing Exp on RVSS-data$[a]$ and the public input g (or any other element in G_q) to output $g' = g^a$. We use this capability of Exp in the threshold key generation for the Cramer-Shoup cryptosystem (see Appendix A).

We note that the Exp protocol is also secure under parallel composition.

Definition 14 *We call an execution of protocol* Exp *on input a correct joint secret-sharing* RVSS-data$[a]$ *and some element* $m \in G_q$ successful *if it outputs a public value* $A = m^a$ *where* a *is the unique shared secret defined by* RVSS-data$[a]$.

Protocol Exp, $(2t+1 \leq n)$

Public Input: element $m \in G_q$
Other Input: secret-sharing RVSS-data$[a]$ (See Fig.4-7)
Public Output: value $A = m^a$, where a is defined by RVSS-data$[a]$

1. Each player $P_i \in Qual$, broadcasts $A_{ik} = m^{c_{ik}}$ for $k = 0, ..., t$, where c_{ik}'s are the coefficients of $f_{a_i}(z)$ from RVSS-data$[a]$. Denote $A_i = A_{i0}$.
2. Each player P_j checks the Feldman verification equation for the values broadcast by all the other players $P_i \in Qual$:

$$m^{\alpha_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \bmod p \tag{4.2}$$

   If the check fails for an index i, P_j *complains* against P_i by broadcasting values $\alpha_{ij}, \hat{\alpha}_{ij}$ that agree with verification function $F_{a_i}(z)$ established in RVSS-data$[a]$, i.e., such that $g^{\alpha_{ij}} h^{\hat{\alpha}_{ij}} = F_{a_i}(j)$, but which do not satisfy Eq. (4.2).
3. For players P_i who receive at least one valid complaint, the other players run the reconstruction protocol PedVSS-REC on PedVSS-data$[a_i]$ contained in RVSS-data$[a]$ to compute a_i and $A_i = m^{a_i}$ in the clear. Compute $A = \prod_{i \in Qual} A_i$. If any reconstruction fails (outputs null) then the output of Exp is null too.

Simulator SIM_{Exp} of Exp (interacting with $\mathcal{A}$):

Public Input: element $m \in G_q$, public data in RVSS-data$[a]$
SIM_{Exp}'s Private Input: private outputs of all $P_i \in Good$ in RVSS-data$[a]$
the "target output" value $A^* \in G_q$
$\mathcal{A}$'s Private Input: adversary's output in RVSS-data$[a]$

Pick some uncorrupted player (assume P_n) and perform some pre-computations:

(a) Compute $A_{ik} = m^{c_{ik}}$ for $P_i \in Good \setminus \{P_n\}$, $k = 0, \ldots, t$. Denote A_{i0} as A_i
(b) For each $P_i \in (Bad \cap Qual)$, interpolate polynomial f_{a_i} from α_{ij}'s for $P_j \in Good$ in RVSS-data$[a]$. (If RVSS-data$[a]$ is incorrect then some f_{a_i} might be of higher degree than t.)
(c) Compute $A_i = m^{f_{a_i}(0)}$ and set $A^*_{n0} = A^*_n = A^* \cdot \prod_{P_i \in (Qual \setminus \{P_n\})} (A_i)^{-1}$
(d) Compute $A^*_{nk} = (A^*_{n0})^{\lambda_{k0}} \cdot \prod_{P_i \in Bad} (m^{\alpha_{ni}})^{\lambda_{ki}}$ for $k = 1, \ldots, t$, where λ_{ki}'s are derived from Lagrange interpolation coefficients, i.e. they are coefficients computed so that $\prod_{k=0}^{t} (A^*_{nk})^{i^k} = m^{\alpha_{ni}} = m^{f_n(i)}$ for $P_i \in Bad$

1. For $k = 0, ..., t$, SIM_{Exp} broadcasts A^*_{nk} and A_{ik} for $P_i \in Good \setminus \{P_n\}$
2. SIM_{Exp} follows Step 2 of the protocol on behalf of the uncorrupted players, except that the uncorrupted players do not send any complaints against player P_n.
3. SIM_{Exp} follows Step 3 of the protocol on behalf of the uncorrupted players.

Figure 4-9: Exp: Statically Secure Threshold Exponentiation Protocol

Lemma 7 (Robustness of Exp)
Consider an execution of the following sequence of protocols: (1) protocol Ped-IG *(Figure 7-1) which on public input* 1^k *outputs a Pedersen commitment instance* (p, q, g, h)*, (2) some protocol* $\mathcal{P}$ *which on input* (p, q, g, h) *produces outputs which contain some correct joint secret-sharing* RVSS-data$[a]$*, and (3) protocol* Exp *on input* RVSS-data$[a]$ *and some public input* m *in* G_q*.*

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/2$*-threshold adversary, if the output of* $\mathcal{P}$ *contains some* RVSS-data$[a]$ *in* $\mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$*, then, except for probability negligible in the security parameter* k*, a run of* Exp *is* successful.

Proof: Let a be the secret shared with RVSS-data$[a]$ generated by $\mathcal{P}$, let a_i be its additive shares, and f_{a_i} the secret-sharing polynomials. We need to show that value A (computed by the honest players) is indeed $A = m^a$ (under the discrete-log assumption and except for negligible probability). We will show that (under the discrete-log assumption and except for negligible probability) for all $P_i \in Qual$, each uncorrupted player computes $A_i = m^{a_i}$, and thus also computes $A = \prod_{i \in Qual} A_i = \prod_{i \in Qual} m^{a_i} = m^{\sum_{i \in Qual} a_i} = m^a$.

For parties $P_i \in Qual$ against whom a valid complaint has been issued in Step 2, value a_i is publicly reconstructed and A_i set to m^{a_i}. Under the discrete-log assumption, the correct reconstruction of a_i is guaranteed (except for negligible probability) by the robustness property of protocol PedVSS-REC (Lemma 2). Now we need to show that for each $P_i \in Qual$ against whom a valid complaint has *not* been issued, the value A_i is set to m^{a_i} as well. Values $\{A_{ik}\}_{k=0,\ldots,t}$ broadcast by player P_i in Step 1 define a t-degree polynomial $f_{a'_i}$ in $\mathbb{Z}_q$ by equation $m^{f_{a'_i}(z)} = \prod_{k=0}^{t} (A_{ik})^{z^k}$. Since we assume that no valid complaint was issued against P_i then Eq. 4.2 is satisfied for all honest players, and thus $f_{a'_i}$ and f_{a_i} have at least $t+1$ points in common. Hence they are equal, and in particular $A_i = A_{i0} = m^{f_{a'_i}(0)} = m^{f_{a_i}(0)} = m^{a_i}$. □

Secrecy of Exp. The following lemma expresses an essential technical property of protocol Exp and its simulator SIM_{Exp}. Given this lemma, and given that the public output of a simulation of Exp in which SIM_{Exp}'s private input is A^* is equal to either A^* or null (see Fact 1 below), we conclude that for every A in G_q, an adversarial view of a random execution of the (RVSS,Exp) sequence *which outputs* A is the same (has the same distribution) as an adversarial view of a simulation in which the simulator's input is the "target value" A, and which produces this A as its public output. We use this reasoning to prove that the DKG protocol, Figure 4-10, is a secure Distributed Key Generation protocol.

Fact 1 *For every adversary* $\mathcal{A}$ *with history* ah*, for any joint secret-sharing* RVSS-data$[x]$ *(not necessarily a* correct *one), for any* $A^* \in G_q$*, the public output of a simulation of* Exp *via the simulator* SIM_{Exp} *of Figure 4-9, on input* RVSS-data$[a]$*, on* $\mathcal{A}$*'s private input* ah*, on the simulator's* SIM_{Exp} *private input* A^**, is equal either to* A^* *or to* null*.*

Proof: This can be verified by inspection of the simulation procedure SIM_{Exp}, Figure 4-9. Note that if for some P_i the shares α_{ij}, $P_j \in Good$, do not interpolate to a t-degree polynomial then the simulation will eventually output null because some of the uncorrupted players

will send a complaint against P_i and then the output of the PedVSS-REC on PedVSS-data$[a_i]$ will be null, and hence the public output of this simulation will be null too. □

Lemma 8 (Static Secrecy of Exp)
For every $n/2$-threshold static secure-channels adversary $\mathcal{A}$ with history ah, *for every discrete-log instance (p, q, g), for every $h \in G_q$, and for every $m \in G_q$, the following two variables are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a run of* RVSS *(Figure 4-6) on public input (p, q, g, h) and $\mathcal{A}$'s input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a run of* Exp *on inputs* RVSS-data$[a]$ *and m*
- *an adversarial view of the following simulation:*
  - *a run of* RVSS *on public input (p, q, g, h) and $\mathcal{A}$'s input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a simulation of* Exp *with simulator* SIM_{Exp} *(Figure 4-9), on inputs* RVSS-data$[a]$, *m, and on* SIM_{Exp}*'s additional private input A^* picked at random in G_q*

Less Formally: The adversarial view of the execution of RVSS followed by a simulation performed by SIM_{Exp} on a random input A^* looks like a random execution of the (RVSS,Exp) sequence. Given that a public input of a simulation of Exp on simulator's SIM_{Exp} private input A^* can be either A^* or null (see Fact 1 above), this implies that, for every A, a simulation of the (RVSS,Exp) sequence to a target output A looks like a random execution of this sequence which outputs A.
Informally: An execution of Exp on RVSS-data$[a]$ and m hides everything about a shared in RVSS-data$[a]$ apart of the computed public value $A = m^a$.

Remark. In the discussions of threshold protocols which utilize Exp as a building block, we will refer to a *successful simulation* of Exp via simulator SIM_{Exp}, by which we mean an instance of the simulation in which the public output is equal to the input A^* of SIM_{Exp} and not to null. (Compare Fact 1 above.)

Proof: Secrecy of the threshold exponentiation protocol Exp follows immediately from the polynomial secrecy property of RVSS as stated in Lemma 6.

Note that publishing values $A_{nk} = m^{c_{nk}}$ in the protocol Exp is equivalent to publishing a function $\mathcal{F}_{a_n}(z) = \prod_{k=0}^{t}(A_{nk})^{z^k} = m^{\sum_{k=0}^{t} c_k z^k} = m^{f_{a_n}(z)}$. Values A^*_{nk} in the simulation are computed so that they correspond to function $\mathcal{F}^*_{a_n}(z) = m^{f^*_{a_n}(z)}$, where $f^*_{a_n}(0) = a^*_n$ and $f^*_{a_n}$ agrees with adversary's shares, i.e. $f^*_{a_n}(i) = \alpha_{ni}$ for $P_i \in Bad$.

Because A^* is random in G_q, polynomial $f^*_{a_n}$ is a random t-degree polynomial subject to the constraint that it agrees with f_{a_n} defined in RVSS-data$[a]$ on points $P_i \in Bad$. Therefore by Lemma 6 (assume that the player P_S used in the replacement procedure $\mathcal{T}_{RVSS}$ is the player P_n), the above $f^*_{a_n}$ is perfectly compatible with the adversary's view of RVSS which produced RVSS-data$[a]$. Therefore, function $\mathcal{F}^*_{a_n}(z) = g^{f^*_{a_n}(z)}$ is also perfectly compatible with the adversary's view, and the lemma follows. □

Protocol DKG, $(2t+1 \leq n)$

Public Input: Pedersen commitment instance (p, q, g, h)
Public Output: generated public key $y \in G_q$
Other Output (Private and Public):
secret-sharing data RVSS-data$[x]$ (See Fig. 4-7),
where $x = f_x(0) = \log_g y$

1. Players perform protocol RVSS (Fig.4-6) on input (p, q, g, h) to generate a secret-sharing data-structure RVSS-data$[x]$ (See Fig.4-7)
2. Players perform protocol Exp (Fig.4-9) on input a sharing RVSS-data$[x]$ and an element g in G_q to generate the public key $y = g^x$

Simulator SIM_{DKG} of DKG (interacting with $\mathcal{A}$):

Public Input: Pedersen commitment instance (p, q, g, h)
SIM_{DKG}'s Private Input: the "target public key" value $y^* \in G_q$

1. SIM_{DKG} follows Step 1 of the protocol on behalf of the uncorrupted players, i.e. performs RVSS on input (p, q, g, h) to generate a secret-sharing RVSS-data$[x]$
2. SIM_{DKG} simulates Exp by running the SIM_{Exp} of Figure 4-9 on sharing RVSS-data$[x]$, and on SIM_{Exp}'s private input the target value y^*

Figure 4-10: DKG: Statically Secure Distributed Key Generation Protocol

Secure Distributed Key Generation Protocol

As mentioned above, our proposal for a secure DKG protocol h-IG+DKG], which consists of the h-generation protocol h-IG, Figure 7-3, followed by protocol DKG of Figure 4-10, which generates sharing RVSS-data$[x]$ of a uniformly distributed secret key x in $\mathbb{Z}_q$ and extracts the public key $y = g^x$. For notational purposes we define the notion of a *successful execution* of protocol DKG:

Definition 15 *We call an execution of protocol* DKG successful *if and only if it includes a successful instance of* RVSS *and a successful instance of* Exp.

Theorem 1 (Static Security of DKG)
Under the discrete logarithm intractability assumption, protocol h-IG+DKG, which consists of protocol h-IG, Figure 7-3, followed by protocol DKG, *Figure 4-10, is an $n/2$-secure Distributed Key Generation protocol in a static secure-channels adversary model.*

Proof: See the properties of a secure DKG protocol in Definition 11, page 49. It is easy to see that the *correctness* properties of secure DKG are satisfied by h-IG+DKG. Properties (C1) and (C2) are stated in Lemma 4 (note that protocol h-IG+DKG is *sharing-successful*, a notion defined in (C1) of Definition 11, if and only if the RVSS in Step 1 of DKG is successful). Property (C1') follows from Lemmas 4 and 5. Property (C3) is shown in Lemma 7.

It remains for us to show that h-IG+DKG satisfies the *secrecy* property of a secure DKG. We argue that simulator SIM which on input (p, q, g) and a "target public key" $y^* \in G_q$, first follows the h-IG protocol on behalf of the uncorrupted players and then simulates DKG via simulator SIM_{DKG} of Figure 4-10, on public input (p, q, g) and h output by h-IG, and on SIM_{DKG}'s input y^*, satisfies the secrecy property stated in Definition 11. It is a straightforward implication of Lemma 8 and Fact 1, and we argue it formally below.

Consider a random execution of protocol h-IG+DKG conditioned by the constraint that it outputs a specific element y in G_q. Let $pc = (p, q, g, h)$ be a Pedersen commitment instance. Lemma 8 together with Fact 1 implies that the following adversarial views are identically distributed, for all pc, all $\mathcal{A}$ and ah, and for all m and A in G_q:

- an adversarial view of the following sequence of executions:
  1. a *successful* execution of RVSS on pc, with outputs denoted RVSS-data$[a]$
  2. a *successful* execution of Exp on m and RVSS-data$[a]$ *which outputs value* A
- an adversarial view of the following simulation via SIM_{DKG}:
  1. a *successful* execution of RVSS on pc, with outputs denoted RVSS-data$[a]$ (in Step 1 of the simulation of DKG)
  2. a *successful* simulation of Exp on m and RVSS-data$[a]$ and on SIM_{Exp}'s input A (in Step 2 of the simulation of DKG)

     (i.e. a simulation of Exp which outputs A, see a remark after Lemma 8)

Consider a vector $\vec{r}_{h-IG}$ of random inputs used by the adversary and the players (actual or simulated) in the h-IG protocol. Such vector fixes the resulting values h and the adversarial history ah output by h-IG. Therefore, for every y^* in G_q, every DLog instance (p, q, g), every $\vec{r}_{h-IG}$, and for (h, ah) output by h-IG on public inputs (p, q, g) and random inputs $\vec{r}_{h-IG}$, the adversarial view of a simulation of DKG which on input (p, q, g, h) and ah and on simulator's SIM_{DKG} input y produces public output y is the same as the adversarial view of an execution of DKG which produces y. If we take the distribution of the above two views induced by a uniform distribution of the random vector $\vec{r}_{h-IG}$, this implies that for every y and (p, q, g), the adversarial view of a simulation of h-IG+DKG which outputs y^* on public input (p, q, g) and on SIM's input y^* is the same as the adversarial view of an execution of h-IG+DKG which outputs y^* on public input (p, q, g). Thus h-IG+DKG achieves the secrecy property of a secure Distributed Key Generation. □

Note on the simulation technique. We note that for the sake of proving the $n/2$-threshold security of h-IG+DKG, simulator SIM used in the above proof does not need to simulate the h-IG protocol, because the following protocol DKG can be simulated without knowing the trapdoor $\sigma = \log_g h$ of the Pedersen commitment instance (p, q, g, h). This changes when h-IG+DKG is executed as a first step of a run of an optimal-threshold DSS signature scheme TSS-opt of Section 4.4, because we know only how to simulate subsequent executions of the optimal-threshold DSS signature generation protocol DSS-TSig-opt if the simulator knows the trapdoor σ.

4.3 Threshold DSS Signatures

We present further basic building blocks of discrete-log based threshold protocols and we show how to construct a threshold DSS signature generation protocol, which together with the distributed key generation protocol of the preceding section forms a secure threshold DSS signature scheme. We first recall the standard DSS signature scheme, then we present some building blocks of our protocols, and then we construct threshold DSS signature generation. Protocols in this section have only $n/4$-threshold resilience. In Section 4.4 we show how to improve them to optimal $n/2$-threshold resilience. All the protocols we present have non-rewinding simulators and extractors and thus can be securely performed in parallel.

4.3.1 DSS Signature Scheme

The Digital Signature Standard (DSS) [NIS91] is a signature scheme based on the ElGamal [ElG85a] and Schnorr's [Sch91] signature schemes, which was adopted as the US standard digital signature algorithm. Following [GJKR96b]', our description of the DSS protocol uses the notation introduced in [Lan95], which differs from the original presentation of [NIS91] by switching k and k^{-1}. This change allows a clearer presentation of our threshold DSS signature protocols that follow.

Key Generation. A DSS key is composed of public information (p, q, g), a public key y and a secret key x, where:
1. p is a prime number of length l, where l is a multiple of 64 and $512 \leq l \leq 1024$.
2. q is a 160-bit prime divisor of $p - 1$.
3. g is an element of order q in Z_p^*. The triple (p, q, g) is public.
4. x is the secret key of the signer, a random number $1 \leq x < q$.
5. $y = g^x \bmod p$ is the public verification key.

Signature Algorithm. Let M be the message to be signed. The message is first hashed using the hash function SHA-1, let m be the resulting hash value. The signer picks a random number k such that $1 \leq k < q$, calculates $k^{-1} \bmod q$, and sets

$$\begin{aligned} r &= (g^{k^{-1}} \bmod p) \bmod q \\ s &= k(m + xr) \bmod q \end{aligned}$$

The pair (r, s) is a signature of m.

Verification Algorithm. A signature (r, s) of a message M can be publicly verified by first computing the SHA-1 hash m of M and then by checking that $r = (g^{ms^{-1}} y^{rs^{-1}} \bmod p) \bmod q$, where the inverse of s in the exponent is computed modulo q.

4.3.2 Sharing of a Refresh Polynomial

A basic building block of many threshold protocols is a protocol that refreshes a secret-sharing, denoted ZVSS, which first appeared in [BGW88]. Such protocol generates a collective sharing of a "secret" whose value is zero. This is achieved by running a slightly modified RVSS, where each player instead of choosing the secret a_i that he wants to share at random, fixes it as $a_i = 0$. Namely, ZVSS proceeds as RVSS except that in the sharing, polynomials

f_{a_i} and $f_{\hat{a}_i}$ have free coefficients equal to zero, and thus a verification value $g^{f_{a_i}(0)}h^{f_{\hat{a}_i}(0)}$ is equal to 1 (i.e. $F_{a_i}(0) = 1$ for all $P_i \in Qual$). We include this protocol in Figure 4-11. We call the outputs of ZVSS a *zero-sharing*, denoted ZVSS-data$[a]$ (or ZVSS-data$[b]$, ZVSS-data$[c]$, etc), and we call it *correct*, and the execution of ZVSS *successful*, if these outputs form a correct joint secret-sharing, i.e. if they satisfy the properties of Figure 4-7, with the additional property that all the secret-sharing polynomials defined by the shares of the uncorrupted players have the free coefficients equal to zero.[7] Both the robustness and secrecy properties of RVSS naturally carry over to ZVSS. We state these properties in Lemmas 9 and 10 below. In particular, note that the secrecy property of ZVSS is stated similarly to that of RVSS, namely in terms of a simulation of any distributed protocol $\mathcal{P}$ which takes a zero-sharing ZVSS-data as its inputs where the simulator replaces the private data of the (simulated) uncorrupted players according to procedure $\mathcal{T}_{ZVSS}$ of Figure 4-12. This procedure is a straightforward modification of the corresponding procedure $\mathcal{T}_{RVSS}$ (Figure 4-8) used to express the secrecy property of protocol RVSS.

Applications of ZVSS: "Refreshment" of a Secret-Sharing. Notice that if polynomial f_a secret-shares some secret $a = f_a(0)$, then if an execution of ZVSS creates a "refresh" secret-sharing ZVSS-data$[b]$ with a polynomial f_b s.t. $f_b(0) = 0$, then the two polynomials can be added to create a new secret-sharing polynomial f'_a which shares the same secret $a = f'_a(0)$, but is otherwise independent from f_a. In this way we obtain a re-randomization of the secret-sharing of a without changing the secret itself. Indeed, to refresh a secret-sharing data structure RVSS-data$[a]$ we create ZVSS-data$[b]$ with ZVSS, and each player can locally compute its part of a new secret-sharing RVSS-data$[a']$ of the same secret $a' = a$. This is the typical way that the ZVSS protocol is used in threshold cryptography. (The idea that a "zero-sharing" polynomial can be used in this way to re-randomize an existing secret-sharing dates back to [BGW88].) In the threshold protocol discussed in this thesis, ZVSS is utilized in this way in the threshold multiplication protocol of Section 4.3.3 and in the proactive protocols of Appendix B.

Definition 16 *We call an instance of* ZVSS successful *if its outputs* ZVSS-data$[b]$ *form a* correct *joint zero secret-sharing, which means that they form a correct joint secret-sharing which satisfies an additional property that for every $P_j \in Qual$, the secret-shared polynomials $f_{b_j}(z), f_{\hat{b}_j}(z)$ satisfy constraint $f_{b_j}(0) = f_{\hat{b}_j}(0) = 0$, and thus the secret-shared polynomials $f_b(z), f_{\hat{b}}(z)$ satisfy constraint $f_b(0) = f_{\hat{b}}(0) = 0$ too.*

Lemma 9 (Robustness of ZVSS)
Consider an execution of (1) protocol Ped-IG *(Figure 7-1) which on public input 1^k outputs a Pedersen commitment (p, q, g, h), (2) any other protocol $\mathcal{P}$, and (3) protocol* ZVSS *on public input (p, q, g, h).*

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/2$-threshold adversary $\mathcal{A}$, except for probability negligible in the security parameter k, the above instance of ZVSS *is* successful.

[7] See Figure 4-7. By *all* secret-sharing polynomials defined by a correct joint secret-sharing RVSS-data$[a]$ we mean polynomials $f_a, f_{\hat{a}}$, and the polynomials shared by all players in *Qual*, i.e. $f_{a_i}, f_{\hat{a}_i}$ for $P_i \in Qual$.

Protocol ZVSS creates a "refreshment" secret-sharing ZVSS-data, ($2t+1 \leq n$)

[**Note:** The only difference between these protocols and RVSS and RVSS-REC of Figure 4-6 is that here the secret-shared values are equal to zero.]

Threshold Parameter:	t, the degree of the generated polynomials
Public Input:	Pedersen commitment instance (p, q, g, h)
Public Output:	set $Qual \in \{P_1, ..., P_n\}$ of qualified players
	verification function $F_b(z)$
	verification functions $F_{b_i}(z)$ for $P_i \in Qual$
Secret Output of Player P_i:	$f_{b_i}(z), f_{\hat{b}_i}(z)$, secret-sharing polynomials of P_i
	$\beta_{ji}, \hat{\beta}_{ji}$ shares sent by P_j to P_i for $P_j \in Qual$
	$\beta_i, \hat{\beta}_i$, *polynomial* shares of P_i

(a)-(c) Each P_i performs the PedVSS protocol (with threshold parameter t) as a dealer (Steps (a)-(c), Figure 4-3), using polynomials $f_{b_i}(z)$ and $f_{\hat{b}_i}(z)$ s.t. $f_{b_i}(0) = f_{\hat{b}_i}(0) = 0$. All these instances of PedVSS proceed in parallel. We denote the shares P_i sends out as $\beta_{ij} = f_{b_i}(j), \hat{\beta}_{ij} = f_{\hat{b}_i}(j)$, and its public verification function as $F_{b_i}(z) = g^{f_{b_i}(z)} h^{f_{\hat{b}_i}(z)}$.

(d) After Step (c) each player forms a set of non-disqualified players $Qual$. This set excludes also all players P_j such that the verification function they published does not satisfy $F_{b_j}(0) = 1$. Each player computes its final secret-shares as $\beta_i = \sum_{P_j \in Qual} \beta_{ji}$ and $\hat{\beta}_i = \sum_{P_j \in Qual} \hat{\beta}_{ji}$. The verification function $F_b(z)$ is defined as $\prod_{P_j \in Qual} F_{b_j}(z)$.

Reconstruction Protocol ZVSS-REC:

Each P_i broadcasts values $(\beta_i, \hat{\beta}_i)$ s.t. $g^{\beta_i} h^{\hat{\beta}_i} = F_b(i)$. If all shares β_i that pass this verification interpolate into some t-degree polynomial $f_b(z)$ such that $f_b(0) = 0$, then the public output is this polynomial. Otherwise, the public output is null.

Figure 4-11: (ZVSS,ZVSS-REC): Refresh Polynomial Sharing & Reconstruction

Proof Sketch: Protocol ZVSS is a slight modification of RVSS of Figure 4-6. The same proof of the robustness properties of RVSS (Lemma 4) implies that the shares of the honest players interpolate into unique t-degree polynomials $f_{b_j}, f_{\hat{b}_j}$ s.t. $F_{b_j}(z) = g^{f_{b_j}(z)} h^{f_{\hat{b}_j}(z)}$ for $P_j \in Qual$. The only new robustness property is that we demand that all the accepted secret-sharing polynomials $f_{b_j}, f_{\hat{b}_j}$ have the free coefficient equal to zero.

We can show that under the discrete-log assumption this happens except for at most negligible probability by using an almost identical reduction to the one used in the proof of Lemma 1. Namely, we show that if a corrupted player P_j manages to get the honest players to accept, with higher than negligible probability, secret-sharing polynomials $f_{b_j}, f_{\hat{b}_j}$ such that $g^{f_{b_j}(0)} h^{f_{\hat{b}_j}(0)} = 1$ but $(f_{b_j}(0), f_{\hat{b}_j}(0)) \neq (0, 0)$, then there exists a PPT TM algorithm

$\mathcal{E}$, called *extractor*, which can compute discrete-logarithms $\log_g \tilde{g}$ on input a DLog instance (p, q, g) and a random $\tilde{g} \in G_q$. This extractor construction is a slight variation of the one used in the proof of Lemma 1. By following the simulator $\mathsf{SIM}^{(2)}_{h-IG}$ during the h-IG part of the Ped-IG protocol (Figures 7-3 and 7-1), on public input (p, q, g) and $\mathsf{SIM}^{(2)}_{h-IG}$'s private input $\tilde{g}$, extractor $\mathcal{E}$ embeds the instance $\tilde{g}$ of the discrete-log problem into the generated Pedersen commitment instance $h = g^a \tilde{g}^b$. By Lemma 36, page 177, $\mathcal{A}$'s view of this interaction is only negligibly different from its view of an actual h-IG protocol. Therefore if $\mathcal{A}$ has a higher than negligible probability of cheating in an execution of ZVSS which follows h-IG, then it has also a higher than negligible probability of cheating in an execution of ZVSS which follows the above simulation of h-IG. Furthermore, except for negligible probability, $\mathcal{E}$ learns the representation a, b of the created h in bases $g, \tilde{g}$. When in the subsequent execution of ZVSS, the adversary cheats in the way described above, then $\mathcal{E}$ gets two representations $0, 0$ and $f_{b_j}(0), f_{\hat{b}_j}(0)$, of 1 in bases g, h. (Note that $\mathcal{E}$, who has control over $t+1$ players, can reconstruct polynomials f_{b_j} and $f_{\hat{b}_j}$ shared by P_j.) Thus $\mathcal{E}$ can compute $\log_g h$, and consequently also $\log_g \tilde{g}$. □

Lemma 10 (Polynomial Secrecy of ZVSS)
There exists a simulator SIM, *such that for every $n/2$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah, *for any distributed protocol $\mathcal{P}$, for every discrete-logarithm instance (p, q, g), for every $h \in G_q$, the following two variables are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a run of* ZVSS *on public inputs (p, q, g, h) and $\mathcal{A}$'s input* ah, *with outputs denoted* ZVSS-data$[b]$
  - *a run of $\mathcal{P}$ on input* ZVSS-data$[b]$
- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input $\sigma = \log_g h$*
  - *a run of* ZVSS *on public inputs (p, q, g, h) and $\mathcal{A}$'s input* ah, *with outputs denoted* ZVSS-data$[b]$
  - *a replacement of the private data in* ZVSS-data$[b]$ *of the simulated players with the data specified by* ZVSS-data$[b^*] = \mathcal{T}_{ZVSS}$(ZVSS-data$[b], \sigma$) *and then a run of $\mathcal{P}$ on input* ZVSS-data$[b^*]$

Less Formally: A (static) adversarial view of an execution of the (ZVSS,$\mathcal{P}$) sequence is *the same* as an adversarial view of an execution of ZVSS, a *modification* of the private data of the uncorrupted players according to procedure $\mathcal{T}_{ZVSS}$ (which replaces ZVSS-data$[b]$ with a sufficiently random-looking ZVSS-data$[b^*]$), and then an execution of $\mathcal{P}$.
Informally: A static secure-channels $n/2$-threshold adversary learns nothing about polynomial $f_b(z)$ shared in ZVSS protocol except of the shares $f_b(i)$, $P_i \in Bad$, it receives.

Proof Sketch: The proof is a straightforward extension of the polynomial secrecy of a single PedVSS protocol, i.e. of Lemma 3, to the case where the secret-sharing polynomials

ZVSS-data $\rightarrow$ ZVSS-data* **replacement procedure** $\mathcal{T}_{ZVSS}$

Input: public and private data of uncorrupted players in zero-sharing ZVSS-data
Pedersen's trapdoor $\sigma = \log_g h$
Output: (private data of uncorrupted players in) zero-sharing ZVSS-data*

Let $Bad, Good$ be the identities of the corrupted and uncorrupted players ($|Bad| = t$).
Let P_S be any uncorrupted player. Take polynomials $f_{b_S}, f_{\hat{b}_S}$ contained in P_S's outputs in ZVSS-data, i.e. the polynomials this player dealt to create ZVSS-data.
$\mathcal{T}_{ZVSS}$ picks random t-degree polynomials $f^*_{b_S}, f^*_{\hat{b}_S}$ s.t.:

$$f^*_{b_S}(i) = f_{b_S}(i) \text{ for } P_i \in Bad \quad \text{and} \quad f^*_{b_S}(0) = f^*_{\hat{b}_S}(0) = 0$$
$$f^*_{b_S}(z) + \sigma f^*_{\hat{b}_S}(z) \quad = \quad f_{b_S}(z) + \sigma f_{\hat{b}_S}(z) \text{ (for all } z)$$

and then forms ZVSS-data* from ZVSS-data by:
- replacing the private data $f_{b_S}, f_{\hat{b}_S}$ of P_S with $f^*_{b_S}, f^*_{\hat{b}_S}$
- replacing the private data $\beta_{Si}, \hat{\beta}_{Si}$ of each P_i in $Good$ with $\beta^*_{Si} = f^*_{b_S}(i)$ and $\hat{\beta}^*_{Si} = f^*_{\hat{b}_S}(i)$
- replacing the private data β_i, β_i of these players with values $\beta^*_i = \sum_{P_j \in Qual \setminus \{P_S\}} \beta_{ji} + \beta^*_{Si}$ and $\hat{\beta}^*_i = \sum_{P_j \in Qual \setminus \{P_S\}} \hat{\beta}_{ji} + \hat{\beta}^*_{Si}$

Figure 4-12: $\mathcal{T}_{ZVSS}$: Auxiliary procedure for simulation of ZVSS

$f_{b_i}, f_{\hat{b}_i}$ used by player P_i have a free coefficient equal to zero. Then, by the same argument as used in the proof of secrecy of RVSS, i.e. in Lemma 6, the lemma follows.

Namely, note that for any uncorrupted player P_j, the adversarial view of the secret-sharing PedVSS performed by P_j using some t-degree polynomials $f_{b_j}, f_{\hat{b}_j}$ s.t. $f_{b_j}(0) = f_{\hat{b}_j}(0) = 0$ is the same as the adversarial view if P_j used any other polynomials $f^*_{b_j}, f^*_{\hat{b}_j}$ s.t. $f^*_{b_j}(0) = f^*_{\hat{b}_j}(0) = 0$, where $f^*_{b_j}(z) + \sigma f^*_{\hat{b}_j}(z) = f_{b_j}(z) + \sigma f_{\hat{b}_j}(z)$, and f_{b_j} and $f^*_{b_j}$ agree on values corresponding to players $P_i \in Bad$. Therefore, the private data of the uncorrupted players in ZVSS-data[b] can be replaced according to the procedure $\mathcal{T}_{ZVSS}$, and thus the adversarial view of a sequence

$$\text{ZVSS} \rightarrow \text{ZVSS-data} \ ; \ \mathcal{T}_{ZVSS}(\text{ZVSS-data}, \sigma) \rightarrow \text{ZVSS-data}^* \ ; \ \mathcal{P}(\text{ZVSS-data}^*)$$

has the same distribution as the adversarial view of a sequence

$$\text{ZVSS} \rightarrow \text{ZVSS-data} \ ; \ \mathcal{P}(\text{ZVSS-data})$$

□

4.3.3 Threshold Multiplication of Two Shared Secrets

We present a distributed multiplication protocol Mult which computes a product of two shared secrets. Given two secret-sharing data structures RVSS-data[a] and RVSS-data[b], we show how to compute sharing $c_1, ..., c_n$ of product $c = ab$, and then reconstruct c. The

multiplication protocol Mult is very simple. If we have sharings of two t-degree polynomials f_a and f_b such that $f_a(0) = a$ and $f_b(0) = b$, then we can create a sharing of $2t$-degree polynomial $f_c(z) = f_a(z) * f_b(z)$. Namely, given RVSS-data$_t[a]$ and RVSS-data$_t[b]$, each player P_i locally multiplies his polynomial share α_i of a and his polynomial share β_i of b, and the result is a share c_i of $c = ab$ on a polynomial $f_c(z)$ of degree $2t$.

Consequently, the value c can still be reconstructed by polynomial interpolation of enough correct shares. However, the new sharing can be reconstructed only with a lower $t < n/4$ threshold of faults, because we do not create any public verification information for it. Instead, to assure robust reconstruction of c we rely on error-correcting codes. We will use notation ECSS-REC to denote a reconstruction procedure where each player broadcasts its share $c_i = \alpha_i\beta_i$ of $c = ab$ and each player computes the value c by using the Berlekamp-Welch error-correction procedure [BW], which we denote as $c =$ EC-Interpolate$(c_1, \ldots, c_n)$. If $\{c_1, \ldots, c_n\}$ $(4t + 1 \leq n)$ is a set of values broadcast by players during the ECSS-REC procedure, and if at least $3t$ of the values lie on some $2t$-degree polynomial $f_c(z)$, then c is defined as $f_c(0)$ and can be efficiently computed. We will call the outputs of Mult, i.e. the shares c_i of the uncorrupted players and an adversarial history output by the adversary, an *EC-sharing* of c, and we will denote this distributed data-structure as ECSS-data$_{2t}[c]$. We will call such sharing *correct* if the shares of the uncorrupted players interpolate to some $2t$-degree polynomial.

Note that we need a re-randomization procedure to protect the secrecy of the multiplied secret. This randomization is essential because a polynomial of degree $2t$ which is a product of two polynomials of degree t is not a random polynomial, and might therefore expose information about a and b. Such randomization is achieved by generating ZVSS-data$_{2t}[d]$ and adding the resulting shares to the shares of ab (see Figure 4-13).

We note that the above threshold multiplication protocol is a simplified version of the protocols presented in [BGW88, CCD88].[8]

Definition 17 *We call an instance of protocol* Mult *executed on correct joint secret-sharings* RVSS-data$[a, b]$ successful *if and only if the instance of* ZVSS *invoked within this* Mult *was successful.*

Remark. The above definition is more restrictive than a natural definition of Mult's success would be, namely that Mult is successful if it outputs a correct sharing ECSS-data$[c]$ s.t. $c = ab$, where a, b, c are defined by correct sharings RVSS-data$[a]$, RVSS-data$[b]$, ECSS-data$[c]$. We adopt a more restrictive definition strictly for the purpose of simplifying notation when expressing the *secrecy* property of the Mult protocol (Lemma 12 below), as well as the

[8] In contrast to those works, in the case of the threshold DSS protocol we present in this chapter, shared secrets are multiplied only once, thus saving most of the complexity of the solutions in the above works which deal with the problem of repetitive multiplication. This is it is enough that the Mult protocol produces a secret-sharing ECSS-data$_{2t}[c]$, which shares c with a $2t$-degree polynomial and has worse verifiability threshold than Pedersen-VSS-like "joint secret-sharing" RVSS-data$_t[c]$. We note that the idea of avoiding complicated degree-reduction steps when there is only one multiplication to perform appears also in [Bea87]. However, in Section 4.4.2 we present an optimally-resilient $(t < n/2)$ distributed multiplication protocol of [GRR98] which on input a secret-sharing of two variables outputs a secret-sharing of their product which can then be used as an input to another instance of the distributed multiplication protocol. Namely, the threshold multiplication protocol of Section 4.4.2 produces RVSS-data$_t[c]$ as its output.

Statically Secure $n/4$-Threshold Mult, $(4t+1 \leq n)$

Input: sharings RVSS-data$_t[a]$ and RVSS-data$_t[b]$ (See Fig. 4-7)
Output: error-correcting sharing ECSS-data$_{2t}[c]$, which consists of:
- share c_i for each uncorrupted player P_i
- adversarial output ah

(where $c = ab$ if the adversary does not cheat)

1. Players execute ZVSS with security parameter $2t$, to create ZVSS-data$_{2t}[d]$. Denote the polynomial share of P_i in ZVSS-data$_{2t}[d]$ as δ_i.
2. Each player computes its share $c_i = \alpha_i\beta_i + \delta_i$ of a $2t$-degree polynomial f_c which secret-shares $c = ab$.

Reconstruction Protocol ECSS-REC:

Each P_i broadcasts c_i in ECSS-data$_{2t}[c]$ and computes $c = \text{EC-Interpolate}(c_1, \ldots, c_n)$.

Figure 4-13: Mult: $n/4$-Threshold Multiplication of Two Shared Secrets

secrecy properties of other threshold protocols that utilize Mult as their building block, i.e. Reciprocal and DSS-TSig which we present subsequently.

Lemma 11 (Robustness of $n/4$-Threshold Mult)
Consider an execution of the following sequence of protocols: (1) protocol Ped-IG *(Figure 7-1) which on public input 1^k outputs a Pedersen commitment instance (p, q, g, h), (2) some protocol $\mathcal{P}$ which on input (p, q, g, h) produces outputs which contain two instances of joint secret-sharing* RVSS-data$[a, b]$*, and (3) protocol* Mult *on input* RVSS-data$[a, b]$.

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/4$-threshold adversary (i.e. $4t+1 \geq n$), if the output of $\mathcal{P}$ contains two instances RVSS-data$[a, b] \in \mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$ *then, except for probability negligible in the security parameter k, the above instance of* Mult *is* successful.

Moreover, such instance of Mult *outputs a correct EC-sharing* ECSS-data$_{2t}[c]$ *s.t. $c = ab$ where a and b are shared in* RVSS-data$[a, b]$ *and c is shared in* ECSS-data$_{2t}[c]$.

Proof: That Mult is successful except for at most negligible probability follows immediately from the fact that under the discrete-log assumption protocol ZVSS is successful except for at most negligible probability.

From the robustness properties of RVSS and ZVSS it also follows that if these protocols are successful then every honest party P_i holds $\alpha_i, \beta_i, \delta_i$ s.t. shares δ_i of honest players lie on a unique $2t$-degree polynomial f_d s.t. $f_d(0) = 0$, while shares α_i and β_i of honest players lie on unique t-degree polynomials f_a, f_b. Therefore shares c_i broadcast by the honest players in Step 2 all lie on some $2t$-degree polynomial $f_c(z) = f_a(z)f_b(z) + f_d(z)$ s.t. $f_c(0) = f_a(0)f_b(0) + f_d(0) = ab + 0 = ab$. □

The robustness of the reconstruction of c from ECSS-data$_{2t}[c]$ created by Mult is guaranteed by the Berlekamp-Welsh error-correction procedure [BW]:

Fact 2 *When the reconstruction procedure* ECSS-REC *is executed in the presence of $n/4$-threshold adversary (i.e. if $n \geq 4t+1$) on a* correct *EC-sharing* ECSS-data$_{2t}[c]$ *then the public output of this* ECSS-REC *is c shared in* ECSS-data$_{2t}[c]$, *i.e. $c = f_c(0)$ where f_c is the $2t$-degree polynomial defined by the shares of the uncorrupted players.*

Secrecy Property of the Mult Protocol

The Mult protocol does not reveal any information about the secret-sharing polynomials f_a, f_b in RVSS-data$[a]$ and RVSS-data$[b]$. We state this secrecy property of protocol Mult in Lemma 12 below, by considering an adversarial view of an adversary who participates in two instances of the secret-sharing protocol RVSS, which create RVSS-data$[a]$ and RVSS-data$[b]$, in protocol Mult which outputs ECSS-data$_{2t}[c]$ on input RVSS-data$[a,b]$ (with the property that $c = ab$ provided that the adversary does not manage cheat in these protocols), and in some subsequent protocol $\mathcal{P}$ on inputs RVSS-data$[a,b]$ and ECSS-data$_{2t}[c]$, for example a reconstruction of all these secret-sharings via RVSS-REC and ECSS-REC. We will show that this view is identical to the adversarial view of a simulation in which the simulator first follows the protocol of two RVSS's and Mult, and then replaces the private data of the uncorrupted players in RVSS-data$[a,b]$ and ECSS-data$_{2t}[c]$ in a way which corresponds to replacement of all secret-sharings performed by some uncorrupted player with secret-sharing of new uniformly distributed values. If all these protocols are successful (i.e. the adversary does not cheat) then this corresponds to a replacement of the secret-sharings of a, b and $c = ab$ with secret-sharings of new uniformly distributed values a^*, b^* and $c^* = a^*b^*$. In this sense, Lemma 12 states that the Mult protocol does not reveal anything about a, b, c except the fact that $ab = c$. The proof of that Lemma is a straightforward implication of the privacy of RVSS and ZVSS, because the Mult protocol itself is just a creation of zero-sharing RVSS-data$[d]$ via ZVSS and computing $f_c = f_a f_b + f_d$. Therefore, as we show in Lemma 12, any distributed protocol $\mathcal{P}$, which follows an execution of Mult with inputs RVSS-data$[a,b]$ and outputs ECSS-data$[c]$, and which uses RVSS-data$[a,b]$ and ECSS-data$[c]$ as its inputs, can be simulated to random outputs a^*, b^*, c^* subject to the only constraint that $c^* = a^*b^*$. This claim is formalized using the auxiliary simulator procedure $\mathcal{T}_{RVSS}$ which replaces Mult's inputs RVSS-data$[a,b]$ with random RVSS-data$[a^*,b^*]$, and using the auxiliary simulator procedure $\mathcal{T}_{ECSS}$, Figure 4-14, which replaces Mult's outputs ECSS-data$[c]$ with a random EC-sharing ECSS-data$[c^*]$ subject to the constraints that it agrees with ECSS-data$[c]$ on points held by the adversary, and to the constraint that the secret-shared value c^* is equal to a^*b^* defined by the above RVSS-data$[a^*,b^*]$.

Lemma 12 (Static Secrecy of $n/4$-Threshold Mult)
There exists a simulator SIM, *such that for every $n/4$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah, *for any distributed protocol $\mathcal{P}$, for every discrete-logarithm instance (p,q,g), and for every $h \in G_q$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*

Replacement procedure $\mathcal{T}_{ECSS}$

Input: public and private data of uncorrupted players in
EC-sharing ECSS-data$_{2t}[c]$
"target output value" $c^* \in \mathbb{Z}_q$
Output: (private data of uncorrupted players in) EC-sharing ECSS-data$_{2t}[c^*]$

[**Note:** The following procedure works only if the values c_i, $P_i \in Good$, interpolate to a $2t$-degree polynomial.]
Let $Bad, Good$ be the identities of the corrupted and uncorrupted players ($|Bad| = t$).

1. $\mathcal{T}_{ECSS}$ picks a random $2t$-degree polynomial f_c^* s.t. $f_c^*(0) = c^*$ and $f_c^*(i) = f_c(i)$, where f_c is a $2t$-degree polynomial interpolated from values c_i, $P_i \in Good$
2. $\mathcal{T}_{ECSS}$ forms ECSS-data$_{2t}[c^*]$ by replacing shares c_i in ECSS-data$_{2t}[c]$, for each $P_i \in Good$, with $c_i^* = f_c^*(i)$

Figure 4-14: $\mathcal{T}_{ECSS}$: Auxiliary simulation procedure

- *a run of two* successful *instances of* RVSS *(either parallel or sequential), on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a, b]$
- *a* successful *run of* Mult *on input* RVSS-data$[a, b]$, *with outputs* ECSS-data$_{2t}[c]$
- *a run of* $\mathcal{P}$ *on inputs* RVSS-data$[a, b]$, ECSS-data$_{2t}[c]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input* $\sigma = \log_g h$
  - *a run of two* successful *instances of* RVSS *(either parallel or sequential, as in the corresponding step of the sequence above), on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a, b]$
  - *a* successful *run of* Mult *on input* RVSS-data$[a, b]$, *with outputs* ECSS-data$_{2t}[c]$
  - *a replacement of the private data in* RVSS-data$[a, b]$, *and* ECSS-data$_{2t}[c]$ *of the simulated players with*
    - RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma$)
    - RVSS-data$[b^*] = \mathcal{T}_{RVSS}$(RVSS-data$[b], b^*, \sigma$)
    - ECSS-data$_{2t}[c^*] = \mathcal{T}_{ECSS}$(ECSS-data$_{2t}[c], c^*$)
      where $a^*, b^*, c^* \in \mathbb{Z}_q$ *are random subject to the constraint that* $a^*b^* = c^*$

    and then a run of $\mathcal{P}$ *on* RVSS-data$[a^*, b^*]$ *and* ECSS-data$_{2t}[c^*]$

Less Formally: A (static) adversarial view of *successful* executions of RVSS, Mult on their outputs, and then a random run of protocol $\mathcal{P}$, is *the same* as an adversarial view of a successful execution of 2×RVSS, and Mult, a *modification* of the private data of the uncorrupted players via $\mathcal{T}_{RVSS}$ and $\mathcal{T}_{ECSS}$ (which replace the sharings of a, b, c with sharings of random a^*, b^*, c^* subject to the constraint that $c^* = a^*b^*$), and then an execution of $\mathcal{P}$.

Informally: A static secure-channels $n/4$-threshold adversary learns nothing about secret-sharing polynomials f_a, f_b, f_c from an execution of Mult which on input RVSS-data$[a,b]$ outputs ECSS-data$_{2t}[c]$, except of the shares $f_a(i), f_b(i), f_c(i)$, $P_i \in Bad$, that it receives, and apart of the fact that $c = f_c(0)$ is equal to $ab = f_a(0)f_b(0)$.

Note on the Secrecy Property of Mult. The formalization of the secrecy property of Mult is different than that of RVSS, Exp, or ZVSS, in that we claim simulatability only in the case if the initial RVSS secret-sharings which provide the inputs to Mult were *successful*, i.e. if the sharings RVSS-data$[a,b]$ these protocols produced are *correct* (see Section 4.2.4). The reason is that, unlike in the protocols we discussed so far, if the adversary manages to cheat in the creation of the inputs to Mult, i.e. if the adversary cheats in secret-sharing his contribution $f_{a_B}(z) = \prod_{P_i \in Qual \cap Bad} f_{a_i}(z)$ and $f_{b_B}(z) = \prod_{P_i \in Qual \cap Bad} f_{b_i}(z)$ to RVSS-data$[a]$ or RVSS-data$[b]$, then during the Mult protocol he can actually learn some additional information about the contribution of the uncorrupted players f_{a_G}, f_{b_G} to these sharings (where f_{a_G}, f_{b_G} are defined as f_{a_B}, f_{b_B} above but for $P_i \in Good$). For example, if $f_{a_B}(z) = z^{2t}$ and $f_{b_B}(z)(z) = z^{3t}$, but $f_d(z) = f_{d_B}(z) + f_{d_G}(z)$ is a $2t$-degree polynomial, then the adversary learns $f_c(z) = (f_d(z) + f_{a_G}(z)f_{b_G}(z)) + z^2 f_{b_G}(z) + z^3 f_{a_G}(z) + z^5$, during the reconstruction ECSS-REC on ECSS-data$[c]$ produced by Mult(RVSS-data$[a,b]$), and since the first part of that expression is a $2t$-degree polynomial, if $n > 5t$ the adversary can interpolate this polynomial and learn the entire $f_{a_G}(z)$ and $f_{b_G}(z)$.

Since we had to restrict the secrecy claim to executions of Mult on correct inputs RVSS-data$[a,b]$ only, we also restricted the claim only to successful executions of Mult itself, which, given that the inputs are correct, is the same to say that we restrict ourselves to the case when the adversary does not cheat in the ZVSS protocol. This restriction is not necessary, but we do it for convenience and readability of the simulation algorithm.

This does not matter much in practice because, as we show in Lemma 4, under the discrete-log assumption the probability that the adversary manages to cheat in the RVSS protocol is negligible. And so is the probability that, once RVSS-data$[a,b]$ are correct, the adversary cheats during Mult itself. Therefore, under the discrete-log assumption, for every protocol $\mathcal{P}$, there is a negligible statistical difference between the distributions of adversarial views considered in the above Lemma, once it is modified so that, in both cases, instead of *random successful* execution of the (2×RVSS;Mult) sequence, we consider simply a *random* execution of these protocols. Consequently, under the discrete-log assumption, it does not matter what the simulator of Mult does when the adversary with whom it interacts does manage to cheat in any of these protocols.

Proof: This lemma follows straightforwardly from the secrecy of RVSS which shares a and b and from the secrecy of ZVSS which shares the refresh polynomial f_d. Below we argue this implication formally.

Let Correct(RVSS-data$_t[a]$) be a function which outputs 1 if RVSS-data$_t[a]$ is correct and 0 otherwise. Note that this function can operate only on the *adversary's view* in RVSS-data$_t[a]$, because RVSS-data$_t[a]$ is incorrect if and only if some polynomial f_{a_i} (and hence also $f_{\hat{a}_i}$) for $P_i \in Bad \cap Qual$, as defined by the shares α_{ij} sent by P_i to the honest players $P_j \in Good$, has a degree higher than t. (Note, btw, that this cannot happen if $n = 2t + 1$.) Therefore, the equality of distributions of two adversarial views stated in Lemma 6 holds if we restrict

both to executions of RVSS$\rightarrow$ RVSS-data s.t. Correct(RVSS-data) = 1. We will denote a *random successful* run of RVSS as Succ-RVSS $\rightarrow$ RVSS-data.[9]

By Lemma 6, the following two adversarial views have the same distributions:

$$\begin{aligned} 2 \times \text{Succ-RVSS} &\rightarrow \text{RVSS-data}[a, b] \\ \text{Mult}(\text{RVSS-data}[a, b]) &\rightarrow \text{ECSS-data}[c] \\ \mathcal{P}(\text{RVSS-data}[a, b], &\text{ECSS-data}[c]) \end{aligned}$$

and

$$\begin{aligned} 2 \times \text{Succ-RVSS} &\rightarrow \text{RVSS-data}[a, b] \\ \mathcal{T}_{RVSS}(\text{RVSS-data}[a, b], \sigma) &\rightarrow \text{RVSS-data}[a^*, b^*] \\ \text{Mult}(\text{RVSS-data}[a^*, b^*]) &\rightarrow \text{ECSS-data}[c^*] \\ \mathcal{P}(\text{RVSS-data}[a^*, b^*], &\text{ECSS-data}[c^*]) \end{aligned}$$

Note that Mult is a composition of ZVSS$\rightarrow$ ZVSS-data$[d]$ and a local translation of shares RVSS-data$[a, b]$, ZVSS-data$[d] \overset{Mult,\text{St.2}}{\longrightarrow}$ ECSS-data$[c]$ performed in Step 2, Figure 4-13. Therefore, Mult executes successfully on correct RVSS-data$[a, b]$ if and only if the execution of ZVSS inside Mult is itself successful. Therefore, we have a following equality of distribution of adversarial views:

$$\begin{aligned} 2 \times \text{Succ-RVSS} &\rightarrow \text{RVSS-data}[a, b] \\ \text{Succ-Mult}(\text{RVSS-data}[a, b]) &\rightarrow \text{ECSS-data}[c] \\ \mathcal{P}(\text{RVSS-data}[a, b], &\text{ECSS-data}[c]) \end{aligned}$$

and

$$2 \times \text{Succ-RVSS} \rightarrow \text{RVSS-data}[a, b] \tag{4.3}$$

$$\mathcal{T}_{RVSS}(\text{RVSS-data}[a, b], \sigma) \rightarrow \text{RVSS-data}[a^*, b^*] \tag{4.4}$$

$$\text{Succ-ZVSS} \rightarrow \text{ZVSS-data}[d] \tag{4.5}$$

$$(\text{RVSS-data}[a^*, b^*], \text{ZVSS-data}[d]) \overset{Mult,\text{St.2}}{\longrightarrow} \text{ECSS-data}[c^*] \tag{4.6}$$

$$\mathcal{P}(\text{RVSS-data}[a^*, b^*], \text{ECSS-data}[c^*]) \tag{4.7}$$

Note that given that the adversary is static, the last view remains the same if Steps 4.4 and 4.5 in the above sequence are switched, because they both represent only local computation of the (simulated) uncorrupted players. Furthermore, by the secrecy of ZVSS, Lemma 10,

[9] We define Succ-ZVSS, Succ-Mult, etc in the same way, as random *successful* instances of the appropriate protocol. Note that for any protocol $\mathcal{P}$ running on some inputs I, if $\mathcal{P}$ is defined as successful iff its outputs are correct, and if this correctness can be determined from the information visible to the adversary during $\mathcal{P}$, i.e. from the public information produced and from the messages exchanged between the adversary and the uncorrupted players, then we can perform a similar restriction for $\mathcal{P}$ as we did above for RVSS. I.e. it then follows that if the adversarial view of sequence $\mathcal{P}_1 \rightarrow I$; $\mathcal{P}(I)$ is identical to the adversarial view of $\mathcal{P}_2 \rightarrow I; \mathcal{P}(I)$, then also the adversarial views of sequence $\mathcal{P}_1 \rightarrow I$; Succ-$\mathcal{P}(I)$ and $\mathcal{P}_2 \rightarrow I$; Succ-$\mathcal{P}(I)$ are identical.

also restricted to *successful* executions of this protocol, it follows that the adversarial view of the above sequence has the same distribution as the following adversarial view:

$$2 \times \mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[a, b] \tag{4.8}$$
$$\mathsf{Succ\text{-}ZVSS} \rightarrow \mathsf{ZVSS\text{-}data}[d] \tag{4.9}$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[a, b], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[a^*, b^*] \tag{4.10}$$
$$\mathcal{T}_{ZVSS}(\mathsf{ZVSS\text{-}data}[d], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[d^*] \tag{4.11}$$
$$(\mathsf{RVSS\text{-}data}[a^*, b^*], \mathsf{ZVSS\text{-}data}[d^*]) \stackrel{Mult,\mathsf{St.2}}{\longrightarrow} \mathsf{ECSS\text{-}data}[c^*] \tag{4.12}$$
$$\mathcal{P}(\mathsf{RVSS\text{-}data}[a^*, b^*], \mathsf{ECSS\text{-}data}[c^*]) \tag{4.13}$$

Moreover, we argue that a sequence of Steps [4.10; 4.11; 4.12], looks identical, to a static adversary, as the following sequence:

$$(\mathsf{RVSS\text{-}data}[a, b], \mathsf{ZVSS\text{-}data}[d]) \stackrel{Mult\mathsf{St.2}}{\longrightarrow} \mathsf{ECSS\text{-}data}[c] \tag{4.14}$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[a, b], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[a^*, b^*] \tag{4.15}$$
$$\mathcal{T}_{ECSS}(\mathsf{ECSS\text{-}data}[c], c^* = a^*b^*) \rightarrow \mathsf{ECSS\text{-}data}[c^*] \tag{4.16}$$

Since Steps 4.9 and 4.14 together describe a random *successful* execution of Mult, sequence [4.8; 4.9; 4.14; 4.15; 4.16; 4.13] describes exactly the distribution of an adversarial view during a simulation of Mult specified in the claim of the lemma we are proving. Hence, the equality of adversarial views in [4.10; 4.11; 4.12] and [4.14; 4.15; 4.16] implies our lemma.

In other words, and this is an essence of this lemma, we show that the simulator of Mult can *first* perform the Mult protocol on RVSS-data$[a, b]$, the data that the simulator knows, and *then* simulate any protocol $\mathcal{P}$ to some "target outputs" RVSS-data$[a^*, b^*]$ and ECSS-data$[c^*]$ given by transformations of RVSS-data$[a, b]$ and Mult's output ECSS-data$[c]$, as specified by $\mathcal{T}_{RVSS}$ and $\mathcal{T}_{ECSS}$ and by the constraint that $a^*b^* = c^*$.

The equality of [4.10; 4.11; 4.12] and [4.14; 4.15; 4.16] in the eyes of a static adversary, where both are executed on correct RVSS-data$[a, b]$ and ZVSS-data$[d]$ (or more precisely, when both are preceded by [4.8; 4.9]), is easy to see. First note that since the adversary is static, we can switch steps 4.14 and 4.15, and so we need to show that given a view of

$$2 \times \mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[a, b] \quad (4.8)$$
$$\mathsf{Succ\text{-}ZVSS} \rightarrow \mathsf{ZVSS\text{-}data}[d] \quad (4.9)$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[a, b], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[a^*, b^*] \quad (4.10)$$

the adversarial view of the following two sequences are identical:

$$\mathcal{T}_{ZVSS}(\mathsf{ZVSS\text{-}data}[d], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[d^*] \quad (4.11)$$
$$(\mathsf{RVSS\text{-}data}[a^*, b^*], \mathsf{ZVSS\text{-}data}[d^*]) \stackrel{Mult,\mathsf{St.2}}{\longrightarrow} \mathsf{ECSS\text{-}data}[c^*] \quad (4.12)$$

and

$$(\mathsf{RVSS\text{-}data}[a, b], \mathsf{ZVSS\text{-}data}[d]) \stackrel{Mult\mathsf{St.2}}{\longrightarrow} \mathsf{ECSS\text{-}data}[c] \quad (4.14)$$

$$\mathcal{T}_{ECSS}(\mathsf{ECSS\text{-}data}[c], c^* = a^*b^*) \rightarrow \mathsf{ECSS\text{-}data}[c^*] \quad (4.16)$$

This is easy to see because in both cases shares c_i^* in the resulting ECSS-data$[c^*]$ interpolate to a $2t$-degree polynomial f_c^* which is random subject to the constraints that $f_c^*(0) = c^* = a^*b^*$ and that $f_c^*(i) = f_c(i)$ for $P_i \in Bad$, where $f_c(i) = f_a(i)f_b(i) + f_d(i)$, and f_a, f_b, f_d are defined by (the outputs of honest players in) outputs of sequence [4.8; 4.9]. □

Secrecy of Repetitive Execution of Mult

Unfortunately, Lemma 12 does not seem to be general enough to imply the secrecy property of the Mult protocol under *repetitive execution*. Namely, we need to know whether a repeated execution of Mult on secret-sharing RVSS-data$[a]$ and a sequence of secret-sharings RVSS-data$[b_1]$, RVSS-data$[b_2]$, ..., RVSS-data$[b_{p(k)}]$, where $p(z)$ is some polynomial, does not reveal anything about the secret-sharing polynomials f_a, f_{b_1}, f_{b_2}, ..., $f_{b_{p(k)}}$, and the created polynomials f_{c_1}, f_{c_2}, ..., $f_{c_{p(k)}}$, apart of the fact that $f_a(0)f_{b_i}(0) = f_{c_i}(0)$ for all $i = 1, .., p(k)$. Such "repetitive secrecy" of Mult is needed to argue that a repeated use of the DSS signature generation protocol DSS-TSig (Section 4.3.5) does not reveal the key material, and thus allows us to prove the unforgeability of the threshold DSS scheme which uses the DSS-TSig protocol (Theorem 3). The repetitive secrecy of Mult is relevant there because each time DSS-TSig runs to sign some message m_i, it executes Mult on RVSS-data$[x]$ which secret-shared the DSS secret key x and on RVSS-data$[k_i]$ which is a secret-sharing created in the i-th instance of the DSS-TSig protocol to share a one-time DSS secret k_i.

Once we identify a need for such re-formulation of the secrecy property of Mult we should take care of a few further extensions of this property which reflect the use of the Mult protocol in a threshold DSS signature scheme of Section 4.3. First, note that it can be shown by following the arguments of the proof of Lemma 12, that if an execution of Mult which outputs ECSS-data$_{2t}[c]$ on input RVSS-data$[a, b]$ is *preceded* by some distributed protocol $\mathcal{P}$ on inputs RVSS-data$[a, b]$ and followed by some other protocol $\mathcal{P}'$ on inputs (RVSS-data$[a, b]$, ECSS-data$_{2t}[c]$), then the adversarial view of such sequence of executions is still identical to the adversarial view of a simulation of these protocols, in which the simulator performs Mult on RVSS-data$[a, b]$ but instead of executing $\mathcal{P}$ and $\mathcal{P}'$ on RVSS-data$[a, b]$ and ECSS-data$_{2t}[c]$, executes them instead on RVSS-data$[a^*, b^*]$ and ECSS-data$_{2t}[c^*]$, where RVSS-data$[a^*, b^*]$ and ECSS-data$_{2t}[c^*]$ are modifications of sharings RVSS-data$[a, b]$ and ECSS-data$_{2t}[c]$ via $\mathcal{T}_{RVSS}$ and $\mathcal{T}_{ECSS}$ as in Lemma 12. The proof that this simulation produces a correct view is only a slight variation of the proof of Lemma 12.

In Lemma 13 below, we state a more general claim in the sense that we consider an execution of RVSS$\rightarrow$ RVSS-data$[a]$ followed by some protocol $\mathcal{P}_1$(RVSS-data$[a]$) with outputs I_1, then an execution of RVSS$\rightarrow$ RVSS-data$[b]$ followed by some protocol $\mathcal{P}_2$(RVSS-data$[b], I_1$) with outputs I_2, and only then Mult(RVSS-data$[a, b] \rightarrow$ ECSS-data$[c]$) and $\mathcal{P}_3$(RVSS-data$[a$, $b]$, ECSS-data$[c], I_1, I_2$). However, for ease of notation we restrict the generality of this claim by considering protocols $\mathcal{P}_2$ (respectively, $\mathcal{P}_3$) which takes as inputs only the public outputs of $\mathcal{P}_1$ (respectively, $\mathcal{P}_1$ and $\mathcal{P}_2$).

This observation is relevant to the use of protocol Mult in the threshold DSS signature generation protocol DSS-TSig, where Mult is executed on secret-sharing RVSS-data$[x]$ that shares a DSS secret key x and a secret-sharing RVSS-data$[k_i]$ that shares a one-time secret

k_i. However, before a run of the DSS-TSig protocol (and thus before the execution of Mult), sharing RVSS-data$[x]$ is used as an input to the exponentiation protocol which generates the public key $y = g^x$ from RVSS-data$[x]$ (this exponentiation is a part of the key generation protocol with which a threshold DSS scheme must start, see the DKG protocol).

Secondly, the Mult protocol inside DSS-TSig is performed not on sharings RVSS-data$[k_i]$ and RVSS-data$[x]$ but on RVSS-data$[k_i]$ and RVSS-data$[x'_i]$, where RVSS-data$[x'_i]$ is derived in the i-th instance of the DSS-TSig protocol from RVSS-data$[x]$ by scaling all the secret-sharing values by a multiplicative factor r_i and an additive factor m_i, both of which are public values: m_i is the i-th message submitted by the user for signing and r_i is a first part of the DSS signature on m_i, computed as a public value in the first part of the i-th instance of DSS-TSig. Such scaling computation, denoted Scale, is presented in Figure 4-15 below. We will call any such protocol executed on some a, b which are parts of the public data, i.e. they are marked as public by all honest players, as Scale(RVSS-data$[x]$, Sel), where Sel is a selection procedure which identifies variables a and b in the public transcript of an execution of a threshold scheme so far. In the case of the i-th instance of the DSS-TSig protocol, the selection procedure $\mathsf{Sel}_{DSS\text{-}TSig}$ identifies a as the message m_i submitted for signing, and b as the public output r_i of the first part of this instance of DSS-TSig. If the last value is null, i.e. if the adversary managed to cheat in that protocol, then the honest players abandon this instance of DSS-TSig and the scaling is not executed anyway.

"Scaling" of a Joint Secret-Sharing: Scale(RVSS-data$[x], a, b) \rightarrow$ RVSS-data$[x']$

Input: sharing RVSS-data$[x]$
Public Input: additive factor a and multiplicative factor b
(alternatively, the public input is a selecting procedure Sel, with which honest players identify values a, b in the public output produced by the threshold scheme so far)
Output: sharing RVSS-data$[x']$

Each P_i forms RVSS-data$[x']$ as $a + b$RVSS-data$[x]$ as follows:

$$
\begin{aligned}
f_{x'_i}(z)\ ,\ f_{\hat{x}'_i}(z) &\triangleq a + bf_{x_i}(z)\ ,\ a + bf_{\hat{x}_i}(z) \\
\alpha'_{ji}\ ,\ \hat{\alpha}'_{ji} &\triangleq a + b\alpha_{ji}\ ,\ a + b\hat{\alpha}_{ji} \text{ if } P_j \in \mathit{Qual} \text{ and null otherwise} \\
\alpha'_i\ ,\ \hat{\alpha}'_i &\triangleq a + b\alpha_i\ ,\ a + b\hat{\alpha}_i \\
F_{x'_j}(z) &\triangleq a(F_{x_j}(z))^b \text{ if } P_j \in \mathit{Qual} \text{ and null otherwise} \\
F_{x'}(z) &\triangleq a(F_x(z))^b
\end{aligned}
$$

Figure 4-15: Scale: Protocol for "Scaling" a Joint Secret-Sharing

We summarize this discussion by formalizing the "repetitive secrecy" of Mult in a way that is tailored to the use of Mult in the threshold DSS signature scheme of Section 4.3, and can be used directly in the proof of unforgeability of that scheme in Theorem 3, page 106.

Lemma 13 (Static Secrecy of Repetitive Executions of $n/4$-Threshold Mult) *There exists a simulator* SIM, *such that for every $n/4$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah, *for every distributed protocols $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ and any selection procedure* Sel, *for every discrete-logarithm instance (p, q, g) of security parameter k, for every $h \in G_q$, for any polynomial $p(z)$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a* successful *run of* RVSS *on public input (p, q, g, h) and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a run of $\mathcal{P}_1$ on* RVSS-data$[a]$ *with public outputs denoted I_1*
  - *a cycle of the following executions, for $j = 1, ..., p(k)$:*
    - *a* successful *run of* RVSS *on (p, q, g, h) with outputs denoted* RVSS-data$[b_j]$
    - *a run of $\mathcal{P}_2$ on* RVSS-data$[b_j]$ *and I_1 with public outputs denoted I_2*
    - *scaling procedure* Scale(RVSS-data$[a]$, Sel) $\rightarrow$ RVSS-data$[a'_j]$
    - *a* successful *run of* Mult *on* RVSS-data$[a'_j, b_j]$, *with outputs* ECSS-data$_{2t}[c_j]$
    - *a run of $\mathcal{P}_3$ on* (RVSS-data$[a, b_j]$, ECSS-data$_{2t}[c_j], I_1, I_2$)
- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input $\sigma = \log_g h$:*
  - *a* successful *run of* RVSS *on public input (p, q, g, h) and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a replacement of the private data in* RVSS-data$[a]$ *of the simulated players with* RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], \sigma$), *and then a run of $\mathcal{P}_1$ on* RVSS-data$[a^*]$ *with public outputs denoted I_1*
  - *a cycle of the following executions, for $j = 1, ..., p(k)$:*
    - *a* successful *run of* RVSS *on (p, q, g, h) with outputs denoted* RVSS-data$[b_j]$
    - *a replacement of the private data in* RVSS-data$[b]$ *of the simulated players with* RVSS-data$[b^*_j] = \mathcal{T}_{RVSS}$(RVSS-data$[b_j], \sigma$), *and then a run of $\mathcal{P}_2$ on* RVSS-data$[b^*_j]$ *and I_1 with public outputs denoted I_2*
    - *scaling procedure* Scale(RVSS-data$[a]$, Sel) $\rightarrow$ RVSS-data$[a'_j]$
    - *a* successful *run of* Mult *on* RVSS-data$[a'_j, b_j]$, *with outputs* ECSS-data$[c_j]$
    - *a replacement of the private data in* ECSS-data$[c_j]$ *of the simulated players with* ECSS-data$[c^*_j] = \mathcal{T}_{ECSS}$(ECSS-data$[c_j], c^*_j$), *and then a run of $\mathcal{P}_3$ on* (RVSS-data$[a^*, b^*_j]$, ECSS-data$[c^*_j], I_1, I_2$), *where $c^*_j = a'^*_j b^*_j = (\lambda_1 + \lambda_2 a^*) b^*_j$ and λ_1, λ_2 are the public factors identified by* Sel *in the* Scale *procedure above.*

Informally: A static secure-channels $n/4$-threshold adversary learns nothing about secret-sharing polynomials f_a, f_{b_1}, ..., $f_{b_{p(k)}}$ and $f_{c(1)}$, ..., $f_{c_{p(k)}}$ from the executions of Mult which, for each $j = 1, ..., p(k)$, multiply RVSS-data$[a]$ with every RVSS-data$[b_j]$ and output ECSS-data$[c_j]$, apart of the values of these polynomials held by the corrupted players, and apart of the fact that $c_j = f_{c_j}(0)$ is equal to $a'_j b_j = (\lambda_1 + \lambda_2 f_a(0)) f_{b_j}(0)$ where λ_1, λ_2

are the additive and multiplicative factors identified by procedure Sel in the public output of the protocol.

Proof: The proof is a simple extension of the proof of Lemma 12. We will prove it formally by induction on j, the number of loops of Mult.

The base case, for $j = 0$, i.e. that the adversary view of

$$\begin{aligned} &\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[a] \\ &\mathcal{P}_1(\mathsf{RVSS\text{-}data}[a]) \end{aligned}$$

is distributed identically to its view of

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[a] \tag{4.17}$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[a], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[a^*] \tag{4.18}$$
$$\mathcal{P}_1(\mathsf{RVSS\text{-}data}[a]) \tag{4.19}$$

is implied by the secrecy property of RVSS shown in Lemma 6.

Now, for the recursive case, let's assume that the adversary's view of $j-1$ loops of (successful) execution and successful simulation of the above "repetitions of Mult with $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ mixed in" protocol are identically distributed. It means that, by the secrecy of RVSS and ZVSS, j loops of (successful) protocol execution (where we consider only successful executions of RVSS and Mult but *random* executions of $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$, as in the statement of the lemma) look like:

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[a] \quad (4.17)$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[a], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[a^*] \quad (4.18)$$
$$\mathcal{P}_1(\mathsf{RVSS\text{-}data}[a^*]) \rightarrow I_1 \quad (4.19)$$

followed by $j-1$ loops of the simulation (as in the statement of the lemma) and then

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[b_j] \tag{4.20}$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[b_j], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[b_j^*] \tag{4.21}$$
$$\mathcal{P}_2(\mathsf{RVSS\text{-}data}[b_j^*], I_1) \rightarrow I_2 \tag{4.22}$$
$$\mathsf{Scale}(\mathsf{RVSS\text{-}data}[a^*], \mathsf{Sel}) \rightarrow \mathsf{RVSS\text{-}data}[a_j'^*] \tag{4.23}$$
$$\mathsf{Succ\text{-}ZVSS} \rightarrow \mathsf{ZVSS\text{-}data}[d_j] \tag{4.24}$$
$$\mathcal{T}_{ZVSS}(\mathsf{ZVSS\text{-}data}[d_j], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[d_j^*] \tag{4.25}$$
$$(\mathsf{RVSS\text{-}data}[a_j'^*, b_j^*], \mathsf{ZVSS\text{-}data}[d^*]) \stackrel{Mult,\mathsf{St.2}}{\longrightarrow} \mathsf{ECSS\text{-}data}[c_j^*] \tag{4.26}$$
$$\mathcal{P}_3(\mathsf{RVSS\text{-}data}[a^*, b_j^*], \mathsf{ECSS\text{-}data}[c_j^*], I_1, I_2) \tag{4.27}$$

Because the adversary is static we can move Step 4.23 after Step 4.24, and then we note that, similarly to the proof of Lemma 12, that after participating in execution of Steps

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[a] \quad (4.17)$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[a], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[a^*] \quad (4.18)$$

$$\mathcal{P}_1(\text{RVSS-data}[a^*]) \rightarrow I_1 \quad (4.19)$$

($j-1$ loops of the simulation procedure, see the lemma statement)

$$\text{Succ-RVSS} \rightarrow \text{RVSS-data}[b_j] \quad (4.20)$$

$$\mathcal{T}_{RVSS}(\text{RVSS-data}[b_j], \sigma) \rightarrow \text{RVSS-data}[b_j^*] \quad (4.21)$$

$$\mathcal{P}_2(\text{RVSS-data}[b_j^*], I_1) \rightarrow I_2 \quad (4.22)$$

$$\text{Succ-ZVSS} \rightarrow \text{ZVSS-data}[d_j] \quad (4.24)$$

the adversarial view of Steps

$$\text{Scale}(\text{RVSS-data}[a^*], \text{Sel}) \rightarrow \text{RVSS-data}[a_j'^*] \quad (4.23)$$

$$\mathcal{T}_{ZVSS}(\text{ZVSS-data}[d_j], \sigma) \rightarrow \text{RVSS-data}[d_j^*] \quad (4.25)$$

$$(\text{RVSS-data}[a_j'^*, b_j^*], \text{ZVSS-data}[d^*]) \stackrel{Mult,\text{St.2}}{\longrightarrow} \text{ECSS-data}[c_j^*] \quad (4.26)$$

$$\mathcal{P}_3(\text{RVSS-data}[a^*, b_j^*], \text{ECSS-data}[c_j^*], I_1, I_2) \quad (4.27)$$

look the same as the following view:

$$\text{Scale}(\text{RVSS-data}[a], \text{Sel}) \rightarrow \text{RVSS-data}[a_j'] \quad (4.28)$$

$$(\text{RVSS-data}[a_j', b_j], \text{ZVSS-data}[d]) \stackrel{Mult,\text{St.2}}{\longrightarrow} \text{ECSS-data}[c_j] \quad (4.29)$$

$$\mathcal{T}_{ECSS}(\text{ECSS-data}[c_j], c_j^* = (\lambda_1 + \lambda_2 a^*) b_j^*) \rightarrow \text{ECSS-data}[c_j^*] \quad (4.30)$$

$$\mathcal{P}_3(\text{RVSS-data}[a^*, b_j^*], \text{ECSS-data}[c_j^*], I_1, I_2) \quad (4.31)$$

(where λ_1, λ_2 are the factors identified by the above instance of Sel.)

The equality holds for the same reason as in the proof of Lemma 12, i.e. in both cases polynomial $f_{c_i}^*$ interpolated by the shares of the honest players in the resulting ECSS-data$[c^*]$ is a $2t$-degree polynomial which is random subject to the constraints that $f_{c_j}^*(0) = c_j^* = a_j'^* b_j^* = (\lambda_1 + \lambda_2 a^*) b_j^*$, and that $f_{c_j}^*(i) = f_{c_j}(i)$ for $P_i \in Bad$, where $f_{c_j}(z) = (\lambda_1 + \lambda_2 f_a(z)) f_{b_j}(z) + f_{d_j}(z)$, and f_a, f_{b_j}, f_{d_j} are defined by (the outputs of honest players in) outputs of Steps 4.17, 4.20, and 4.24.

Since Steps [4.28; 4.29; 4.30; 4.31] above describe the adversary's view during the j-th loop of a successful simulation, it follows that j loops of successful protocol execution look like j loops of successful simulation, which proves our lemma. □

Remark. The proof of the inductive case in Lemma 13 above is very similar to the proof of Lemma 12, which seems to suggest that there should be a way of isolating this similarity and stating the secrecy property of the Mult protocol in a way that would imply also the above "repetitive secrecy" property.

4.3.4 Threshold Inverse Computation Protocol

In the distributed DSS protocol we are faced with the following problem. Given a shared secret k, we need to generate public value $g^{(k^{-1} \bmod q)}$, without any other revealing information on k. To achieve this we use a protocol Reciprocal due to Bar-Ilan and Beaver [BB89],

which given a secret-sharing of k creates a secret-sharing of $k^{-1} \bmod q$. This protocol, given a sharing RVSS-data$[a]$ creates a sharing RVSS-data$[e]$ where $e = a^{-1}$ by creating a sharing RVSS-data$[b]$ of a random value, multiplying RVSS-data$[a]$ and RVSS-data$[b]$ with Mult and reconstructing the output with ECSS-REC to get $c = ab$ as a public output (both in Figure 4-13). The Reciprocal protocol we present in Figure 4-16 below is secure only against $n/4$-threshold adversary because it uses the $n/4$-threshold multiplication protocol Mult. When we replace this multiplication protocol with an optimal $t/2$-threshold multiplication protocol Mult-opt of Section 4.4.2, Reciprocal gains an optimal threshold resilience too.

Statically Secure $n/4$-Threshold Reciprocal, $(4t + 1 \leq n)$

Input: sharing RVSS-data$[a]$ (See Fig. 4-7)
Output: sharing RVSS-data$[e]$
(where $e = a^{-1} \bmod q$ if the adversary does not cheat)

1. Players perform RVSS to create a new sharing RVSS-data$[b]$.
2. Players compute ECSS-data$[c] \leftarrow$ Mult(RVSS-data$[a]$, RVSS-data$[b]$)
3. Players compute $c \leftarrow$ ECSS-REC(ECSS-data$[c]$), see Figure 4-13
4. Players compute RVSS-data$[e] \leftarrow$ Scale(RVSS-data$[b], 0, c^{-1}$), see Figure 4-15

Simulator SIM_{Rpcl} of Reciprocal **(interacting with $\mathcal{A}$):**

Public Input:	public data in RVSS-data$[a]$
SIM_{Rpcl}**'s Private Input:**	private outputs of all $P_i \in$ *Good* in RVSS-data$[a]$
$\mathcal{A}$**'s Private Input:**	adversary's output in RVSS-data$[a]$

1.-2. SIM_{Rpcl} follows Steps 1-2 of Reciprocal on behalf of the uncorrupted players.

3. SIM_{Rpcl} picks c^* uniformly in $\mathbb{Z}_q$ and replaces the private data of the simulated players with the data specified by ECSS-data$[c^*] = \mathcal{T}_{ECSS}$(ECSS-data$[c], c^*$).
Then SIM_{Rpcl} performs the ECSS-REC protocol on input ECSS-data$[c^*]$ on behalf of the simulated players.

4. SIM_{Rpcl} follows Step 4 of Reciprocal on behalf of the uncorrupted players.
(Note that if the adversary does not cheat then the public output of the previous step is c^* chosen by SIM_{Rpcl}, and thus the scaling factor used in this step is $(c^*)^{-1}$.)

Figure 4-16: Reciprocal: $n/4$-Threshold Inverse Computation Protocol

In [BB89] it is proven that such protocol is secure against an eavesdropping $n/2$-threshold or halting $n/3$-threshold adversary, i.e. that on input a secret-sharing of a it correctly computes a sharing of a^{-1} and reveals no extra information (i.e. is simulatable). Intuitively, the value c revealed in the protocol gives no information on a since c is the product of a

with a random element b. We show that in the presence of a malicious $n/4$-threshold adversary, protocol Reciprocal does not reveal any information on either the input secret-sharing polynomial f_a in RVSS-data$[a]$ nor the output secret-sharing polynomial f_e in RVSS-data$[e]$, apart of the shares of f_a and f_e held by the corrupted players, and apart of the fact that they share values which are inverses of one another, i.e. $ae = f_a(0)f_e(0) = 1$. We formalize this secrecy property of Reciprocal in Lemma 15.

However, the secrecy property of Reciprocal, just like the secrecy of protocol Mult which is used inside Reciprocal, is stated in terms of executions which run on correctly-formed inputs, i.e. on a correct secret-sharing, and during which the adversary does not manage to cheat. Therefore, for notational convenience, in the statement of the robustness property of Reciprocal in Lemma 14 below, we define a *successful* execution of Reciprocal as an execution which not only outputs a correct secret-sharing of $e = a^{-1}$ where a is defined by the correct inputs to Reciprocal, but an execution in which the adversary did not cheat in any of the intermediate steps.

Definition 18 *We call an instance of protocol* Reciprocal *executed on correct joint secret-sharing* RVSS-data$[a]$ successful *if and only if this execution included successful instances of protocols* RVSS *and* Mult *(invoked by Steps 1 and 2 of* Reciprocal*).*

Remark. This definition is more restrictive than a natural definition of correctness of an Reciprocal protocol. Similarly as in the case of Mult, we adopt this definition for conciseness of expressing the *secrecy* property of Reciprocal (Lemma 15 below), and of threshold protocols that build on Reciprocal, i.e. the threshold DSS protocol DSS-TSig.

Lemma 14 (Robustness of $n/4$-Threshold Reciprocal)
Consider an execution of (1) protocol Ped-IG *(Figure 7-1) on public input 1^k, which outputs a Pedersen commitment instance (p, q, g, h), (2) some protocol $\mathcal{P}$ on input (p, q, g, h), whose outputs contain some joint secret-sharing* RVSS-data$[a]$*, and (3) protocol* Reciprocal *on input* RVSS-data$[a]$.

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/4$-threshold adversary (i.e. $4t + 1 \geq n$), if the output of $\mathcal{P}$ contains a joint secret-sharing RVSS-data$[a] \in \mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$*, then the above instance of* Reciprocal *is successful except for probability at most negligible in the security parameter k.*

Moreover, such instance of Reciprocal *outputs a correct joint secret-sharing* RVSS-data$[e]$ *which shares value $e = a^{-1} \bmod q$, where a is secret-shared in* RVSS-data$[a]$.

Proof: The first part of the lemma follows immediately from the fact that under the discrete-log assumption both RVSS and Mult are successful except at most for a negligible probability.

By the robustness property of Mult, if the inputs RVSS-data$[a]$ are correct and the RVSS and Mult of Step 1 and 2 in Reciprocal are successful, then the output c computed in Step 2 is equal to $c = ab$ where $b = f_b(0)$ is the secret shared in RVSS-data$[b]$, and $a = f_a(0)$ is defined by RVSS-data$[a]$. Furthermore, all the correctness properties of RVSS-data$[e]$ are inherited from the correctness properties of RVSS-data$[b]$ because data in RVSS-data$[e]$ output by each player is just a scaling of RVSS-data$[b]$ by factor c^{-1}. Therefore, the output

of a successful instance of Reciprocal on correct input RVSS-data$[a]$ forms a correct joint secret-sharing RVSS-data$[e]$ such that $e = f_e(0) = c^{-1} f_b(0) = (ab)^{-1} b = a^{-1} \bmod q$. □

Lemma 15 (Static Secrecy of $n/4$-Threshold Reciprocal)
There exists a simulator SIM, *such that for every $n/4$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah, *for any distributed protocol $\mathcal{P}$, for every discrete-logarithm instance (p, q, g), and for every $h \in G_q$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a successful run of* RVSS *on public input (p, q, g, h) and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a successful run of* Reciprocal *on* RVSS-data$[a]$, *with outputs denoted* RVSS-data$[e]$
  - *a run of $\mathcal{P}$ on input* RVSS-data$[a]$ *and* RVSS-data$[e]$
- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input $\sigma = \log_g h$:*
  - *a successful run of* RVSS *on public input (p, q, g, h) and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a successful simulation of* Reciprocal *(see a remark below) via simulator procedure* SIM_{Rpcl}, *Figure 4-16, on input* RVSS-data$[a]$, *whose output we denote as* RVSS-data$[e]$
  - *a replacement of the private data in* RVSS-data$[a, e]$ *of the simulated players via transformations* RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma$) *and* RVSS-data$[e^*] = \mathcal{T}_{RVSS}$(RVSS-data$[e], e^*, \sigma$), *where a^*, e^* are random elements in $\mathbb{Z}_q$ subject to the constraint that $a^* e^* = 1$, and then an execution of $\mathcal{P}$ on* RVSS-data$[a^*, e^*]$

Less formally: A (static) adversarial view of *successful* executions of RVSS, Reciprocal, and then a random run of protocol $\mathcal{P}$, is *the same* as an adversarial view of a successful execution of RVSS, a successful simulation of Reciprocal, a *modification* of the private data of the uncorrupted players via two $\mathcal{T}_{RVSS}$ transformations which replace the sharings of a, e with sharings of values a^* and e^* which are random subject to the constraint that $a^* e^* = 1$, and then an execution of $\mathcal{P}$.
Informally: A static secure-channels $n/4$-threshold adversary learns nothing about either secret-sharing polynomials f_a and f_e from an execution of Reciprocal which on input RVSS-data$[a]$ outputs RVSS-data$[e]$, apart of the shares of these polynomials received by the corrupted players, and apart of the fact that $ae = f_a(0) f_e(0) = 1$.

Remark. By a "*successful simulation* of Reciprocal via simulator procedure SIM_{Rpcl}", we designate an instance of an interaction between an adversary and the simulator SIM_{Rpcl} of Figure 4-16, in which the RVSS and the Mult protocols which SIM_{Rpcl} performs on behalf of the uncorrupted players in Steps 1-2, were successful. Thus a successful simulation of Reciprocal via SIM_{Rpcl} is defined similarly to a successful *execution* of Reciprocal.

Proof: The proof follows immediately from the secrecy property of protocol Mult. Intuitively, a simulation of Reciprocal and the subsequent protocol $\mathcal{P}$ is a particular case of simulating a distributed protocol running on outputs of Mult. We provide the details below.

For any distributed protocol $\mathcal{P}$ which takes RVSS-data$[a, e]$ as its inputs, let $\mathcal{P}'$ be a protocol which takes (RVSS-data$[a, b]$, ECSS-data$[c]$) as inputs, performs Steps 3-4 of Reciprocal, i.e. performs ECSS-REC(ECSS-data$[c]$) and RVSS-data$[e] \leftarrow$ Scale(RVSS-data$[b], 0, c^{-1}$) (see Figure 4-16), and then performs $\mathcal{P}$ on the resulting RVSS-data$[a, e]$. By the secrecy property of Mult (Lemma 12), given an adversary who witnesses successful executions of RVSS$\rightarrow$ RVSS-data$[a]$, RVSS$\rightarrow$ RVSS-data$[b]$, and Mult(RVSS-data$[a, b]$) $\rightarrow$ ECSS-data$[c]$, an adversarial view of an execution of $\mathcal{P}'$ on (RVSS-data$[a, b]$, ECSS-data$[c]$) is distributed identically as a view of (first the above three protocols $2 \times$ RVSS+Mult and then) an execution of $\mathcal{P}'$ on RVSS-data$[a^*, b^*]$, ECSS-data$[c^*]$ instead, where a^*, b^*, c^* are random elements of $\mathbb{Z}_q$ subject to the constraint that $a^*b^* = c^*$.

Now note that SIM_{Rpcl} takes a random c^* in $\mathbb{Z}_q$, substitutes ECSS-data$[c^*] = \mathcal{T}_{ECSS}($ ECSS-data$[c], c^*)$ and performs the first step of $\mathcal{P}'$, i.e. ECSS-REC on ECSS-data$[c^*]$. Note that since we consider only the case where RVSS-data$[a, b]$ are correct and Mult was successful, the public output of this ECSS-REC is the c^* value chosen by the simulator. Then SIM_{Rpcl} performs the second step of $\mathcal{P}'$, i.e. RVSS-data$[e]$ = Scale(RVSS-data$[b], 0, (c^*)^{-1}$). Subsequently the simulator we are considering picks a^* at random in $\mathbb{Z}_q$, computes $e^* = (a^*)^{-1} \bmod q$, and performs $\mathcal{P}$ on inputs RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma)$ and on RVSS-data$[e^*] = \mathcal{T}_{RVSS}$(RVSS-data$[e], e^*, \sigma)$.

Note that since RVSS-data$[b]$ is correct, steps

$$\text{RVSS-data}[e] = \text{Scale}(\text{RVSS-data}[b], 0, (c^*)^{-1}) \tag{4.32}$$

$$e^* = (a^*)^{-1} \bmod q \tag{4.33}$$

$$\text{RVSS-data}[e^*] = \mathcal{T}_{RVSS}(\text{RVSS-data}[e], e^*, \sigma) \tag{4.34}$$

are equivalent to

$$b^* = c^*(a^*)^{-1} \tag{4.35}$$

$$\text{RVSS-data}[b^*] = \mathcal{T}_{RVSS}(\text{RVSS-data}[b], b^*, \sigma) \tag{4.36}$$

$$\text{RVSS-data}[e^*] = \text{Scale}(\text{RVSS-data}[b^*], 0, (c^*)^{-1}) \tag{4.37}$$

In both procedures RVSS-data$[e^*]$ equals to RVSS-data$[e]$ = Scale(RVSS-data$[b], 0, (c^*)^{-1}$), except that the private data of one simulated player P_{S} chosen by the simulator (see the $\mathcal{T}_{RVSS}$ procedure in Figure 4-8) contains a random t-degree polynomial $f^*_{e_{\mathrm{S}}}$ which agrees with $f_{e_{\mathrm{S}}}$ on $P_i \in Bad$ and such that $e^* = \sum_{P_i \in Qual \setminus \{P_{\mathrm{S}}\}} f_{e_i}(0) + f^*_{e_{\mathrm{S}}}(0)$ is equal to $(a^*)^{-1}$. (Note that since RVSS-data$[b]$ is correct then RVSS-data$[e]$ is correct too, all hence the above polynomials f_{e_i}, $P_i \in Qual \setminus \{P_{\mathrm{S}}\}$, are well defined.) □

4.3.5 Threshold DSS Scheme

We first present a threshold DSS signature generation protocol and then we discuss the security of a threshold DSS signature scheme which employs this protocol and the key generation protocol DKG presented in Section 4.2.

Threshold DSS Signature Generation Protocol

We put together the threshold multiplication, the threshold inverse-computation, and the threshold exponentiation protocols into a $n/4$-threshold DSS signature generation protocol DSS-TSig. We present this protocol in Figure 4-17. Recall that in the DSS signature scheme, to sign message m, where m is a hash of an actual message M, with the private key x, the signer picks a random $k \in \mathbb{Z}_q$, computes its inverse $e = k^{-1} \bmod q$, and the signature is a pair (r, s) where $r = (g^e \bmod p) \bmod q$, and $s = k(m + xr) \bmod q$. (See Section 4.3.1 for more details about DSS.)

Note that value $r' = g^e \bmod p$ can be computed from a valid signature (r, s) and the public key $y = g^x \bmod p$ as $r' = g^{m/s} y^{r/s} \bmod p$. (Recall that the DSS verification equation checks whether $r = (g^{m/s} y^{r/s} \bmod p) \bmod q$.) Therefore, in a threshold DSS signature generation protocol, value r' can be a public output along with values r and s. Thus the threshold DSS generation protocol proceeds as follows. First the players execute a coin-flip protocol RVSS to generate RVSS-data$[k]$, a sharing of a random value k. Then they compute a sharing RVSS-data$[e]$ of its inverse $e = k^{-1}$ with a threshold inverse computation protocol Reciprocal on input RVSS-data$[k]$. Then they compute public value $r' = g^e$ (and $r = r' \bmod q$) by running the distributed exponentiation protocol Exp on input RVSS-data$[e]$. Then the players create a secret-sharing RVSS-data$[x']$ of $x' = m + xr$ by scaling the secret-sharing RVSS-data$[x]$ by a multiplicative factor r and an additive constant m using the Scale procedure. Finally, the players compute public value $s = kx'$ by running the distributed multiplication protocol Mult on inputs RVSS-data$[k]$ and RVSS-data$[x']$.

The $n/4$ value of the threshold is acquired from a $n/4$-threshold resistance of Mult (and consequently of Reciprocal). In Section 4.4 we show that if these subprotocols are replaced by their optimally-resilient versions then the resulting threshold DSS signature protocol is optimally resilient too.

Definition 19 *We call an execution of* DSS-TSig *(on inputs $m \in G_q$, a correct joint secret-sharing* RVSS-data$[x]$*, and on $y = g^x$ where x is uniquely defined by* RVSS-data$[x]$*)* successful *if during this execution protocols* RVSS, Reciprocal, Exp, *and* Mult *performed in Steps 1,2,3,5 of* DSS-TSig *were all successful.*

Lemma 16 (Robustness of $n/4$-Threshold DSS-TSig)
Consider an execution of (1) protocol Ped-IG *(Figure 7-1) on public input 1^k, which outputs a Pedersen commitment instance (p, q, g, h), (2) some protocol $\mathcal{P}$ on input (p, q, g, h), whose outputs contain some joint secret-sharing* RVSS-data$[x]$ *and a public value $y = g^x$, and (3) protocol* DSS-TSig *on inputs* RVSS-data$[x]$*, y, and some element $m \in G_q$.*

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/4$-threshold adversary (i.e. $4t+1 \geq n$), if the output of $\mathcal{P}$ contains a correct joint secret-sharing RVSS-data$[x] \in \mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$ *and the public value $y = g^x$,a then for every $m \in \mathbb{Z}_q$, the above instance of* DSS-TSig *is successful, except at most for a probability negligible in k.*

Furthermore, if DSS-TSig *is successful then it outputs a valid DSS signature on m under the public key y, i.e. a pair (r, s), such that $r = (g^{ms^{-1}} y^{rs^{-1}} \bmod p) \bmod q$, where the inverse of s is computed modulo q.*

Statically Secure $n/4$-Threshold DSS-TSig, $(4t+1 \leq n)$

Input:	secret-sharing RVSS-data$[x]$ of the secret key (see Fig. 4-7), public key $y = g^x$, (hashed) message $m \in \mathbb{Z}_q$ to be signed
Adversarial Input:	adversarial history ah (including adversary's output in RVSS-data$[x]$)
Public Output:	(r, s), the DSS signature on message m under key y

1. Players compute RVSS-data$[k] \leftarrow$ RVSS
2. Players compute RVSS-data$[e] \leftarrow$ Reciprocal(RVSS-data$[k]$) to share $e = k^{-1}$
3. Players compute $r' \leftarrow$ Exp(RVSS-data$[e], g$) and then $r \rightarrow r' \bmod q$
4. Players compute RVSS-data$[x'] \leftarrow$ Scale(RVSS-data$[x], m, r$)
5. Players compute ECSS-data$[s] \leftarrow$ Mult(RVSS-data$[x']$, RVSS-data$[k]$)
6. Players reconstruct $s \leftarrow$ ECSS-REC(ECSS-data$[s]$)

Simulator $\mathsf{SIM}_{DSS-TSig}$ of DSS-TSig (interacting with $\mathcal{A}$):

Public Input:	public data in RVSS-data$[x]$, public key $y^* \in G_q$, (hashed) message $m \in \mathbb{Z}_q$
$\mathsf{SIM}_{DSS-TSig}$**'s Private Input:**	private outputs of all $P_i \in$ *Good* in RVSS-data$[x]$ "target output" signature (r^*, s^*), two elements in $\mathbb{Z}_q$
$\mathcal{A}$**'s Private Input:**	adversarial history ah (including the adversary's output in RVSS-data$[x]$)

1. $\mathsf{SIM}_{DSS-TSig}$ follows Step 1 of DSS-TSig on behalf of the uncorrupted players.
2. $\mathsf{SIM}_{DSS-TSig}$ simulates Reciprocal on RVSS-data$[k]$ with SIM_{Rpcl} of Figure 4-16.
3. $\mathsf{SIM}_{DSS-TSig}$ computes $r'^* = g^{m/s^*}(y^*)^{r^*/s^*} \bmod p$ and simulates Exp via SIM_{Exp} of Figure 4-9 on input RVSS-data$[e]$ and g and on SIM_{Exp}'s target output r'^*.

4.-5. $\mathsf{SIM}_{DSS-TSig}$ follows Steps 4-5 of DSS-TSig on behalf of the uncorrupted players.

6. $\mathsf{SIM}_{DSS-TSig}$ replaces the private data of the uncorrupted players via transformation ECSS-data$[s^*] \leftarrow \mathcal{T}_{ECSS}$(ECSS-data$[s], s^*$) and participates in ECSS-REC on the resulting ECSS-data$[s^*]$ on behalf of the uncorrupted players.

Figure 4-17: DSS-TSig: $n/4$-Threshold DSS Signature Generation

Proof: The lemma follows from the robustness of the building-block protocols RVSS, Reciprocal, Exp, and Mult.

Assume that the discrete-log is intractable. Denote the protocol $\mathcal{P}$ in the statement of this lemma as $\mathcal{P}_0$. By the robustness of RVSS (Lemma 4) it follows that in Step 1 the players create a joint secret-sharing RVSS-data$[k]$, except for negligible probability. If Step 1 does succeed, then by the robustness property of Reciprocal (Lemma 14), where protocol $\mathcal{P}$ in that lemma is protocol $\mathcal{P}_0$ followed by Step 1 of DSS-TSig), Step 2 produces a correct

joint secret-sharing RVSS-data[e] such that $e = k^{-1}$, again except for negligible probability. If Step 2 succeeds too, then by the robustness property of Exp (Lemma 7, where protocol $\mathcal{P}$ in that lemma is protocol $\mathcal{P}_0$ followed by Steps 1-2 of DSS-TSig), Step 3 produces a correct joint secret-sharing RVSS-data[e] such that $e = k^{-1}$, except for negligible probability. If Step 3 succeeds then so does Step 4, because the correctness properties of RVSS-data[x'] are inherited from the correctness properties of RVSS-data[x]. Finally, if all these steps are successful, then by the robustness property of Mult (Lemma 11), where protocol $\mathcal{P}$ in that lemma is protocol $\mathcal{P}_0$ followed by Steps 1-4 of DSS-TSig), Step 5 produces as public input value $s = kx' = k(m + xr)$, except for negligible probability. Therefore, except for negligible probability, $r = (g^{1/k} \bmod p) \bmod q$, and $1/k = (m + xr)s^{-1} = ms^{-1} + xrs^{-1} \bmod q$, and thus $g^{1/k} = g^{ms^{-1}} y^{rs^{-1}} \bmod p$. Thus the lemma follows. □

We state the secrecy property of the DSS-TSig protocol in terms of an adversary who participates first in the distributed key generation protocol DKG, which creates (via protocol RVSS) a secret-sharing RVSS-data[x] of a secret key x, and a public key $y = g^x$ (via protocol Exp on input RVSS-data[x] and g), and second in an execution of DSS-TSig which produces a DSS signature (r, s) on inputs RVSS-data[x], y, and some message m. We consider the view of an adversary who participates in either the execution of these protocols or in their simulation, to a random target public key y^* and a random target DSS signature (r^*, s^*) of m under that key. Furthermore, we consider only instances of DSS-TSig on *correct* RVSS-data[x] and $y = g^x$, and we consider only *successful* instances of such executions. Similarly we consider only successful instances of a simulation of DSS-TSig which on simulator's input r^*, s^* produce this signature as a public output. We show in Lemma 17 below that the above two distributions of adversarial views are identical. In Theorem 3 we extend the argument to prove this lemma to the case of a *repeated* execution of DSS-TSig on a series of messages $m_1, ..., m_{p(k)}$ adaptively chosen by the adversary.

Lemma 17 (Static Secrecy of $n/4$-Threshold DSS-TSig)
There exists a simulator SIM, *such that for every $n/4$-threshold static secure-channels adversary $\mathcal{A}$ with some adversarial history* ah, *for every discrete-logarithm instance (p, q, g), for every $h \in G_q$, and for every m in $\mathbb{Z}_q$, the following adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a successful run of* DKG *(Figure 4-10) on public input (p, q, g, h) and adversarial input* ah, *which outputs sharing* RVSS-data[x] *and a public key y (see remark 1 below).*
  - *a successful run of* DSS-TSig *(Figure 4-17) on inputs* RVSS-data[x], *y, and m (see remark 1 below).*
- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM*:*
  - *a successful simulation of* DKG *via simulator* SIM_{DKG}, *Figure 4-10 (see remark 2 below), on public input (p, q, g, h), adversarial input* ah, *and simulator's* SIM_{DKG}

target input value y^* chosen uniformly in G_q. We denote the secret-sharing created in this simulation as RVSS-data$[x]$

- *a successful simulation of* DSS-TSig *via simulator* $\mathsf{SIM}_{DSS-TSig}$*, Figure 4-17 (see remark 3 below), on inputs* RVSS-data$[x], y^*, m$ *and simulator's* $\mathsf{SIM}_{DSS-TSig}$ *target input value (r^*, s^*), where (r^*, s^*) is a random DSS signature on m under the public key y^* (i.e. $r^* = (g^{1/k^*} \bmod p) \bmod q$ and $s^* = k^*(m + x^* r^*) \bmod q$, where $x^* = \log_g y^*$, and k^* is chosen uniformly in $\mathbb{Z}_q$)*

Less Formally: An adversarial view of an execution of the key generation protocol DKG which outputs a sharing of a secret key RVSS-data$[x]$ and a public key $y = g^x$, followed by an execution of DSS-TSig on some message m, is *the same* as an adversarial view of a *simulation* of DKG to a random target public key y^* and a *simulation* of DSS-TSig to the target a *random* DSS signature under y^* on m.
Informally: A static secure-channels $n/4$-threshold adversary learns nothing new about the secret key x from the DSS signature generation protocol beyond the resulting signature (r, s) on message m. (Note that the adversary does know the public key y, which reveals some information about x.)

Remark 1. Note that in the above statement we consider the adversarial view of only *successful* executions of DKG followed by DSS-TSig. Recall that in such execution the joint secret-sharing RVSS-data$[x]$ output by DKG is correct, and the public key y is equal to g^x where x is defined in RVSS-data$[x]$. Similarly, a successful instance of DSS-TSig, executed on such inputs, outputs a valid signature (r, s) on m under the public key y (i.e. a pair $r, s \in \mathbb{Z}_q$ s.t. $r = (g^{m/s} y^{r/s} \bmod p) \bmod q$).

Remark 2. Similarly as in Lemma 15 where we expressed the static secrecy of the Reciprocal protocol, here by a "*successful simulation of* DKG *via simulator* SIM_{DKG}" we designate an instance of interaction between an adversary and the simulator SIM_{DKG} of Figure 4-10, in which the RVSS protocol performed on behalf of the uncorrupted players in Step 1 is successful, i.e. outputs a correct joint secret-sharing RVSS-data$[x]$, and in which the simulation via simulator SIM_{Exp} of the Exp protocol in Step 2 is also successful, in the sense that its public output is equal to y^*, the input to SIM_{Exp}, rather than to null. (Compare Fact 1, page 74.)

Remark 3. In the same manner, by a "*successful simulation of* DSS-TSig *via simulator* $\mathsf{SIM}_{DSS-TSig}$" we designate an instance of interaction between an adversary and the simulator $\mathsf{SIM}_{DSS-TSig}$, during which all the instances of protocols and simulations were successful themselves. Namely: (1) The RVSS protocol performed by $\mathsf{SIM}_{DSS-TSig}$ on behalf of the uncorrupted players in Step 1 is successful; (2) The simulation of Reciprocal via SIM_{Rpcl} in Step 2 is successful (see Lemma 15 and a subsequent remark); (3) The simulation of Exp via SIM_{Exp} in Step 3 is successful (see a remark after Lemma 8); and (4) The execution of Mult in Step 5 is successful.

It is easy to see, by the same argument as used in the proof of robustness of DSS-TSig (Lemma 16 above), that if the above conditions are satisfied then the public output of this simulation is a pair (r^*, s^*) which was an input to $\mathsf{SIM}_{DSS-TSig}$.

Remark 4. We do not use the above lemma directly to prove that a *repeated* execution of the DSS-TSig protocol on different messages does not leak any new information about the

secret key. Such proof is an essence of proving the unforgeability property of the threshold DSS scheme which utilizes the DSS-TSig protocol (i.e. Theorem 3, page 106). However, the proof of Theorem 3 is a straightforward extension of the proof of this lemma. It should be possible to express a property of the DSS-TSig protocol, which together with the "repetitive secrecy" property of Mult stated in Lemma 13, implies Theorem 3 directly. Even though the above statement does not seem to imply it, we include it to as an exercise which builds confidence in the security of the threshold DSS resulting from a combination of the distributed key generation protocol DKG and the threshold signature generation protocol DSS-TSig.

Proof: The proof follows straightforwardly from the secrecy properties of the component subprotocols used in DSS-TSig, i.e. from the secrecy properties of RVSS, Exp, Mult, and Reciprocal.

The main thing to notice is that the distribution of the (y^*, r'^*, s^*) triple used by the simulator SIM are distributed as if they are picked according to the following process: First elements x^* and k^* are picked uniformly and independently in $\mathbb{Z}_q$, and then the following computations are performed: $y^* = g^{x^*}$, $e^* = (k^*)^{-1}$, $r'^* = g^{e^*}$, $r^* = r'^* \bmod q$, $x'^* = m + x^* r^*$, and $s^* = k^* x'^*$. By the secrecy of Mult, the adversarial view of the successful executions of DKG and DSS-TSig is identical to its view of the following sequence:

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[x] \tag{4.38}$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[x], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[x^*] \tag{4.39}$$
$$\mathsf{Succ\text{-}Exp}(\mathsf{RVSS\text{-}data}[x^*]) \rightarrow y^* \tag{4.40}$$
$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[k] \tag{4.41}$$
$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[k], \sigma) \rightarrow \mathsf{RVSS\text{-}data}[k^*] \tag{4.42}$$
$$\mathsf{Succ\text{-}Reciprocal}(\mathsf{RVSS\text{-}data}[k^*]) \rightarrow \mathsf{RVSS\text{-}data}[e^*] \tag{4.43}$$
$$\mathsf{Succ\text{-}Exp}(\mathsf{RVSS\text{-}data}[e^*]) \rightarrow r'^* \tag{4.44}$$
$$r'^* \bmod q \rightarrow r^* \tag{4.45}$$
$$\mathsf{Scale}(\mathsf{RVSS\text{-}data}[x], m, r^*) \rightarrow \mathsf{RVSS\text{-}data}[x'] \tag{4.46}$$
$$\mathsf{Mult}(\mathsf{RVSS\text{-}data}[x', k]) \rightarrow \mathsf{ECSS\text{-}data}[s] \tag{4.47}$$
$$\mathcal{T}_{ECSS}(\mathsf{ECSS\text{-}data}[s], s^* = k^*(m + x^* r^*)) \rightarrow \mathsf{ECSS\text{-}data}[s^*] \tag{4.48}$$
$$\mathsf{ECSS\text{-}REC}(\mathsf{ECSS\text{-}data}[s^*]) \rightarrow s^* \tag{4.49}$$

(we use Lemma 13 for the case $p(k) = 1$), where k^* and x^* are chosen uniformly and independently in $\mathbb{Z}_q$. We will argue that the above view is identical to what the adversary sees in the simulation of DKG and DSS-TSig via the SIM procedure described in this lemma, i.e. to the following sequence:

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[x] \tag{4.50}$$
$$\mathsf{Succ\text{-}SIM}_{Exp}(\mathsf{RVSS\text{-}data}[x], y^*) \rightarrow y^* \tag{4.51}$$
$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[k] \tag{4.52}$$
$$\mathsf{Succ\text{-}SIM}_{Rpcl}(\mathsf{RVSS\text{-}data}[k]) \rightarrow \mathsf{RVSS\text{-}data}[e] \tag{4.53}$$

$$\text{Succ-SIM}_{Exp}(\text{RVSS-data}[e], r'^*) \rightarrow r'^* \tag{4.54}$$
$$r'^* \bmod q \rightarrow r^* \tag{4.55}$$
$$\text{Scale}(\text{RVSS-data}[x], m, r^*) \rightarrow \text{RVSS-data}[x'] \tag{4.56}$$
$$\text{Mult}(\text{RVSS-data}[x', k]) \rightarrow \text{ECSS-data}[s] \tag{4.57}$$
$$\mathcal{T}_{ECSS}(\text{ECSS-data}[s], s^*) \rightarrow \text{ECSS-data}[s^*] \tag{4.58}$$
$$\text{ECSS-REC}(\text{ECSS-data}[s^*]) \rightarrow s^* \tag{4.59}$$

where, as we have stated above, values y^*, r^*, s^* are distributed as g^{x^*}, $(g^{1/k^*} \bmod p) \bmod q$ and $k^*(m + x^* r^*)$ for uniformly chosen x^*, k^*.

Since y^* is chosen uniformly in G_q, the equality of adversarial views of Steps [4.38; 4.39; 4.40] and [4.50; 4.51] is proven in Lemma 8, which expresses the secrecy property of protocol Exp.

It thus remains only to show that given the adversarial view of Steps [4.50; 4.51] (or [4.38; 4.39; 4.40], since they are the same), the adversarial view of Steps [4.52; 4.53; 4.54] is the same as its view of [4.41; 4.42; 4.43; 4.44]. Note that the view of Step 4.54 is the same as a view of a sequence

$$\mathcal{T}_{RVSS}(\text{RVSS-data}[e], e^*) \rightarrow \text{RVSS-data}[e^*] \tag{4.60}$$
$$\text{Succ-Exp}(\text{RVSS-data}[e^*]) \rightarrow r'^* \tag{4.61}$$

where $e^* = 1/k^* = \log_g r'^*$ and r'^* is the value used by SIM. (This is argued in the proof of the secrecy of Exp in Lemma 8.) Therefore a view of sequence [4.52; 4.53; 4.54] is the same as a view of [4.52; 4.53; 4.60; 4.61].

Now note that by the secrecy property of Reciprocal, Lemma 15, since $e^* = 1/k^*$ is random in $\mathbb{Z}_q$, the adversarial view of the latter sequence is equal to the view of a following sequence:

$$\text{Succ-RVSS} \rightarrow \text{RVSS-data}[k^*] \tag{4.62}$$
$$\text{Succ-Reciprocal}(\text{RVSS-data}[k^*]) \rightarrow \text{RVSS-data}[e^*] \tag{4.63}$$
$$\text{Succ-Exp}(\text{RVSS-data}[e^*]) \rightarrow r'^* \tag{4.64}$$

Therefore, by the secrecy property of RVSS, Lemma 6, this view is the same as the view of sequence [4.41; 4.42; 4.43; 4.44]. Hence the lemma follows. □

Threshold DSS Signature Scheme

We can now prove the security of an $n/4$-threshold DSS signature scheme TSS, which consists of the triple of protocols Ped-IG+DKG, DSS-TSig, and Ver,[10] where:

- Ped-IG+DKG denotes a sequence of two protocols executed one after the other, first the protocol Ped-IG of Figure 7-1 in which the participating players on input a security parameter pick a Pedersen commitment instance (p, q, g, h), and then the "proper" distributed key generation protocol DKG of Figure 4-10

[10] See the definition of a secure signature scheme in Section 2.4.

- DSS-TSig is the threshold DSS signature generation protocol of Figure 4-17
- Ver is the DSS verification algorithm described in Section 4.3.1

Technically, Lemma 17 shows that a successful execution of the DKG protocol followed by a *single* successful execution of the threshold signature protocol DSS-TSig can be simulated to the target of a random public key and a random DSS signature under that key. It turns out that a similar argument, but based on the secrecy property of the *repetitive* use of protocol Mult, can show that a *repeated* successful execution of the DSS-TSig signature generation protocol on different messages can also be simulated to a target sequence of random DSS signatures under the given public key on the messages submitted by the adversary. This implies that a threshold adversary who participates in an execution of the DKG protocol followed by executions of DSS-TSig on the messages of his choice learns nothing useful beyond the signatures output by these instances of DSS-TSig, at least in the case that all the DSS-TSig instances were successful. We first notice, see Theorem 2 below, that the TSS scheme is *robust*, and that in particular, under the discrete-log intractability assumption, each of the DKG and DSS-TSig executions is indeed successful except for at most negligible probability. It thus follows that under the discrete-log assumption there is at most negligible statistical difference between the adversarial view of a random execution of the TSS scheme and an execution in which all the instances of DKG and DSS-TSig are successful. This in turn implies that, under the discrete-log assumption, we can reduce an adversary that breaks the unforgeability property of the threshold DSS signature scheme TSS to an adversary that succeeds in an adaptive chosen message attack against a traditional, i.e. centralized, DSS signature scheme. The reason is that given the adversary of the first type we can build the adversary of the second type by simulating the view of the first adversary based solely on the answers of a DSS oracle. We prove this claim in Theorem 3 below.

For the theorem below see the definition of robustness of a threshold signature scheme in Definition 3 in Section 2.4.

Theorem 2 (Robustness of $n/4$-Threshold DSS Scheme)
Under the discrete-log intractability assumption, TSS = (Ped-IG+DKG, DSS-TSig, Ver) *is a robust threshold signature scheme in the presence of a static secure-channels $n/4$-threshold adversary.*

Proof: The theorem follows straightforwardly from the robustness of DL-IG (Lemma 34), the robustness of the h-IG+DKG=[h-IG;DKG] protocol (Theorem 1), and the robustness of the DSS-TSig protocol (Lemma 16). Assume that discrete-log is hard. First, by the correctness properties of Ped-IG=[DL-IG;h-IG] (Lemma 34), a Pedersen commitment (p, q, g, h) of the required security parameter is always produced. By Theorem 1, page 76 the output of DKG on (p, q, g, h) is a correct secret-sharing RVSS-data$[x]$ and a public key $y = g^x$, except for a probability negligible in the security parameter k. Secondly, every time the adversary triggers an execution of the DSS-TSig protocol on RVSS-data$[x]$, y, and m = SHA-1(M) by broadcasting '`[sign,`M`]`', then by the robustness of DSS-TSig, this protocol produces as public output a pair (r, s) which is a valid DSS signature on M under the public key y, except again for negligible probability. □

For the theorem below see the definition of CMA security of a signature scheme and of unforgeability of a threshold signature scheme, i.e. Definition 1 and 2 in Section 2.4.

Theorem 3 (Static Unforgeability of $n/4$-Threshold DSS Scheme)
If the DSS signature scheme is CMA secure, then TSS = (Ped-IG+DKG, DSS-TSig, Ver) *is an unforgeable threshold signature scheme in the presence of a static secure-channels $n/4$-threshold adversary.*

Proof: Assume that there exists a static $n/4$-threshold adversary $\mathcal{A}$ that, with probability higher than negligible, forges DSS after participating in the distributed key generation protocol Ped-IG+DKG and in the multiple instances of the threshold DSS signature generation protocol DSS-TSig invoked on messages of his choice. In other words, assume that $\mathcal{A}$ breaks the unforgeability property of TSS. We show that such attack can be translated into an adaptive chosen message forgery against regular DSS. We accomplish this reduction by constructing an efficient algorithm SIM which, given access to an oracle $\mathcal{O}_{DSS}$ that implements a standard DSS signature scheme (see Definition 1 in Section 2.4 which defines such an oracle), creates a DSS forgery with probability higher than negligible. (In terms of Definition 1, the adversarial algorithm that forges DSS is the efficient algorithm SIM using the above efficient algorithm $\mathcal{A}$ as a subprocedure.) The strategy of SIM is to first perform the Ped-IG protocol on behalf of the honest players and then to simulate the DKG and the DSS-TSig protocols to the adversary $\mathcal{A}$ via the simulation procedure SIM_{TSS} of Figure 4-18 below.

We will first argue that it follows from the secrecy properties of the building block subprotocols used in the TSS threshold scheme that the adversarial view of a random successful execution of the TSS scheme is distributed identically to the adversarial view of a random successful simulation of TSS via SIM. We then argue that under the discrete-log assumption the adversarial view of a successful run of TSS is at most negligibly different from its view of a random run of TSS. It will thus follow that if $\mathcal{A}$ has a higher than negligible probability of creating a forgery then so does SIM. This will prove our theorem because the assumption that DSS is CMA secure implies the discrete-log intractability assumption.

First notice that both an execution and a simulation of the TSS scheme starts with an instance of protocol Ped-IG. Therefore, for every k and every initial adversarial history ah, and for every (p, q, g, h) and ah′, the public output and the adversarial output of some instance of an execution of Ped-IG on inputs 1^k and ah, we consider the remaining steps of an execution of TSS executing on inputs (p, q, g, h) and ah′, and the remaining steps of simulation of TSS via SIM (i.e. the simulation procedure SIM_{TSS} of Figure 4-18), also executing on (p, q, g, h) and ah′. We call an execution of these remaining steps of TSS the "TSS proper", and similarly we refer to its simulation via simulator SIM_{TSS} as to the "simulation of TSS proper".

We define a *successful execution* of TSS proper on public input a Pedersen commitment (p, q, g, h) as an execution in which all the building block protocols are successful, i.e.:

(e1) The initial DKG is successful (see Definition 15, page 76)

(e2) Each instance of the DSS-TSig protocol invoked by $\mathcal{A}$ is successful (see Definition 19, page 99)

Simulation procedure for a threshold DSS signature scheme

1. On public input (p, q, g, h), SIM_{TSS} triggers the DSS oracle $\mathcal{O}_{DSS}$ on input (p, q, g). The oracle then picks a random DSS secret key x^*, chosen uniformly in $\mathbb{Z}_q$, and outputs a public key $y^* = g^{x^*} \bmod p$.
2. SIM_{TSS} simulates to $\mathcal{A}$ the DKG protocol using the simulator SIM_{DKG} of Figure 4-10 on public input (p, q, g, h) and on the simulator's SIM_{DKG} private input the target value y^*. Note that a successful instance of such simulation creates a correct secret sharing RVSS-data$[x]$.
3. SIM_{TSS} performs the following loop until the adversary $\mathcal{A}$ broadcasts message '`[signed,` M`,`(s,r)`]`' such that $\mathsf{Ver}(y^*, M, (s, r)) =$ `pass` and such that a message '`[sign,`M`]`' has not appeared on the broadcast channel before. In such case, SIM_{TSS} outputs M and the signature (s, r) as a valid forgery against the instance of the DSS scheme chosen by the oracle $\mathcal{O}_{DSS}$.
   (a) SIM_{TSS} waits till the adversary $\mathcal{A}$ broadcasts message '`[sign,`M`]`', i.e. till $\mathcal{A}$ submits some message M to be signed by the threshold signature scheme TSS.
   (b) SIM_{TSS} gives M to the DSS oracle $\mathcal{O}_{DSS}$, which outputs a random DSS signature (r^*, s^*) on M under the public key y^*. Namely, it computes a SHA-1 hash $m \in \mathbb{Z}_q$ of M, picks k^* uniformly in $\mathbb{Z}_q$, and outputs $r^* = (g^{1/k^*} \bmod p) \bmod q$, and $s^* = k^*(m + x^* r^*) \bmod q$.
   (c) SIM_{TSS} simulates to $\mathcal{A}$ the DSS-TSig protocol using the simulator $\mathsf{SIM}_{DSS-TSig}$ of Figure 4-17 on input RVSS-data$[x]$ established in the above simulation of DKG, on public key y^*, on message m, and on the simulator's $\mathsf{SIM}_{DSS-TSig}$ private input the target signature pair (r^*, s^*).

   (To designate the variables involved in j-th instance of the above loop, we write $M^{(j)}$, $m^{(j)}$, $k^{(j)*}$, etc.)

Figure 4-18: SIM_{TSS}: Simulator for threshold DSS signature scheme TSS

Similarly we define a *successful simulation* of the TSS proper via SIM_{TSS} on public input a Pedersen commitment (p, q, g, h) as an instance of an interaction of the adversary and SIM_{TSS} which satisfies the following constraints:

(s1) An instance of simulation of the initial DKG is successful in the sense of Remark 2 on page 102. In other words, an instance of simulation of DKG via simulator SIM_{DKG} in Step 2 outputs a correct joint secret-sharing RVSS-data$[x]$ and a public key y^* rather than null, where y^* is an input to SIM (and to SIM_{DKG})

(s2) Each instance of simulation of the DSS-TSig protocol, i.e. each loop of Step 3 in Figure 4-18, is successful in the sense of Remark 3 on page 102. In other words, each instance of simulation of DSS-TSig via simulator $\mathsf{SIM}_{DSS-TSig}$ of Figure 4-17 satisfies the following conditions: (1) The RVSS protocol performed by $\mathsf{SIM}_{DSS-TSig}$ on behalf of the uncorrupted players in Step 1 is successful (see Definition 13, page 66); (2)

The simulation of Reciprocal via SIM_{Rpcl} in Step 2 is successful (see Lemma 15 and a subsequent remark); (3) The simulation of Exp via SIM_{Exp} in Step 3 is successful (see a remark after Lemma 8); and (4) The execution of Mult in Step 5 is successful (see Definition 17, page 83).

Since $\mathcal{A}$ is an efficient algorithm let $p(z)$ be a polynomial s.t. for all k the number of messages that $\mathcal{A}$ submits for signing is upper bounded by $p(k)$. We claim that for every Pedersen instance (p, q, g, h) and adversarial history ah', the adversarial view of a random successful execution of TSS proper is identically distributed to the adversarial view of a random successful simulation of TSS proper. Note that in Lemma 17 we in fact already proved a special case of the above statement for $p(k) = 1$, i.e. for a special case of an adversary that asks for a signature on just one message.

From the "repetitive secrecy" property of Mult, Lemma 13, it follows that the adversarial view of $p(k)$ loops of a successful execution of TSS proper is distributed identically as the adversarial view of the following process:

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[x] \tag{4.65}$$

$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[x], \sigma, x^*) \rightarrow \mathsf{RVSS\text{-}data}[x^*] \tag{4.66}$$

$$\mathsf{Succ\text{-}Exp}(\mathsf{RVSS\text{-}data}[x^*]) \rightarrow y^* \tag{4.67}$$

followed by $p(k)$ loops of the following process, for $j = 1, ..., p(k)$:

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[k_j] \tag{4.68}$$

$$\mathcal{T}_{RVSS}(\mathsf{RVSS\text{-}data}[k_j], \sigma, k_j^*) \rightarrow \mathsf{RVSS\text{-}data}[k_j^*] \tag{4.69}$$

$$\mathsf{Succ\text{-}Reciprocal}(\mathsf{RVSS\text{-}data}[k_j^*]) \rightarrow \mathsf{RVSS\text{-}data}[e_j^*] \tag{4.70}$$

$$\mathsf{Succ\text{-}Exp}(\mathsf{RVSS\text{-}data}[e_j^*] \rightarrow r_j'^* \tag{4.71}$$

$$r_j'^* \bmod q \rightarrow r_j^* \tag{4.72}$$

$$\mathsf{Scale}(\mathsf{RVSS\text{-}data}[x], m_j, r_j^*) \rightarrow \mathsf{RVSS\text{-}data}[x_j'] \tag{4.73}$$

$$\mathsf{Mult}(\mathsf{RVSS\text{-}data}[x_j', k_j]) \rightarrow \mathsf{ECSS\text{-}data}[s_j] \tag{4.74}$$

$$\mathcal{T}_{ECSS}(\mathsf{ECSS\text{-}data}[s_j], s_j^*) \rightarrow \mathsf{ECSS\text{-}data}[s_j^*] \tag{4.75}$$

$$\mathsf{ECSS\text{-}REC}(\mathsf{ECSS\text{-}data}[s_j^*]) \rightarrow s_j^* \tag{4.76}$$

where m_j's are SHA-1 hashes of messages M_j submitted by $\mathcal{A}$, element x^* is uniformly chosen in $\mathbb{Z}_q$, and for each j element k_j^* is uniform in $\mathbb{Z}_q$ and $s_j^* = k_j^*(m_j + x^* r_j^*)$ where $r_j^* = (g^{1/k_j^*} \bmod p) \bmod q$. Note that this is the same distribution as that of a random secret key x^* and random DSS signatures (r_j^*, s_j^*) under the public key $y^* = g^{x^*}$ for each m_j.

First, note that, by the secrecy property of Exp, the adversarial Steps (4.65-4.67) is identical to its view of a successful simulation of DKG via SIM_{TSS}. Secondly, note that for every j, the (4.68-4.76) sequence of Steps is the same as Steps (4.41-4.49) in the proof of Lemma 17. Therefore we can apply the same argument as used in the lemma and it follows that the adversarial view of such sequence, for each j, is distributed identically its view of the following loop:

$$\mathsf{Succ\text{-}RVSS} \rightarrow \mathsf{RVSS\text{-}data}[k_j] \tag{4.77}$$

$$\text{Succ-SIM}_{Rpcl}(\text{RVSS-data}[k_j]) \rightarrow \text{RVSS-data}[e_j] \tag{4.78}$$
$$\text{Succ-SIM}_{Exp}(\text{RVSS-data}[e_j], r_j'^*) \rightarrow r_j'^* \tag{4.79}$$
$$r_j'^* \bmod q \rightarrow r_j^* \tag{4.80}$$
$$\text{Scale}(\text{RVSS-data}[x], m_j, r_j^*) \rightarrow \text{RVSS-data}[x_j'] \tag{4.81}$$
$$\text{Mult}(\text{RVSS-data}[x_j', k_j]) \rightarrow \text{ECSS-data}[s_j] \tag{4.82}$$
$$\mathcal{T}_{ECSS}(\text{ECSS-data}[s_j], s_j^*) \rightarrow \text{ECSS-data}[s_j^*] \tag{4.83}$$
$$\text{ECSS-REC}(\text{ECSS-data}[s_j^*]) \rightarrow s_j^* \tag{4.84}$$

where values r^*, s^* are distributed as $(g^{1/k_j^*} \bmod p) \bmod q$ and $k_j^*(m_j + x^* r_j^*)$ for a uniformly chosen k_j^* and for $x^* = \log_g y^*$. Since the above sequence is exactly what happens in a successful simulation of TSS proper, this shows our claim, that adversarial views of a successful execution of TSS proper and its successful simulation are identical.

Now, by the robustness property of TSS, under the discrete-log intractability assumption a random execution of TSS proper is successful except of at most negligible probability. It follows that for every $(p, q, g, h), \mathsf{ah}'$ output by Ped-IG on $1^k, \mathsf{ah}$, there is at most negligible statistical difference between the adversarial views of a random execution of TSS proper and a random simulation of TSS proper via SIM_{TSS}. Therefore, for every 1^k and ah, there is also at most negligible statistical difference between the adversarial view of a run of TSS and its simulation via SIM. It follows (refer to Definition 2, page 34) that if there is a higher than negligible probability that $\mathcal{A}$ forges, i.e. that an adversarial run $\text{out}_{\tau,\mathcal{A}}(k)$ of TSS with $\mathcal{A}$ contains a public key y and a message `'[signed,`M`,`(s,r)`]'` such that $\text{Ver}(y, M, (s, r)) = \texttt{pass}$, but no request `'[sign,`$M$`]`, then there is also a higher than negligible probability that the public output of a simulation of TSS via SIM has the same property, i.e. that it contains a public key and $\mathcal{A}$'s forgery under that key. Since a public key such transcript can contain is necessarily equal to y^* given to SIM by $\mathcal{O}_{DSS}$, it follows that SIM breaks the CMA security of centralized DSS, which completes the proof. □

Theorem 4 *If the DSS signature scheme is unforgeable under the adaptive chosen message attack, then* TSS = (Ped-IG+DKG, DSS-TSig, Ver) *is a secure $n/4$-threshold signature scheme.*

Proof: Follows immediately from Theorems 3 and 2. □

4.4 Optimally Resilient Threshold DSS Signatures

We present protocols that lead to an optimally resilient $n/2$-threshold DSS signature scheme. The crucial piece is the $n/2$-threshold multiplication protocol Mult-opt of [GRR98] and the simultaneous proof protocol. This section is based on material which appeared in [CGJ+99].

4.4.1 Simultaneous Zero-Knowledge Proof

Some protocols we propose (for example the Mult-opt protocol presented in the following section) achieve robustness against a minority of faults by having each player perform a zero-knowledge proof of knowledge. However, one should employ zero-knowledge proofs in distributed protocol with care. Note that if every player engages in n zero-knowledge proofs with n other players, each acting as a verifier, then we need to employ proofs that remain zero-knowledge when performed in parallel with polynomially-many dishonest verifiers. Furthermore, to prove robustness of the resulting distributed protocols, we need to use proof systems which remain proofs of knowledge when performed in parallel with polynomially-many dishonest provers.

Rather than designing efficient parallelizable zero-knowledge proof of knowledge systems, we give a general technique of a *simultaneous proof* protocol, which achieves the effect of $O(n^2)$ zero-knowledge proofs of knowledge (where each of the n players proves something to each of the other players) in a single 3-move public-coin honest-verifier zero-knowledge proof, where the public coin is implemented with a distributed generation of a random challenge by all the participating players, i.e. with a *coin-flip* protocol.[11] The idea to implement in this way a parallel execution of honest-verifier public-coin zero-knowledge proofs in the threshold setting appeared in [BMR90].

Consider a Three-round Honest-verifier Public-coin Zero-knowledge Proof of knowledge system. We will designate such proof systems by an acronym *THPZP*. Let $\mathcal{R}$ be a polynomial-time computable relation, i.e. a set of pairs (y, w) for which there exists a fixed polynomial $p(z)$ such that $|w| \leq p(|y|)$ for each $(y, w) \in \mathcal{R}$, and for which there exists a polynomial-time algorithm which given a pair (y, w) decides if it is in $\mathcal{R}$. (From now on, we will simply say "relation" when we mean a polynomial-time computable relation.) It follows that $L_{\mathcal{R}} = \{y \mid \exists_w\ (y, w) \in \mathcal{R}\}$ is in NP, because for each $y \in L_{\mathcal{R}}$ there exists a short (polynomial in y) proof of membership of y in $L_{\mathcal{R}}$. A THPZP proof for relation $\mathcal{R}$ is a protocol between two machines, a prover P and a verifier V acting on common input an element y, called a "public value", in which P can show to V knowledge of element w s.t. $(y, w) \in \mathcal{R}$, called a "witness" of y in $L_{\mathcal{R}}$. In Appendix F we give formal definitions of the proof-of-knowledge property (Definition 28) and the honest-verifier public-coin zero-knowledge property (Definition 30) of the THPZP proof system. Because the THPZP proof takes only three rounds and assumes an honest verifier which only picks a public coin, such proof system is fully specified by a triple of algorithms $P^{(1)}, V, P^{(2)}$, and a function $\mathcal{D} : \{0,1\}^* \to \{0,1\}^*$, s.t. $\mathcal{D}(y)$ is a distribution according to which an honest verifier should pick a public random coin, when proving knowledge of a witness for a given public input y.

[11] Recall that 3-move zero-knowledge proofs cannot exist for cheating verifiers if the underlying problem is not in BPP [GK96, IS93]. Thus, the distributed nature of the verifier in our implementation is essential for "forcing honesty".

Assume that a triple of algorithms $P^{(1)}, V, P^{(2)}$ specifies a discrete-log based Three-round Honest-verifier Public-coin Zero-Knowledge Proof-of-knowledge proof system THPZP for relation $\mathcal{R}$.

The THPZP proof proceeds as follows:

Public Input: public value (p, q, y) in $L_{\mathcal{R}}$, s.t. p, q are primes and q divides $p-1$
Prover's Input: witness w s.t. $((p, q, y), w) \in \mathcal{R}$

P $\longrightarrow$ V: $M \leftarrow P^{(1)}((p, q, y), w, r)$ where r is P's random input
P $\longleftarrow$ V: Random coin R, called a "challenge", chosen uniformly in $\mathbb{Z}_q$
P $\longrightarrow$ V: $m \leftarrow P^{(2)}((p, q, y), w, R, r)$
V: Accepts the proof if the verification $V(m, R, M, (p, q, y))$ accepts

(See Figures F-1, F-2, and F-3, for examples of such THPZP proof systems.)

Figure 4-19: Framework for a THPZP Proof System

In Figure 4-19 we describe the framework for a discrete-log based THPZP proof system. It considers only *discrete-log based* THPZP's (Definition 26, Appendix F), i.e. proof systems for relations in which a public value in $L_{\mathcal{R}}$ always includes a pair of primes p, q s.t. q divides $p-1$, and the distribution $\mathcal{D}(y)$ according to which an honest verifier should pick its public coins is always a uniform distribution over $\mathbb{Z}_q$. This is a property of all the THPZP's we enumerate in Appendix F, namely Schnorr's [Sch91] proof system THPZP-DL (Figure F-1) of knowledge of discrete logarithm, a proof system THPZP-Rep (Figure F-2) of knowledge of (equal) representations in some bases, and Cramer-Damgard [CD98] proof system THPZP-MULT (Figure F-3) of knowledge of three committed values which stand in a multiplicative relationship.

Assume that THPZP $= (P^{(1)}, V, P^{(2)})$ is a discrete-log based three-round honest-verifier public-coin zero-knowledge proof of knowledge system for some relation $\mathcal{R}$. The simultaneous proof protocol SP-THPZP based on the proof system THPZP runs on public inputs a Pedersen commitment instance (p, q, g, h) and a vector of public values $\vec{y} = (y_1, ..., y_n)$ such that each (p, q, y_i) is in $L_{\mathcal{R}}$. The private input of each uncorrupted player P_i is a witness w_i such that $((p, q, y_i), w_i) \in \mathcal{R}$, and a random input r_i. First each player P_i follows algorithm $P^{(1)}((p, q, y_i), w_i, r_i)$ and broadcasts the generated commitment M_i. Then the players perform the coin-flip protocol to generate a challenge R, a random number in $\mathbb{Z}_q$, i.e. they perform the RVSS (Figure 4-6) to generate a secret-sharing RVSS-data$[R]$, and then publicly reconstruct R by performing RVSS-REC (Figure 4-6) on this secret-sharing. Each player P_i then broadcasts its response $m_i = P^{(2)}((p, q, y_i), w_i, R, r_i)$. Finally, each player applies the test $V(m_i, R, M_i, (p, q, y_i))$ for each P_i to determine if P_i passes the proof. Clearly, if the RVSS and RVSS-REC protocols do not fail then all the uncorrupted players make the same decision as to which players pass the simultaneous proof.

To prove that the uncorrupted players do not leak any information to the adversary during the simultaneous proof protocol, we need an additional property of the THPZP proof system we use. Namely, we need the THPZP proof system to be *coin-first simulatable*

Simultaneous Proof Protocol SP-THPZP, $(2t+1 \leq n)$

Underlying THPZP Proof System: Assume a triple of algorithms $P^{(1)}, V, P^{(2)}$ specifies a discrete-log based coin-first simulatable THPZP proof system (see Figure 4-19) for some relation $\mathcal{R}$. Let SIM_{THPZP} be the simulator that exhibits the coin-first simulatable property of this proof system (see Definition 31 in Appendix F.)

Public Input: Pedersen commitment instance (p, q, g, h)
public values $y_1, ..., y_n$ s.t. each $y_i \in L_{\mathcal{R}}$
Private Input of P_i: witness w_i s.t. $((p, q, y_i), w_i) \in \mathcal{R}$
$\mathcal{A}$'s Private Input: adversarial history ah

1. Each P_i broadcasts $M_i = P^{(1)}((p, q, y_i), w_i, r_i)$ for random $r_i \in \mathbb{Z}_q$
2. Players perform protocol RVSS followed by RVSS-REC (Figure 4-6) to create a secret-sharing RVSS-data$[R]$ and then reconstruct publicly the shared value R. If RVSS-REC outputs null then the players abort the protocol.
3. Each P_i broadcasts $m_i = P^{(2)}((p, q, y_i), w_i, R, r_i)$. If $V(m_j, R, M_j, (p, q, y))$ is not satisfied then each P_i removes P_j from *Qual*, and we say that P_i fails SP-THPZP.

Simulator SIM_{SP} of SP-THPZP (interacting with $\mathcal{A}$):

Public Input: Pedersen commitment instance (p, q, g, h), values $y_1, ..., y_n$
SIM_{SP}'s Input: Pedersen's trapdoor $\sigma = \log_g h$
$\mathcal{A}$'s Private Input: adversarial history ah

1. SIM_{SP} picks random $R^* \in \mathbb{Z}_q$, and for each uncorrupted player $P_i \in$ *Good*, it runs the simulator SIM_{THPZP} on instance (p, q, g), public value y_i, and coin R^*, to get messages (M_i^*, m_i^*). SIM_{SP} then broadcasts values M_i^*, for $P_i \in$ *Good*
2. SIM_{SP} follows RVSS on behalf of the uncorrupted players to create secret-sharing RVSS-data$[R]$, replaces the data of the uncorrupted players as RVSS-data$[R^*] = \mathcal{T}_{RVSS}$(RVSS-data$[R], R^*, \sigma$) and performs RVSS-REC on RVSS-data$[R^*]$ on behalf of the uncorrupted players as in Step 2 of SP-THPZP.
3. SIM_{SP} broadcasts values m_i^*, for $P_i \in$ *Good*, on behalf of the uncorrupted players.

Figure 4-20: Framework for a Simultaneous Proof Protocol SP-THPZP

(Definition 31, Appendix F), which means that its zero-knowledge property is exhibited by a simulator SIM which, on input only the public value $y = (p, q, y')$, first chooses the verifier's coin R at random in $\mathbb{Z}_q$, and then generates the rest of the transcript, i.e. messages M^*, m^* which stand for the prover's messages to the verifier. This property of a THPZP proof system allows us to construct an efficient simulator for a simultaneous proof protocol based on such THPZP. We notice that all the THPZP proof systems we enumerate in Appendix F

are coin-first simulatable.[12]

In Figure 4-20 we present a framework for a simultaneous proof protocol, together with a simulation procedure for such protocol. We designate an instantiation of the simultaneous proof protocol by acronym SP-X if it is based on a (discrete-log based coin-first simulatable THPZP) proof system X. For example, by SP-THPZP-DL we designate the protocol of Figure 4-20 where triple $(P^{(1)}, V, P^{(2)})$ is a THPZP-DL proof system (Figure F-1).

Lemma 18 (The Simultaneous Proof Protocol is Zero-Knowledge)
Let (p, q, g, h) be a Pedersen commitment instance. Let THPZP *be a discrete-log based coin-first simulatable three-rounds honest-verifier public-coin zero-knowledge proof-of-knowledge proof system for relation $\mathcal{R}$. For every static secure-channels $n/2$-threshold adversary $\mathcal{A}$ with some adversarial history* ah*, for every vector of public values $\vec{y} = (y_1, ..., y_n)$ and witnesses $\vec{w} = (w_1, ..., w_n)$ s.t. $((p, q, y_i), w_i) \in \mathcal{R}$ for each $P_i \in$ Good, the following two variables have identical distribution:*

- *an adversarial view of an execution of* SP-THPZP *on public inputs $(p, q, g, h, \vec{y})$, private inputs w_i of each player $P_i \in$ Good, and adversarial input* ah
- *an adversarial view of a simulation of* SP-THPZP *on public inputs $(p, q, g, h, \vec{y})$, simulator's* SIM_{SP} *input $\sigma = \log_g h$, and adversarial input* ah

Informally: A static adversary learns no information about the witnesses w_i held by the uncorrupted players $P_i \in$ *Good* from participating in the simultaneous proof protocol.

Proof: It follows from the coin-first simulatability property of the THPZP proof system (Definition 31) and from the secrecy property of the (RVSS,RVSS-REC) coin-flip protocol expressed in Lemma 6.

Namely, first, by Lemma 6, the adversarial view of Step 2 of SIM_{SP} (Figure 4-20), i.e. a view of an execution of RVSS followed by a simulation of RVSS-REC to output R^* picked by SIM_{SP}, is identically distributed to the adversarial view of an execution RVSS followed by an execution of RVSS-REC, as in Step 2 of the SP-THPZP protocol. Secondly, by the coin-first simulatability property of the THPZP proof system (see Definition 31), it follows that for each $P_i \in$ *Good*, messages (M_i^*, m_i^*) are distributed identically to what the adversary expects to see in the SP-THPZP protocol which generates challenge R^*. Thus the lemma follows. □

The zero-knowledge property of the simultaneous proof protocol implies also the following witness-hiding property:

Lemma 19 (The Simultaneous Proof Protocol is Witness-Hiding)
Let (p, q, g, h) be a Pedersen commitment instance. Let THPZP *be a discrete-log based coin-first simulatable three-rounds honest-verifier public-coin zero-knowledge proof-of-knowledge proof system for relation $\mathcal{R}$. For every static secure-channels $n/2$-threshold adversary $\mathcal{A}$*

[12] In a similar vain, to prove the equivalent of the proof-of-knowledge property for the simultaneous proof we put additional restriction on the *extractor* that exhibits the proof-of-knowledge property of the underlying THPZP. See the discussion preceding Lemma 20 for more details.

with some adversarial history ah, *for every vector of public values* $\vec{y} = (y_1, ..., y_n)$ *and every two vectors of witnesses* $\vec{w} = (w_1, ..., w_n)$ *and* $\vec{w}' = (w'_1, ..., w'_n)$ *s.t.* $((p, q, y_i), w_i) \in \mathcal{R}$ *and* $((p, q, y_i), w'_i) \in \mathcal{R}$ *for each* $P_i \in$ *Good, the following two variables have identical distributions:*

- *an adversarial view of an execution of protocol* SP-THPZP *on public inputs* $(p, q, g, h, \vec{y})$, *private inputs* w_i *of each player* $P_i \in$ *Good, and adversarial input* ah
- *an adversarial view of an execution of protocol* SP-THPZP *on public inputs* $(p, q, g, h, \vec{y})$, *private inputs* w'_i *of each player* $P_i \in$ *Good, and adversarial input* ah.

Proof: This lemma is immediately implied by Lemma 18, just like a zero-knowledge property of a proof system implies its witness-hiding property. Namely, we note that Lemma 18 states that both the distribution D of an adversarial view of a run of SP-THPZP on inputs $(p, q, g, h, \vec{y}, \vec{w}, \text{ah})$ and the distribution D' of such view of a run of SP-THPZP on inputs $(p, q, g, h, \vec{y}, \vec{w}', \text{ah})$ are identical to the distribution of an adversarial view of a simulation on public inputs $(p, q, g, h, \vec{y})$, SIM_{SP}'s input σ and adversary's input ah. Thus in particular D and D' are identical. □

We would like to claim the following robustness property of the simultaneous proof protocol. Namely, that if any player P_i controlled by the adversary has a higher than negligible chance of passing in a simultaneous proof on public value y_i, then a correct witness w_i s.t. $(y_i, w_i) \in \mathcal{R}$ can be efficiently extracted from such adversary. However, the construction of an extractor that exhibits such property seems to require that the proof-of-knowledge property of the underlying THPZP proof system is exhibited in a particular way. Namely, that the extractor which exhibits the proof-of-knowledge property of the underlying THPZP system, on input a public value y, queries the prover by sending to it only true random coins, picked uniformly in $\mathbb{Z}_q$. This is because the random challenges in the SP protocol are generated via the coin-flip protocol (RVSS;RVSS-REC), which means that only a uniform distribution of coins (or computationally indistinguishable from uniform) can be simulated to the adversary. Since we consider only discrete-log based THPZP proof systems, this is equivalent to requiring that their extractors generate challenges in distribution $\mathcal{D}(y)$ specified by the THPZP proof system. In Definition 29 in Appendix F we define such extraction as *oblivious extraction.* We note that all the THPZP proof systems we include in Appendix F have oblivious extractors.

Lemma 20 (The Simultaneous Proof is a Proof of Knowledge)
Let THPZP *be a discrete-log based three-rounds honest-verifier public-coin zero-knowledge proof-of-knowledge proof system for a relation* $\mathcal{R}$, *with knowledge error at most* $K(|y|) \leq 2^{-c|y|}$ *for some constant* $c > 0$. *Assume furthermore that* THPZP *has an oblivious extractor.*

Consider an execution of the following sequence of protocols: (1) protocol Ped-IG *(Figure 7-1) on public input* 1^k, *which outputs a Pedersen commitment instance* (p, q, g, h), *(2) some protocol* $\mathcal{P}$ *on input* (p, q, g, h), *whose public output contains a vector of values* $\vec{y} = (y_1, ..., y_n)$ *and whose private inputs of each uncorrupted player* P_i *contain a witness* w_i *s.t.* $((p, q, y_i), w_i) \in \mathcal{R}$, *and (3) protocol* SP-THPZP *on inputs* $(p, q, g, h, \vec{y}, \vec{w})$.

Under the discrete logarithm intractability assumption, for every polynomial $p(z)$ there exists an efficient (PPT TM) algorithm $\mathcal{E}_{SP}$, such that for every static secure-channels $n/2$-threshold adversary $\mathcal{A}$ the following two properties hold:

1. *$\mathcal{E}_{SP}$ interacts with $\mathcal{A}$ on common input 1^k by performing the above specified sequence of protocols* Ped-IG, $\mathcal{P}$, *and* SP-THPZP *on behalf of the uncorrupted players.*

   (However, $\mathcal{E}_{SP}$ is allowed to interact with a copy *of algorithm $\mathcal{A}$ on the side.)*

2. *There exists k_0 s.t. for all $k \geq k_0$, with probability at least $1 - \frac{1}{p(k)}$, the extractor $\mathcal{E}_{SP}$ produces as a* private *output a set of witnesses $\{w_i\}_{P_i \in BadQual}$ s.t. $((p, q, y_i), w_i) \in \mathcal{R}$ for every $P_i \in BadQual$ where $BadQual = Bad \cap Qual$ is the set of players who are corrupted but who passed in the above instance of* SP-THPZP.

Proof: The proof is an elementary application of the proof-of-knowledge (with an oblivious-extractor) property of the underlying proof system and the fact that under the discrete-log assumption the coin-flip protocol produces coins with a distribution at most negligible different from uniform, and hence the witnesses of the corrupted players who pass the proof can be extracted (except for at most negligible probability) from their responses to repeated execution of the coin-flip protocol. We fill in the details of this argument below.

Let $BadQual = Bad \cap Qual$ be a non-empty set of corrupted players which pass the SP-THPZP protocol in the above interaction of $\mathcal{E}$ and $\mathcal{A}$. Let $(p, q, g, h, \vec{y}, \vec{w}, \mathsf{ah})$ be the inputs to this instance of SP-THPZP, and let $r_{\mathcal{A}}$ be the random input used by $\mathcal{A}$ in this instance of sequence Ped-IG, $\mathcal{P}$, and SP-THPZP. Let $r_{\mathcal{E}}$ be the random input used by $\mathcal{E}$ in this interaction during Ped-IG, $\mathcal{P}$, and Step 1 of SP-THPZP. For any polynomial $p(z)$ we consider two types of players in *Bad*: Set S_1 of those players whose probability of passing the simultaneous proof on these inputs is at least $T_p(k) = \frac{1}{2np(k)}$, and set S_2 of those players for whom it is lower, where the probability is taken over the random coins of the uncorrupted players (i.e. the extractor $\mathcal{E}$ who controls them) used in the remaining steps of SP-THPZP. We will show a polynomial $p'(z)$ s.t. by interacting with a *copy* of $\mathcal{A}$ executing on the same public, private, and random inputs as in the original interaction, machine $\mathcal{E}$, running in time $p'(k)$, extracts the witnesses w_i for all players $P_i \in S_1$ except for probability at most $\frac{1}{2p(k)}$ for all large enough k. Since the event that any player in S_2 passes the proof in the original interaction of $\mathcal{A}$ and $\mathcal{E}$ (i.e. the event that $S_2 \cap BadQual \neq \emptyset$) happens with a probability smaller than $tT_p(k)$ (this is because in the worst case set S_2 has t elements), it will follow that for all large enough k, except of probability at most $tT_p(k) + \frac{1}{2p(k)} < nT_p(k) + \frac{1}{2p(k)} < \frac{1}{p(k)}$, machine $\mathcal{E}$ learns witnesses w_i for all players in *BadQual*. Since this is true for any polynomial $p(z)$, as well as any $(p, q, g, h, \vec{y}, \vec{w}, \mathsf{ah})$ and $r_{\mathcal{A}}, r_{\mathcal{E}}$, the lemma will follow.

At the end of the interaction with $\mathcal{A}$, machine $\mathcal{E}$ repeats the following interaction with a *copy* of $\mathcal{A}$ for $p'(k)$ times, where $p'(z)$ is a polynomial we will define below. In each of these interaction machine $\mathcal{A}$ runs on the same random input $r_{\mathcal{A}}$ as in the "original" interaction with $\mathcal{E}$. In each of the $p'(k)$ new interactions with $\mathcal{A}$, machine $\mathcal{E}$ simply performs the prescribed sequence of protocols Ped-IG, $\mathcal{P}$, and SP-THPZP, on behalf of the uncorrupted players, using the same randomness $r_{\mathcal{E}}$ throughout protocols Ped-IG, $\mathcal{P}$, and in Step 1 of protocol SP-THPZP, but using fresh randomness in the remaining steps of SP-THPZP.

Then $\mathcal{E}$ gathers all the transcripts of an interaction with each $P_i \in BadQual$ (i.e. generated coins $R^{(j)}$ and the responses $m_i^{(j)}$ of P_i for $j = 1, .., p'(k)$) and uses the same procedure as the oblivious extractor of the underlying THPZP proof system to compute a witness w_i s.t. $((p, q, y_i), w_i) \in \mathcal{R}$. (For each P_i the inputs to this extraction procedure are (p, q, g, h, y_i, M_i) and $\{R^{(j)}, m_i^{(j)}\}_{j=1,...,p'(k)}$.) Note that the oblivious extractor might not output any witness for some P_i's. However, as we argue below, for all large enough k, this extraction does work for all players $P_i \in S_1$ except for at most $\frac{1}{2p(k)}$ probability.

Notice that since in protocols Ped-IG and $\mathcal{P}$ both sides used the same randomness, these protocols have the same outputs $(p, q, g, h, \vec{y}, \vec{w}, \mathsf{ah})$. Moreover, since both sides continue to use the same randomness in Step 1 of SP-THPZP, the commitment messages M_i sent by all corrupted players in that step are the same as in the original run of SP-THPZP. Note that for each player $P_i \in S_1$, the probability that a single interaction of $\mathcal{A}$ and $\mathcal{E}$ produces a valid transcript of the THPZP proof under the public value y_i is at least $T_p(k)$ for all large enough k (where the probability is taken over the random coins that $\mathcal{E}$ uses in Steps 2-3 of SP-THPZP in this interaction). Since THPZP is a proof of knowledge with knowledge error at most 2^{-ck}, and since its extractor is oblivious, it holds for each $P_i \in S_1$ that there exist polynomial $q(z)$ such that for all large enough k, if the random coins $R^{(1)}, ..., R^{(p(k))}$ were distributed independently and uniformly in $\mathbb{Z}_q$ and if the oblivious extractor is used on $p'(k) > q(k)/\left(T_p(k) - \frac{1}{2^{ck}}\right)$ instances of the THPZP proof then it outputs a correct witness w_i except for probability at most $2^{-|(p,q,y_i)|} \leq 2^{-k}$.

Now, under the discrete-logarithm assumption, by the robustness property of RVSS and RVSS-REC (Lemmas 4 and 5), each instance of the coin-flip protocol of Step 2 of SP-THPZP passes except of negligible probability. Furthermore, coin $R^{(j)}$ created in the j-th successful instance of RVSS is distributed uniformly in $\mathbb{Z}_q$. It follows that, for every instance of the interaction of $\mathcal{E}$ and $\mathcal{A}$, the statistical difference between the distribution of the public coin created in this instance and the uniform distribution over $\mathbb{Z}_q$ is a negligible function of k. It follows in particular that the statistical difference between the string of coins $R^{(1)}, ..., R^{(p'(k))}$ produced in this interaction and a uniform distribution over $(\mathbb{Z}_q)^{p'(k)}$ is at most $\frac{1}{2np(k)}$ for all large enough k.

Putting the two facts together we have that for all large enough k, for all $P_i \in S_1$, the extractor $\mathcal{E}$ learns a witness w_i for y_i except for probability at most $2^{-k} + \frac{1}{2np(k)} < \frac{1}{2np(k)}$. Therefore the probability that there is some P_i in S_1 whose witness is not learned by $\mathcal{E}$ is at most $t\frac{1}{2np(k)} < \frac{1}{2p(k)}$ for all large enough k. By the argument of the previous paragraphs, this completes the proof of the lemma. □

4.4.2 Optimally Resilient Multiplication of Shares

We include the $n/2$-threshold multiplication protocol Mult-opt of [GRR98], modified by the use of a simultaneous proof to implement the zero-knowledge proofs between each pair of players.[13] The idea of the protocol of [GRR98] is to simplify the multiplication protocol of [BGW88] and not to rely on error-correcting codes for resilience, as in the $n/4$-threshold Mult protocol of Section 4.3.3. They achieve it by making a fuller use of the verification

[13] In the original proposal of [GRR98], robustness was provided by pairwise zero-knowledge proofs of knowledge, which was less efficient.

information supplied by Pedersen verifiable secret-sharing. Assume for simplicity of exposition that $n = 2t + 1$. Assume that secrets a, b are shared with RVSS-data$_t[a, b]$ and that we want to compute a sharing RVSS-data$_t[v]$ of $v = ab$. Note that each player P_i holds shares α_i, β_i of t-degree polynomials f_a and f_b, and hence each player holds value $v_i = \lambda_i \alpha_i \beta_i$ s.t. $v = ab = \sum_{i=1}^{n} v_i$ for appropriate Lagrange interpolation coefficients λ_i. Therefore, if each P_i creates a sharing PedVSS-data$[v_i]$ by running Pedersen secret-sharing protocol PedVSS on input v_i, and then all these sharings are added together as in the RVSS protocol, then the result is a joint secret-sharing RVSS-data$_t[v]$. Note that this is the same protocol as RVSS (Section 4.2.4), except that instead of sharing a random value, each player P_i shares input $v_i = \lambda_i \alpha_i \beta_i$ derived from its private shares α_i, β_i in RVSS-data$[a, b]$.

To ensure that every player shares a correctly computed v_i, each player performs a zero-knowledge proof that value v_i committed to in $F_{v_i}(0)$, the verification information which is a part of the created sharing PedVSS-data$[v_i]$, is a product of values α_i and $\lambda_i \beta_i$ committed to in $F_a(i)$ and $(F_b(i))^{\lambda_i}$. To guarantee robustness of this protocol we need these proofs to be proofs of knowledge, so that proper witnesses can be efficiently extracted from any cheating players. We can achieve the effect of having each pair of parties conduct such proof between one another with a simultaneous proof protocol presented in the previous section. Therefore the ZKPK proof system we need must be a three-move public-coin honest-verifier zero-knowledge proof of knowledge, which is coin-first simulatable and has an oblivious extractor. Cramer and Damgard [CD98] proposed such proof system for proving knowledge of values $a, b, c = ab$ committed in public commitments A, B, C. We recall this proof system, which we denote as THPZP-MULT, in Figure F-3 in Appendix F.

Since $v = \sum_{P_i \in Part} v_i$, if any player P_k is caught cheating, the other players recover the missing share $v_k = \lambda_k \alpha_k \beta_k$ by reconstructing shares α_k and β_k with protocol Recon. We present the Recon protocol in Figure 4-22 as a reconstruction of λ_k from RVSS-data$[a]$ only. Clearly, reconstruction of β_k from RVSS-data$[b]$ works the same way and should be performed in parallel. Recon relies on the fact that each polynomial share α_k in RVSS-data$[a]$ is itself *additively* shared as $\alpha_k = \sum_{P_i \in Qual} \alpha_{ik}$, in the same way that the generated secret a is additively shared as a sum of P_i's inputs a_i. Note that to reconstruct α_k we need to also publicly reconstruct α_{kk}, which might not be broadcast by the already faulty player P_k. We reconstruct this missing additive share by reconstructing the whole secret-sharing polynomial $f_{a_k}, f_{\hat{a}_k}$ in the standard way (i.e. via reconstruction of PedVSS-data$[a_k]$ shared by P_k). Note that the values that are published in Recon are already known to the adversary who corrupted player P_k.

Efficiency Considerations. In case of malicious failures protocol Mult-opt resorts to its subprotocol Recon to reconstruct the needed shares of the faulty players. This subprocedure is invoked for every fault, and furthermore, procedure Recon itself is restarted every time a malicious fault occurs within it. However, note that there can be a total of t faults within the lifetime of any threshold cryptosystem, because every time a corrupted party exhibits a fault, it is removed permanently from the set of the participating players. Thus the amortized additional cost incurred by running all the required instances of the Recon protocol is negligible.

For notational purposes we introduce a notion of a successful execution of Mult-opt, defined below in a similar to a successful execution of Mult.

Statically Secure $n/2$-Threshold Mult-opt, $(2t+1 \leq n)$

Input: sharings RVSS-data$_t[a]$ and RVSS-data$_t[b]$ (See Fig. 4-7)
Public Output: sharing RVSS-data$_t[v]$ where $v = ab$

1. Let *Part* be an arbitrary set of $2t+1$ players. Each player $P_i \in Part$ computes its additive share $v_i = \lambda_i \alpha_i \beta_i$ of v for appropriate interpolation coefficients λ_i.
2. Players perform a RVSS protocol, where each player in *Part* uses v_i as its input instead of picking it at random (players not in *Part* do not participate as dealers, only as receivers). Denote the created data as RVSS-data$[v]$.
3. Each P_i proves that v_i committed in $F_{v_i}(0)$ is the product of α_i and $\lambda_i \beta_i$ in $F_a(i)$ and $(F_b(i))^{\lambda_i}$. To do that, the players perform a simultaneous proof protocol SP-THPZP-MULT of Figure 4-20 based on the [CD98] proof system THPZP-MULT (Figure F-3, Appendix F).
4. If some P_k fails either in this proof or in the RVSS sharing of Step 2, his input values α_k, β_k are reconstructed from RVSS-data$[a,b]$ via the reconstruction protocol Recon of Figure 4-22, his v_k is publicly computed, and other players incorporate it in the RVSS-data$[v]$ data, for example by taking constant secret-sharing polynomials $f_{v_k}(z) = v_k$, $f_{\hat{v}_k}(z) = 0$.

Figure 4-21: Mult-opt: $n/2$-Threshold Multiplication of Two Shared Secrets

Definition 20 *We call an instance of protocol* Mult-opt *executed on correct joint secret-sharings* RVSS-data$[a,b]$ successful *if and only if (1) the* RVSS *protocol performed in Step 2, Figure 4-21, is successful; (2) Each v_i, $P_i \in Bad$, shared in that step is correctly computed, i.e. it is equal to $v_i = \lambda_i \alpha_i \beta_i$ where α_i, β_i are defined by* RVSS-data$[a,b]$*; (3) If some player $P_i \in Bad$ is disqualified in Step 2 then the subsequent reconstruction of their shares α_i, β_i via protocol* Recon *is successful, i.e. proper shares defined by* RVSS-data$[a,b]$ *are publicly reconstructed as α_i, β_i, and thus the proper value $v_i = \lambda_i \alpha_i \beta_i$ is publicly computed.*

Lemma 21 (Robustness of $n/2$-Threshold Mult-opt**)**
Consider an execution of the following sequence of protocols: (1) protocol Ped-IG *(Figure 7-1) which on public input 1^k outputs a Pedersen commitment instance (p,q,g,h), (2) some protocol $\mathcal{P}$ which on input (p,q,g,h) produces outputs which contain two instances of joint secret-sharing* RVSS-data$[a,b]$*, and (3) protocol* Mult-opt *on input* RVSS-data$[a,b]$.

Under the discrete logarithm intractability assumption, in the presence of a static secure-channels $n/2$-threshold adversary (i.e. $2t+1 \geq n$), if the output of $\mathcal{P}$ contains two instances RVSS-data$[a,b] \in \mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$ *then, except for probability negligible in the security parameter k, the above instance of* Mult-opt *is* successful.

Moreover, such instance of Mult-opt *outputs a correct joint secret-sharing* RVSS-data$[v]$ *s.t. $v = ab$ where a and b are defined by* RVSS-data$[a,b]$.

Proof: Assume that the discrete logarithm problem is intractable. Assume that there is a higher than negligible probability that some $n/2$-threshold adversary (a non-uniform family

Share Reconstruction Protocol Recon, $(2t+1 \leq n)$

Input: joint secret-sharing RVSS-data$[a]$
the identity of the guilty party P_k
Public Output: polynomial share α_k held by P_k, contained in RVSS-data$[a]$

1. Each P_i broadcasts $\alpha_{ik}, \hat{\alpha}_{ik}$, the shares P_i sent to P_k during formation of RVSS-data$[a]$, and $\alpha_{ki}, \hat{\alpha}_{ki}$, the shares P_k sent to P_i
2. Each share $\alpha_{ik}, \hat{\alpha}_{ik}$ is publicly verified against the verification data F_{a_i}. If $g^{\alpha_{ik}} h^{\hat{\alpha}_{ik}} \neq F_{a_i}(k)$ then P_i is faulty and its shares $\alpha_{ik}, \hat{\alpha}_{ik}$, together with the whole secret-sharing polynomials $f_{a_i}, f_{\hat{a}_i}$ are publicly reconstructed via PedVSS-REC on PedVSS-data$[a_i]$, i.e. each P_j broadcasts $\alpha_{ij}, \hat{\alpha}_{ij}$, and the polynomials are interpolated from shares that pass verification $g^{\alpha_{ij}} h^{\hat{\alpha}_{ij}} = F_{a_i}(j)$
3. Shares $\alpha_{ki}, \hat{\alpha}_{ki}$ are verified against data F_{a_k}, the secret-sharing polynomials $f_{a_k}, f_{\hat{a}_k}$ are interpolated from shares that pass this verification, and $\alpha_{kk}, \hat{\alpha}_{kk}$ are publicly computed as $f_{a_k}(k), f_{\hat{a}_k}(k)$
4. α_k is publicly computed as $\sum_{P_i \in Qual} \alpha_{ik}$

Figure 4-22: Recon: Share Reconstruction Protocol

of PPT TM's) $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, ...)$ breaks the robustness property of the Mult-opt protocol. We will show a contradiction, namely we construct a discrete-log family DL and an efficient algorithm (a non-uniform family of PPT TM's) $\mathcal{E} = (\mathcal{E}_1, \mathcal{E}_2, ...)$, called an extractor, which on input an instance (p, q, g) of security parameter k in DL, and a random element $\tilde{g}$ in G_q, interacts with $\mathcal{A}$ and computes $\log_g \tilde{g}$ with probability higher than negligible. In other words, we reduce breaking Mult-opt to computing discrete logs.

The reduction is simple: Since we have seen earlier that under a discrete-log assumption a computationally bound adversary can cheat in the secret-sharing and reconstruction protocols only with a negligible probability, the only way that remains for an adversary to cheat in the Mult-opt protocol is for some corrupt player P_i to pass the simultaneous proof of Step 3 even if the value v_i shared by this player is not equal to $\lambda_i \alpha_i \beta_i$. It follows from the proof-of-knowledge property of the simultaneous proof that for every player that passes the proof, the extractor can extract, except for negligible probability, the correct witness for the proven statement. Therefore, our extractor will on one hand know the shares $\alpha_i, \hat{\alpha}_i, \beta_i, \hat{\beta}_i, v_i, \hat{v}_i$ that belong to P_i (note that the extractor controls the majority of the players and hence can interpolate all the secret-sharings), and on the other it will extract from $\mathcal{A}$ values $\alpha'_i, \hat{\alpha}'_i, \beta'_i, \hat{\beta}'_i, v'_i, \hat{v}'_i$ which map to the same commitments, i.e. $g^{\alpha_i} h^{\hat{\alpha}_i} = g^{\alpha'_i} h^{\hat{\alpha}'_i}$ (and the same for β's and v's), and which satisfy the multiplicative relationship $\lambda_i \alpha'_i \beta'_i = v'_i$. Therefore, either $\alpha_i = \alpha'_i$, $\beta_i = \beta'_i$, and $v_i = v'_i$, and hence $\lambda_i \alpha_i \beta_i = v_i$, or the extractor gets two representations of some value in G_q, and hence can compute $\log_g h$. Using simulator $\mathsf{SIM}^{(2)}_{h-IG}$ (Figure 7-3), the extractor $\mathcal{E}$ can imbed an instance of the discrete-log problem, a random $\tilde{g}$ in G_q, in value h generated in h-IG so that $h = g^a \tilde{g}^b$ for some values a, b known to $\mathcal{E}$. (Compare the proof of robustness of Pedersen's VSS in Lemma 3.) Therefore learning

$\log_g h$ allows $\mathcal{E}$ to learn $\log_g \tilde{g}$. This is the main argument, and below we fill out its details.

Note that if P_i is disqualified in either Step 2 or Step 3 then v_i is computed from α_i and β_i reconstructed by Recon (Step 4). Since Recon is just another variation of Pedersen VSS reconstruction protocol, the proof that Recon is robust under the discrete-log assumption is virtually identical to the proof of Lemma 2. Therefore, under the discrete-log assumption, the probability that the adversary cheats in Step 2, i.e. that the players fail to create a correct joint secret sharing RVSS-data$[v]$, plus the probability that any execution of Recon fails to reconstruct a proper value, is a negligible function of k. Therefore, if $\mathcal{A}$ has a higher than negligible probability of cheating in Mult-opt, it must have a higher than negligible probability that some $P_i \in Bad$ passes the SP-THPZP-MULT proof but s.t. the value v_i this player shared in Step 2 is not equal to $\lambda_i \alpha_i \beta_i$. We will say in such case that "P_i (or $\mathcal{A}$) cheats in the SP-THPZP-MULT protocol".

It follows that there exists polynomial $p_{\mathcal{A}}(z)$ s.t. for all k_0 there exists $k \geq k_0$ s.t. $\mathcal{A}_k$ cheats in SP-THPZP-MULT with probability at least $1/p_{\mathcal{A}}(k)$. By the same argument as we used in the proof of robustness of Pedersen Verifiable Secret Sharing (Lemma 3), for every k we consider only the execution of the DL-IG part of the Ped-IG protocol where the vector $\vec{r_k}$ random coins used by the adversary and the servers maximizes $\mathcal{A}$'s subsequent cheating in SP-THPZP-MULT. Let DL be a family of discrete-log instances which are output by Ped-IG on those coins. Extractor $\mathcal{E}_k$, on input an instance (p, q, g) of security parameter k in DL and an element $\tilde{g}$ in G_q, runs Ped-IG to $\mathcal{A}_k$ on random coins of $\mathcal{A}_k$ and the servers specified by vector $\vec{r_k}$. This produces (p, q, g) as a public output.

After this execution of DL-IG the extractor $\mathcal{E}$ simulates to $\mathcal{A}$ the execution of h-IG (the second part of Ped-IG) using simulator $\mathsf{SIM}^{(2)}_{h-IG}$ (Figure 7-3) on input $(p, q, g, \tilde{g})$. Under the discrete-log assumption, by Lemma 36, $\mathsf{SIM}^{(2)}_{h-IG}$ runs in polynomial time and the difference between $\mathcal{A}$'s view of h-IG and $\mathcal{A}$'s view of the interaction with SIM_{h-IG} is negligible. In particular, it is smaller than $1/(2p_{\mathcal{A}}(k))$ for all large-enough k, and hence for all k_0 there exists $k \geq k_0$ s.t. the probability that $\mathcal{A}_k$ cheats in the subsequent execution of SP-THPZP-MULT is at least $1/(2p_{\mathcal{A}}(k))$. Furthermore, by the same lemma, there exists k_{SIM} s.t. for all $k \geq k_{SIM}$, $\mathcal{E}$ receives from $\mathsf{SIM}^{(2)}_{h-IG}$ values $a, b \in \mathbb{Z}_q$ such that $g^a \tilde{g}^b = h$ where h is output in this (simulated) run of h-IG, except for probability at most $1/(4p_{\mathcal{A}}(k))$. Therefore, for every k_0 there exists $k \geq k_0$ s.t. with probability at least $1/(4p_{\mathcal{A}}(k))$, $\mathcal{A}$ cheats in the subsequent SP-THPZP-MULT *and* $\mathcal{E}$ knows a, b s.t. $g^a \tilde{g}^b = h$.

After this simulation of h-IG the extractor $\mathcal{E}$ performs protocols $\mathcal{P}$ and Mult-opt by playing the part of the uncorrupted players. Note that since $\mathcal{E}$ controls the majority of the players, by the correctness properties of joint secret-sharing RVSS-data$[a]$ and RVSS-data$[b]$ created by $\mathcal{P}$, the extractor knows all shares $\alpha_i, \hat{\alpha}_i$ and $\beta_i, \hat{\beta}_i$ for all $P_i \in Qual$ s.t. $g^{\alpha_i} h^{\hat{\alpha}_i} = F_a(i)$ and $g^{\beta_i} h^{\hat{\beta}_i} = F_b(i)$. By the robustness property of RVSS (Lemma 4), in Step 2 of Mult-opt a correct joint secret-sharing RVSS-data$[v]$ is generated, except for a negligible probability. For example we can upper-bound it by $1/(8p_{\mathcal{A}}(k))$ for all large enough k. In particular it follows that for all k_0 there exists $k \geq k_0$ s.t. with probability at least $1/(8p_{\mathcal{A}}(k))$ the adversary cheats in SP-THPZP-MULT *and* $\mathcal{E}$ holds shares $(v_i, \hat{v}_i) = (f_{v_i}(0), f_{\hat{v}_i}(0))$ for every $P_i \in Bad \cap Qual$. ($\mathcal{E}$ reconstructs them from shares $(\nu_{ij}, \hat{\nu}_{ij}) = (f_{v_i}(j), f_{\hat{v}_i}(j))$ that it holds for every $P_j \in Good$.)

By the proof-of-knowledge property of SP-THPZP-MULT, i.e. by Lemma 20 in conjunc-

tion with Lemma 41, $\mathcal{E}$ can run an efficient subprocedure, the extractor $\mathcal{E}_{SP}$ of Lemma 20, and compute, except for probability at most $1/(16p_{\mathcal{A}}(k))$, for all corrupted players $P_i \in Part$ which pass the SP-THPZP-MULT protocol of Step 3, witnesses $\alpha'_i, \hat{\alpha}'_i, \beta'_i, \hat{\beta}'_i, v'_i, \hat{v}'_i$ s.t. $g^{\alpha'_i} h^{\hat{\alpha}'_i} = F_a(i)$, $g^{\beta'_i} h^{\hat{\beta}'_i} = F_b(i)$, $g^{v'_i} h^{\hat{v}'_i} = F_{v_i}(0)$, and such that $\lambda_i \alpha'_i \beta'_i = v'_i$.

It follows that for all k_0 there exists $k \geq k_0$ s.t. with probability at least $1/(16p_{\mathcal{A}}(k))$ the extractor $\mathcal{E}$ learns all the above values but $\lambda_i \alpha_i \beta_i \neq v_i$ because $\mathcal{A}$ cheated in the SP-THPZP-MULT protocol. But in such case one of the following three inequalities must hold: either $(\alpha'_i, \hat{\alpha}'_i) \neq (\alpha_i, \hat{\alpha}_i)$, or $(\beta'_i, \hat{\beta}'_i) \neq (\beta_i, \hat{\beta}_i)$, or $(v'_i, \hat{v}'_i) \neq (v_i, \hat{v}_i)$. In either case $\mathcal{E}$ can compute $\log_g h$, and hence $\log_g \bar{g}$, because this means that $\mathcal{E}$ holds two different representations of some number, either $F_a(i)$, or $F_b(i)$, or $F_{v_i}(0)$, in bases g, h.

Therefore, if discrete-log is hard then RVSS-data$[v]$ contains, except for negligible probability, a proper secret-sharing of each $v_i = \lambda_i \alpha_i \beta_i$ for $P_i \in Part$, and therefore $v = \sum_{P_i \in Qual} \lambda_i \alpha_i \beta_i = ab$. □

Secrecy Property of Mult-opt

As for the $n/4$-threshold multiplication protocol Mult, we state the secrecy property of Mult-opt in terms of the ability of an efficient simulator to execute Mult-opt(RVSS-data$[a, b]$) $\rightarrow$ RVSS-data$[v]$ and then to simulate any *subsequent* protocol $\mathcal{P}$ that follows an execution of Mult-opt on secret-sharings RVSS-data$[a^*, b^*, v^*]$ which are related to RVSS-data$[a, b, v]$, but where a^*, b^*, v^* are random numbers in G_q subject only to the constraint that $a^* b^* = v^*$. To express this property we introduce an auxiliary simulation procedure $\mathcal{T}_{Mult}$, Figure 4-23, which can be thought of as replacing the private data related to each secret-sharing PedVSS-data$[v_i]$, for all $P_i \in Good$.

Note, however, that this replacement procedure does not produce all the secret data that the uncorrupted players should have as outputs of Mult-opt. Namely, the simulated players receive only the new polynomial shares $\nu^*_i, \hat{\nu}^*_i$ of the new secret v^*. They do not receive new additive shares $v^*_i, \hat{v}^*_i$ of this secret, nor do they receive the shares of their own component secret-sharing polynomials $f^*_{v_i}, f^*_{\hat{v}_i}$. (However, each uncorrupted player P_i does keep the shares $\nu_{ij}, \hat{\nu}_{ij}$ of their secret-sharing polynomials they sent to the corrupted players $P_j \in Bad$, and shares $\nu_{ji}, \hat{\nu}_{ji}$ received from these players.) We make this explicit in Figure 4-23 by having all these values over-written with question mark signs. This means that any protocol $\mathcal{P}$ that runs on the outputs of Mult-opt can be simulated only if $\mathcal{P}$ does not use (i.e. take as inputs) all the question-marked data. (See also the paragraph below on a more general secrecy property of the Mult-opt protocol.)

Lemma 22 (Static Secrecy of $n/2$-Threshold Mult-opt)
There exists a simulator SIM, *such that for every $n/2$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah, *for any distributed protocol $\mathcal{P}$, for every discrete-logarithm instance (p, q, g), and for every $h \in G_q$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a run of two* successful *instances of* RVSS *(either parallel or sequential), on public input (p, q, g, h) and adversarial input* ah, *with outputs denoted* RVSS-data$[a, b]$

Replacement procedure $\mathcal{T}_{Mult}$

Input: public and private data of uncorrupted players in RVSS-data$[v]$
(in fact data Inc-RVSS-data$[v] \subset$ RVSS-data$[v]$ is enough, see below)
"target output value" $v^* \in \mathbb{Z}_q$
Pedersen's trapdoor $\sigma = \log_g h$

Output: (private data of uncorrupted players in) Inc-RVSS-data$[v^*]$

Let $Bad, Good$ be the identities of the corrupted and uncorrupted players ($|Bad| = t$).
Let Inc-RVSS-data$[v]$ stands for an "incomplete" data-structure RVSS-data$[v]$, where the private data of each uncorrupted player P_i consists only of its polynomial shares $\nu_i, \hat{\nu}_i$.
Note that the secret-sharing polynomials $f_v, f_{\hat{v}}$ are defined by the shares $\nu_i, \hat{\nu}_i$ held by all the uncorrupted players.
$\mathcal{T}_{Mult}$ picks random t-degree polynomials $f_v^*, f_{\hat{v}}^*$ s.t.:

$$\begin{aligned} f_v^*(0) &= v^* \\ f_v^*(i) &= f_v(i) \text{ for } P_i \in Bad \\ f_v^*(z) + \sigma f_{\hat{v}}^*(z) &= f_v(z) + \sigma f_{\hat{v}}(z) \text{ (for all } z) \end{aligned}$$

and then forms Inc-RVSS-data$[v^*]$ from Inc-RVSS-data$[v]$ by replacing the private data $\nu_i = f_v(i)$ and $\hat{\nu}_i = f_{\hat{v}}(i)$ of each $P_i \in Good$ with $\nu_i^* = f_v^*(i)$ and $\hat{\nu}_i^* = f_{\hat{v}}^*(i)$

Figure 4-23: $\mathcal{T}_{Mult}$: Auxiliary simulation procedure

  - *a* successful *run of* Mult-opt *on input* RVSS-data$[a, b]$, *with outputs* RVSS-data$[v]$
  - *a run of* $\mathcal{P}$ *on inputs* RVSS-data$[a, b]$, Inc-RVSS-data$[v]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input* $\sigma = \log_g h$
  - *a run of two* successful *instances of* RVSS *(either parallel or sequential, as in the corresponding step of the sequence above), on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a, b]$
  - *a* successful *run of* Mult-opt *on input* RVSS-data$[a, b]$, *with outputs* RVSS-data$[v]$
  - *a replacement of the private data in* RVSS-data$[a, b]$, *and* RVSS-data$[v]$ *of the simulated players with*
    * RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma$)
    * RVSS-data$[b^*] = \mathcal{T}_{RVSS}$(RVSS-data$[b], b^*, \sigma$)
    * Inc-RVSS-data$[v^*] = \mathcal{T}_{Mult}$(RVSS-data$[v], v^*, \sigma$)
      where $a^*, b^*, v^* \in \mathbb{Z}_q$ *are random subject to the constraint that* $a^* b^* = v^*$

    and then a run of $\mathcal{P}$ *on* RVSS-data$[a^*, b^*]$, Inc-RVSS-data$[v^*]$

Remark. Notation Inc-RVSS-data$[a]$ in the Lemma above stands for an "incomplete" joint secret-sharing distributed data structure RVSS-data$[a]$, where the private data of each honest player P_i consists only of its polynomial shares $\alpha_i, \hat{\alpha}_i$. In this way we restrict the

type of protocols $\mathcal{P}$ that can be efficiently simulated on the outputs of Mult-opt. Namely, only protocols which take as their inputs no more but the polynomial shares output by each honest player in protocol Mult-opt. Note that this is also the only data given to the uncorrupted players in Inc-RVSS-data$[v^*]$ output by the simulation procedure $\mathcal{T}_{Mult}$ of Figure 4-23. (See also a paragraph below on a discussion of a more general secrecy property of the Mult-opt protocol.)

Proof: This lemma follows straightforwardly from the polynomial secrecy of RVSS, from the polynomial secrecy of Pedersen VSS that each player performs in Step 2, and from the witness-hiding property of the simultaneous proof protocol of Step 3.

For simplicity of notation assume that $Part = Bad \cup Good$, i.e. that $n = 2t + 1$. Consider two successful executions of RVSS which create correct sharings RVSS-data$[a]$ and RVSS-data$[b]$, and a successful execution of Mult-opt on RVSS-data$[a, b]$ which creates correct sharing RVSS-data$[v]$ where $v = ab$. By the polynomial secrecy of RVSS (Lemma 6), the distribution of outputs of such two instances of RVSS is identical to the distribution of outputs of two instances of RVSS followed by a replacement of the private data of the uncorrupted players via procedure $\mathcal{T}_{RVSS}$(RVSS-data$[a, b], \sigma) \rightarrow$ RVSS-data$[a^*, b^*]$. It remains for us to show that the distribution of Inc-RVSS-data$[v^*]$ output by a successful execution of Mult-opt on RVSS-data$[a^*, b^*]$ is identical to the distribution of Inc-RVSS-data$[v^*]$ output by Mult-opt(RVSS-data$[a, b]) \rightarrow$ RVSS-data$[v]$ followed by replacement $\mathcal{T}_{Mult}$(RVSS-data$[v], v^*, \sigma) \rightarrow$ Inc-RVSS-data$[v^*]$, where $v^* = a^*b^*$.[14]

Let f_{v_i}, $P_i \in Part$, and $f_v(z) = \sum_{P_i \in Part} f_{v_i}(z)$, be the polynomials in RVSS-data$[v]$ created in a successful Mult-opt on RVSS-data$[a, b]$. Let $v_i^* = \lambda_i f_a^*(i) f_b^*(i)$ for $P_i \in Good$ where f_a^*, f_b^* are given in RVSS-data$[a^*, b^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a, b], \sigma)$. For each $P_i \in Good$, let $f_{v_i}^*$ be a random t-degree polynomial s.t. $f_{v_i}^*(0) = v_i^*$ and $f_{v_i}^*(j) = f_{v_i}(j)$ for all $P_j \in Bad$. Let $f_v^*(z) = \sum_{P_i \in Bad} f_{v_i}(z) + \sum_{P_i \in Good} f_{v_i}^*(z)$. It therefore follows from the polynomial secrecy of PedVSS (Lemma 3) that the output of a successful execution of Step 2 on RVSS-data$[a, b]$ followed by a replacement of $\mathcal{T}_{PedVSS}$(PedVSS-data$[v_i]) \rightarrow$ PedVSS-data$[v_i^*]$ is distributed as the output of a successful execution of this step on RVSS-data$[a^*, b^*]$. It thus follows that the output of a successful execution of this step followed by a replacement of $\mathcal{T}_{Mult}$(RVSS-data$[v], v^*, \sigma) \rightarrow$ Inc-RVSS-data$[v^*]$ is distributed as Inc-RVSS-data$[v^*]$ output by a successful execution of this step on RVSS-data$[a^*, b^*]$.

In particular it follows that values $F_a(i)$, $F_b(i)$, and F_{v_i} for $P_i \in Good$, are distributed identically in the two distributions. Therefore, by the witness-hiding property of the SP-THPZP-MULT protocol of Step 3, the distribution of an adversarial view of that step is also identical in the two cases.

Note that only a corrupt player can fail in the RVSS or the simultaneous proof protocol, and hence protocol Recon triggered by Step 4 of Mult-opt can only be executed if the identity P_k designates an already corrupt player. Therefore all the public information generated in this step is already contained in the adversarial history. □

[14]Note that the outputs of Mult-opt form a distributed data structure RVSS-data, but in the above statement we are concerned only with the outputs Inc-RVSS-data included in this data structure RVSS-data (see the remark above). Note furthermore that the data structures Inc-RVSS-data$[v]$ and Inc-RVSS-data$[v^*]$ contain adversarial views of the instances of the Mult-opt protocol that produced them.

More general secrecy property of Mult-opt

Among the threshold protocols discussed in this work, only protocol RVSS-REC can be executed on only the Inc-RVSS-data$[a]$ part of joint secret-sharing RVSS-data$[a]$. In all other threshold protocols which take a joint secret-sharing RVSS-data$[a]$ as their inputs the honest players require more than just their polynomial shares $\alpha_i, \hat{\alpha}_i$. However, if we look closer into what exactly these protocols need, we see that each honest party P_i in protocols Mult, and Reciprocal, as well as protocol Mult-opt itself and protocol Reciprocal-opt of the next section, takes as inputs only its polynomial shares $\alpha_i, \hat{\alpha}_i$ as well as (1) polynomial shares $\alpha_{ji}, \hat{\alpha}_{ij}$ received from each $P_j \in Bad$; and (2) polynomial shares $\alpha_{ij}, \hat{\alpha}_{ji}$ that P_i sent to each $P_j \in Bad$. In other words, the only other data the uncorrupted players need is the data that the adversary has already seen. Therefore the same argument given in the proof of Lemma 22 above implies that all the above protocols can also be securely executed on the outputs RVSS-data$[v]$ of protocol Mult-opt, i.e. that such execution can be simulated to "hit" any target v^* using the same auxiliary simulation procedure $\mathcal{T}_{Mult}$ of Figure 4-23.[15] However, since in the threshold DSS signature scheme (Section 4.4.3) the secret-sharing output by Mult-opt is always immediately reconstructed via RVSS-REC, Lemma 22 suffices to prove the security of this particular threshold scheme.[16]

However, the threshold exponentiation protocol Exp of Section 4.2.5, as well as its adaptively secure version Ad-Exp presented in Chapter 5, *does* need (some of) the remaining data in a joint secret-sharing RVSS-data$[a]$ of the uncorrupted players. Namely, in protocol Exp each party P_i needs its "component" secret-sharing f_{a_i}, while in the adaptively secure version Ad-Exp of this protocol each party P_i needs its additive share $a_i = f_{a_i}(0)$. Therefore our analysis of the security of Mult-opt does not imply any secrecy property of, for example, the following sequence of threshold protocols:

$$\begin{aligned} 2 \times \text{RVSS} &\rightarrow \text{RVSS-data}[a, b] \\ \text{Mult-opt}(\text{RVSS-data}[a, b]) &\rightarrow \text{RVSS-data}[v] \\ \text{Exp}(\text{RVSS-data}[a], m) &\rightarrow m^a \\ \text{Exp}(\text{RVSS-data}[b], m) &\rightarrow m^b \\ \text{Exp}(\text{RVSS-data}[v], m) &\rightarrow m^v \end{aligned}$$

Where by "secrecy property" we mean a property we would *like* to claim of such protocol, i.e. that the above sequence does not reveal anything more about the generated shared secrets a and b except of values g^a, g^b, and g^{ab}.

We note that it is likely that under the decisional Diffie-Hellman assumption, a simulator for the above protocol constructed using our techniques[17] presents an adversarial view which is *computationally indistinguishable* from an adversarial view of an execution of this

[15] We note that this argument holds only for the static adversarial model.

[16] The outputs of Mult-opt are used differently in the threshold Cramer-Shoup decryption protocol, and we need a separate argument why this protocol is secure. (See Appendix A.)

[17] Namely, a simulator which on input random $A^* = g^{a^*}$, $B^* = g^{b^*}$, $V^* = g^{a^*b^*}$ for random a^*, b^* in $\mathbb{Z}_q$, performs $2 \times$ RVSS $\rightarrow$ RVSS-data$[a, b]$ and Mult-opt(RVSS-data$[a, b]$) $\rightarrow$ RVSS-data$[v]$ and then simulates the three instances of Exp via SIM_{Exp} of Figure 4-9, first on inputs RVSS-data$[a]$, A^*, then on RVSS-data$[b]$, B^*, and then on RVSS-data$[v]$, V^*.

protocol. However, even if Mult-opt functions well in this sense in the above example, such property does not seem as generally useful as the secrecy properties we can show for the $n/4$-threshold protocols Mult or Reciprocal, or the $n/2$-threshold protocol Reciprocal-opt of the next section.

Note that adding a refreshment sharing ZVSS to the above Mult-opt does not improve its secrecy property. (I.e. creating ZVSS $\rightarrow$ ZVSS-data$[z]$ and refreshing the secret-sharing of value ab by outputting RVSS-data$[v']$ = RVSS-data$[v]$+ZVSS-data$[z]$ instead of RVSS-data$[v]$ itself, compare Section 4.3.2.) This is because all the secret-sharing polynomials f_{z_i} in ZVSS-data$[z]$ have free coefficients equal to zero (at least for $P_i \in Good$). Therefore if a revised Mult-opt protocol outputs secret-sharing polynomials $f_{v'_i}(z) = f_{v_i}(z) + f_{z_i}(z)$, then values $v'_i = f_{v'_i}(0)$ will still be equal to $v_i = f_{v_i}(0)$. Hence they will still be equal to $\lambda_i \alpha_i \beta_i$, and thus they will still be correlated with the secret-sharing polynomials f_a, f_b. In other words, the distribution of data in RVSS-data$[a, b, v']$ is different than in RVSS-data$[a^*, b^*, v'^*]$ where RVSS-data$[a^*]$ = $\mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma$), RVSS-data$[b^*]$ = $\mathcal{T}_{RVSS}$(RVSS-data$[b], b^*, \sigma$), RVSS-data$[v'^*]$ = $\mathcal{T}_{RVSS}$(RVSS-data$[v'], a^*b^*, \sigma$), and a^*, b^* are random in $\mathbb{Z}_q$.

Instead, Mult-opt *can* be amended so that it has the same type of secrecy property as the other protocols, and hence can be more generally useful as a building block in larger threshold protocols. To do that we need to refresh the sharing RVSS-data$[v]$ output by Mult-opt in a manner which re-randomizes all its component secret-sharings. For example, the above Mult-opt can be followed by RVSS $\rightarrow$ RVSS-data$[r]$, RVSS-REC(RVSS-data$[r]$) $\rightarrow$ r, and then local scaling RVSS-data$[v']$ = RVSS-data$[v]$ + RVSS-data$[r]$ $-$ r, which means that every party P_i re-computes its component secret-sharing polynomial for example as $f_{v'_i}(z) = f_{v_i}(z) + f_{r_i}(z) - \frac{r}{|Qual|}$, and hence $v'_i = v_i + r_i - \frac{r}{|Qual|}$ and $v' = \sum_{P_i \in Qual} v'_i = \sum_{P_i \in Qual}(v_i + r_i - \frac{r}{|Qual|}) = v + r - r = v = ab$. Then the correlation between values $v'_1, ..., v'_n$ and the secret-sharing polynomials f_a and f_b will be lost, except for the proper constraint that v' is equal to ab.

Security of Repetitive Execution of Mult-opt

The above secrecy property of protocol Mult-opt can be extended to the case of its *repeated* execution, just like in Section 4.3.3 we argued that the $n/4$-threshold multiplication protocol Mult remains secret if it is executed repetitively. By a repetitive execution of a threshold multiplication protocol we mean that it is used to compute sharings of values $c_j = a * b_j$ for $j = 1, ..., p(k)$ for some polynomial $p(z)$, where values a and $b_1, ..., b_{p(k)}$ are secret-shared as RVSS-data$[a]$, and RVSS-data$[b_1]$, ..., RVSS-data$[b_{p(k)}]$. We stated such property for Mult in Lemma 13 in Section 4.3.3. The $n/2$-threshold multiplication protocol Mult-opt also remains secure when performed repetitively in the sense that Lemma 22 can be extended to claim that the following adversarial views are identically distributed:

- an adversarial view of a sequence of *successful executions* of the following protocol: RVSS $\rightarrow$ RVSS-data$[a]$ and some protocol $\mathcal{P}_1$(RVSS-data$[a]$), followed by the following loop for $j = 1, ..., p(k)$: RVSS $\rightarrow$ RVSS-data$[b_j]$, some protocol $\mathcal{P}_2$(RVSS-data$[b_j]$), Mult-opt(RVSS-data$[a, b_j]$) $\rightarrow$ RVSS-data$[v_j]$, and some protocol $\mathcal{P}_3$(RVSS-data$[a, b_j]$, Inc-RVSS-data$[v_j]$).

- an adversarial view of a sequence of *successful simulations* of the above process,

where the simulator performs RVSS $\rightarrow$ RVSS-data$[a]$ on behalf of the uncorrupted players, replaces RVSS-data$[a^*] \leftarrow \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma)$ for a random $a^* \in \mathbb{Z}_q$, performs $\mathcal{P}_1$(RVSS-data$[a^*]$), and then for $j = 1, ..., p(k)$ it executes RVSS $\rightarrow$ RVSS-data$[b_j]$ on behalf of the uncorrupted players, replaces RVSS-data$[b_j^*] \leftarrow \mathcal{T}_{RVSS}$(RVSS-data$[b_j], b_j^*, \sigma)$ for a random $b_j^* \in \mathbb{Z}_q$, executes $\mathcal{P}_2$(RVSS-data$[b_j^*]$), runs Mult-opt(RVSS-data$[a, b_j]) \rightarrow$ RVSS-data$[v_j]$, assigns Inc-RVSS-data$[v_j^*] \leftarrow \mathcal{T}_{Mult}$(RVSS-data$[v_j]$, $v_j^*, \sigma)$ where $v_j^* = a^* b_j^*$, and runs $\mathcal{P}_3$(RVSS-data$[a^*, b_j^*]$, Inc-RVSS-data$[v_j^*]$).

Moreover, we can prove a more general statement of the above putting all the extensions captured in Lemma 13, Section 4.3.3, i.e. where protocols $\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3$ pass public information between one another, and Mult-opt is executed not on RVSS-data$[a]$ but on RVSS-data$[a']$ = Scale(RVSS-data$[a]$, Sel) where procedure Sel selects the scaling factors from the public output of the threshold protocol so far. The formal proof is a straightforward extension of Lemma 22, just like the proof of Lemma 13 is a straightforward extension of Lemma 12.

4.4.3 Optimally Resilient Computation of Inverses and DSS Signatures

The optimal-threshold versions of distributed protocols for computing inverses and for distributed DSS signature generation are identical to their $n/4$-threshold versions except that in place of the $n/4$-threshold multiplication protocol Mult (which includes the reconstruction of the secret-shared product), the players execute an optimal-threshold multiplication subprotocol Mult-opt followed by a reconstruction of the secret-shared product via protocol RVSS-REC. For ease of following the security arguments that follow, we present these optimal-threshold protocols, Reciprocal-opt and DSS-TSig-opt, in Figures 4-24 and 4-25.

Robustness of protocols Reciprocal-opt and DSS-TSig-opt is implied by virtually the same arguments which we used to show the robustness of Reciprocal and DSS-TSig in Lemmas 14 and 16. The only part of those arguments that needs to change is that references to the robustness properties of Mult and the reconstruction procedure ECSS-REC need to be replaced with references to robustness of Mult-opt and RVSS-REC. Therefore, since both the claims and the proofs of these properties are virtually identical to those for the $n/4$-threshold versions described previously, we omit them here.

The statements and the arguments of the secrecy properties of both Reciprocal-opt and DSS-TSig-opt are also very similar to those of Reciprocal and DSS-TSig. For completeness, we include below the secrecy property of Reciprocal-opt in Lemma 23. Finally, in Theorems 6 and 5 we state the security of the optimal-threshold DSS signature scheme TSS-opt resulting from a combination of the distributed key generation protocol Ped-IG+DKG (see Section 4.3.5), the optimal-threshold DSS signature generation protocol DSS-TSig-opt presented here, and the DSS signature verification procedure Ver.

Optimal-Threshold Inverse Computation Protocol

To express the privacy property of the Reciprocal-opt protocol we define a notion of its successful execution, similarly as we did for protocol Reciprocal in Section 4.3.4.

Definition 21 *We call an instance of protocol* Reciprocal-opt *executed on correct joint secret-sharing* RVSS-data[a] successful *if and only if this execution included successful instances of protocols* RVSS, Mult-opt, *and* RVSS-REC *(invoked in Steps 1, 2, and 3).*

Statically Secure $n/2$-Threshold Reciprocal-opt, $(2t+1 \leq n)$

Input: sharing RVSS-data[a] (See Fig. 4-7)
Output: sharing RVSS-data[e]
(where $e = a^{-1} \bmod q$ if the adversary does not cheat)

1. Players execute RVSS to create a new sharing RVSS-data[b]
2. Players compute RVSS-data[c] $\leftarrow$ Mult-opt(RVSS-data[a], RVSS-data[b])
3. Players compute $c \leftarrow$ RVSS-REC(RVSS-data[c])
4. Players compute RVSS-data[e] $\leftarrow$ Scale(RVSS-data[b], $0, c^{-1}$), see Figure 4-15

Simulator $\mathsf{SIM}_{Rpcl-opt}$ of Reciprocal-opt **(interacting with $\mathcal{A}$):**

Public Input:	public data in RVSS-data[a]
$\mathsf{SIM}_{Rpcl-opt}$**'s Private Input:**	private outputs of all $P_i \in$ *Good* in RVSS-data[a] Pedersen's trapdoor $\sigma = \log_g h$
$\mathcal{A}$**'s Private Input:**	adversary's output in RVSS-data[a]

1.-2. $\mathsf{SIM}_{Rpcl-opt}$ follows Steps 1-2 of Reciprocal-opt on behalf of the good players.

3. $\mathsf{SIM}_{Rpcl-opt}$ picks c^* uniformly in $\mathbb{Z}_q$ and replaces the private data of the simulated players with the data specified by Inc-RVSS-data[c^*] = $\mathcal{T}_{Mult}$(RVSS-data[c], c^*, σ).
Then $\mathsf{SIM}_{Rpcl-opt}$ performs RVSS-REC on input Inc-RVSS-data[c^*] on behalf of the simulated players.

4. $\mathsf{SIM}_{Rpcl-opt}$ follows Step 4 of Reciprocal-opt on behalf of the uncorrupted players.
(Note that if the adversary does not cheat then the public output of the previous step is c^* chosen by $\mathsf{SIM}_{Rpcl-opt}$, and thus the scaling factor used here is $(c^*)^{-1}$.)

Figure 4-24: Reciprocal-opt: $n/2$-Threshold Inverse Computation Protocol

Lemma 23 (Static Secrecy of $n/2$-Threshold Reciprocal-opt) *There exists a simulator* SIM, *such that for every $n/2$-threshold static secure-channels adversary $\mathcal{A}$ with history* ah, *for any distributed protocol $\mathcal{P}$, for every discrete-logarithm instance (p, q, g), and for every $h \in G_q$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*

  - *a successful run of* RVSS *on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a successful run of* Reciprocal-opt *on input* RVSS-data$[a]$, *with outputs denoted* RVSS-data$[e]$
  - *a run of* $\mathcal{P}$ *on input* RVSS-data$[a]$ *and* RVSS-data$[e]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input* $\sigma = \log_g h$*:*
  - *a successful run of* RVSS *on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a successful simulation of* Reciprocal-opt *via* $\mathsf{SIM}_{Rpcl-opt}$ *of Figure 4-24 (see a remark below) on input* RVSS-data$[a]$ *and the additional input* $\sigma = \log_g h$ *of simulator* $\mathsf{SIM}_{Rpcl-opt}$*, with outputs denoted* RVSS-data$[e]$
  - *a replacement of the private data in* RVSS-data$[a, e]$ *of the simulated players via transformations* RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma$) *and* RVSS-data$[e^*] = \mathcal{T}_{RVSS}$(RVSS-data$[e], e^*, \sigma$)*, where* a^*, e^* *are random elements in* $\mathbb{Z}_q$ *subject to the constraint that* $a^*e^* = 1$*, and then an execution of* $\mathcal{P}$ *on* RVSS-data$[a^*, e^*]$

Less formally: A (static) adversarial view of *successful* executions of RVSS, Reciprocal-opt, and then a random run of protocol $\mathcal{P}$ is *the same* as an adversarial view of a successful execution of RVSS, a successful simulation of Reciprocal-opt, a *modification* of the private data of the uncorrupted players via two $\mathcal{T}_{RVSS}$ transformations which replace the sharings of a, e with sharings of a^*, e^* which are random subject to the constraint that $a^*e^* = 1$, and then an execution of $\mathcal{P}$.
Informally: A static secure-channels $n/2$-threshold adversary learns nothing about the secret-sharing polynomials in RVSS-data$[a, e]$ from an execution of Reciprocal which on input RVSS-data$[a]$ produces a secret-sharing RVSS-data$[e]$, apart of the shares of these polynomials received by the corrupted players, and apart of the fact that $ae = f_a(0)f_e(0) = 1$.

Remark. By "*successful simulation* of Reciprocal-opt via simulator procedure $\mathsf{SIM}_{Rpcl-opt}$", we designate an instance of an interaction between an adversary and the simulator of Figure 4-24, in which the RVSS, Mult-opt, and RVSS-REC protocols which $\mathsf{SIM}_{Rpcl-opt}$ performs on behalf of the uncorrupted players in Steps 1-3, were successful. Thus a successful simulation of Reciprocal-opt via $\mathsf{SIM}_{Rpcl-opt}$ is defined similarly to a successful *execution* of Reciprocal-opt.

Proof Sketch: The reasoning is virtually identical to the proof of secrecy of the $n/4$-threshold inverse-computation protocol Reciprocal in Lemma 15. Namely, the secrecy property of Reciprocal-opt follows straightforwardly from the secrecy of Mult-opt, i.e. from Lemma 22. As in that proof, we see that the reconstruction RVSS-REC(RVSS-data$[c]$) $\rightarrow c$, the scaling Scale(RVSS-data$[b], 0, c^{-1}$) $\rightarrow$ RVSS-data$[e]$ (Steps 3-4 of Reciprocal-opt) and the subsequent run of $\mathcal{P}$(RVSS-data$[a, e]$) are a particular example of some protocol $\mathcal{P}'$ executing on inputs RVSS-data$[a, b]$ and Inc-RVSS-data$[c]$. (Note that this protocol takes only Inc-RVSS-data$[c]$ as an input, because that's all that RVSS-REC needs.) Therefore by the

secrecy property of Mult-opt, the successful execution of these steps is distributed as their simulation where RVSS-data$[a, b, c]$ are replaced with RVSS-data$[a^*, b^*, c^*]$ where $c^* = a^*b^*$ and a^*, b^* are uniform in $\mathbb{Z}_q$. Note that in the simulation of Reciprocal-opt and in subsequent replacement of RVSS-data$[a, e]$ with RVSS-data$[a^*, e^*]$, the simulator picks c^* at random but then picks a^*, e^* subject to the constraint that $a^*e^* = 1$, which is equivalent to the constraint that $a^*b^*/c^* = 1$, i.e. that $a^*b^* = c^*$. Therefore Lemma 23 follows from Lemma 22. □

Optimal-Threshold DSS Signature Scheme

The optimal-threshold DSS signature generation protocol DSS-TSig-opt, Figure 4-25, differs from its $n/4$-threshold version DSS-TSig only by a substitution of the optimal-threshold multiplication protocol Mult-opt in place of its $n/4$-threshold version Mult. We include a simulator $\mathsf{SIM}_{DSS-TSig-opt}$ of this protocol in Figure 4-25. However, the reader might recall that the secrecy property of the threshold signature computation protocol turned out to be equivalent to a particular case of the unforgeability property of the threshold signature *scheme*, namely the case when the signature generation protocol is executed on a single message. (See Section 4.3.5, esp. Remark 4 on page 102.) Therefore here we proceed immediately to proving the unforgeability of the optimal-threshold DSS signature scheme which utilizes the DSS-TSig-opt protocol.

We can now prove the security of an $n/2$-threshold DSS signature scheme TSS-opt = (Ped-IG+DKG, DSS-TSig-opt, Ver), which is very similar to the $n/4$-threshold DSS signature scheme TSS = (Ped-IG+DKG, DSS-TSig, Ver) of Section 4.3.5. The only difference is the replacement of the $n/4$-threshold DSS signature generation protocol DSS-TSig with its optimal-threshold version DSS-TSig-opt presented above. Recall that by Ped-IG+DKG we denote two protocols run one after the other, first the Pedersen instance generation protocol Ped-IG and then the "proper" distributed key generation protocol DKG. Recall also that Ver denotes the standard DSS signature verification procedure. The following two theorems state the unforgeability and the robustness of the threshold DSS signature scheme TSS-opt. For the definitions of security properties of a threshold signature scheme see Section 2.4. We note that the proofs of Theorems 5 and 6 below are very similar to the proofs of the corresponding theorems (Theorems 2 and 3) about the $n/4$-threshold DSS signature scheme TSS.

Theorem 5 (Robustness of $n/2$-Threshold DSS Scheme)
Under the discrete-log intractability assumption, TSS-opt = (Ped-IG+DKG, DSS-TSig-opt, Ver) *is a robust threshold signature scheme in the presence of a static secure-channels $n/2$-threshold adversary.*

Proof Sketch: Robustness of TSS-opt follows straightforwardly from the robustness of Ped-IG, DKG, and DSS-TSig-opt, in the same way as robustness of TSS follows from robustness of Ped-IG, DKG, and DSS-TSig, see Theorem 2, page 105. □

Theorem 6 (Static Unforgeability of $n/2$-Threshold DSS Scheme)
If the DSS signature scheme is CMA secure, then TSS-opt = (Ped-IG+DKG, DSS-TSig-opt, Ver) *is an unforgeable threshold signature scheme in the presence of a static secure-channels $n/2$-threshold adversary.*

Proof Sketch: The proof of the unforgeability of the $n/2$-threshold scheme TSS-opt is almost identical to the proof of the unforgeability of the $n/4$-threshold scheme TSS in Theorem 3 in Section 4.3.5. The only difference between TSS-opt and TSS is the substitution of DSS-TSig with DSS-TSig-opt. However, the $n/2$-threshold DSS-TSig-opt can be simulated only if the simulator knows the Pedersen commitment trapdoor $\sigma = \log_g h$. This brings a few slight but necessary modifications to the argument we use in the proof of Theorem 3.

First, the simulator SIM_{TSS} used in the proof of Theorem 3 requires $\sigma = \log_g h$ as an additional private input, and its Step Step 3c (see Figure 4-18) must be modified so that SIM_{TSS} simulates the DSS-TSig-opt protocol to $\mathcal{A}$ via simulator $\mathsf{SIM}_{DSS-TSig-opt}$ by passing to that simulator as its private inputs the target signature (r^*, s^*) *and* the trapdoor value σ.

Secondly, the simulation of a run of TSS-opt via SIM_{TSS} must start not by an *execution* of Ped-IG = (DL-IG, h-IG) on behalf of the uncorrupted players, but by an execution of DL-IG and a *simulation* of h-IG via simulator $\mathsf{SIM}^{(1)}_{h-IG}$ of Figure 7-3. If $\mathsf{SIM}^{(1)}_{h-IG}$ outputs null as a private output then the simulation process is abandoned. Otherwise, if it outputs $\sigma = \log_g h$ then the simulator proceeds to simulate the remaining steps of TSS-opt via SIM_{TSS} on its additional private input σ.

Third, as in the proof of Theorem 3 we argue that an adversarial view of a successful execution of the remaining steps of TSS-opt (i.e. everything that follows the initialization protocol Ped-IG) and a view of a successful simulation of these steps via SIM_{TSS}, are identically distributed. The difference from Theorem 3 is that the above is true if and only if SIM_{TSS} has σ as a private input. The notions of a *successful* execution of the remaining steps of TSS-opt and of a *successful* simulation via SIM_{TSS} are defined similarly as in the proof of Theorem 3, but for TSS-opt instead of TSS. The proof of the identity of these views is virtually identical to the argument used in Theorem 3, except here it is implied by the "repetitive secrecy" of Mult-opt rather than Mult and by the secrecy property of Reciprocal-opt rather than Reciprocal.

The last modification is that we have to take into account the fact that the simulation of h-IG via simulator $\mathsf{SIM}^{(1)}_{h-IG}$ might fail to produce trapdoor σ as a private output. Let $p_{\mathcal{A}}$ be the polynomial s.t. for every k_0 there exists $k \geq k_0$ s.t. $\mathcal{A}$ creates a forgery in a run of TSS-opt on public input 1^k with probability at least $\frac{1}{p_{\mathcal{A}}(k)}$. It follows from the robustness property of TSS-opt that since an adversarial view of a successful run of (the remaining steps of) TSS-opt on any $(p, q, g, h), \mathsf{ah}''$ output by Ped-IG on 1^k and ah is distributed identically as in a successful simulation via SIM_{TSS} with the additional private input $\sigma = \log_g h$ then, under the discrete-log assumption, there is a negligible statistical difference between an adversarial view of a *random* run of (the remaining steps of) TSS-opt on these inputs and a *random* simulation via SIM_{TSS} with σ. In other words, it follows that for all large enough k, for all $(p, q, g, h), \mathsf{ah}''$ output by h-IG there is at most $\frac{1}{3p_{\mathcal{A}}(k)}$ probability that $\mathcal{A}$ outputs a different output in the simulation via SIM_{TSS} on $(p, q, g, h), \mathsf{ah}''$ and $\sigma = \log_g h$ then in the execution of (the remaining steps) of TSS-opt on these inputs. By Lemma 35, there is an efficient simulator $\mathsf{SIM}^{(1)}_{h-IG}$ which for every $(p, q, g), \mathsf{ah}'$ output by DL-IG simulates h-IG to $\mathcal{A}$ on these inputs and returns $\sigma = \log_g h$ as a private output with probability at least $1 - \frac{1}{3p_{\mathcal{A}}(k)}$ for all large enough k. It follows that for all large enough k, for all $(p, q, g), \mathsf{ah}'$ output by Ped-IG on $1^k, \mathsf{ah}$, there is at most $\frac{1}{3p_{\mathcal{A}}(k)} + \frac{1}{3p_{\mathcal{A}}(k)} = \frac{2}{3p_{\mathcal{A}}(k)}$ probability that $\mathcal{A}$ outputs a different output in the simulation via $\mathsf{SIM}^{(1)}_{h-IG}$ an SIM_{TSS} on $(p, q, g), \mathsf{ah}'$ then

Statically Secure $n/2$-Threshold DSS-TSig-opt, $(2t+1 \leq n)$

Input:	secret-sharing RVSS-data$[x]$ of the secret key (see Fig. 4-7), public key $y = g^x$, (hashed) message $m \in \mathbb{Z}_q$ to be signed
Adversarial Input:	adversarial history ah, adversary's output in RVSS-data$[x]$
Public Output:	(r, s), the DSS signature on message m under key y

1. Players compute RVSS-data$[k] \leftarrow$ RVSS
2. Players perform RVSS-data$[e] \leftarrow$ Reciprocal-opt(RVSS-data$[k]$) to share $e = k^{-1}$
3. Players compute $r' \leftarrow$ Exp(RVSS-data$[e], g)$ and then $r \rightarrow r' \bmod q$
4. Players compute RVSS-data$[x'] \leftarrow$ Scale(RVSS-data$[x], m, r)$
5. Players compute RVSS-data$[s] \leftarrow$ Mult-opt(RVSS-data$[x']$, RVSS-data$[k]$)
6. Players reconstruct $s \leftarrow$ RVSS-REC(RVSS-data$[s]$)

Simulator $\mathsf{SIM}_{DSS-TSig-opt}$ of DSS-TSig-opt (interacting with $\mathcal{A}$):

Let RVSS-data$[x]$ be an element of $\mathcal{RVSS\text{-}DATA}_{(p,q,g,h)}$

Public Input:	public data in RVSS-data$[x]$, public key $y^* \in G_q$, (hashed) message $m \in \mathbb{Z}_q$
$\mathsf{SIM}_{DSS-TSig-opt}$**'s Private Input:**	private outputs of all $P_i \in$ *Good* in RVSS-data$[x]$ Pederson's trapdoor value $\sigma = \log_g h$ "target output" signature $(r^*, s^*) \in (\mathbb{Z}_q \times \mathbb{Z}_q)$
$\mathcal{A}$**'s Private Input:**	adv. hist. ah, adversary's output in RVSS-data$[x]$

1. $\mathsf{SIM}_{DSS-TSig-opt}$ follows Step 1 of DSS-TSig-opt on behalf of the good players.
2. $\mathsf{SIM}_{DSS-TSig-opt}$ simulates Reciprocal-opt with simulator's $\mathsf{SIM}_{Rpcl-opt}$ of Figure 4-24 on input RVSS-data$[k]$ and $\mathsf{SIM}_{Rpcl-opt}$'s additional input σ.
3. $\mathsf{SIM}_{DSS-TSig-opt}$ computes $r'^* = g^{m/s^*}(y^*)^{r^*/s^*} \bmod p$ and simulates Exp with the simulator SIM_{Exp} of Figure 4-9 on input RVSS-data$[e]$ and g and the SIM_{Exp}'s target output r'^*. If the simulation is successful then all players compute the public output as $r^* = r'^* \bmod q$.

4.-5. $\mathsf{SIM}_{DSS-TSig-opt}$ follows Steps 4-5 of DSS-TSig-opt on behalf of the good players.

6. $\mathsf{SIM}_{DSS-TSig-opt}$ replaces the private data of the good players via transformation Inc-RVSS-data$[s^*] \leftarrow \mathcal{T}_{Mult}$(RVSS-data$[s], s^*)$ and participates in RVSS-REC on the resulting RVSS-data$[s^*]$ on behalf of the good players.

Figure 4-25: DSS-TSig-opt: $n/2$-Threshold DSS Signature Generation

in the execution of (all the steps after the initial DL-IG) of TSS-opt on these inputs. Since this holds for all output of Ped-IG, and since for all k_0 there exists $k > k_0$ s.t. $\mathcal{A}$ outputs a forgery with probability at least $\frac{1}{p_{\mathcal{A}}(k)}$, it follows that for all k_0 there exists $k > k_0$ s.t. there is at least $\frac{1}{p_{\mathcal{A}}(k)} - \frac{2}{3p_{\mathcal{A}}(k)} = \frac{1}{3p_{\mathcal{A}}(k)}$ probability that $\mathcal{A}$ outputs a forgery in the *simulation* of TSS-opt on 1^k, ah. Since a forgery in the simulation of TSS-opt must be a DSS forgery under the public key y^* given to the simulator by the oracle $\mathcal{O}_{DSS}$, this creates a reduction from $\mathcal{A}$ to a successful CMA attack on DSS, which proves the theorem. □

Theorems 6 and 5 immediately imply the following:

Theorem 7 *If the DSS signature scheme is unforgeable under the adaptive chosen message attack, then* TSS-opt = (Ped-IG+DKG, DSS-TSig-opt, Ver) *is a secure* $n/2$*-threshold signature scheme.*

Chapter 5

Adaptive Threshold Cryptosystems

In this chapter we show how to modify the efficient threshold protocols of Chapter 4 so that they become *provably* resilient against an *adaptive* adversary. In an adaptive model, the adversary can decide which server to corrupt at any moment during the operation of a threshold scheme, until the adversary exhausts the admissible threshold of corruptions. By contrast, in a static adversarial model considered in Chapter 4, the adversary must decide upon the identities of the players that will ever become corrupted before the threshold scheme starts. As we argued in the introduction, an adaptive adversary is a more realistic model of centrally-coordinated attacks facing fault-tolerant distributed systems than a static model considered in the previous Chapter.[1] Unfortunately, we do not know how to prove the protocols presented in Chapter 4 adaptively secure if they are left unmodified. It is known, however, that an adaptive adversary is strictly more powerful than a static one, as there do exist protocols which are secure in the static model but which can be shown insecure in the adaptive model [CFGN96, Can00, CDD+99].[2] Consequently, it is important to construct protocols which provably resist an adaptive adversary.

As we mentioned in the introduction, it is known how to perform general multi-party computation in the adaptive adversarial model ([BGW88, CCD88] and the subsequent results referred to in Section 1.1), but the computational cost of adaptively-secure threshold DSS or Cramer-Shoup schemes based on these general results would be higher by orders of magnitude than the cost of statically-secure threshold schemes constructed from the building blocks we give in Chapter 4.[3] In the present chapter we show that a few subtle modifications to the protocols of Chapter 4 yield protocols that can be proven secure in the adaptive adversary model. In particular, the modified protocols require only slightly more computation and communication than the statically-secure protocols of Chapter 4. This section is based on material published in [CGJ+99] and [JL00].[4]

We will argue that the fundamental building block of the threshold protocols of Chap-

[1] See the "*Scheduling of Faults*" paragraph in Section 2.2, page 26, for a further discussion of the adaptive and the static models of scheduling of faults.

[2] Interestingly, [CDD+99] exhibit a protocol which resists even computationally-unlimited static attackers but succumbs to computationally-bounded adaptive attackers.

[3] For a further discussion of efficiency comparisons see Section 1.2.

[4] Most of the material presented here is given in [CGJ+99], but we also incorporate some modifications introduced by [JL00]. See footnote 7, page 137 for further discussion.

ter 4, Pedersen's Verifiable Secret Sharing protocol PedVSS and the Distributed Coin-Flip protocol RVSS, are already secure against an adaptive adversary. The protocols that rely heavily on RVSS, namely the simultaneous proof protocol SP-THPZP, as well as the optimal-threshold multiplication Mult-opt and inverse computation Reciprocal-opt, also remain secure in the adaptive model.

The only protocol that needs to change is the threshold exponentiation protocol Exp presented in Section 4.2.5, which computes a public output $A = m^a$ from secret-sharing RVSS-data$[a]$ of a secret value a. To tolerate an adaptive adversary we modify this protocol in two ways: (1) The players use their *additive* shares a_i instead of the *polynomial* shares α_i of the secret a (see Figure 4-7); (2) To ensure that the players use their additive shares correctly they need to perform a simultaneous zero-knowledge proof of knowledge of an appropriate witness. We denote this adaptively secure version of the exponentiation protocol as Ad-Exp and we present it in detail in Section 5.2.1. The distributed key generation protocol DKG and the optimal-threshold DSS signature generation protocol DSS-TSig-opt which both use the threshold exponentiation protocol Exp as a building block do not need to change themselves. One needs only to replace the statically secure exponentiation building block Exp that they use with its adaptively secure version Ad-Exp.

Maintaining Private Channels in the Adaptive Erasure-Enabled Model. In this chapter we provide protocols which are adaptively secure under a crucial simplifying assumption that the participating players can reliably erase some of their local data. Using the terminology established in the introduction, we consider here an adaptive erasure-enabled adversarial model. In this model we can implement private channels between the players with a simple and inexpensive technique of [BH92]. Consequently, in the proofs of security of the protocols in this section we consider a secure (i.e. private and authenticated) channels communication model.

In the [BH92] implementation of private channels, initially each *pair* of players (P_i, P_j) establishes a shared secret seed s_{ij} with a key-exchange protocol. Then every message transmitted from P_i to P_j is encrypted under a new symmetric key, where both players compute the fresh key material by running a pseudorandom generator on the shared seed s_{ij}. The key, and all previous states of the pseudorandom number generator are *erased* at the time the key is used. This adds virtually no computational overhead to the protocols beyond the cost of symmetric encryption itself because every encryption method requires the use of a pseudorandom number generator as a source of randomness. Therefore we can assume private channels in this chapter. However, in Chapter 6 we remove the assumption that local erasure is available, and we show that private channels can be efficiently implemented for threshold schemes in the adaptive and erasure-free model with a novel primitive of a *simultaneously secure* encryption.

Problems with Simulating an Adaptive Adversary. Recall that we prove that our protocols have certain secrecy properties by showing that the adversary does not learn anything from them except of public inputs and outputs. Recall furthermore that our (only) technique for showing such property is to exhibit a simulator which produces the adversarial view of a protocol given only its inputs and outputs. However, it is more difficult to simulate a protocol when an adversary is adaptive. An adaptive adversary can corrupt any player at any time (as long as the adversary has not already corrupted the

allowed threshold of players), and at that point the simulator needs to be able to provide the attacker with the current internal state of the broken party. In particular, this information must be consistent with the information previously seen by the adversary. Providing this information is typically the main difficulty in proving adaptive security of protocols. We will show that the simulator of the statically-secure exponentiation protocol Exp shown in Figure 4-9, fails in this way in the adaptive setting. This does not necessarily mean that this protocol is insecure in the presence of an adaptive attacker, but it shows, at least, that current analytical techniques are insufficient to prove its adaptive security.

Here we argue the difficulty of simulating a view of the adaptive adversary with a simplified example taken from the Exp protocol of Figure 4-9. Let a triple (p, q, g) be a discrete logarithm instance. Assume that each server P_i holds a random secret value $\alpha_i \in Z_q$ such that values $\alpha_1, ..., \alpha_n$ encode, via Shamir polynomial secret sharing of degree t, a secret quantity $a \in Z_q$.[5] We want the players to compute $A = m^a \bmod p$ by having each player P_i broadcast a value $m^{\alpha_i} \bmod p$, from which A can be computed via interpolation in the exponent. Namely, for any group G of $t+1$ players, $A = m^a = \prod_{i \in G} (m^{\alpha_i})^{\lambda_i} \bmod p$, where λ_i's are the known Lagrange coefficients such that $a = \sum_{i \in G} \lambda_i \alpha_i \bmod q$. To prove that the adversary learns no other information about the value a throughout the protocol except $A = m^a$, we have to show a simulator which generates the adversary's view of this protocol given only A. Specifically, the simulator must produce (1) values $m^{\alpha_1}, ..., m^{\alpha_n}$ which agree with A; and (2) For each player P_i that the adversary decides to corrupt, value α_i which agrees with value m^{α_i} that the simulator broadcasts on behalf of P_i.

However, an adaptive adversary might not corrupt any players initially, and decide which ones to corrupt only after seeing the broadcast m^{α_i} values. He will then expect the secret exponents α_i of the corrupted servers to be consistent with the m^{α_i} values that he has just observed. Since the α_i's encode the secret a (which is unknown to the simulator) via Shamir secret sharing with a polynomial of degree t, the simulator cannot know more than t of these values. The simulation of such a protocol *is* possible in the case of a static attacker, where the simulator knows in advance the set of corrupted players. A simulator which faces a static adversary can act as follows.[6] Let $T = Bad$ be the subset of t players corrupted by the (static) adversary. Note that since the simulator controls the majority of the players, it can interpolate the secret-sharing polynomial f_a and compute each share $\alpha_i = f_a(i)$, $P_i \in T$. The simulator then computes the appropriate values m^{α_i} for $P_i \in T$, and uses these values and A to compute the remaining $n - t$ values m^{α_i} via interpolation in the exponent. We could try to simplistically extend this procedure to the adaptive model by having the simulator guess the t-element subset T of servers that will be eventually corrupted. However, the chance that the simulator guesses which players the adaptive adversary will corrupt is a negligible function of the number of participating players n.

The Single Inconsistent Player Simulation Technique. We overcome the above difficulty as follows. We design all our adaptively secure threshold protocols so that their simulators can reveal a consistent state for all simulated players except possibly for one

[5] In other words, there exists a t-degree polynomial $f(z)$ s.t. $\alpha_i = f(i) \bmod q$ for all $i \in \{1, ..., n\}$ and $a = f(0) \bmod q$. Therefore, the random values $\alpha_1, ..., \alpha_n$ are t-wise independent.

[6] This simulation procedure corresponds to the simulator SIM_{Exp} of the statically-secure exponentiation protocol Exp presented in Figure 4-9.

player, whom we will call "inconsistent". We will show that such simulators are sufficient for the proofs of security of threshold schemes. This technique is motivated by the following observations about the threshold exponentiation protocol discussed above: (1) Value $A = m^a$ can be reconstructed from the joint secret-sharing RVSS-data$[a]$ if every player publishes $A_i = m^{a_i}$ where a_i is an *additive* share of a, i.e. $\sum_{P_i \in Qual} a_i = a$ (and then $A = m^a = \prod_{P_i \in Qual} A_i$), instead of using the *polynomial* shares α_i as described above; (2) Joint secret-sharing RVSS-data$[a]$ contains as public information a Pedersen's commitment $F_{a_i}(0) = g^{a_i} h^{\hat{a}_i}$ for each additive share a_i, and thus we can enforce robustness of the above protocol with a simultaneous proof protocol in which each player proves to all others that its broadcast value m^{a_i} corresponds to such commitment. We will use a zero-knowledge proof-of-knowledge system for proving knowledge of equal representations to achieve this.

The use of additive shares in the above exponentiation protocol enables the simulator to reveal a consistent state for all simulated players except of one, whom we call "inconsistent". Let *Good* be the set of currently uncorrupted players. Note that since PedVSS is perfectly secret (Lemma 3), any subset of $|Good|$-1 additive shares of the uncorrupted players is statistically independently from the shared secret a. This allows us to simulate efficiently a protocol that outputs some value A^* without knowing $a^* = \log_g(A^*)$ as follows. The simulator can reveal the actual additive shares a_i in RVSS-data$[a]$ for all uncorrupted players $P_i \in Good$ except of one player $P_S \in Good$ whose identity is chosen at random among the currently uncorrupted players. The simulator broadcasts the corresponding values $A_i = m^{a_i}$ for $P_i \in Good \setminus \{P_S\}$, interpolates the additive shares $a_i = f_{a_i}(0)$ for $P_i \in (Bad \cap Qual)$ (the simulator holds more than t values of each secret-shared polynomial f_{a_i} in RVSS-data$[a]$), and computes $A^*_S = A^*/(\prod_{P_i \in Qual \setminus \{P_S\}} m^{a_i})$. If the adversary ever corrupts any of the players in *Good* except of P_S, the simulator will be able to reveal an internal state of that player which is consistent with value m^{a_i} the adversary has seen in the above simulation. If on the other hand the adversary corrupts player P_S then the simulation will fail as the simulator cannot know $\log_g(A^*_S)$, or otherwise the simulator would be able to compute $\log_g(A^*)$. However, because the inconsistent player was chosen at random and because regardless of the identity of P_S the adversarial view of the simulation *until the corruption of* P_S looks identical to the adversarial view of the actual protocol, the probability that the adversary corrupts the player P_S chosen by the simulator is equal to $t/n < 1/2$. Consequently, with at most $1/2$ probability the simulator will have to rewind the adversary and attempt the simulation again. Thus the simulation of the protocol succeeds after expected two trials.

Recall that we compose our protocol building-blocks into threshold signature or decryption schemes, and that the simulator for the threshold scheme is a composition of the simulators of its subprotocols. In order to compose simulators of the adaptively-secure protocols which present a perfect view of their protocols except when the adversary corrupts one inconsistent player, this player is picked at the beginning of the simulation of the whole threshold scheme. If the adversary does not corrupt this player, the simulation process passes without ever rewinding. If the adversary does corrupt this player at some point, the whole simulation of a threshold signature or decryption scheme is repeated from scratch. Because the expected number of such trials is at most two, such simulation process still establishes a reduction from the security of a threshold scheme to the security of an

underlying signature scheme or a cryptosystem.[7] We define the notion of a protocol which is simulatable in the presence of an $n/2$-threshold secure-channels adaptive adversary in the above manner as a "single-inconsistent-player-simulatable" protocol:

Definition 22 (Single-Inconsistent-Player-Simulatable Protocol)
We call a protocol Single-Inconsistent-Player-Simulatable *if there exists a non-rewinding simulator* SIM *which takes as input an identity of some player P_S, called "inconsistent", such that for every $n/2$-threshold secure-channels adaptive adversary, if P_S is uncorrupted when the simulation starts then an adversarial view of the simulation is distributed identically to the adversarial view of the execution of the protocol, until the moment the adversary corrupts player P_S (or until the end, if the adversary does not corrupt P_S).*

Note that in the above definition we do not specify the inputs of the simulator. Such inputs depend on the inputs and outputs of the simulated protocol, and we specify them for each protocol separately.

Another important feature of single-inconsistent-player-simulatable protocols, is that they are secure under parallel composition. (Compare a discussion of parallel composition of threshold protocols in the presence of a static adversary at the end of Section 4.2.4, page 71.) If a threshold scheme calls for an execution of multiple instances of some single-inconsistent-player-simulatable building-block protocol in parallel, then such execution can be simulated by running multiple instances of the simulator of that building-block protocol in parallel, each instance using the same player P_S as an inconsistent player. If P_S is not compromised, all instances are simulated successfully without rewinding. Otherwise, if P_S is compromised, then the simulation attempt fails and the simulation of the whole threshold scheme begins from scratch. However, by the same argument we used above, this happens with at most $t/n < 1/2$ probability, and thus provides an efficient reduction argument for the security of a threshold scheme.

Conventions for Simulators of Adaptive Protocols. In Section 3.2 we argued that if the adversary is static then in the security proofs one can assume the worst case that the adversary corrupts its threshold of t players before the protocol starts. As one can see from our earlier discussion, this is no longer the worst case when the adversary is adaptive. Therefore in the description of simulators in the security proofs of this section,

[7] The single inconsistent player technique was independently discovered by [CGJ+99] and [FMY99b]. However, it is here used somewhat differently than in the simulation arguments used by [CGJ+99, FMY99b]. There the inconsistent player is picked at random in the simulation of *each* subprotocol. Unlike here, the simulator then rewinds *within* a given subprotocol in case of a corruption of a chosen inconsistent player, and the independence between simulations of consecutive subprotocols is achieved by requiring the players to erase most of the local data produced during each subprotocol. The proof technique we use here is a subtle but crucial modification of these ideas proposed in the subsequent work of [JL00] and termed "persistently inconsistent player technique" to stress that unlike in the analysis used by [CGJ+99, FMY99b] here the identity of an inconsistent player stays fixed throughout the simulation of a threshold scheme.

One consequence of this analytic technique is that a simulation is never rewound. In case of eventual failure it simply begins from scratch. This one-pass simulation has a simplifying effect of eliminating a subtle issue of so-called "post-execution corruptions", which comes up in composition of secure multi-party protocols in the adaptive model, but only if the simulators used to prove the security of these protocols use local rewinding. For a discussion of the role of post-execution corruptions in the composition of adaptively-secure multi-party protocols we refer to [Can00].

by *Bad* and *Good* we denote the sets of *currently* corrupted and uncorrupted players. When an adaptive adversary corrupts some player during a simulation, the simulator reveals a current (simulated) state of that player to the adversary, and the sets *Good* and *Bad* are appropriately modified to reflect this corruption.

In the descriptions of adaptive simulation processes for our threshold protocols we do not *explicitly* describe the state of a simulated player that the simulator gives to the adversary when such player is corrupted. Instead, this state is described *implicitly* by the following conventions: (1) In the description of a simulator we use the same variable names as in the corresponding protocol. When we say that "the simulator participates in the protocol on behalf of the uncorrupted players", we mean that the simulator generates the appropriate information and stores it for every simulated player. This is also the default information that the simulator reveals to the adversary as an internal state of some simulated player that becomes corrupted. (2) In the descriptions of our simulators we sometimes use a "starred" variable notation to designate all *modifications* to the state of the simulated players performed by the simulator. We often say that the simulator "replaces the private data of the uncorrupted players" via various "replacement procedures", like $\mathcal{T}_{RVSS}$, Figure 4-8, or $\mathcal{T}_{Mult}$, Figure 4-14. In such cases we implicitly assume that if some simulated player becomes later corrupted, the simulator reveals this new state for that player.

5.1 Adaptive Security of PedVSS-based threshold protocols

In this section we will show that the fundamental building block of our threshold protocols, the Pedersen Verifiable Secret Sharing protocol PedVSS remains secure in the adaptive setting without any modifications. We will also explain why many protocols that use PedVSS as their main building block remain secure in the adaptive model as well. Specifically, we argue that the following protocols described in Chapter 4 have the same robustness and secrecy properties in the adaptive as in the static model: PedVSS, PedVSS-REC, RVSS, RVSS-REC, ZVSS, the simultaneous proof protocol, Mult-opt, and Reciprocal-opt.[8]

5.1.1 Adaptive Security of PedVSS, RVSS, and ZVSS

Robustness Property of PedVSS, RVSS, and ZVSS in the Adaptive Setting

The arguments that PedVSS, PedVSS-REC, and ZVSS are robust in the adaptive model are very similar to the ones we gave in the static setting in Lemmas 1, 2 and 9 in Section 4.2.4 and 4.3.2. The only modification to these arguments we have to make is the following. To reduce adversary's cheating in any of these protocols to computing discrete logarithms we construct an extractor that first simulates the h-IG protocol to the adversary via simulator $\mathsf{SIM}^{(2)}_{h-IG}$, Figure 7-3, page 174. This simulator needs to pick an inconsistent player P_S for whom it will not know the value $x_S = \log_g(y_S)$. (This allows the simulator to embed its input instance of a discrete-log problem into the generated Pedersen commitment value h.) However, in the adaptive setting, if the adversary eventually corrupts this chosen player

[8]Protocols Mult and Reciprocal remain secure in the adaptive setting as well, but we skip them and concentrate on their optimal-threshold versions Mult-opt and Reciprocal-opt instead.

P_S then the extractor cannot show a consistent state of that player to the adversary and the extraction process has to start from scratch. (I.e. the interaction between the extractor and the adversary is scrapped and a new extraction attempt, i.e. a new interaction of the extractor and the adversary, is executed on fresh random inputs.) By Lemma 36 in Chapter 7, the adversary's view of an interaction with $\mathsf{SIM}^{(2)}_{h-IG}$ is statistically indistinguishable from its view of h-IG until the adversary corrupts the chosen player P_S. Therefore if the extractor picks P_S at random among the uncorrupted players then the probability that the adversary corrupts player P_S is at most negligibly higher than $t/n < 1/2$. Therefore if the probability that the adversary "cheats" in PedVSS, PedVSS-REC, or ZVSS protocols is higher than negligible, then the probability that in a single run of an extraction process this adversary does not corrupt P_S *and* cheats is still higher than negligible. Since in such case the extractor can extract an answer to the input discrete-log problem, these protocols remain robust in the adaptive model.

As in the static model, the robustness of protocols RVSS and RVSS-REC in the adaptive model is argued almost identically as the robustness of PedVSS and PedVSS-REC. (Compare Lemmas 4 and 5 in Section 4.2.4.)

Secrecy Property of PedVSS in the Adaptive Setting

PedVSS retains also its secrecy property in the adaptive model. Furthermore, as in the static model, the secrecy of RVSS and ZVSS in the adaptive model follows immediately from the secrecy of PedVSS. (Compare Lemmas 6 and 10 in Sections 4.2.4 and 4.3.2.)

To express the secrecy property of PedVSS in the adaptive model we have to rephrase the statement we give for the static model, i.e. Lemma 3, page 62, as follows: (1) The statement holds for *every* player P playing the role of the dealer, not just for every *uncorrupted* player P; (2) In the static model we considered an adversarial views of a random PedVSS with P as a dealer, but here we need to consider a *random run of* PedVSS *during which* P *does not become corrupted*; (3) To make the secrecy property of PedVSS more useful in exhibiting security of threshold protocols in the adaptive setting we give the simulator more flexibility in the way it modifies the sharing PedVSS-data$[x]$ to PedVSS-data$[x^*]$ by changing the data of the uncorrupted players. Namely, the simulator can pick *any* set, denoted *Fixed*, that includes all the currently corrupted players *Bad*, but has at most t elements and does not include the dealer P, and change the secret-sharing polynomial from f_x to f^*_x while keeping it fixed for all players in *Fixed*, i.e. making $f^*_x(0) = x^*$ instead of $f_x(0) = x$ but keeping $f^*_x(i) = f_x(i)$ for all $P_i \in$ *Fixed*. We include the modified statement of this lemma below and we put the modified instructions for the $\mathcal{T}_{RVSS}$ procedure in Figure 5-1.

Lemma 24 (Adaptive Polynomial Secrecy of PedVSS)
There exists a simulator SIM *s.t. for every $n/2$-threshold adaptive secure-channels adversary $\mathcal{A}$ with history* ah*, for any distributed protocol $\mathcal{P}$, for every discrete-log instance (p, q, g), for every $h \in G_q$, for every two elements x, x^* in $\mathbb{Z}_q$, for* every *player P playing the role of the dealer, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a run of protocol* PedVSS *with dealer P* during which P is not corrupted, *on*

public input (p, q, g, h), $\mathcal{A}$*'s input* ah, *and* P*'s input* x^*, *with outputs denoted* PedVSS-data$[x^*]$

– *a run of* $\mathcal{P}$ *on input* PedVSS-data$[x^*]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *whose private inputs are* (σ, x^*), *where* $\sigma = \log_g h$

  – *a run of protocol* PedVSS *with dealer* P during which P is not corrupted, *on public input* (p, q, g, h), $\mathcal{A}$*'s input* ah, *and* P*'s input* x, *with outputs denoted* PedVSS-data$[x]$

  – *a replacement of the private data of the simulated players in* PedVSS-data$[x]$ *with the data specified by* PedVSS-data$[x^*] = \mathcal{T}_{PedVSS}$(PedVSS-data$[x], x^*, \sigma, Fixed$), where $Fixed$ is any subset of at most t players s.t. $Bad \subseteq Fixed$, *and then a run of* $\mathcal{P}$ *on* PedVSS-data$[x^*]$

PedVSS-data$[x] \rightarrow$ PedVSS-data$[x^*]$ **replacement procedure** $\mathcal{T}_{PedVSS}$

Input: (*Implicit:* identity of the dealer P)
public and private data of uncorrupted players in PedVSS-data$[x]$
target value x^*
Pedersen's trapdoor $\sigma = \log_g h$
set $Fixed$ of at most t players s.t. $Bad \subseteq Fixed$
Output: (private data of uncorrupted players in) secret-sharing PedVSS-data$[x^*]$

Assume that the dealer P has been so far uncorrupted.
Let $f_x, f_{\hat{x}}$ be its outputs in PedVSS-data$[x]$.
$\mathcal{T}_{PedVSS}$ picks random t-degree polynomials $f_x^*, f_{\hat{x}}^*$ subject to the following constraints:

$$f_x^*(i) = f_x(i) \text{ for } P_i \in Fixed \text{ and } f_x^*(0) = x^*$$
$$f_x^*(z) + \sigma f_{\hat{x}}^*(z) = f_x(z) + \sigma f_{\hat{x}}(z) \text{ (for all } z)$$

$\mathcal{T}_{PedVSS}$ forms PedVSS-data$[x^*]$ by replacing the private data of each player P_i in $Good$ with $\alpha_i^* = f_x^*(i)$ and $\hat{\alpha}_i^* = f_{\hat{x}}^*(i)$, and the data $f_x, f_{\hat{x}}$ of dealer P with $f_x^*, f_{\hat{x}}^*$.

Figure 5-1: $\mathcal{T}_{PedVSS}$: Auxiliary procedure for adaptive simulation of PedVSS

Proof Sketch: The proof is virtually identical to the proof of Lemma 3, page 62. Just like in the static setting, it still holds that for every secret-sharing polynomials $f_x, f_{\hat{x}}$ used by dealer P, for every random input $r_{\mathcal{A}}$ of the adversary s.t. set $Bad_{(f_x, f_{\hat{x}}, r_{\mathcal{A}})}$ of players corrupted at the end of an instance of PedVSS on (P's and $\mathcal{A}$'s) inputs $f_x, f_{\hat{x}}, r_{\mathcal{A}}$ does not include P, for every set $Fixed$ s.t. $BAD_{(f_x, f_{\hat{x}}, r_{\mathcal{A}})} \subseteq Fixed$ and $|Fixed| \leq t$, the adversarial view of PedVSS on these inputs is identical to its view of PedVSS on inputs $f_x^*, f_{\hat{x}}^*, r_{\mathcal{A}}$ where $f_x^*, f_{\hat{x}}^*$ are any secret-sharing polynomials s.t. $f_x^*(i) = f_x(i)$ for $P_i \in Fixed$ and $f_x^*(z) + \sigma f_{\hat{x}}^*(z) = f_x(z) + \sigma f_{\hat{x}}(z)$. In other words the outputs PedVSS-data$[x^*] = \mathcal{T}_{PedVSS}$(PedVSS-data$[x], x^*, \sigma, Fixed$) where PedVSS-data$[x]$ = PedVSS$(f_x, f_{\hat{x}}, r_{\mathcal{A}})$ are equal

to PedVSS-data$[x^*]$ = PedVSS$(f_x^*, f_{\hat{x}}^*, r_{\mathcal{A}})$. The lemma follows if we range this equality over random polynomials $f_x, f_{\hat{x}}$ and vectors $r_{\mathcal{A}}$ s.t. $P \notin Bad_{(f_x, f_{\hat{x}}, r_{\mathcal{A}})}$. □

Secrecy Property of RVSS and ZVSS in the Adaptive Setting

The adaptive secrecy of the "random" VSS protocol RVSS, Figure 4-6, follows immediately from the adaptive secrecy of PedVSS, in the same way as in the static setting. However, to express the secrecy property of RVSS in the adaptive setting we have to modify the statement given for the static model, i.e. Lemma 6, page 68. This modification corresponds to the one we made above for PedVSS, i.e. we allow the simulator to change the data of the uncorrupted players associated while keeping it fixed in some superset *Fixed* of the set of the currently corrupted players *Bad*. In other words, a statement of the *adaptive* secrecy of RVSS is just like Lemma 6 except that we let the simulator SIM replace the private data in RVSS-data$[x]$ of the simulated players with the data specified by RVSS-data$[x^*]$ = $\mathcal{T}_{RVSS}$(RVSS-data$[x], \sigma, , Fixed$) where *Fixed* is any set of at most t players which includes the currently corrupted ones. Procedure $\mathcal{T}_{RVSS}$ of Figure 4-8 should be modified accordingly:

1. $\mathcal{T}_{RVSS}$ must use set *Fixed*, passed as an input, instead of *Bad* to calculate the new data in RVSS-data$[x^*]$;

2. We require that the special player P_S used by the simulator in this recalculation is not just currently uncorrupted but outside of set *Fixed*.

The same applies to the ZVSS protocol, Section 4.3.2, which shares a "refresh" polynomial. To state its secrecy property in the adaptive setting we need to modify Lemma 10, page 81, so that the simulator SIM replaces the private data of the uncorrupted players ZVSS-data$[b^*]$ = $\mathcal{T}_{ZVSS}$(ZVSS-data$[b], \sigma, P_S, Fixed$) where *Fixed* is, as above, any superset of the set of currently corrupted players s.t. $|Fixed| \leq t$. Similarly, the replacement procedure $\mathcal{T}_{ZVSS}$ of Figure 4-12 needs to be modified to (1) use set *Fixed* in place of the set of corrupted players *Bad*; (2) require that a special player P_S is not in *Fixed*; and (3) take set *Fixed* as an input.

5.1.2 Adaptive Security of the Simultaneous Proof Protocol

Adaptive Zero-Knowledge Property of the Simultaneous Proof

The simultaneous proof protocol (Figure 4-20, page 112) has a weaker "zero-knowledge" property in the adaptive model. Namely, the simulator can present a perfect view of the protocol only up to the moment the adversary corrupts the special "inconsistent" player chosen by the simulator, and only if the simulator knows witnesses w_i for values y_i in relation $A_{p,q}$ for *all* uncorrupted players $P_i \in$ *Good except for the inconsistent player* P_S (we use the notation of Figure 4-20).[9] Recall that in the static model the simulation of any SP protocol was possible even if the simulator did not know *any* witnesses for the uncorrupted players.

[9] In fact, the adaptive simulation of the simultaneous proof protocol remains successful if the simulator knows all witnesses except for *any constant number* of the simulated players. However, in the threshold protocols we present the simulator assumes only at most one inconsistent player.

Accordingly, we need to modify the simulator algorithm SIM_{SP} of the simultaneous proof protocol (see Figure 4-20) for the adaptive setting. The modified simulator SIM_{SP} is shown in Figure 5-2 below.

Simulator SIM_{SP} of SP-THPZP (interacting with $\mathcal{A}$)

Underlying THPZP Proof System: Assume a triple of algorithms $P^{(1)}, V, P^{(2)}$ specifies a discrete-log based coin-first simulatable THPZP proof system (see Figure 4-19) for some relation $\mathcal{R}$. Let SIM_{THPZP} be the simulator that exhibits the coin-first simulatable property of this proof system (see Definition 31 in Appendix F.)

Public Input:	Pedersen commitment instance (p, q, g, h), values $y_1, ..., y_n$
SIM_{SP}**'s Input:**	Pedersen's trapdoor $\sigma = \log_g h$
	identity P_S of the inconsistent player among *Good*
	witnesses w_i for all uncorrupted players P_i except of P_S
$\mathcal{A}$**'s Private Input:**	adversarial history ah

1. For each player $P_i \in \mathit{Good} \setminus \{P_S\}$ the simulator follows the SP-ZKPK protocol, i.e. broadcasts $M_i = P^{(1)}((p, q, y_i), w_i, r_i)$ for random $r_i \in \mathbb{Z}_q$ on behalf of all $P_i \in \mathit{Good} \setminus \{P_S\}$

   For player P_S the simulator picks random $R^* \in \mathbb{Z}_q$ and runs the simulator SIM_{THPZP} on instance (p, q, g), public value y_S, and coin R^*, to get messages (M_S^*, m_S^*). SIM_{SP} then broadcasts M_S^* on behalf of player P_S

2. SIM_{SP} follows RVSS on behalf of the uncorrupted players to create secret-sharing RVSS-data$[R]$.

   Then SIM_{SP} replaces the data of the uncorrupted players as RVSS-data$[R^*] = \mathcal{T}_{RVSS}(\text{RVSS-data}[R], R^*, \sigma, \mathit{Fixed})$, where *Fixed* is the current set of corrupted players, and performs RVSS-REC on RVSS-data$[R^*]$ on behalf of the uncorrupted players as in Step 2 of SP-THPZP.

3. SIM_{SP} follows the protocol on behalf of players $P_i \in \mathit{Good} \setminus \{P_S\}$ and broadcasts values $m_i = P^{(2)}((p, q, y_i), w_i, R^*, r_i)$ on their behalf.

   For player P_S the simulator broadcasts value m_S^*.

Figure 5-2: Adaptive Simulation of the Simultaneous Proof Protocol SP-THPZP

Moreover, the "zero-knowledge" and the "witness-hiding" properties of the simultaneous proof protocol in the adaptive setting need to be stated somewhat differently than in Lemmas 18 and 19, page 113, which state these properties in the context of the static adversarial model. Since in the adaptive model the adversary can corrupt, albeit with at most 1/2 probability, the one player P_S for whom the simulator SIM_{SP} (Figure 5-2) does not have a valid witness, an adversarial view of the simulation is identical to an adversarial view of the protocol only up to the moment the adversary corrupts player P_S. In other words, the simultaneous proof protocol is single-inconsistent-player-simulatable (see Definition 22, page 137). The formal statement of this property follows (we use a different font to stress

the differences between this statement and Lemma 18):

Lemma 25 (The Simultaneous Proof Protocol is Single-Inconsistent-Player--Simulatable, i.e. "Zero-Knowledge" in the Adaptive Setting)
Let (p, q, g, h) be a Pedersen commitment instance. Let THPZP *be a discrete-log based coin-first simulatable three-rounds honest-verifier public-coin zero-knowledge proof-of-knowledge proof system for relation $\mathcal{R}$. For every* adaptive *secure-channels $n/2$-threshold adversary $\mathcal{A}$ with some adversarial history* ah*, for every vector of public values $\vec{y} = (y_1, ..., y_n)$ and witnesses $\vec{w} = (w_1, ..., w_n)$ s.t. $((p, q, y_i), w_i) \in \mathcal{R}$ for each $P_i \in Good$,* for every player P_S uncorrupted at the beginning the SP-THPZP protocol, *the following two variables have identical distribution:*

- *an adversarial view of an execution of* SP-THPZP *on public inputs $(p, q, g, h, \vec{y})$, private inputs w_i of each player $P_i \in Good$, and adversarial input* ah, up to the moment the adversary corrupts player P_S (or until the end if the adversary does not corrupt P_S)

- *an adversarial view of a simulation of* SP-THPZP *on public inputs $(p, q, g, h, \vec{y})$, simulator's* SIM_{SP} *inputs a trapdoor $\sigma = \log_g h$,* an identity of player P_S, and witnesses w_i for all players $P_i \in Good \setminus \{P_S\}$, *and adversarial input* ah, up to the moment the adversary corrupts player P_S (or until the end if the adversary does not corrupt P_S)

Proof: The proof is virtually identical to the one given in the static setting for Lemma 18. By the adaptive secrecy property of RVSS an adversarial view of Step 2 is identical in the simulation and the protocol execution. Furthermore, by the coin-first simulatability property of the THPZP proof system the simulation of this proof on behalf of player P_S is perfectly distributed unless the adversary corrupts this player. □

Adaptive Witness-Hiding Property of the Simultaneous Proof

Similarly, in the adaptive setting we need to express differently the "witness-hiding" property of the simultaneous proof stated for the static setting in Lemma 19, page 113. The point of the property is that the adversary sees no difference between two instances of the simultaneous proof protocol in which the vectors of witnesses $\vec{w} = (w_1, ..., w_n)$ and $\vec{w}' = (w'_1, ..., w'_n)$ held by the players differ on the points corresponding to the *currently* uncorrupted parties. This is not true in the adaptive setting where, if $w_i \neq w_i'$ for any P_i, then the adversary who corrupts this player sees the difference between an instance of SP on input $\vec{w}$ and an instance on $\vec{w}'$. However, if $\vec{w}$ and $\vec{w}'$ agree on the points hold corresponding to all the *eventually* corrupted parties, then the adaptive adversary can tell no difference between an execution of SP-THPZP on $\vec{w}$ and on $\vec{w}'$. This formalization of the witness-hiding property of the simultaneous proof protocol turns out to be useful in the adaptive setting, for example to argue the adaptive security of the threshold multiplication protocol in the following section.

We re-state the "witness-hiding" property of the simultaneous proof protocol modified for the adaptive setting in the lemma below. The differences from the statement for the static case in Lemma 19 are emphasized by a different font:

Lemma 26 (The Simultaneous Proof Protocol is "Witness-Hiding" in the Adaptive Setting)
Let (p, q, g, h) be a Pedersen commitment instance. Let THPZP *be a discrete-log based coin-first simulatable three-rounds honest-verifier public-coin zero-knowledge proof-of-knowledge proof system for relation $\mathcal{R}$. For every adaptive secure-channels $n/2$-threshold adversary $\mathcal{A}$ with some adversarial history* ah*, for every vector of public values $\vec{y} = (y_1, ..., y_n)$,* for every set Bad_{fin} of players that the adversary eventually corrupts, *for every two vectors of witnesses $\vec{w} = (w_1, ..., w_n)$ in W and $\vec{w}' = (w'_1, ..., w'_n)$ s.t. $((p, q, y_i), w_i) \in \mathcal{R}$ and $((p, q, y_i), w'_i) \in \mathcal{R}$ for each P_i,* and such that $w_i = w'_i$ for each $P_i \in Bad_{fin}$, *the following two adversarial views are identically distributed:*

- *an adversarial view of an execution of protocol* SP-THPZP *on public inputs $(p, q, g, h, \vec{y})$, private inputs w_i of* each uncorrupted player, *and adversarial input* ah
- *an adversarial view of an execution of protocol* SP-THPZP *on public inputs $(p, q, g, h, \vec{y})$, private inputs w'_i of* each uncorrupted player, *and adversarial input* ah

Proof: This lemma is immediately implied by the witness-hiding property of the underlying ZKPK proof system THPZP used within the SP-THPZP protocol. The distribution of the adversarial view of the protocol performed by every eventually corrupted player is the same in the two cases because the inputs of these players are the same in both cases. On the other hand, the adversarial view of the protocol performed by the players that never get corrupted is identical in the two cases as well, because for every coin R and every $y_i \in L_{\mathcal{R}}$, the distribution of messages (M_i, m_i) generated by THPZP is independent of the choice of a witness w_i. This is a witness-hiding property of the THPZP proof system, which is a straightforward consequence of its zero-knowledge property (see Definition 30 in Appendix F): If an adversarial view of a run of a THPZP proof (on some witness) is identical to a view of its simulation (where the simulator has no access to *any* witness), then the view of two instances of the proof on two different witnesses must be identical too. □

Adaptive Proof-of-Knowledge Property of the Simultaneous Proof

The robustness property of the simultaneous proof protocol (Lemma 20, page 114) also holds in the adaptive adversary model. The same statement as given in Lemma 20 applies to the adaptive model. The proof of this property in the adaptive setting is very similar to the corresponding proof in the static setting too (i.e. to the proof of Lemma 20). The only difference is that we must define the sets S_1 and S_2 (see the proof of Lemma 20 on page 114) must be defined more carefully. Namely, we define set S_1 as the set of all players P_i s.t. the probability that P_i gets corrupted at the end of the SP-THPZP protocol *and* passes this protocol, i.e. that P_i is in $Bad \cap Qual$ at the end of the SP-THPZP instance, is higher than $T_p(k) = \frac{1}{2np(k)}$ (where the probability is taken as in the static case over the random coins of the uncorrupted players in Steps 2–3 of SP-THPZP). The rest of the argument remains identical, because the threshold $\frac{1}{2np(k)}$ is set so that it can tolerate the fact that now the set S_1 can include all n players. (In the way that it was defined in the static case, S_1 could have at most t players.)

5.1.3 Adaptive Security of Threshold Multiplication Protocol Mult-opt

The optimal-threshold multiplication protocol Mult-opt (Figure 4-21, page 118) also remains secret and robust in the adaptive adversary model.

Robustness Property of Mult-opt in the Adaptive Setting

The robustness of Mult-opt in the adaptive model follows from the robustness properties of the coin-flip protocol and from the proof-of-knowledge property of the simultaneous proof protocol. As we argued above, both properties hold in the adaptive model as well as in the static model. The formal statement of robustness of Mult-opt in the adaptive setting is the same as the one we gave for the static model in Lemma 21 (and Definition 20).

However, the proof of this lemma has to be modified slightly in the adaptive model. The essence of the proof of robustness of Mult-opt (page 118), is that by the proof-of-knowledge property of the simultaneous proof, one can extract from the adversary's cheating in Mult-opt two different representations of some number in G_q in bases g, h, and thus compute $\log_g h$. If during the simulation of the h-IG protocol that precedes Mult-opt the extractor embeds an input instance of the discrete-log problem in the generated value h, he can then translate computing $\log_g h$ to computing the discrete log on this input instance. This proof goes through in the adaptive model as well as the static one with only one modification, the same one we needed to make in Section 5.1.1 to argue that protocols PedVSS, RVSS, and ZVSS remain robust in the adaptive model. Namely, and we recap this argument verbatim from Section 5.1.1, to embed an input DLog instance in the generated value h the extractor needs to simulate protocol h-IG via simulator $\mathsf{SIM}^{(2)}_{h-IG}$, Figure 7-3, page 174. This simulator needs to pick an inconsistent player P_S for whom it will not know the value $x_S = \log_g(y_S)$. However, in the adaptive setting, if the adversary eventually corrupts this chosen player P_S then the extractor cannot show a consistent state of that player to the adversary and the extraction process has to start from scratch. By Lemma 36 in Chapter 7, the adversary's view of an interaction with $\mathsf{SIM}^{(2)}_{h-IG}$ is statistically indistinguishable from its view of h-IG until the adversary corrupts the chosen player P_S. Therefore if the extractor picks P_S at random among the uncorrupted players then the probability that the adversary corrupts player P_S is at most negligibly higher than $t/n < 1/2$. Therefore if the probability that the adversary "cheats" in Mult-opt is higher than negligible, then the probability that in a single run of an extraction process this adversary does not corrupt P_S *and* cheats is still higher than negligible. Since in such case the extractor can extract an answer to the input discrete-log problem, the proof of robustness of Mult-opt goes through in the adaptive setting.

Secrecy Property of Mult-opt in the Adaptive Setting

Note that the secrecy property of the Mult-opt protocol stated for the static setting in Lemma 22, page 121, shows secrecy in the sense of "perfect simulatability" only of the Inc-RVSS-data[v] part of the joint secret-sharing RVSS-data[v] output by Mult-opt (see Figure 4-23). In other words, we show that the Mult-opt protocol leaks no information about the shared factors a and b (and their shared product v) only if the subsequent threshold

protocols which operate on the generated sharing RVSS-data$[v]$ are restricted to operate on the Inc-RVSS-data$[v]$ part of this sharing.

We noted in Section 4.4.2 that a stronger secrecy/simulatability property *can* be stated about the Mult-opt protocol. This remains true in the adaptive model, and we come back to this point in a separate paragraph below. However, we first state in the adaptive model the secrecy property of Mult-opt which corresponds to Lemma 22. As we said, the simulation of Mult-opt via the auxiliary simulation procedure $\mathcal{T}_{Mult}$ of Figure 4-23 can replace only the "final" polynomial shares $\nu_i, \hat{\nu}_i$ held by the simulated players in the secret-sharing RVSS-data$[v]$ produced by Mult-opt (and thus it can be used to simulate subsequent protocols which only take these shares as their inputs). This is because the "component" secret-sharing polynomials $f_{v_i}, f_{\hat{v}_i}$ of each player P_i are correlated with the input secret-sharing polynomials f_a, f_b, namely $f_{v_i}(0) = \lambda_i f_a(i) f_b(i)$ (i.e. "$v_i = \lambda_i \alpha_i \beta_i$"). Since an adaptive adversary can corrupt some players after the simulator replaces Inc-RVSS-data$[v]$ with Inc-RVSS-data$[v^*]$,[10] the inability of the auxiliary simulation procedure $\mathcal{T}_{Mult}$ to consistently replace other private data of all the simulated players in RVSS-data$[v]$ is a problem in the adaptive model.

There are two solutions to this problem. We present a simple solution first, and postpone a more complicated one to the separate subsection below on more general adaptive secrecy property of Mult-opt. The easy solution comes about when we notice that since we are interested (for now) only in the simulatability of protocols which take the Inc-RVSS-data$[v]$ part of the outputs of Mult-opt as their sole inputs, we can simply require the players to *erase* all other private data in RVSS-data$[v]$ at the end of the Mult-opt protocol. In this way the simulation procedure $\mathcal{T}_{Mult}$ of Figure 4-23 becomes successful in consistently modifying *all* the private data held by the uncorrupted players. In other words, we modify the Mult-opt protocol of Figure 4-21 by adding the following step at the end: Each player P_i erases its secret-sharing polynomials $f_{v_i}, f_{\hat{v}_i}$, as well as all shares $\nu_{ji}, \hat{\nu}_{ji}$ for all $P_j \in$ *Part* it received from other players. Therefore the outputs of Mult-opt consist now only of the Inc-RVSS-data$[v]$ data-structure, where the private data of each uncorrupted player P_i includes only its "final" polynomial shares $\nu_i, \hat{\nu}_i$.

The secrecy property of the so modified Mult-opt protocol can be stated as follows:

Lemma 27 (Adaptive Secrecy of $n/2$-Threshold Mult-opt)
There exists a simulator SIM*, such that for every $n/2$-threshold adaptive secure-channels adversary $\mathcal{A}$ with history* ah*, for any distributed protocol $\mathcal{P}$, for every discrete-logarithm instance (p, q, g), and for every $h \in G_q$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*
  - *a run of two* successful *instances of* RVSS *(either parallel or sequential), on public input (p, q, g, h) and adversarial input* ah*, with outputs denoted* RVSS-data$[a, b]$
  - *a* successful *run of* Mult-opt(RVSS-data$[a, b]$)*, with outputs* Inc-RVSS-data$[v]$

[10] For example, this is how we simulate protocols Reciprocal-opt and DSS-TSig-opt, both of which contain a Mult-opt $\rightarrow$ RVSS-data$[v]$; RVSS-REC(RVSS-data$[v]$) sequence. A simulator of these protocols needs to replace the private data of the simulated players in secret-sharing RVSS-data$[v]$ output by Mult-opt to "hit" some target output value v^* in the reconstruction via RVSS-REC.

– *a run of* $\mathcal{P}$ *on inputs* RVSS-data$[a, b]$, Inc-RVSS-data$[v]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input* $\sigma = \log_g h$

  – *a run of two* successful *instances of* RVSS *(either parallel or sequential, as in the corresponding step of the sequence above), on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a, b]$

  – *a* successful *run of* Mult-opt(RVSS-data$[a, b]$), *with outputs* Inc-RVSS-data$[v]$

  – *a replacement of the private data in* RVSS-data$[a, b]$, *and* Inc-RVSS-data$[v]$ *of the simulated players with*

  * RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma, Fixed_1$)
  * RVSS-data$[b^*] = \mathcal{T}_{RVSS}$(RVSS-data$[b], b^*, \sigma, Fixed_2$)
  * Inc-RVSS-data$[v^*] = \mathcal{T}_{Mult}$(Inc-RVSS-data$[v], v^*, \sigma, Fixed_3$)
    where $a^*, b^*, v^* \in \mathbb{Z}_q$ *are random subject to the constraint that* $a^*b^* = v^*$, and $Fixed_1, Fixed_2, Fixed_3$ are any sets of at most t players which contain all the currently corrupted players

  and then a run of $\mathcal{P}$ *on inputs* RVSS-data$[a^*, b^*]$, Inc-RVSS-data$[v^*]$

Proof: The proof follows the same logic as the proof of the static secrecy of Mult-opt in Lemma 22, page 121. By the adaptive secrecy of RVSS, an adversarial view of sequence

$$\begin{aligned} 2 \times \text{RVSS} &\rightarrow \text{RVSS-data}[a, b] \\ \text{Mult-opt}(\text{RVSS-data}[a, b]) &\rightarrow \text{Inc-RVSS-data}[v] \\ \mathcal{P}(\text{RVSS-data}[a, b], &\text{Inc-RVSS-data}[v]) \end{aligned}$$

is distributed as a view of

$$\begin{aligned} 2 \times \text{RVSS} &\rightarrow \text{RVSS-data}[a, b] \\ (a^*, b^*) &\leftarrow_{\$} \mathbb{Z}_q \times \mathbb{Z}_q \\ \mathcal{T}_{RVSS}(\text{RVSS-data}[a], a^*, \sigma, Fixed_1) &\rightarrow \text{RVSS-data}[a^*] \\ \mathcal{T}_{RVSS}(\text{RVSS-data}[b], b^*, \sigma, Fixed_2) &\rightarrow \text{RVSS-data}[b^*] \\ \text{Mult-opt}(\text{RVSS-data}[a^*, b^*]) &\rightarrow \text{Inc-RVSS-data}[v^*] \\ \mathcal{P}(\text{RVSS-data}[a^*, b^*], &\text{Inc-RVSS-data}[v^*]) \end{aligned}$$

where $Fixed_1$ and $Fixed_2$ are some sets of at most t players which include the players corrupted when Mult-opt begins. Because the adversary always sees if an execution of Mult-opt is successful or not, the identity holds also if we restrict the above sequences to only *successful* executions of Mult-opt. We will show that the distribution of Inc-RVSS-data$[v^*]$ output by the latter sequence of executions where $Fixed_1, Fixed_2$ are some sets of at most t players which include the players corrupted at the *end* of the Mult-opt protocol,[11] is identical to the

[11] Although no efficient simulator can guess, before an execution of Mult-opt, the identities of players an

distribution of Inc-RVSS-data$[v^*]$ output by Mult-opt(RVSS-data$[a,b]$) $\rightarrow$ RVSS-data$[v]$ followed by replacement $\mathcal{T}_{Mult}$(RVSS-data$[v], v^*, \sigma, Fixed_3$) $\rightarrow$ Inc-RVSS-data$[v^*]$, where $v^* = a^*b^*$ and $Fixed_3$ is some set of at most t players which includes all players corrupted at the end of Mult-opt.

It follows from the adaptive secrecy of PedVSS that the distribution of Inc-RVSS-data$[v^*]$ is the same in both cases. Assume for simplicity of notation that $Part = P_1, ..., P_n$ and that $Fixed_1 = Fixed_2$. Let Bad_{fin} be the set of players corrupted at the end of Mult-opt. Note that $Bad_{fin} \subseteq Fixed_1 = Fixed_2$. Let $Fixed_3$ be any set of at most t players s.t. $Bad_{fin} \subseteq Fixed_3$. In both cases the inputs of players in $Fixed_1 = Fixed_2$ are determined by their parts in RVSS-data$[a,b]$. For the players in $Part \setminus Fixed_1$, in one case they execute PedVSS on inputs $v_i^* = \lambda_i \alpha_i^* \beta_i^*$ and in the other on inputs $v_i = \lambda_i \alpha_i \beta_i$. By the adaptive secrecy of PedVSS, an adversarial view of the latter execution is identical, for each $P_i \in Part \setminus Fixed_1$, to a view of an execution of PedVSS followed by a replacement of the private data of the uncorrupted players in the generated PedVSS-data$[v_i]$ by PedVSS-data$[v_i^*]$ = $\mathcal{T}_{PedVSS}$(PedVSS-data$[v_i], v_i^* = \lambda_i \alpha_i^* \beta_i^*, \sigma, Fixed_3$). Therefore, as in the static case, the final polynomial shares $\nu_j = f_v(j) = \sum_{P_i \in Part} f_{v_i}(j)$ of each player $P_j \in Part \setminus Fixed_3$ can be replaced by the simulator with values $\nu_j^* = f_v^*(j)$ where f_v^* is a random t-degree polynomial which agrees with f_v on $P_j \in Fixed_3$, and such that $f_v^*(0) = \sum_{P_i \in Part \setminus Fixed_1} f_{v_i}^*(0) + \sum_{P_i \in Fixed_1} f_{v_i}(0) = \sum_{P_i \in Part \setminus Fixed_1} \lambda_i \alpha_i^* \beta_i^* + \sum_{P_i \in Fixed_1} \lambda_i \alpha_i \beta_i = a^*b^*$. Since this is exactly the replacement that SIM performs via the auxiliary simulation procedure $\mathcal{T}_{Mult}$, and since all the other private data related to the execution of Mult-opt are erased, it follows that the distribution of Inc-RVSS-data$[v^*]$ is the same in the two cases, and hence the lemma follows. □

More General Adaptive Secrecy Property of Mult-opt

First we note that the adaptive secrecy property expressed in the above lemma can be generalized, as in the static case, to the case of a *repetitive* execution of Mult-opt, see Lemma 13. Furthermore, we note that data erasure is not necessary to prove adaptive secrecy of the Mult-opt protocol. It merely simplifies the proof of adaptive secrecy of this protocol. Indeed, one can note that even with the data-erasure modification we introduced to Mult-opt above, an uncorrupted player P_i does not modify its data in the input secret-sharings RVSS-data$[a,b]$. Hence it still keeps α_i, β_i, from which $v_i = f_{v_i}(0)$ can be computed. Since it is precisely the relationship between v_i and RVSS-data$[a,b]$ that causes difficulty in the simulation, the erasure of the *remaining part* of each "component" secret sharing PedVSS-data$[v_i]$, i.e. the erasure by each P_i of all other coefficients of its polynomials $f_{v_i}, f_{\hat{v}_i}$, is not essential in the proof of an adaptive secrecy of Mult-opt. Moreover, if the uncorrupted players leave all the data in the generated RVSS-data$[v]$, protocol Mult-opt remains adaptively secret in a more general sense than expressed in Lemma 27 above. Namely, similarly as in the static case (we discuss this issue in detail in Section 4.4.2), Mult-opt remains secret in the sense of perfect simulatability of any subsequent protocol in which the

adaptive adversary will corrupt throughout this protocol, here we are considering an adversarial view of an interaction with a *hypothetical* simulator. We use this hypothetical interaction only to show that the adversarial view of the protocol execution is distributed identically to the view of an *actual* simulation via the efficient simulator SIM described in the statement of this lemma.

uncorrupted players P_i use, apart of their data in Inc-RVSS-data$[v]$, also the polynomial shares $\nu_{ji}, \hat{\nu}_{ji}, \nu_{ij}, \hat{\nu}_{ij}$ they either sent to or received from the *eventually corrupted* players. As we mentioned in Section 4.4.2, this includes protocols Mult-opt and Reciprocal-opt, but does not include Exp or its adaptive version Ad-Exp presented later on in this chapter.

Without erasure, simulation of the Mult-opt protocol is more complicated. The simulator SIM still simply performs the Mult-opt protocol on RVSS-data$[a, b]$ on behalf of the uncorrupted players, but once it is done, and once it replaces the private data of the uncorrupted players in RVSS-data$[a, b]$ and Inc-RVSS-data$[v]$ via procedures $\mathcal{T}_{PedVSS}$ (using sets $Fixed_1, Fixed_2$, which, for simplicity of discussion we will assume to be equal) and $\mathcal{T}_{Mult}$ (using set $Fixed_3$), as described in Lemma 27, the simulator needs also to replace the remaining private data in RVSS-data$[v]$ of each player $P_i \in Fixed_1$. It can do this as follows. It leaves intact shares $\nu_{ji}, \hat{\nu}_{ji}$ received by P_i from *other* players $P_j \in Part$, but replaces P_i's own secret-sharing polynomials $f_{v_i}, f_{\hat{v}_i}$ with random t-degree polynomials $f^*_{v_i}, f^*_{\hat{v}_i}$ subject to the constraints that they (1) agree with $f_{v_i}, f_{\hat{v}_i}$ at zero (i.e. $(f^*_{v_i}(0), f^*_{\hat{v}_i}(0)) = (f_{v_i}(0), f_{\hat{v}_i}(0)) = v_i, \hat{v}_i$, because P_i is in set $Fixed_1 = Fixed_2$, and hence its value $v^*_i = f^*_{v_i}(0)$ must be equal to $f_{v_i}(0) = \lambda_i \alpha_i \beta_i$); (2) agree with $f_{v_i}, f_{\hat{v}_i}$ on values corresponding to other players in $Fixed_1$; (3) agree with shares $\nu^*_i, \hat{\nu}^*_i$ chosen by $\mathcal{T}_{Mult}$, namely $f^*_{v_i}(i) = \nu^*_{ii} = \nu^*_i - \sum_{P_j \in Part \setminus \{P_i\}} \nu_{ji}$ and $f^*_{\hat{v}}(i) = \hat{\nu}^*_{ii} = \hat{\nu}^*_i - \sum_{P_j \in Part \setminus \{P_i\}} \hat{\nu}_{ji}$; and (4) agree also with the verification information F_{v_i}. It follows from the adaptive secrecy of PedVSS that such $f^*_{v_i}, f^*_{\hat{v}_i}$ are distributed identically to what the adversary expects to see in an actual execution of Mult-opt.

5.1.4 Adaptive Security of Inverse Computation Protocol Reciprocal-opt

The $n/2$-threshold inverse computation protocol Reciprocal-opt (Figure 4-24, page 127) remains secure in the adaptive model as well. Its robustness follows trivially from the adaptive robustness of RVSS and Mult-opt. As for the proof of secrecy, the simulation procedure of Figure 4-24 already works against the adaptive adversary. For simplicity of analysis, we assume that in the adaptive setting the Mult-opt step of Reciprocal-opt is modified by the use of data erasure as discussed in Section 5.1.3 above, and hence in Step 2 (Figure 4-24) the players compute just Inc-RVSS-data$[c]$ and not RVSS-data$[c]$. However, as we mention above, such erasure is not necessary to prove adaptive security of Reciprocal.

To state the adaptive secrecy property of Reciprocal-opt we make similar modifications as for the other threshold protocols discussed above, namely we give more versatility to the simulator in the way it replaces the private data of the uncorrupted players in RVSS-data$[a]$ and RVSS-data$[e]$, the inputs and the outputs of Reciprocal-opt. We also need to clarify that in the adaptive setting the simulator $\mathsf{SIM}_{Rpcl-opt}$ of Figure 4-24, page 127, after computing Inc-RVSS-data$[c] \leftarrow$ Mult-opt(RVSS-data$[a, b]$), modifies the data of the simulated players via procedure Inc-RVSS-data$[c^*] = \mathcal{T}_{Mult}$(Inc-RVSS-data$[c], c^*, \sigma, Bad$), i.e. it keeps fixed all the data associated to the *currently* corrupted players Bad.

Lemma 28 (Adaptive Secrecy of $n/2$-Threshold Reciprocal-opt) *There exists a simulator* SIM, *such that for every $n/2$-threshold adaptive secure-channels adversary $\mathcal{A}$ with history* ah, *for any distributed protocol $\mathcal{P}$, for every discrete-logarithm instance (p, q, g), and for every $h \in G_q$, the following two adversarial views are identically distributed:*

- *an adversarial view of the following sequence of protocol executions:*

  - *a successful run of* RVSS *on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a successful run of* Reciprocal-opt *on input* RVSS-data$[a]$, *with outputs denoted* RVSS-data$[e]$
  - *a run of* $\mathcal{P}$ *on input* RVSS-data$[a]$ *and* RVSS-data$[e]$

- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *which has a private input* $\sigma = \log_g h$*:*
  - *a successful run of* RVSS *on public input* (p, q, g, h) *and adversarial input* ah, *with outputs denoted* RVSS-data$[a]$
  - *a successful simulation of* Reciprocal-opt *via* $\mathsf{SIM}_{Rpcl-opt}$ *of Figure 4-24 (modified as discussed above) on input* RVSS-data$[a]$ *and the additional input* $\sigma = \log_g h$ *of simulator* $\mathsf{SIM}_{Rpcl-opt}$, *with outputs denoted* RVSS-data$[e]$
  - *a replacement of the private data in* RVSS-data$[a, e]$ *of the simulated players via transformations*
    * RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma, Fixed_1$)
    * RVSS-data$[e^*] = \mathcal{T}_{RVSS}$(RVSS-data$[e], e^*, \sigma, Fixed_2$)

    where a^*, e^* *are random elements in* $\mathbb{Z}_q$ *subject to the constraint that* $a^*e^* = 1$, and where $Fixed_1, Fixed_2$ are any sets of at most t players which contain all the currently corrupted players, *and then an execution of* $\mathcal{P}$ *on* RVSS-data$[a^*, e^*]$

Proof Sketch: As in the static case, the proof of secrecy of Reciprocal-opt is a straightforward application of the secrecy property of Mult-opt. As in that proof, the reconstruction RVSS-REC(Inc-RVSS-data$[c]$) $\rightarrow c$, the scaling Scale(RVSS-data$[b], 0, c^{-1}$) $\rightarrow$ RVSS-data$[e]$ (Steps 3-4 of Reciprocal-opt) and the subsequent run of $\mathcal{P}$(RVSS-data$[a, e]$) are a particular example of some protocol $\mathcal{P}'$ executing on inputs RVSS-data$[a, b]$ and Inc-RVSS-data$[c]$. Therefore by the adaptive secrecy property of Mult-opt, the successful execution of these steps is distributed as in a simulation where RVSS-data$[a, b]$ are replaced by RVSS-data$[a^*, b^*]$ generated via two $\mathcal{T}_{PedVSS}$ procedures using any sets $Fixed_1$ and $Fixed_2$ of at most t players which include all players corrupted at the end of Reciprocal-opt, while Inc-RVSS-data$[c]$ is replaced by Inc-RVSS-data$[c^*]$ generated via procedure $\mathcal{T}_{Mult}$ using as set $Fixed_3$ the set of all players corrupted just before the reconstruction protocol RVSS-REC of Step 3. Therefore Lemma 28 follows from Lemma 27. □

5.2 Adaptively Secure Key Generation and DSS Signatures

5.2.1 Adaptively Secure Threshold Exponentiation Protocol

The Distributed Key Generation protocol DKG, introduced in the context of a static adversary in Section 4.2.5, consists of a joint secret-sharing protocol RVSS which creates a sharing RVSS-data$[x]$ of a random secret key x, and a threshold exponentiation protocol Exp executed on RVSS-data$[c]$ and on element g in G_q, to produce the public key $y = g^x$. In the preceding section we argued that the RVSS protocol remains secure in the adaptive setting.

However, as we have argued in the paragraph on "Problems with Simulating an Adaptive Adversary" in the introduction to this chapter, the threshold exponentiation protocol Exp (Figure 4-9, page 73) is not known to be adaptively secure.

In Figure 5-3 we present an adaptively secure version Ad-Exp of this protocol which was proposed in [CGJ+99, FMY99b]. We will follow the terminology of the subsequent work of [JL00] where it was called an "additive exponentiation" protocol, in contrast with the statically-secure "polynomial exponentiation" protocol Exp. The difference is in how value m^a, for any element $m \in G_q$, is extracted from the joint secret-sharing RVSS-data[a]. In the "additive" method, every player P_i broadcasts value m^{a_i} where a_i is its additive share in RVSS-data[a] (see Figure 4-7 for notation), and then all players compute $m^a = \prod_{i \in Qual} m^{a_i}$. In the "polynomial" method, each player broadcasts value m^{α_i} where α_i is its polynomial share in RVSS-data[a] , and m^a is computed by interpolation of any $t+1$ of these values using Lagrange coefficients. We discussed why in the adaptive setting we need to replace the statically-secure "polynomial" exponentiation method with an above "additive" exponentiation in the "Single Inconsistent Player Technique" paragraph in the introduction to this chapter.

To guarantee robustness, each player proves, via a simultaneous proof protocol SP-THPZP-Rep that the representation of $F_{a_i}(0) = g^{a_i} h^{\hat{a}_i}$ in bases g, h is the same as the representation of $A_i = m^{a_i}$ in bases $m, 1$, which proves that P_i either produced A_i by exponentiating m to the proper value a_i, or that P_i knows some other representation $(a'_i, \hat{a}'_i)$ of $F_{a_i}(0)$ in bases g, h not equal to $(a_i, \hat{a}_i)$. However, under the discrete-log assumption the latter case happens with only negligible probability, because otherwise we could build an extractor who would use this ability of the corrupted player to efficiently compute discrete logarithms. We present the details of the "additive" exponentiation protocol in Figure 5-3.

We express the secrecy property of protocol Ad-Exp as the property of single-inconsistent-player-simulatability of this protocol.

Lemma 29 (Single-Inconsistent-Player-Simulatability of Ad-Exp)
For every $n/2$-threshold adaptive secure-channels adversary $\mathcal{A}$ with history ah, *for every discrete-log instance (p, q, g), for every $h \in G_q$, for every $m \in G_q$, and for every initially uncorrupted player P_S, the following two adversarial views are have identical distribution:*

- *an adversarial view of the following sequence of protocol executions until the adversary corrupts player P_S (or until the end if the adversary does not corrupt P_S)*
  - *a* successful *run of* RVSS *(Figure 4-6) on public input (p, q, g, h) and $\mathcal{A}$'s input* ah, *with outputs denoted* RVSS-data[a]
  - *a run of* Ad-Exp *on inputs* RVSS-data[a] *and m*
- *an adversarial view of the following simulation, where the uncorrupted players are controlled by the simulator* SIM *whose private input is $\sigma = \log_g h$, until the adversary corrupts player P_S (or until the end if the adversary does not corrupt P_S)*
  - *a* successful *run of* RVSS *on public input (p, q, g, h) and $\mathcal{A}$'s input* ah, *with outputs denoted* RVSS-data[a]

Adaptively Secure Exponentiation Protocol Ad-Exp

Public Input: element $m \in G_q$
Other Input: secret-sharing RVSS-data$[a]$ (See Fig.4-7)
Public Output: value $A = m^a$, where a is defined by RVSS-data$[a]$

1. Each $P_i \in Qual$ broadcasts $A_i = m^{a_i}$.
2. Each player P_i proves knowledge of equal representation of element A_i in bases $m, 1$ and element $F_{a_i}(0)$ in bases g, h. To do that, the players perform a simultaneous proof protocol SP-THPZP-Rep of Figure 4-20 based on the proof system THPZP-Rep for showing equality of representations (Figure F-2, Appendix F).
3. If some P_i fails, a_i and $A_i = m^{a_i}$ are reconstructed publicly via the PedVSS-REC protocol on input PedVSS-data$[a_i]$ contained in RVSS-data$[a]$.
4. Value $A = m^a = \prod_{P_i \in Qual} A_i$ is publicly computed.

Simulator SIM_{adExp} **of** Ad-Exp (interacting with $\mathcal{A}$):

Public Input: element $m \in G_q$, public data in RVSS-data$[a]$
SIM_{adExp}**'s Private Input:** private outputs of all $P_i \in Good$ in RVSS-data$[a]$
the "target output" value $A^* \in G_q$
Pedersen trapdoor value $\sigma = \log_g h$
identity P_S of the inconsistent player among *Good*
$\mathcal{A}$**'s Private Input:** adversary's output in RVSS-data$[a]$

1. SIM_{adExp} interpolates values a_i and computes $A_i = m^{a_i}$ for $P_i \in Bad \cap Qual$. For all players $P_i \in Good \setminus \{P_S\}$, SIM_{adExp} broadcasts correct values $A_i = m^{a_i}$, while for P_S it broadcasts $A_S^* = A^* / \prod_{P_i \in Qual \setminus \{P_S\}} A_i$.
2. SIM_{adExp} simulates the simultaneous proof protocol SP-THPZP-Rep of Step 2 via simulator SIM_{SP} of Figure 5-2 on SIM_{SP}'s inputs σ, P_S, and the witnesses $a_i, \hat{a}_i$ s.t. $A_i = m^{a_i}$ and $F_{a_i}(0) = g^{a_i} h^{\hat{a}_i}$ for all players in *Good* except of P_S.

3.-4. SIM_{adExp} performs Steps 3-4 on behalf of the uncorrupted players. (Note that players controlled by SIM_{adExp} cannot fail Step 2.)

Note that if the adversary does not manage to cheat then the public output value is reconstructed as A^*

Figure 5-3: Ad-Exp: Adaptively Secure Threshold Exponentiation Protocol

– *a simulation of* Ad-Exp *with simulator* SIM_{adExp} *(Figure 5-3), on* RVSS-data$[a]$ *and* m*, and on* SIM_{adExp}*'s additional private inputs the trapdoor* σ*, the identity of player* P_S*, and an element* A^* *picked at random in* G_q

Proof: It suffices to show that unless the adversary corrupts P_S, SIM_{adExp} executed on RVSS-data$[a], m, \sigma, P_S, A^*$ produces an adversarial view identical to a random run of the protocol Ad-Exp on RVSS-data$[a^*] = \mathcal{T}_{RVSS}$(RVSS-data$[a], a^*, \sigma, Fixed$) where $a^* = \log_m A^*$,

set *Fixed* includes all *eventually corrupted* players, and the $\mathcal{T}_{RVSS}$ procedure uses the same player P_S as an inconsistent player. This follows immediately from the adaptive zero-knowledge property of the simultaneous proof protocol SP-THPZP-Rep. □

Adaptive Robustness Property of Ad-Exp

To express the robustness property of protocol Ad-Exp, we need to pinpoint a notion of *successful* execution of this protocol:

Definition 23 *We call an instance of protocol* Ad-Exp *executed on inputs a* correct *joint secret-sharing* RVSS-data$[a]$ *and element* $m \in G_q$ successful *if and only if (1) Each* A_i, $P_i \in Bad$, *broadcast in that step is correctly computed, i.e. it is equal to* m^{a_i} *where* a_i *is defined as* $f_{a_i}(0)$ *interpolated from shares of the uncorrupted players in* RVSS-data$[a]$*; (2) If some player* $P_i \in Bad$ *fails in the proof of Step 2 then the subsequent reconstruction of their share* a_i *via protocol* PedVSS-REC *is successful, and thus the proper value* $A_i = m^{a_i}$ *is publicly computed.*

It follows that if Ad-Exp is successful on correct RVSS-data$[a]$ then its public output is m^a where a is defined by (the outputs of the uncorrupted players in) RVSS-data$[a]$. Given that definition, the robustness property of Ad-Exp in the adaptive setting is expressed in the same way as the robustness of protocol Exp in the static setting, i.e. as in Lemma 7.

The proof of that property is very similar to the proof of robustness of Mult-opt, another protocol which uses a simultaneous proof, in the adaptive setting. Namely, we use the same idea to construct an extractor which on input some discrete-log instance $(p, q, g, \tilde{g})$ simulates the Ped-IG protocol to embed $\tilde{g}$ in the generated value h. In the adaptive setting, to simulate Ped-IG in this way the extractor needs to pick an inconsistent player P_S among the simulated players, and hope that this player is not corrupted later on. If the adversary then has a higher than negligible chance of cheating in the Ad-Exp protocol, then by the adaptive robustness of the PedVSS-REC protocol used in Step 3 of Ad-Exp, the only chance it has of cheating is to cheat in the simultaneous proof protocol. But if the adversary does that then by the adaptive proof-of-knowledge property of the simultaneous proof, an extractor can with higher than negligible probability extract witnesses $(a'_i, \hat{a}'_i)$ from each cheating P_i. If the adversary cheats then this witness is different than the correct values $(a_i, \hat{a}_i)$ which the extractor can interpolate from RVSS-data$[a]$. But then the extractor gets two representations of $F_{a_i}(0)$ in bases g, h, which allows him to compute $\log_g h$, and then translate it into $\log_g \tilde{g}$, and thus to efficiently compute discrete logarithms. As in the argument for adaptive robustness of Mult-opt, this works only if the adversary does not corrupt the inconsistent player P_S. However the probability that the adversary corrupts this player is at most negligible different than $1/2$. Therefore if the adversary does corrupt that player the extraction can start from scratch, and this process still establishes an efficient reduction between a higher than negligible margin of adversarial cheating in this protocol and breaking of the discrete-log assumption.

5.2.2 Adaptively Secure Threshold DSS Scheme

The adaptively-secure threshold DSS signature generation Ad-DSS-TSig-opt follows the same procedure as its statically-secure counterpart DSS-TSig-opt presented for the static

model in Figure 4-25, page 131, except that Ad-DSS-TSig-opt uses the above adaptively-secure Ad-Exp protocol in place of the statically-secure Exp used by DSS-TSig-opt. Similarly, any call to the *simulator* SIM_{Exp} of Exp should be replaced with calls to the simulator SIM_{adExp} of Ad-Exp. The proof of unforgeability of the resulting threshold DSS signature scheme Ad-TSS-opt = (Ped-IG+DKG, Ad-DSS-TSig-opt, Ver), is the same as the proof we gave for Theorem 6, page 129, which stated the unforgeability of the TSS-opt scheme in the static setting. The only difference between the two arguments is that here the simulator SIM_{TSS} needs to pick an inconsistent player P_S at random among the initially uncorrupted players. If that player is never corrupted, by the adaptive secrecy of the building-block protocols used in Ad-TSS-opt, the adversarial view of a successful simulation is identical to its view of a successful protocol. If that player becomes corrupted, the whole simulation process against the threshold adversary $\mathcal{A}$ starts anew. However, since this happens with probability at most negligibly over 1/2, this procedure establishes a reduction between unforgeability of Ad-TSS-opt and unforgeability of the standard, "centralized", DSS scheme.

Chapter 6

Erasure-Free Adaptive Threshold Cryptosystems

The adaptive security of protocols of Chapter 5 was proven in the private-channels communication model. To efficiently implement such private channels in the adaptive model we resorted to the assumption that the participating players can reliably erase local data. This is a significant drawback because secure erasure of data is hard to achieve in practice.[1] In this chapter we show that in the adaptively secure threshold setting it is possible to get rid of the need of secure data erasure, and hence that efficient threshold protocols can be executed on standard hardware and operating systems. We model such computation formally as an *erasure-free* model of computation, where the adversary is effectively allowed to examine the entire computational history of a party it corrupts.

By proposing erasure-free adaptive threshold cryptosystems that efficiently compute specific arithmetic operations in the threshold setting (e.g. shared multiplication, shared exponentiation, shared computation of DSS signatures, etc), we provide the first efficient non-erasing protocols for adaptively secure computation of non-trivial multi-party functions over insecure channels. No such protocols were known before, because although general secure multi-party computation protocols are known in the adaptive erasure-free model assuming private channels [BGW88, CCD88], it is not known how to efficiently implement private channels in this model. Unlike in the static adversary setting, where private channels can be implemented with the conventional semantically secure encryption [GM84], implementing private channels in the adaptive-adversary model is a difficult task which, in general, seems to require the players to use a *non-committing encryption* [CFGN96]. In the previous chapter we relied on the efficient implementation of such encryption provided by [BH92], which works in the erasure-enabled computational model. Polynomial-time implementations of the non-committing encryption in the erasure-free model were given in [CFGN96, Bea97, DN00], but these implementations carry a vary large communication and computation overhead, from quadratic to linear in the security parameter.

In this chapter we show that the adaptively secure protocols of Chapter 5 can be proven secure in the erasure-free model if the private channels are implemented with a new encryp-

[1] See the "Assumption of Reliable Data Erasure" paragraph on page 27 in Section 2.2 for a discussion of this issue.

tion primitive which we call a *simultaneously secure* encryption.[2] A simultaneously secure encryption is a stronger notion than the semantically secure encryption. It is weaker, however, than the non-committing encryption defined by [CFGN96]. It has a property that a simulator which simulates a transmission of an encrypted message between two uncorrupted players can create a quasi-ciphertext message that looks like a ciphertext but is statistically independent from a plaintext. Furthermore, if the adversary corrupts the receiver of this quasi-ciphertext, the simulator can open it as a valid encryption of some message. The simultaneously secure encryption is weaker than the non-committing encryption in two aspects: (1) Upon the corruption of a receiver, the simulator can open such quasi-ciphertexts in only one way; and (2) All bets are off if the adversary corrupts the sender. Our implementation of this novel encryption primitive has an efficiency comparable to the ElGamal encryption [ElG85b] from which our construction is derived. Its security is based on the DDH intractability assumption. In the next chapter we discuss these issues in more detail.

6.1 Private Channels in the Adaptive Erasure-Free Model

6.1.1 General Multi-Party Protocols and Non-committing Encryption

If servers can reliably erase their local data, then we can assume secret communication between players in any adaptively-secure distributed protocol because it can be implemented in that model with an inexpensive technique due to Beaver and Haber [BH92].[3] However, if erasures are not allowed, implementing private channels for any multi-party protocol that is provably secure in the adaptive-adversary and secure-channels model is more complicated.

Here we sketch an argument why implementing private channels in the adaptive erasure-free model seems to require a "non-committing" property of encryption, and therefore cannot be realized with a conventional encryption. Suppose some multi-party protocol was proven secure against an adaptive adversary assuming that the communication between players was implemented with physically secure, and hence private, channels.[4] In the current absence of other techniques for proving security of multi-party protocols, this means that this protocol has an efficient *simulator* SIM which simulates the protocol in the secure-channels model.[5] If the private channels were implemented with conventional encryption, and if we attempted to use the above simulator SIM to construct a simulator for the resulting protocol in the insecure-channels model, then such simulation would fail. Suppose that the protocol calls for a transmission of some secret value between a pair of players, and suppose that the adaptive adversary does not corrupt these players before this transmission, but does corrupt either of them afterwards. In this case, at the time of this transmission the simulator has no knowledge of the transferred secret, but must give to the adversary a ciphertext that corresponds to this transmission. Subsequently, when the adversary corrupts either of the participating players, the simulator learns the transferred secret from the underlying

[2] It is an interesting open question whether this new encryption primitive can implement private channels for some *general* secure multi-party computation protocol.

[3] See page 134 in Chapter 5 for a description of this implementation.

[4] [BGW88, CCD88] provide just such protocols for efficient computation of any multi-party function which can be realized with a polynomial-depth circuit. See also Appendix D.

[5] See the discussion of the simulation technique in Section 2.4.

simulator SIM. However, since in general this secret can be an arbitrary value, the simulator is faced with a task of "opening" the ciphertext it provided as a valid encryption of any given plaintext, namely a plaintext chosen by the algorithm SIM. This is, however, an impossible task if a regular encryption is used to encrypt the data. Indeed, in conventional implementations of encryption, a ciphertext can serve as a commitment to the plaintext, and hence can be opened in only one way.

Canetti at al. [CFGN96] solve this problem by implementing a new type of encryption scheme called *non-committing encryption*, which has the property that the simulator can produce ciphertexts that, upon a corruption of either a sender or a receiver, can be opened by the simulator as corresponding to *any given cleartext*. However, the existing non-committing encryption constructions of [CFGN96, Bea97, DN00] are quite inefficient, because they require an overhead of either $O(k^2)$ or $O(k)$ ciphertext bits for each securely transferred plaintext bit, where k is a security parameter.[6]

Note that if the adversary is non-adaptive, then the above difficulty in the simulation vanishes. Indeed, any multi-party protocol which is non-adaptively secure in the secure-channels model can be efficiently implemented, based on some hardness assumptions, in the more realistic insecure-channels model using conventional encryption. Given that the best adaptively-secure and non-adaptively-secure multi-party protocols have comparable efficiency in the secure-channels model, the current best-known techniques of [DN00] imply that in the more realistic insecure-channels model, resistance to adaptive adversary incurs either $O(k)$ penalty in efficiency, which makes such resistance too burdensome, or, by the techniques of [BH92], it requires an assumption of reliable local erasure, which is non-trivial to satisfy in practice as well.

6.1.2 Our Threshold Protocols and Simultaneously Secure Encryption

Our efficient implementation of private channels in the adaptive erasure-free model for our threshold protocols, i.e. the adaptively-secure threshold protocols of Chapter 5, relies on two observations: (1) Our simulators "know" all the secret information of all but one "inconsistent" uncorrupted player; and (2) The only messages that need to be secretly transmitted between players are shares generated by secret-sharing protocols. We elaborate on these two observations below.

Observation (1): In the simulation of our adaptive protocols over private channels, there is only one *inconsistent player*, i.e. only one player for whom the simulator cannot present an internal state that would be consistent with the adversary's view.[7] When any of the remaining "consistent" players is corrupted, the simulator can open all the information generated by such player, including the messages sent, and this information is consistent with what the adversary expects to see. Namely, this information, together with the other information the adversary has seen in the simulation, has the same probability distribution

[6] Namely, the solution of [CFGN96] assumes one-way trapdoor permutations and carries a $O(k^2)$ overhead, [Bea97] gives a solution that makes a stronger assumption of decisional Diffie-Hellman and reduces the overhead to $O(k)$ bits, and the construction of [DN00] weakens the assumption to one-way trapdoor permutations again and maintains the $O(k)$ overhead.

[7] See the introduction to Chapter 5 for an explanation of the single inconsistent player simulation method.

as the corresponding information the adversary gains in an interaction with an actual run of a protocol.

In this chapter we will show a stronger property of the simulation of these protocols. Namely, that there is no difference between the adversary's view of the simulation and of the protocol, even if this view is extended by all the messages sent by the "consistent" players in the clear. In particular, to satisfy this property, the simulator must prepare the messages of all the "consistent" simulated players at the time these messages should be sent. Intuitively, the reason why the simulators of the adaptively-secure protocols of the previous chapter have this property is that on behalf of all but one of the simulated players our simulators follow the actual prescribed protocol of these players. More formally, the actions of all but one of the simulated players have the same probability distribution as the actions of these players in the actual random execution of the prescribed protocol.

A crucial consequence of this property of the simulation of our adaptively-secure threshold protocols is that regardless of the encryption we use, the secure-channels adversarial view extended by the encrypted messages sent by all the "consistent" players has the same distribution in the simulation and in the protocol. This observation is reflected in the requirements (2) and (3) in the Definition 25, page 163, which specifies the class of multi-party protocols for which we are able to efficiently implement private channels in the adaptive erasure-free model.

Observation (2): For the secret information generated by the single inconsistent player we need a separate argument. Here we make two observations: (2a) First note that the simulator never has to worry about the case when the adversary corrupts that player, because, as we argue in the introduction to Chapter 5 (see esp. Definition 22, page 137), it is fine if the whole simulation fails in that case.[8] (2b) Secondly, note that the general problem described in Section 6.1.1 above, of simulating a multi-party protocol for adaptive adversary in the insecure-channels model, does not appear in the simulation of most threshold protocols, because it is unusual for a threshold protocol to prescribe that some two players transmit a *true secret* between each other, i.e. a quantity whose value is unknown by the simulator unless the adversary corrupts either the sender or the receiver. Instead, in most threshold protocols, including the ones we propose, all the data that is sent privately between players are *random shares* corresponding to secret-sharing protocols performed by the senders.[9]

Therefore the task of our simulator is easier than the general case described in Section 6.1.1 on two counts. The simulator does have to simulate in an adaptive setting an

[8]Note, however, that the general problem of simulating adaptive erasure-free protocols over insecure-channels described in Section 6.1.1 above, remains if the adversary is constrained to corrupting only the *receiver* of the ciphertext. However, the subsequent observation (2b) helps us solve the case of the receiver corruptions as well.

[9]In the general secure multi-party computation in the adaptive model, the simulator algorithm has access to the private inputs of the corrupted players *at the moment* these players become corrupted. (See for example the MPC formalizations of [MR91, Can00].) By a *true secret* we refer to a variable whose distribution is unknown to the simulator *before* a corruption. The distribution is unknown to the simulator in the sense that the simulator cannot produce a value with this distribution. For example in our adaptive threshold exponentiation protocol Ad-Exp (Figure 5-3, page 152), the true secret is the secret key $x = \log_g y$ because the simulator, on input y, cannot produce a value with this distribution. On the other hand, any t shares sent by any of the players controlled by the simulator are not *true secrets*, because these shares are distributed as independent random values in $\mathbb{Z}_q$, and hence the simulator can produce such values.

adversarial view of the transmission of messages over insecure channels sent by a player whose internal state is unknown (i.e. the inconsistent player), but: (2a) The adversary can corrupt only some subset of the *receivers* of these messages, not its sender; and (2b) The ciphertexts are not encryptions of arbitrary (and unknown) secrets but of *shares* generated in a random t-threshold secret-sharing of an arbitrary (and unknown) secret. In particular, if the simulator secret-shares a random secret instead, any subset of t of the shares it generates will be distributed as in the actual protocol, and hence at the time of creating each ciphertext the simulator knows the plaintext to which it can later safely decommit when the receiver of this ciphertext gets corrupted.

Defining Simultaneous Security of Encryption

Therefore, to simulate the actions of the inconsistent player, we need an encryption scheme with the property that if the adversary sees the ciphertexts of some n plaintexts and chooses any t of them to be opened, the adversary learns nothing about the *remaining* plaintexts. We generalize this property as *simultaneous security* of an encryption scheme. In Figure 6-1 and Definition 24 below we formalize it in terms of indistinguishability of the ciphertexts that remain unopened by the adversary. The interaction between the adversary A and the encryption oracle S described in Figure 6-1 is similar to the corresponding interaction in the standard definition of indistinguishability of encryption, where S picks a *single* key pair (e, d), sends the public key e to A, who then chooses two probability distributions M_1 and M_2 over *single* k-bit plaintext strings and sends them to S. Then the oracle S picks a random bit b, picks a *single* string m according to the distribution M_b, and sends an encryption of m to A, who outputs a guess b' as to whether m was picked in M_1 or M_2. The interaction between A and S in Figure 6-1 differs from this on two counts: (1) The interaction operates on n encryption instances and messages in parallel, and (2) The adversary can ask for opening of any *legal subset* of the ciphertexts, in the sense that corresponds to corrupting the receivers of these messages, i.e. the adversary obtains their secret keys. See also the notes 1-4 below for some explanatory comments to our definition of simultaneous security.[10]

Definition 24 (Simultaneous Security of Encryption Scheme)
$E = (KeyGen, Enc, Dec)$ *is a* simultaneously secure *encryption scheme if, for any set L of legal subsets of indices $\{1, ..., n\}$ and for any probabilistic poly-time algorithm A that interacts with an algorithm S as in Figure 6-1, the probability that $b' = b$ is negligibly different from* $1/2$.
In other words, A cannot distinguish between encryptions of random elements in D_0 and random elements in D_1.

Note 1: The simultaneous security notion for encryption can be contrasted with the recent definition of selective security for *commitment schemes* of Dwork et al. [DNRS99]. Our notion of simultaneous security of encryption is dictated by its application to the multi-party protocols, but a [DNRS99]-like notion of selective security can also be considered for encryption schemes. A crucial difference between the two notions is that here each of the

[10] One can propose other notions of simultaneous security, based on semantic security or non-malleability ([GM84, DDN91]). Also other variants of the indistinguishability definition we give are possible.

An interaction between the adversary A and the oracle S, which implements a *simultaneously secure encryption scheme* E.

k is a security parameter and $n \leq poly(k)$
$E = (KeyGen, Enc, Dec)$ is a public-key encryption scheme, i.e.:
randomized alg. $KeyGen$ outputs a public key e and a secret key d on input 1^k,
randomized alg. Enc outputs a ciphertext c on input (m, e)
algorithm Dec outputs a cleartext m on input (c, d)
L is any subset of the set of all subsets of indices $\{1, ..., n\}$.
(L defines the set of *legal subsets* of indices in $\{1, ..., n\}$.)

S :	Compute n independent key pairs $(e_i, d_i) \leftarrow KeyGen_{\$}(1^k)$, for $i = 1, ..., n$
$S \longrightarrow A$:	Send $e_1, ..., e_n$
A :	Pick two probability distributions D_0, D_1 over n blocks of k-bit long plaintext strings, subject to the constraint that for all legal sets of indices $I = \{i_1, ..., i_{n'}\}$ in L, the probability distribution of $\vec{m}\|_I = (m_{i_1}, ..., m_{i_{n'}})$ is the same for $\vec{m} = (m_1, ..., m_n)$ picked in D_0 and in D_1
$S \longleftarrow A$:	Send (an efficient description of) D_0, D_1
S :	Pick a random bit b and pick n blocks of plaintext $m_1, ..., m_n$ at random according to distribution D_b. Encrypt the plaintext blocks $c_i \leftarrow Enc_{\$}(m_i, e_i)$ for $i = 1, ..., n$
$S \longrightarrow A$:	Send $c_1, ..., c_n$
A :	Pick a *legal subset* of indices $I \subseteq \{1, ..., n\}$ i.e. some I in L
$S \longleftarrow A$:	Send I
$S \longrightarrow A$:	Send the decryption key d_i for each $i \in I$
A :	Compute a single bit final output b'

Figure 6-1: Scenario for the *Simultaneous Security* of an Encryption Scheme

n ciphertexts is created under a separate instance of the encryption scheme. Similarly, by "opening" of some of the encryptions, we mean learning the corresponding secret key, and not the randomness used in the encryption. In other words, we open the data that corresponds to the selected receiver of the ciphertexts, instead of the data known to the sender. The relation between the two notions is not known.

Note 2: Note that the probability distributions D_0 and D_1 are constrained so that A can distinguish encryptions of their random elements only on the basis of the ciphertexts that remain unopened, because for any legal subset of indices I that A gets to open, the distribution of the opened plaintexts $\{\vec{m}|_I; \vec{m} \in D_0\}$ and $\{\vec{m}|_I; \vec{m} \in D_1\}$ are equal.

Note 3: Note furthermore that the oracle S in Figure 6-1 operates on behalf of both the sender and the n receivers $P_1, ..., P_n$. S runs the key generation algorithm on behalf of each of the receivers, hands the public keys to the adversary, which corresponds to the transmission of these keys from the receiver to the sender on the insecure channels, encrypts the messages and makes them public on behalf of the sender, and then lets the adversary pick any subset $I \in L$ of the receivers whose private keys the adversary will

learn. If D_0 is the probability distribution of plaintexts sent by the single inconsistent player during the simulation, while D_1 is a distribution of plaintexts send by the same player during the execution of a protocol, and if the set of legal subsets L consists of all t-element subsets of $\{1, ..., n\}$ (i.e. every element of L is the set of indices of players which can be together corrupted by a t-threshold adversary), then the above interaction between A and S corresponds to an interaction between the adaptive threshold adversary who attacks either the real or the simulated network. We formalize this point in Theorem 9 below.

Note 4: We could relax the restriction that distributions $\{m_I;\ m \in D_0\}$ and $\{m_I;\ m \in D_1\}$ are the same for every $I \in L$, and require instead that they are computationally indistinguishable. With such relaxation in the definition of simultaneous security, Theorem 9 would hold even if restriction (3) in Definition 25 was relaxed to admit computationally indistinguishable simulations. In other words, such encryption would implement private channels for a multi-party protocol whose simulator in the private-channels model produced an adversarial view that is computationally indistinguishable, rather than identical, to the adversarial view of the protocol. Furthermore, our encryption scheme E of Figure 6-1 satisfies such stronger notion of simultaneous security.

Implementing a Simultaneously Secure Encryption Scheme

Surprisingly, conventional encryption schemes are not known to be simultaneously secure. Indeed, we do not know how to argue why the adversary should not learn anything from the ciphertexts that he does not get to open. This difficulty arises from our inability to reduce an adversary that *does* learn something from such view to solving some presumably hard computational problem. To prove such reduction we would have to embed an instance of such hard problem into the ciphertexts that make up the crucial part of the adversary's view, which in our case are the unopened ciphertexts $\{c_i\}_{i \notin I}$. However, we do not know beforehand which of the n ciphertexts the adversary wants to leave unopened. We could embed the instance of the hard problem in all the ciphertexts, but then we would be in trouble trying to open them as regular encryptions if the adversary asks for the secret keys that correspond to these instances. Therefore, proving simultaneous security of encryption seems to require that we are able to embed an instance of a hard problem *and* that the resulting ciphertext looks valid if the adversary learns a secret key. This seems hard to do for conventional encryption schemes. However, below we present a slight modification of the ElGamal encryption scheme [ElG85b], which allows us to do this, and which is consequently provably simultaneously secure.

We present an efficient implementation of a simultaneously secure encryption in Figure 6-2. Its security is based on the DDH assumption. Our encryption is a simple modification of ElGamal encryption [ElG85b]. We require two generators g, h of a prime-order subgroup G_q of ${\mathbb{Z}_p}^*$, we replace a single private key $x \in \mathbb{Z}_q$ with a pair $x, y \in \mathbb{Z}_q$, the corresponding public key is then $P = g^x h^y$ instead of $P = g^x$, and the ciphertext is a triple $(g^r, h^r, P^r m)$ instead of a pair $(g^r, P^r m)$. The idea to extend ElGamal in this way is similar to the use of the DDH assumption in the work of Cramer and Shoup [CS98].

Theorem 8 *Under the Decisional Diffie-Hellman assumption, encryption E is simultaneously secure.*

Common parameters: primes p, q, s.t. $|q| = k$, the security parameter, element $g \in G_q$, random $h \in G_q$, efficient encoding of k-bit strings m as elements of G_q.
Alice's input: Message $m \in G_q$ she wants to transmit to Bob.

Bob: Picks random $x, y \in \mathbb{Z}_q$, sends $P = g^x h^y \bmod p$ to Alice
Alice: Picks random $r \in Z_q$, sends $A = g^r$, $B = h^r$, $C = P^r m$ (all mod p) to Bob
Bob: Decrypts $m = C/(A^x B^y) \bmod p$

Figure 6-2: Simultaneously Secure Encryption Scheme E

Proof: Take any PPT algorithm A which interacts with S as in Figure 6-1. The cornerstone of the proof is a construction of a "cheating encryption" algorithm E^* for which we will exhibit the following crucial properties: i) The views $A^{S(E^*,0)}$ and $A^{S(E^*,1)}$ of A interacting with S that uses E^* instead of E and fixes the value of bit b as, respectively, 0 or 1, are *exactly the same*; and ii) Under the DDH assumption, views $A^{S(E,0)}$ and $A^{S(E^*,0)}$, as well as $A^{S(E,1)}$ and $A^{S(E^*,1)}$, are indistinguishable. It will consequently follow that, under DDH, views $A^{S(E,0)}$ and $A^{S(E,1)}$ are indistinguishable, and hence the probability that $b = b'$ when A interacts with S that implements E can be only negligibly different from $1/2$.

As we noticed, S of Figure 6-1 performs on behalf of both the receiver and the sender, and hence the "cheating encryption" E^* can be an algorithm where the sender and the receiver are controlled by the same party. It proceeds as follows. The keys (x, y) and P are picked as in E, but the ciphertext (A, B, C) is formed by picking two random values $r_1 \neq r_2$ in $\mathbb{Z}_q$, and computing $A = g^{r_1}$, $B = g^{r_2}$, $C = A^x B^y m$ (all mod p). If the adversary gets the secret key (x, y), it can decrypt $m = C/(A^x B^y) \bmod p$.

First note that A's view of the interaction with S which uses E^* instead of E does not depend on the bit b, because for every index i, data $(e_i, c_i) = (P, A, B, C)$ that A observes is independent from the value of m_i. All messages are as likely, because for every $m \in G_q$ there exists a unique pair x, y s.t. $P = g^x h^y \bmod p$ and $C = A^x B^y m \bmod p$, as these two equations imply two independent linear equations on x and y modulo q. Therefore, the subset of indices I picked by A is independent from $\vec{m}$. Furthermore, since for every I in L value of $\vec{m}|_I$ has the same distribution for $\vec{m}$ picked in D_0 and for $\vec{m}$ picked in D_1, the distribution of the total A's view $(\vec{e}, \vec{c}, \vec{m}|_I)$ is also independent of whether p is picked according to D_0 or D_1. It follows that $A^{S(E^*,0)} = A^{S(E^*,1)}$.

We now show that under the DDH assumption, for both $b = 0$ and $b = 1$, views $A^{S(E,b)}$ and $A^{S(E^*,b)}$ are indistinguishable. Fix b. Assume that the difference $Prob[b \leftarrow A^{S(E,b)}] - Prob[b \leftarrow A^{S(E^*,b)}]$ is higher than negligible for random $g, h \in G_q$. We can construct an algorithm A' which breaks the DDH assumption as follows. On a DDH instance (g, u, v, w), A' sets $h = u$ and interacts with A as S would, except that instead of encryption E it follows another encryption-like procedure E^{**}. Keys x, y and P are computed as in E and E^*, but to encrypt plaintext m, A' picks at random $\alpha, \beta \in \mathbb{Z}_q$ and outputs $A = v^\alpha g^\beta$, $B = w^\alpha h^\beta$ and $C = A^x B^y m$ (all mod p). Note that if (u, v, w) is distributed as (u, g^r, u^r) for uniform $u \in G_q, r \in \mathbb{Z}_q$ then the (A, B) parts of every ciphertext seen by A are distributed as if S used E, and if (u, v, w) are distributed as (u, g^{r_1}, u^{r_2}) for uniform $u \in G_q, r_1, r_2 \in \mathbb{Z}_q$, then values (A, B), again in every ciphertext seen by A, are distributed as if S used E^*.

Consequently, if A' outputs the final output bit b' of A, then A' distinguishes between these two distributions and thus breaks the DDH assumption. □

In the threshold protocols we propose, generator h is not guaranteed to be uniformly distributed in G_q. Therefore, to prove simulatability of our threshold protocols which use encryption E (i.e. Theorem 9 below) we need a version of the above theorem which proves simultaneous security of E for a specific distribution of $h \in G_q$ that is output by the h-generation protocol used by our threshold schemes, namely the protocol in Figure 7-3, page 174.

Lemma 30 *Under the Decisional Diffie-Hellman assumption, E is a simultaneously secure encryption scheme if h is output by the h-generation protocol of Figure 7-3.*

Proof: We proceed as in the proof of the theorem above, except that in the reduction to DDH, the algorithm A', on instance $(\tilde{g}, G, \tilde{G})$ of DDH (distributed either as $\tilde{g}, g^r, \tilde{g}^r$ for uniform $\tilde{g} \in G_q, r \in \mathbb{Z}_q$ or as $\tilde{g}, g^{r_1}, \tilde{g}^{r_2}$ for uniform $\tilde{g} \in G_q, r_1, r_2 \in \mathbb{Z}_q$), picks $b \in \mathbb{Z}_q$ and follows option ii) in the simulation of the h-generation protocol of Figure 7-3 (see Chapter 7) so that $h = g^a \tilde{g}^b$ and A' knows a and b. Then A' assigns $v = G$, $w = G^a \tilde{G}^b$, and proceeds as in the proof of the theorem above. Then the value of h is distributed as in a real protocol, and the output of E^{**} looks again as either E or E^*, depending on which distribution the triple $\tilde{g}, G, \tilde{G}$ comes from. □

Note that the above proof works also if values $\tilde{g}$ in the DDH instances are not uniform in G_q, because regardless of the distribution of $\tilde{g}$, value $h = g^a \tilde{g}^b$ remains distributed as the output of the h-picking protocol if the algorithm A' in the proof above picks b at random in $\mathbb{Z}_q$.

Implementing Private Channels with Simultaneously Secure Encryption

Finally we are ready to state the requirements on the multi-party protocols for which a simultaneously secure encryption scheme can implement private channels:

Definition 25 (Insecure-Channels-Enabled Protocol)
We call a multi-party protocol Insecure-Channels-Enabled *if it is secure against the adaptive adversary in the secure-channels erasure-free model, and if its simulator algorithm* SIM *satisfies the following additional requirements:*

(1) At the beginning of the simulation SIM *randomly selects among the simulated players a constant number of players which will be called* inconsistent.[11] *If the adversary corrupts some inconsistent player then the simulation can be restarted (i.e. any single pass of the simulation is allowed to fail in the case that some inconsistent player is corrupted).*

(2) All messages sent by the simulated (both consistent and inconsistent) players are prepared by SIM *at the time of sending.*

(3) The adversarial view extended by all secret messages sent by the consistent players, is

[11] All our protocols can be simulated with at most *one* inconsistent player. Here we present a more general notion where the number of such inconsistent players is any constant.

perfect (i.e. the view of the simulation and of the protocol are identically distributed), until the moment the adversary corrupts some inconsistent player.

Note 1: Requirement (3) can be stated more formally as follows. Consider a data ensemble (view,msgs) produced by an algorithm which picks an inconsistent player at random and interacts with an adaptive adversary, where view is the adversarial view of the interaction, and msgs are all the (remaining) messages produced by the "consistent" players. We require that for all adaptive adversaries, ensemble (view,msgs) is identically distributed when the algorithm follows the simulator SIM, and when the algorithm follows the actual protocol for all simulated players.

Note 2: In some of our protocols, namely in the threshold multiplication protocol I.C.E.-Mult of Figure 6-3, there are exactly $2t + 1$ participating players, and the simulator must pick the one inconsistent player among the simulated players that belong to that group. To claim that this protocol is Secure-Channels-Enabled we have to slightly modify requirement (1) as follows. In the set-up stage, simulator SIM participates in a selection of a group of $2t + 1$ players (which will be the only participants in some later protocol Mult), and then selects at random one inconsistent player among the simulated players that belong to this group.

Theorem 9 *If a multi-party protocol is Insecure-Channels-Enabled, and if every private message transmission between two players in this protocol is implemented with an application of a fresh instance of a simultaneously secure encryption scheme to every message, then the resulting protocol is a secure multi-party protocol in the insecure-channels erasure-free model.*

Note: By "an application of a fresh instance of an encryption scheme to every message" we mean the following use of the encryption scheme. In every communication round when some messages need to be sent secretly between some players, we need an extra round in which each receiver picks a fresh and one-time instance of the encryption scheme per each k-bit long message that is to be secretly transmitted to it, and sends each public key to the appropriate sender. Then each sender encrypts each k-bit long secret message using an appropriate public key and sends the ciphertext to the receiver.

Proof: The simulator in the insecure-channels model simply follows the algorithm of the simulator SIM, except that it implements every private message transmission with a fresh instance of the simultaneously secure encryption, as described in note above. We argue that for any adaptive adversary in the insecure-channels model, the following two views are indistinguishable. His view $\mathsf{view}_{\mathsf{sim}}$ of a run of the simulation up to the moment he corrupts one of the inconsistent players; and his view $\mathsf{view}_{\mathsf{prot}}$ of a run of the actual protocol, up to the moment he corrupts the same player. Once the above indistinguishability of two views is established, it is easy to see that the protocol is efficiently simulatable in the insecure-channels model. Since by property (1) the constant number of the inconsistent players are chosen at random, and since the adversary cannot distinguish a real from a simulated run of the protocol until he corrupts one of these players, the probability that he corrupts them in the simulated protocol run can differ only negligibly from constant. Hence the expected time before a random pass of the simulation succeeds (i.e. the adversary does not

corrupt any inconsistent players throughout the simulation) is only negligibly different from a constant. Since every pass of the simulation is efficient, so is the whole process.

Now we argue that views $\mathsf{view}_{\mathsf{sim}}$ and $\mathsf{view}_{\mathsf{prot}}$ are indistinguishable. Since by property (3), the view of the secure-channels adversary and the messages sent by the consistent players are identically distributed in the simulation and in the actual protocol (up to the moment the adversary corrupts some inconsistent player), the insecure-channels adversary's view of the actions of the consistent players is the same as what the adversary expects to see in the protocol, regardless if the adversary corrupts these players or not, and regardless of what encryption is used to implement secure channels.

As for the inconsistent players, by property (2), SIM creates their messages at the time of sending, and hence can use the encryption procedure to encrypt them and hand them to the adversary. We will argue that for every communication round, the distribution of messages associated with secret message transmissions from the inconsistent players is indistinguishable between the protocol and the simulation. Let D_0 be the distribution, specified by SIM and the adversary, of n' k-bit long secret messages sent out in some communication round of the simulation by all the inconsistent players (e.g. if the protocol specifies that every pair of players exchanges two k-bit secret messages in this round, and there are three inconsistent players, then $n' = 6(n-1)$). Let D_1 be the corresponding distribution of messages of these players during a regular protocol. Let L, the set of legal subsets of $\{1, ..., n'\}$, be the set of all subsets of these messages whose cleartexts can be eventually seen by a t-threshold adaptive adversary who does not corrupt any of the inconsistent players. It follows from property (3) that for every $I \in L$, distributions $\{m_I;\ m \in D_0\}$ and $\{m_I;\ m \in D_1\}$ are the same. From the definition of simultaneous security it follows that the computationally restricted adversary who adaptively corrupts t of the n receivers of messages sent out by the inconsistent players, cannot distinguish between the protocol and the simulation. He could possibly distinguish the two if he corrupted one of the inconsistent players, i.e. one of the senders, but by property (1) of the simulation, we do not have to worry about this case. □

Efficiency Considerations. The use of the encryption procedure required by Theorem 9 is non-standard because we need a fresh instance of a public-key encryption per each transmitted message. In other words, in every communication round in which parties send to one another some secret messages, each receiver must first send to each sender as many fresh public keys as there will be transmitted messages. This does not necessarily have to double the number of communication rounds in the multi-party protocol, because these public keys can be piggy-backed on the messages exchanged in the preceding communication round. Secondly, if our encryption E is used, and if we compare it with the standard use of ElGamal encryption (i.e. a single public key used for all data sent to a given player), the total amount of work per player per $|q|$-bit message involves 7 exponentiations with E (4 by the receiver, including the creation of a public-key, and 3 by the sender), while the standard use of ElGamal takes 3 exponentiations (2 to encrypt and 1 to decrypt). As for the communication overhead, our use of E generates $4|p|$ bits of communication per one $|q|$-bit message sent, which is twice more than the standard use of ElGamal.

6.2 Adaptive Erasure-Free Threshold Protocols

Most of the adaptively-secure protocols of Chapter 5 are insecure-channels-enabled without any modifications. We argue this point in Section 6.2.1 below. The only protocol that requires modification is the threshold multiplication protocol Mult-opt of Figure 4-21, page 118. We modify this protocol to make it insecure-channels-enabled, and we present this modified version, denoted I.C.E.-Mult, in Section 6.2.2.

6.2.1 Insecure-Channels-Enabled Protocols based on RVSS and Ad-Exp

The security properties of our threshold protocols are implied by the properties of our main building block, the parallel execution of the Pedersen VSS, i.e. protocol RVSS (Figure 4-6, page 65). Below we argue that this protocol is insecure-channels-enabled:

Lemma 31 *The* RVSS *protocol is Insecure-Channels-Enabled.*

Proof: The simulator that exhibits this property is the simulator described in the Lemma that states the static security of RVSS, i.e. Lemma 6, page 68. As we discuss in Section 5.1.1, in the adaptive setting this simulator works as follows:

- SIM executes RVSS$\rightarrow$ RVSS-data$[x]$
- SIM replaces the private data of the currently uncorrupted players by applying procedure $\mathcal{T}_{RVSS}$(RVSS-data$[x], \sigma, Fixed$) (see Figure 4-8 and the discussion of the modifications needed in the adaptive setting in Section 5.1.1), where $\sigma = \log_g h$ and $Fixed$ is any set of at most t players which includes the set of currently corrupted players

Note that, as we explain in Section 5.1.1, this simulation implies the secrecy of the RVSS protocol in the adaptive secure-channels model. Secondly, this protocol is also erasure-free. Designate the uncorrupted players used by SIM in procedure $\mathcal{T}_{RVSS}$ as the "inconsistent" player $P_{\bar{S}}$. (Here this name is misleading because in the private-channels model SIM can reveal a consistent state for all the players it controls.) Then the property (1) of Definition 25 is trivially satisfied. Note that the distribution of the messages produced by all the remaining players is the same in the simulation as it is in a random execution of the RVSS protocol which produces the same outputs. Therefore requirements (2) and (3) of Definition 25 are satisfied. □

By the same argument as in the above lemma, it easily follows that the two other adaptively secure protocols based on RVSS, the "zero-sharing" protocol ZVSS, and the simultaneous proof protocol (we argue their adaptive security in Sections 5.1.1 and 5.1.2) are also insecure-channels-enabled. We argue separately that the same property is maintained by the adaptively secure threshold exponentiation protocol Ad-Exp presented in Section 5.2.1.

Lemma 32 *The* Ad-Exp *protocol is Insecure-Channels-Enabled.*

Proof: The simulator SIM_{adExp} that exhibits secrecy of this protocol in the adaptive secure-channels model is contained in Figure 5-3, page 152 (see also Lemma 29). Note that the

Ad-Exp protocol does not need erasures. Property (1) of Definition 25 is satisfied too. As in the proof of adaptive secrecy of Ad-Exp in Lemma 29 and as in the proof Lemma 31 above, the distribution of the messages produced by all the consistent players in the RVSS that precedes Ad-Exp is the same in the simulation on input A^* as it is in a random execution of RVSS which creates RVSS-data$[a^*]$ where $a^* = \log_g A^*$. Therefore requirements (2) and (3) of Definition 25 are satisfied with regards to the RVSS protocol that precedes Ad-Exp. Therefore the adversary's view of values A_i of the consistent players revealed in Step 1 is distributed as in the actual protocol too. Value A_S^* is distributed correctly because $A_S^* \cdot \prod_{P_i \in Qual \setminus \{P_S\}} A_i = A^*$. The view produced in Step 2 is correct as well because the simultaneous proof protocol is insecure-channels-enabled. Finally, Steps 3 and 4 produce no new information to the adversary. Therefore the distribution of the extended adversarial view of the Ad-Exp protocol is identical to the distribution of the extended view of its simulation. □

By the same argument as in the above Lemma 32, it easily follows that two adaptively-secure threshold protocols based on Ad-Exp, namely DKG, the Distributed Key Generation protocol for discrete logarithm-based schemes (see Section 4.2.5, page 71), and CS-DKG, the Threshold Cramer-Shoup Key Generation protocol (Figure A-1, page 193), are also insecure-channels-enabled.

6.2.2 Insecure-Channels-Enabled Threshold Multiplication

The only adaptively-secure protocol of Chapter 5 that needs to be modified to be insecure-channels-enabled is the threshold multiplication protocol Mult-opt, presented in Figure 4-21, page 118 (see also the discussion of its adaptive security in Section 5.1.3). We modify this protocol to make it insecure-channels-enabled and we present this modification, denoted I.C.E.-Mult, in Figure 6-3 below.

As in the Mult-opt protocol, the players in group $Part$[12] already hold additive shares v_i of $v = ab$. The problem faced already in the static adversary model is that these shares are not independently distributed, and thus protocol Mult-opt re-randomizes this sharing before v is reconstructed. In the static and adaptive erasure-enabled model this re-randomization is performed with the Pedersen's VSS. (See the RVSS protocol executed in Step 2 of Mult-opt.) Such execution of RVSS would not be insecure-channels-enabled because the simulator could not produce a view of all the messages produced by all except of one of the simulated players. This is because the messages sent by each uncorrupted player determine the secret-sharing polynomial this player shares, and hence determine the value v_i shared by this player. Since these values differ for at least $t+1$ simulated players in the execution of the Mult-opt and in its simulation (see the proof of adaptive secrecy of this protocol in Lemma 27), the extended view of the simulation differs from the extended view of the protocol execution.

Therefore, in the I.C.E.-Mult protocol, each player shares its contribution v_i to $v = ab$ in an *additive* manner, instead of sharing it *polynomially* with PedVSS. Namely, each

[12]For simplicity of presentation we assume that $n = 2t+1$. If $n > 2t+1$ then the threshold multiplication protocol can be carried out by some arbitrarily chosen group $Part$ of $2t+1$ players.

Threshold Multiplication I.C.E.-Mult : RVSS-data$[a]$, RVSS-data$[b]$ $\rightarrow$ ADD-data$[ab]$

Input: Sharings RVSS-data[a], RVSS-data[b], elements $g, h \in G_q$
Output: Additive sharing ADD-data[c] of $c = ab = \sum_{i=1}^{n} \lambda_i \alpha_i \beta_i \bmod q$, which consists of set of players *Part*, secret shares $c_i, \hat{c}_i$ held by each $P_i \in Part$, and public verification values $C_i = g^{c_i} h^{\hat{c}_i}$.

1. Let *Part* be an arbitrary set of $2t+1$ players that participate in this protocol. Each player P_i computes its additive share $v_i = \lambda_i \alpha_i \beta_i$ of c, picks random $\hat{v}_i$, broadcasts $V_i = g^{v_i} h^{\hat{v}_i}$, and proves that v_i committed in V_i is the product of α_i and $\lambda_i \beta_i$ in $F_a(i)$ and $(F_b(i))^{\lambda_i}$. This is done with a simultaneous proof protocol using a ZKPK system of [CD98] (Figure F-3, Appendix F). If some P_k fails this proof, it is removed from *Part*, his input values α_k, β_k are reconstructed from RVSS-data$[a, b]$ (via the reconstruction protocol Recon of Figure 4-22), his v_k is publicly computed, and all players remaining in *Part* recompute their secret shares v_i and the public verification values V_i so that $\sum_{P_i \in Part} v_i = c$.

2. The players re-randomize the resulting additive sharing of c. Every P_i picks random values $v_{ij}, \hat{v}_{ij}$ s.t. $\sum_{P_j \in Part} v_{ij} = v_i$ and $\sum_{P_j \in Part} \hat{v}_{ij} = \hat{v}_i$, broadcasts all values $V_{ij} = g^{v_{ij}} h^{\hat{v}_{ij}}$, and sends $v_{ij}, \hat{v}_{ij}$ to every other player P_j in *Part*. Each P_i tentatively computes its new additive shares $c_i = \sum_{P_j \in Part} v_{ji}$ and $\hat{c}_i = \sum_{P_j \in Part} \hat{v}_{ji}$ of c, and the public verification values are tentatively defined as $C_i = \prod_{P_j \in Part} V_{ji}$ for all $P_i \in Part$.

3. Receivers verify that $V_{ij} = g^{v_{ij}} h^{\hat{v}_{ij}}$. If P_j feels cheated it broadcasts a complaint and P_i replies by broadcasting correct $v_{ij}, \hat{v}_{ij}$. If some P_k responds incorrectly or if $V_k \neq \prod_{P_j \in Part} V_{kj}$, then P_k is removed from *Part*, his input values α_k, β_k are reconstructed as in Step 1, and secret shares $c_i, \hat{c}_i$ and the public verification values C_i are recomputed so that $\sum_{P_i \in Part} c_i = c$, and $g^{c_i} h^{\hat{c}_i} = C_i$.

4. **Reconstruction:** Each P_i broadcasts $c_i, \hat{c}_i$ s.t. $g^{c_i} h^{\hat{c}_i} = C_i$. If verification fails for some P_k, a procedure from Step 3 above is used to recompute shares $c_i, \hat{c}_i$, which are then resubmitted. When all faults are eliminated, c is computed as $\sum_{P_i \in Part} c_i$.

Simulation: (on inputs secret-sharings RVSS-data$[a]$ and RVSS-data$[b]$, and on the additional SIM's input a trapdoor $\sigma = \log_g h$, a "target" value c^* in $\mathbb{Z}_q$, and an identity of the "inconsistent" player P_S)

1-3. SIM follows the I.C.E.-Mult protocol on behalf of all uncorrupted players.

4. SIM interpolates a and b. SIM broadcasts $c_S^* = c^* - (ab - c_S)$ and $\hat{c}_S^*$ s.t. $c_S + \sigma \hat{c}_S = c_S^* + \sigma \hat{c}_S^*$ on behalf of player P_S. For all other players P_j, SIM broadcasts the correct values $c_j, \hat{c}_j$. If some reconstruction and share-recomputation procedure (Steps 1,3 or 4) is triggered (note that only the shares of the corrupted players are subject to reconstruction), SIM performs the Recon protocol on behalf of the uncorrupted players. Note that unless the adversary cheats, the public output is equal to c^*.

Figure 6-3: I.C.E.-Mult: Insecure-Channels-Enabled Threshold Multiplication

player P_i in *Part* simply sends random values v_{ij} in $\mathbb{Z}_q$, subject to the constraint that $v_i = \sum_{P_j \in Part} v_{ij}$, to each other player $P_j \in Part$. Then the new additive shares of v (which we denote also by $c = v$) are $c_i = \sum_{P_j \in Part} v_{ji}$.[13] For robustness which preserves perfect secrecy, similarly as in the Pedersen's VSS protocol, these shares are accompanied by associated random values $\hat{v}_{ij}$, and player P_i broadcasts verification values $V_{ij} = g^{v_{ij}} h^{\hat{v}_{ij}}$, for all P_j. The resulting shares c_i are then also accompanied by associated random values $\hat{c}_i = \sum_{P_j \in Part} \hat{v}_{ji}$ and the verification values $C_i = \prod_{P_j \in Part} V_{ji}$.

We denote the resulting "additive secret-sharing" data structure made of c_i's, $\hat{c}_i$'s and C_i's as ADD-data$[c]$. Such "additive sharing" of $c = ab$ can be robustly used only together with the joint secret-sharings RVSS-data$[a, b]$. Essentially, the polynomial secret-sharings of a and b serve as backups in case any player P_j in *Part* fails in the construction of ADD-data$[c]$ or misuses or withholds its c_j when ADD-data[c] is used in some later protocol (for example the insecure-channels-enabled DSS signatures I.C.E.-DSS-TSig or Cramer-Shoup decryption I.C.E.-CS-TDec, see Section 6.2.3). In case of such failure of some player P_k, its shares λ_k and β_k are reconstructed via protocol Recon in Figure 4-22. As in the Mult-opt protocol, two parallel instances of Recon reconstruct α_k and β_k, and v_k is publicly computed as $\lambda_k a_k b_k$ for the appropriate Lagrange coefficient λ_k.

When the value v_i of a cheating player is recomputed, values v_i (in Step 1) or c_i (Steps 3 and 4) of all the remaining players must also be adjusted so that $\sum_{P_i \in Part \setminus \{P_k\}} v_i = \sum_{P_i \in Part \setminus \{P_k\}} c_i = c$. For example in Step 1 of I.C.E.-Mult such recomputation can work as follows. Each $P_i \in Part$ can add $v_k/|Part|$ to its v_i, and each V_i is multiplied by $g^{v_k/|Part|}$. In Steps 3 and 4 of I.C.E.-Mult it is the $c_i, \hat{c}_i$ and C_i values that need to be recomputed, for example by each $P_i \in Part$ adding $v_{ik} + v_k/|Part|$ to its c_i and $\hat{v}_{ik}$ to its $\hat{c}_i$, and each C_i being multiplied by $g^{v_k/|Part|} V_{ik}$.

Lemma 33 *The multiplication protocol* I.C.E.-Mult *is Insecure-Channels-Enabled.*

Proof: The one inconsistent player in the simulation is player P_S. It is easily seen that all the messages of the simulated players are formed by SIM at the time of their sending. It remains to argue that the secure-channels adversarial view and the messages of the consistent players produced by SIM are distributed as in the random run of the protocol. We fix a random element $c^* \in \mathbb{Z}_q$ and argue that the following two data ensembles D are identically distributed: D_1 made of the secure-channels adversary's view and the messages sent by the consistent players during the simulation of the protocol I.C.E.-Mult as described in Figure 6-3 with value c^* input in Step 4, and D_2 made of the corresponding data produced in a random run of the protocol I.C.E.-Mult that outputs c^*.

By Lemma 31, the adversarial view and the messages of the inconsistent players during the creation of RVSS$[a]$ and RVSS$[b]$ in D_1 are distributed identically to a random run of these protocols in D_2, where a and b are picked in a different distribution (namely in D_2 values a, b are random numbers s.t. $a * b = c^*$). The two views of the simultaneous proof protocol in Step 1 are also identical by the witness-hiding property of this protocol.

[13] Such re-randomization of an additive secret-sharing structure was also used by Frankel et al. in a "2sum-to-2sum" protocol [FMY99a-b]. The slight difference is that here all $2t+1$ players participate in this protocol.

(Note that this protocol uses secret channels only to perform a coin-flip, and this coin-flip is performed in the same way in the execution and the simulation of I.C.E.-Mult.) Note also that in the case of any failures, only the data that is already known to the adversary is published during the reconstruction protocol(s) Recon.

Assume without loss of generality that $Part = \{1, ..., 2t+1\}$ and that the set of players that are eventually corrupted, denoted *Bad*, is the set $\{1, ..., t\}$. Consider values v_i, v_{ij}, c_j, $1 \leq i, j \leq 2t+1$ generated by Steps 2-4 (the argument about the corresponding values $\hat{v}_i, \hat{v}_{ij}, \hat{c}_j$ is identical). Note that v_{ii} and v_{ni}, for $i = t+1, ..., n$, are the only related values that are *excluded* from ensemble D. We argue that the remaining values, when sampled from D_1 or D_2, have the same distribution given all the information in D seen so far. Note that given that the adversary sees t shares $\alpha_1, \beta_1, ..., \alpha_t, \beta_t$, and that the pair of secrets a, b has a different distribution in D_1 and D_2, the vector of values $v_{t+1}, ..., v_n$ also has a different distribution in D_1 and D_2. If you think of the process of generating data in Mult as picking independent random values v_{ij} for $i \neq j$ and then determining values v_{ii} from them and from v_i's, then values v_{ij}, for $i \neq j$, have the same distributions (uniform and independent) in both D's. Values v_{ii}, and hence c_i, for $i = 1, .., t$, have the same distribution in D_1 and D_2 because values $v_1, .., v_t$ have the same distribution in D_1 and D_2. Values c_j, for $j = t+1, ..., n-1$, are each determined as $v_j + \sum_{i \neq j}(v_{ij} - v_{ji})$, but since each v_{nj}, $j \neq n$ is an independent random value which is not included in D, these c_j's are also distributed as independent random values. Finally, value c_n is in both D_1 and D_2 distributed identically, as $v - \sum_{j \neq n} c_j$. Thus D_1 and D_2 are identically distributed. □

6.2.3 Remaining Insecure-Channels-Enabled Protocols

The remaining adaptively-secure threshold protocols we discuss in Chapter 5, as well as the additional protocols of a threshold Cramer-Shoup cryptosystem protocol (Appendix A) become insecure-channels-enabled once the Mult-opt building block is replaced with the insecure-channels-enabled I.C.E.-Mult protocol of Figure 6-3. Thus, by modifying in this way the threshold inverse computation protocol Reciprocal-opt (Figure 4-24, page 127), we obtain its insecure-channels-enabled version I.C.E.-Reciprocal. Similarly, by substituting I.C.E.-Mult for Mult-opt in the adaptively-secure threshold DSS signature generation protocol Ad-DSS-TSig (Figure 4-25, page 131), or in the adaptively-secure threshold Cramer-Shoup decryption protocol CS-TDec (Figure A-2, page 194), we obtain their insecure-channels-enabled versions I.C.E.-DSS-TSig and I.C.E.-CS-TDec.

Chapter 7

Distributed Generation of a Pedersen Commitment Instance

Our threshold protocols rely heavily on a discrete-log based trapdoor commitment scheme due to Pedersen [Ped91a] which we described in Section 3.1. All the discrete–log based threshold schemes we discuss start by executing a threshold protocol for generating an instance of the Pedersen trapdoor commitment scheme. This protocol, denoted Ped-IG and presented it in Figure 7-1, starts by invoking protocol DL-IG, Figure 7-2, which on input a security parameter picks an instance (p, q, g) of the discrete-log problem. Secondly, protocol Ped-IG invokes protocol h-IG, Figure 7-3, which allows the players to jointly generate the fourth element h needed to establish the Pedersen's commitment instance. We note that these protocols need to be executed only once, during the set-up of any threshold scheme designed using the tools provided in this thesis. In particular, the speed of these set-up protocols is not a primary concern in practice.

This chapter is based on material previously published in [JL00].

Protocol Ped-IG, $(2t + 1 \leq n)$

Public Input: security parameter k, encoded in unary
Public Output: values (p, q, g, h) where (p, q, g) is a DLog instance and $h \in \mathbb{Z}_q$

1. Players generate DLog instance (p, q, g) by running DL-IG (Figure 7-2) on 1^k
2. Players generate an element $h \in G_q$ by running h-IG (Figure 7-3) on (p, q, g)

Figure 7-1: Ped-IG: Distributed Generation of Pedersen Commitment Instance

Distributed Generation of Discrete-Log Instance (p, q, g)

As discussed in the introduction, the distributed DLog instance generation protocol DL-IG relies on a verifiable discrete-log instance generator $IG = (G, V)$ (see Definition 6). On

input a security parameter k, some arbitrarily chosen participating player broadcasts values $(p, q, g, \mathsf{proof}) = G(1^k)$, and each other player verifies this value using procedure $V(p, q, g, \mathsf{proof})$. (Alternatively, the player who picks the DLog instance can broadcast also the randomness he used in its execution of G, which allows all other players to verify the output by re-running G.) If the verification does not work, some other player is asked to generate a DLog instance, until the verification passes. Note that all honest players have the same view of the DLog instance which is output by this protocol.

Protocol DL-IG, $(2t + 1 \leq n)$

Assume $IG = (G, V)$ is a verifiable discrete-log instance generator (see Definition 6)

Public Input: security parameter k, encoded in unary
Public Output: DLog instance (p, q, g)

1. An arbitrarily chosen player executes algorithm G on input 1^k and broadcasts its output $(p, q, g, \mathsf{proof})$
2. Each player locally verifies if $V(p, q, g, \mathsf{proof})$. If the verification fails, then it broadcasts a complaint. If more than t players broadcast a complaint, another player is chosen and Step 1 is run again. The first (p, q, g) instance that is agreed upon by a majority of players defines the public output of the protocol.

Figure 7-2: DL-IG: Distributed Generation of the Discrete-Log Instance

Distributed Generation of Pedersen Commitment value h

For the proofs of security of the threshold schemes we propose in this thesis, we need the protocol h-IG which given the DLog instance (p, q, g) generates an additional element h in G_q to have the following properties:

1. It can be simulated so that the simulator can learn the trapdoor $\log_g h$ of the chosen commitment
2. It can also be simulated so that the simulator can embed another instance of the discrete logarithm problem into the generated commitment.
3. It does not assume secret channels between the players

Note that we do not require (and indeed do not achieve) that the generated h is uniformly distributed in G_q.

In Figure 7-3 we present such protocol h-IG and we include the two simulation procedures. The h-IG protocol is straightforward. Every player P_i picks x_i at random, broadcasts $y_i = g^{x_i}$, proves to all others that it knows $x_i = \log_g(y_i)$ via an appropriate ZKPK proof system, and $h = \prod_{P_i \in Qual} y_i$, where $Qual$ is a set of players which pass the ZKPK proof. The ZKPK proof system employed by this protocol must preserve the zero-knowledge and

the proof-of-knowledge properties under parallel composition. We present such proof system in Appendix G.

Simulator $\mathsf{SIM}^{(1)}_{h-IG}$ which shows property 1 above is used for proving secrecy of the DLog-based threshold protocols we present in this thesis. All such protocols take the generated Pedersen's commitment instance (p, q, g, h) as a public input. The knowledge of the trapdoor $\sigma = \log_g h$ of the jointly generated commitment instance allows the simulator of the subsequent threshold protocols to open the commitments of the players it controls in the way it chooses, which leads to efficient simulation of these protocols.

Similarly, simulator $\mathsf{SIM}^{(2)}_{h-IG}$ which shows property 2 above is used to prove robustness of the subsequent threshold protocols. It is an invariant in the threshold protocols we propose in this thesis that there exists an efficient extractor which can play the part of the uncorrupted players of a given protocol, and if an adversary manages to cheat, the extractor can translate such cheating into computing $\log_g h$. If that extractor has previously followed $\mathsf{SIM}^{(2)}_{h-IG}$ to simulate the initial h-IG protocol, then it can embedded an instance of the discrete logarithm problem $(p, q, g, \tilde{g})$ into the generated h, by causing the (simulated) protocol to output $h = g^a \tilde{g}^b$ for known values a, b. If the adversary cheats in the subsequent protocols, the extractor can extract $\log_g h$, and thus compute $\log_g \tilde{g} = (\log_g h - a)/b$.

Alternative Approaches to Distributed Generation of Pedersen Commitment

In the static model and the adaptive but secure-channels model (i.e. the models we consider in Chapters 4 and 5), a random element $h \in G_q$ can be efficiently obtained via collective coin flipping protocols of general multi-party computation [BGW88, CDD+99]. Namely, each player verifiably shares a random secret, the random coin r is the sum of all the shared secrets which is publicly reconstructed, and h is set as $r^{(p-1)/q}$ (assuming that q^2 does not divide $p-1$). Such method was used in statically secure DKG protocol of [GJKR99] and in the adaptive threshold protocols of [CGJ+99], which assumed erasure (and hence could implement secure channels efficiently as in [BH92], see page 134). However, since each VSS protocol requires secret channels, such protocol will not work well in the adaptive erasure-free model of Chapter 6, where secure channels would need to be implemented with computationally intensive general non-committing encryption [CFGN96, Bea97] and incur at least $O(k)$ communication blow-up. (We explain the problem of implementing secure channels in the adaptive erasure-free model in Section 6.1.2). Therefore the protocol h-IG we chose implements h-generation directly, i.e. without using secret channels, with a parallelizable zero-knowledge proof of knowledge of discrete logarithm ZKPK-DL described in Appendix G. Similar protocol for generating h was also proposed by Frankel et al. [FMY99b, FMY99a], but they use general witness-hiding ZKPK protocols, while the known ZKPK we use has a modest advantage of taking only 5 rounds.

Lemma 34 (Robustness of DL-IG, h-IG, and hence Ped-IG) *For every $2t+1 \geq n$, in the presence of any t-threshold (static or adaptive) adversary, protocol DL-IG outputs a DLog instance (p, q, g) of security k in some discrete-log instance family DL, while protocol h-IG executed on input a DLog instance (p, q, g) outputs an element $h \in G_q$.*

Proof: DL-IG outputs (p, q, g) only if it is verified as a DLog instance by at least one honest player. h-IG outputs $h \in G_q$ because for each $P_i \in Qual$, all honest players verify

Protocol h-IG, $(2t+1 \leq n)$

Public Input: DLog instance (p, q, g)
Public Output: value $h \in G_q$

1. Each P_i generates random $x_i \in \mathbb{Z}_q$ and broadcasts $y_i = g^{x_i}$. Each player P_i s.t. $y_i^q \neq 1 \bmod p$ is disqualified from the rest of the protocol, i.e. removed from the set $Qual = \{P_1, ..., P_n\}$.
2. For each $(P_i, P_j) \in Qual$, P_i proves to P_j the knowledge of $\log_g(y_i)$ via protocol ZKPK-DL (Fig. G-1). These proofs proceed *in parallel*, as shown in Fig. 7-4.
3. Players broadcast their judgments about which other players pass the ZK proof. Each player accused of not passing the ZK proof by more than t other players is removed from $Qual$. Each player then sets a public output as $h = \prod_{P_j \in Qual} y_j$.

Simulation (option i): $\mathsf{SIM}^{(1)}_{h-IG}$ interacting with $\mathcal{A}$

Public Input: DLog instance (p, q, g)
$\mathcal{A}$'s Input: any auxiliary information z of length polynomial in $|q|$
$\mathsf{SIM}^{(1)}_{h-IG}$'s Private Output: $\sigma = \log_g h$, where h is output by (simulated) h-IG, or null

$\mathsf{SIM}^{(1)}_{h-IG}$ follows the protocol on behalf of the uncorrupted players. When $\mathsf{SIM}^{(1)}_{h-IG}$ reaches the end, it takes a group $G = Bad \cap Qual$ of corrupted players which are not disqualified. $\mathsf{SIM}^{(1)}_{h-IG}$ then runs another copy of the algorithm $\mathcal{A}$ (on the same random input) on the side, and attempts to extract the values $x_i = \log_g(y_i)$ for all $P_i \in G$. (The details of this extraction are in the proof of Lemma 35.) If it succeeds then $\mathsf{SIM}^{(1)}_{h-IG}$ outputs $\sigma = \log_g h = \sum_{P_i \in G} x_i + \sum_{P_i \in Good} x_i$, otherwise $\mathsf{SIM}^{(1)}_{h-IG}$ outputs null.

Simulation (option ii): $\mathsf{SIM}^{(2)}_{h-IG}$ interacting with $\mathcal{A}$

Public Input: DLog instance (p, q, g)
$\mathsf{SIM}^{(2)}_{h-IG}$'s Private Input: element $\tilde{g} \in G_q$
$\mathcal{A}$'s Input: any auxiliary information z of length polynomial in $|q|$
$\mathsf{SIM}^{(2)}_{h-IG}$'s Private Output: elements (a, b) in $\mathbb{Z}_q$, s.t. $h = g^a \tilde{g}^b$, where h is output by (simulated) run of h-IG, or null

$\mathsf{SIM}^{(2)}_{h-IG}$ follows h-IG on behalf of the uncorrupted players in Steps 1-2, except that for one "inconsistent" player P_S, it broadcasts $y_S = \tilde{g}^b$ for a random $b \in \mathbb{Z}_q$. $\mathsf{SIM}^{(2)}_{h-IG}$ then simulates the ZKPK-DL proofs between P_S and each $P_i \in Bad$. (See Lemma 36.) If this simulation fails, $\mathsf{SIM}^{(2)}_{h-IG}$ *fails*. Otherwise it proceeds to Step 3 and performs it on behalf of the uncorrupted players. Finally, it attempts to extract the values $x_i = \log_g(y_i)$ for all players P_i in $G = Bad \cap Qual$. (This extraction process is identical as in $\mathsf{SIM}^{(1)}_{h-IG}$.) If the extraction is successful, $\mathsf{SIM}^{(2)}_{h-IG}$ outputs the representation $(\sum_{P_j \in Qual \setminus \{P_S\}} x_j, b)$ of the generated h in bases $g, \tilde{g}$. Otherwise $\mathsf{SIM}^{(2)}_{h-IG}$ outputs null.

Figure 7-3: h-IG: Dist. Generation of value h in the Pedersen Commitment

The following is a ZKPK-DL proof of Figure G-1 performed in parallel by each pair (P_i, P_j) of servers in *Qual*, where P_i plays the role of the prover and P_j the role of the verifier. The protocol is performed over authenticated point-to-point links, with no recourse to broadcast, and no need for secrecy on the links.

Public Inputs: discrete-logarithm instance (p, q, g), values $y_1, ..., y_n \in G_q$
P_i's Private Input: $x_i \in \mathbb{Z}_q$ such that $y_i = g^{x_i} \bmod p$.

1. Each P_i picks a trapdoor value $\alpha_{ij} \in \mathbb{Z}_q$ and sends $h_{ij} = g^{\alpha_{ij}}$ to P_j, for every P_j
2. Each P_j chooses random coins $R_{ij}, \hat{R}_{ij}$ in $\mathbb{Z}_q$ and sends a commitment $C_{ij} = g^{R_{ij}} h_{ij}^{\hat{R}_{ij}}$ to P_i, for every P_i
3. Each P_i chooses a random r_{ij} in $\mathbb{Z}_q$ and sends $M_{ij} = g^{r_{ij}}$ to P_j, for every P_j
4. Each P_j opens its commitments C_{ij} by sending $(R_{ij}, \hat{R}_{ij})$ to P_i, for every P_i
5. Each P_i verifies that $g^{R_{ij}} h_{ij}^{\hat{R}_{ij}} = C_{ij}$, for every P_j. If so, then it sends $m_{ij} = r_{ij} + x_i R_{ij}$ and α_{ij} to P_j.
6. P_j *accepts the proof* of P_i if $g^{\alpha_{ij}} = h_{ij}$ and $g^{m_{ij}} = y_i^{R_{ij}} M_{ij}$. Otherwise we say that "P_j believes that P_i does not pass the ZK proof."

Figure 7-4: Step 2 of h-IG: Parallel Execution of $n \times n$ ZKPK-DL proofs

that $y_i \in G_q$ by checking if $y_i^q = 1 \bmod p$. □

Lemma 35 (Simulation of h-IG, Option i) *For every polynomial $p(k)$ there exists a (PPT TM) simulator $\mathsf{SIM}_{h-IG}^{(1)}$ (Figure 7-3), such that for every $n/2$-threshold (static or adaptive) adversary $\mathcal{A}$, the following two properties hold:*

1. *For any DLog instance (p, q, g), the following two variables are identically distributed:*
   (a) $\mathsf{View}_{h-IG,\mathcal{A}}(p, q, g)$, *an adversarial view of a random execution of the h-IG protocol on public input (p, q, g)*
   (b) $\mathsf{View}_{SIM_{h-IG}^{(1)},\mathcal{A}}(p, q, g)$, *an adversarial view of a simulation, i.e. a computation of $\mathsf{SIM}_{h-IG}^{(1)}$ and $\mathcal{A}$ on public input (p, q, g)*
2. *There exists k_0 s.t. for all $k \geq k_0$, for every discrete-log instance (p, q, g) of security parameter k, simulator $\mathsf{SIM}_{h-IG}^{(1)}$ interacting with $\mathcal{A}$ on public input (p, q, g), with probability at least $1 - \frac{1}{p(k)}$ produces a private output $\sigma = \log_g h$, where h is the public output of this interaction (i.e. the simulation of h-IG).*

Proof: Property (1) is easy to see. $\mathsf{SIM}_{h-IG}^{(1)}$ follows the protocol on behalf of the uncorrupted players till the end of h-IG. The fact that $\mathsf{SIM}_{h-IG}^{(1)}$ runs a *copy* of $\mathcal{A}$ on the side does not matter.

As for property (2), we first explain in more detail how $\mathsf{SIM}^{(1)}_{h-IG}$ extracts values $x_i = \log_g(y_i)$ from all the bad players that passed the ZKPK-DL proof with some good player. This process of extraction is similar to the construction used to show the proof-of-knowledge property of the ZKPK-DL proof system, i.e. Lemma 43 of Appendix G. The difference is that there the extractor needed to extract a witness from one prover, while here $\mathsf{SIM}^{(1)}_{h-IG}$ needs to extract witnesses from many provers at the same time.

$\mathsf{SIM}^{(1)}_{h-IG}$ performs the h-IG protocol on behalf of the uncorrupted players, and thus knows all values $x_i = \log_g y_i$ for $P_i \in \mathit{Good}$. At the end of the protocol, $G = \mathit{Bad} \cap \mathit{Qual}$ is the set of corrupted players which are not disqualified. For each $P_i \in G$, let G_i be the group of players in *Good* such that for each $P_j \in G_i$, P_i passed in the ZKPK-DL proof with P_j as a verifier. If P_i is in set G at the end of the protocol then it must be that P_i passed the ZKPK-DL proof with at least one uncorrupted player controlled by $\mathsf{SIM}^{(1)}_{h-IG}$. Therefore $\mathsf{SIM}^{(1)}_{h-IG}$ knows $\alpha_{ij} = \log_g h_{ij}$ for each $P_i \in G$ and $P_j \in G_i$.

Let $p()$ be any polynomial. For all large enough k's, $\mathsf{SIM}^{(1)}_{h-IG}$ can learn x_i for each P_i in G, with probability at least $1 - \frac{1}{p(k)}$ by the following procedure. Let $p'(k) = 6np''(k)p(k)$, where $p''(k)$ is some small polynomial such that $p''(k) \geq \ln(2p(k))$ for all large enough k's. $\mathsf{SIM}^{(1)}_{h-IG}$ repeats the following loop for $p'(k)$ times. $\mathsf{SIM}^{(1)}_{h-IG}$ runs the h-IG protocol again, interacting with a *copy* of $\mathcal{A}$ on the same adversarial input and the same random tape. In this re-run, $\mathsf{SIM}^{(1)}_{h-IG}$ sends the exact same messages on behalf of the uncorrupted players in Step 1 and then in Step 2 (i.e. in the parallel ZKPK-DL protocol of Figure 7-4), it sends still the same messages on behalf of the uncorrupted players in Rounds 1-3. However, in Round 4, $\mathsf{SIM}^{(1)}_{h-IG}$ uses fresh random coins $(R'_{ij}, \hat{R}_{ij})$, for each $P_i \in G$ and $P_j \in G_i$ s.t. $R'_{ij} \neq R_{ij}$ and $R_{ij} + h_{ij}\hat{R}_{ij} = R'_{ij} + h_{ij}\hat{R}'_{ij}$, where $(R_{ij}, \hat{R}_{ij})$ are the coins used by $\mathsf{SIM}^{(1)}_{h-IG}$ in the original interaction with $\mathcal{A}$. For all other (P_j, P_i) pairs s.t. $P_j \in \mathit{Good}$, $\mathsf{SIM}^{(1)}_{h-IG}$ simply follows the protocol. If for some $P_j \in G_i$, P_i responds correctly in Round 5 then $\mathsf{SIM}^{(1)}_{h-IG}$ learns $\log_g y_i$, as in the proof of the proof-of-knowledge property of the ZKPK-DL protocol (Lemma 43). Namely, $\mathsf{SIM}^{(1)}_{h-IG}$ computes $x_i = \log_g(y_i) = (m_{ij} - m'_{ij})/(R_{ij} - R'_{ij})$. $\mathsf{SIM}^{(1)}_{h-IG}$ repeats this interaction with a copy of $\mathcal{A}$ for $p'(k)$ times. If at the end there is $P_i \in G$ which never responds correctly in Round 5 to any $P_j \in G_i$, then $\mathsf{SIM}^{(1)}_{h-IG}$ outputs null. Otherwise, $\mathsf{SIM}^{(1)}_{h-IG}$ learns $x_i = \log_g y_i$ for each $P_i \in \mathit{Qual}$ and hence can output $\sigma = \log_g h = \sum_{P_i \in \mathit{Qual}} x_i$.

We argue that the above process is successful except for probability at most $\frac{1}{p(k)}$. This argument is very similar to the one in the proof of Lemma 43 in Appendix G, except that here we need to adapt it to the case where the simulator extracts the witnesses from many provers at the same time. Let X_i be the event that P_i is in set G, i.e. that P_i passes the ZKPK-DL proof with at least one uncorrupted player, and that $\mathsf{SIM}^{(1)}_{h-IG}$ never extracts x_i, i.e. that in all the $p'(k)$ repetitions of the parallel ZKPK-DL protocol, player P_i fails the proof with the same uncorrupted players. We want to upper-bound the probability that $\mathsf{SIM}^{(1)}_{h-IG}$ outputs null, i.e. the probability $\mathit{FAIL} = \Pr\left[\bigcup_{P_i \in \mathit{Bad}} X_i\right]$, taken over the random choices of $\mathcal{A}$ and $\mathsf{SIM}^{(1)}_{h-IG}$. Clearly, $\mathit{FAIL} \leq \sum_{P_i \in \mathit{Bad}} \Pr[X_i] \leq n \Pr[X_{\max}]$, where $i = \max$ is an index for which $\Pr[X_i]$ is the largest.

Denote by IJ some subset of pairs $(P_i, P_j) \in \mathit{Bad} \times \mathit{Good}$. Denote by $\vec{C}_{IJ}$ any set of values $C_{ij} \in G_q$ for $(i,j) \in IJ$. Denote by "$P_{\max}$ passes on $\vec{C}_{IJ}$" the event that $P_{\max}$ passes the ZKPK-DL proof with some $P_j \in \mathit{Good}$ s.t. $(\max, j) \in IJ$, and that the messages C_{ij} for each $(i,j) \in IJ$ sent by the uncorrupted players in Round 2 of Step 2 are equal to the values specified in $\vec{C}$. This event is well-defined only for the set of pairs of indices IJ s.t. $(\max, j) \in IJ$ for some P_j. Fix the random coins of the adversary and consider a probability threshold $T = \frac{1}{2np(k)}$, and let's define a set of "good" sets $(IJ, \vec{C}_{IJ})$ as follows:

$$\text{We call } (IJ, \vec{C}_{IJ}) \text{ "good" iff } \Pr[P_{\max} \text{ passes on } \vec{C}_{IJ}] \geq T$$

where the probability is taken over *those* coins of the uncorrupted players s.t. for every $(i,j) \in IJ$, values $(R_{ij}, \hat{R}_{ij})$ sent by the uncorrupted players P_j are subject to the constraint that $g^{R_{ij}} h^{\hat{R}_{ij}} = C_{ij}$, for every C_{ij} in $\vec{C}_{IJ}$.

Denote by "$P_{\max}$ passes *again* on $\vec{C}_{IJ}$" the event that in an interaction of $\mathsf{SIM}^{(1)}_{h-IG}$ with a *copy* of $\mathcal{A}$, in at least one of the $p'(k)$ repetitions of this interaction, $P_{\max}$ passes the ZKPK-DL proof with some $P_j \in \mathit{Good}$ s.t. $(\max, j) \in IJ$, and that the messages C_{ij} for each $(i,j) \in IJ$ sent by the uncorrupted players in Round 2 of Step 2 are equal to the values specified in $\vec{C}$. Let $p_{\vec{C}}$ denote the following probability:

$$p_{\vec{C}_{IJ}} = \Pr[P_{\max} \text{ passes } \textit{again} \text{ on } \vec{C}_{IJ} \mid \vec{C}_{IJ} \text{ is good}]$$

It is easy to see that if polynomial $p'(k)$ is sufficiently larger than $1/T$ (i.e. sufficiently larger than $np(k)$), then probability $\overline{p_{\vec{C}_{IJ}}} = 1 - p_{\vec{C}_{IJ}}$ is negligible. In particular, we make $6p''(k) = p'(k)/(np(k))$ sufficiently large so that the probability $\overline{p_{\vec{C}_{IJ}}}$ is smaller than $\frac{1}{2np(k)}$. This argument is elementary and it is identical to the one given in the proof of Lemma 43 in Appendix G. Furthermore, by the similar logic as in the proof of Lemma 43, $\Pr[X_{\max}]$ can be upper-bounded by $T + \overline{p_{\vec{C}_{IJ}}}$:

$$\begin{aligned}
\Pr[X_{\max}] \;\leq\; & \Pr[P_{\max} \text{ passes on } \vec{C}_{IJ} \mid \vec{C}_{IJ} \text{ good}] \; * \\
& \quad * \Pr[P_{\max} \text{ does not pass } \textit{again} \text{ on } \vec{C}_{IJ} \mid \vec{C}_{IJ} \text{ good}] \; + \\
& + \; \Pr[P_{\max} \text{ passes on } \vec{C}_{IJ} \mid \vec{C}_{IJ} \text{ bad}] \; * \\
& \quad * \Pr[P_{\max} \text{ does not pass } \textit{again} \text{ on } \vec{C}_{IJ} \mid \vec{C}_{IJ} \text{ bad}] \\
< \; & 1 * \overline{p_{\vec{C}_{IJ}}} + T * 1 \;\leq\; \left(\frac{1}{2np(k)} + \frac{1}{2np(k)} \right) \;\leq\; \frac{1}{np(k)}
\end{aligned}$$

Therefore, $\mathit{FAIL} \leq n \Pr[X_{\max}] < \frac{1}{p(k)}$. □

Lemma 36 (Simulation of h-IG, Option ii) *Under the discrete-log intractability assumption, for every polynomial $p(k)$ there exists a (PPT TM) simulator $\mathsf{SIM}^{(2)}_{h-IG}$ (Figure 7-3), such that for every $n/2$-threshold (static or adaptive) adversary $\mathcal{A}$, for any discrete-log instance family DL, for (p,q,g) a DLog instance of security parameter k in DL, and for every $\tilde{g} \in G_q$, the following two properties hold:*

1. *The statistical difference between the following variables is a negligible function of k:*

(a) $\mathsf{View}_{h-IG,\mathcal{A}}(p,q,g)$, *an adversarial view of a random execution of the* h-IG *protocol on public input* (p,q,g)

(b) $\mathsf{View}_{SIM^{(2)}_{h-IG},\mathcal{A}}((p,q,g);\tilde{g})$, *an adversarial view of a simulation, i.e. a computation of* $\mathsf{SIM}^{(2)}_{h-IG}$ *and* $\mathcal{A}$ *on public input* (p,q,g) *and* $\mathsf{SIM}^{(2)}_{h-IG}$*'s input* $\tilde{g}$*, given that the adversary does not corrupt the inconsistent player* P_S *(see remark below).*

2. *There exists* k_0 *s.t. for all* $k \geq k_0$*, simulator* $\mathsf{SIM}^{(2)}_{h-IG}$ *interacting with* $\mathcal{A}$ *on public input* (p,q,g) *and* $\mathsf{SIM}^{(2)}_{h-IG}$*'s input* $\tilde{g}$*, with probability at least* $1-\frac{1}{p(k)}$ *outputs a representation* (a,b) *of* h *in bases* $g,\tilde{g}$ *(i.e. a pair s.t.* $g^a\tilde{g}^b = h$*), where* h *is a public output of this interaction (i.e. the simulation of* h-IG*).*

Remark. Note that in the static adversary model property 1 means that the adversarial view of the simulation is statistically indistinguishable from the view of the protocol, because any P_S chosen among the uncorrupted players *Good* by $\mathsf{SIM}^{(2)}_{h-IG}$ is never corrupted by the adversary.

Proof: First we explain how $\mathsf{SIM}^{(2)}_{h-IG}$ simulates the ZKPK-DL proofs between P_S and each $P_i \in Bad$ (see the description of $\mathsf{SIM}^{(2)}_{h-IG}$ in Figure 7-3). We use the notation of Figure 7-4 for Step 2 of the h-IG protocol.

$\mathsf{SIM}^{(2)}_{h-IG}$ simulates the proofs performed by P_S as follows. In Round 4 of Step 2, $\mathsf{SIM}^{(2)}_{h-IG}$ collects all coins $R_{S,j}$ that the corrupted players P_j open correctly. $\mathsf{SIM}^{(2)}_{h-IG}$ rewinds the adversary $\mathcal{A}$ to the beginning of Round 3 (i.e. restarts algorithm $\mathcal{A}$ and runs it until the beginning of Round 3, on the same random input of $\mathcal{A}$, and sending the same messages on behalf of the uncorrupted players), and uses the $R_{S,j}$ values above to prepare messages $m^*_{S,j}$ and $M^*_{S,j}$ which make P_S pass the ZKPK-DL proof against each P_j that uses $R_{S,j}$ in Round 3 again. Namely, by the construction used in the proof of the ZK property of ZKPK-DL in Lemma 44 in Appendix G, $\mathsf{SIM}^{(2)}_{h-IG}$ picks $m^*_{S,j}$ uniformly in $\mathbb{Z}_q$, and computes $M^*_{S,j} = g^{m^*_{S,j}} y_S^{-R_{S,j}}$. For each player $P_j \in Bad$ who opens its commitment as $R_{S,j}$ again, $\mathsf{SIM}^{(2)}_{h-IG}$ sends back $m^*_{S,j}$ and α_{ij}. For players $P_j \in Bad$ who do not open their commitments correctly at all, $\mathsf{SIM}^{(2)}_{h-IG}$ sends nothing. If any player $P_j \in Bad$ opens its commitment correctly, but as a different coin $R'_{S,j} \neq R_{S,j}$, then $\mathsf{SIM}^{(2)}_{h-IG}$ fails. For all other players in *Good*, $\mathsf{SIM}^{(2)}_{h-IG}$ just follows their protocol. If $\mathsf{SIM}^{(2)}_{h-IG}$ does not fail in simulating Step 2, it proceeds to Step 3 of h-IG and just follows the protocol on behalf of the uncorrupted players. We call this second run of Steps 1-2 (and followed by Step 3) a "re-run" of the h-IG protocol.

In the same way as in the proof of Lemma 44, we can show that if the probability that any player P_j controlled by the adversary $\mathcal{A}$ opens its commitment $C_{S,j}$ as one coin $R_{S,j}$ in the first run and then another coin $R^*_{S,j}$ in the re-run (where for any DLog instance (p,q,g) the probability is taken over random coins of the players and the adversary $\mathcal{A}$), is not negligible (i.e. higher than $\frac{1}{p(k)}$ for some polynomial $p()$, for all k_0, and for some $k \geq k_0$), then we can construct an efficient algorithm $\mathcal{E}$ which, interacting with $\mathcal{A}$, can compute

(with the same probability $\frac{1}{p(k)}$) discrete logarithm $\log_g h$ of a random input value h in G_q. Namely, $\mathcal{E}$, similarly to $\mathsf{SIM}^{(1)}$ in the proof of Lemma 44 embeds h into each instance of the Pedersen Commitment $h_{\mathrm{S},j}$ it sends to player P_j (for example $h_{\mathrm{S},j} = h^{\beta_{\mathrm{S},j}}$ for a random $\beta_{\mathrm{S},j}$ in $\mathbb{Z}_q$). $\mathcal{E}$ runs the protocol to Round 4 once and then re-runs it once again. If any P_j controlled by $\mathcal{A}$ opens its commitment $C_{\mathrm{S},j}$ correctly but differently in the two instances then $\mathcal{E}$ gets two representations of $C_{\mathrm{S},j}$ in bases $g, h_{\mathrm{S},j}$, and hence can compute $\log_g h_{\mathrm{S},j}$, and hence can compute $\log_g h$.

Therefore, under the discrete-log assumption, this probability can only be negligible in the security parameter, and consequently $\mathsf{SIM}^{(2)}_{h-IG}$ passes the re-run of the interaction with $\mathcal{A}$ except for negligible probability. Until Round 5, the distribution of the messages seen by $\mathcal{A}$ in this re-run is the same as in the protocol, because all the players except of P_{S} follow the protocol, messages $M^*_{\mathrm{S},j}$ sent by P_{S} in Round 3 are chosen uniformly and independently in G_q. In Round 5 the adversary $\mathcal{A}$ sees (for all $P_j \in \mathit{Bad}$) either messages $m^*_{\mathrm{S},j}$ and $m_{i,j}$ for $P_i \in \mathit{Good} \setminus \{P_{\mathrm{S}}\}$ which satisfy the relation imposed by the protocol, or $\mathcal{A}$ sees no good messages from player P_{S}. However, as we argued above, the latter happens only with a negligible probability. Consequently, the statistical difference between $\mathcal{A}$'s view of an interaction with $\mathsf{SIM}^{(2)}_{h-IG}$ and $\mathcal{A}$'s view of a random execution of h-IG is negligible which proves property (1).

The above discussion implies that under the discrete-log assumption, for any $p(k)$ there exists k_0 s.t. for all $k \geq k_0$ simulator $\mathsf{SIM}^{(2)}_{h-IG}$ passes to the end of h-IG except for at most $\frac{1}{2p(k)}$ probability. In this case $\mathsf{SIM}^{(2)}_{h-IG}$ follows the protocol of $\mathsf{SIM}^{(1)}_{h-IG}$ to extract values x_i for $P_i \in (\mathit{Bad} \cap \mathit{Qual})$. Namely, as argued in Lemma 35, there exists polynomial $p'(k)$ s.t. $\mathsf{SIM}^{(2)}_{h-IG}$ runs a *copy* of $\mathcal{A}$ on the side $p'(k)$ times and attempts to extract all the above values. (Note that since $\mathsf{SIM}^{(2)}_{h-IG}$ interacts with a copy of $\mathcal{A}$, the adversarial view of the simulation does not change.) By the same argument which was used to show property (2) of Lemma 35, there is k'_0 s.t. for all $k \geq k'_0$, $\mathsf{SIM}^{(2)}_{h-IG}$ learns all the values x_i for $P_i \in (\mathit{Bad} \cap \mathit{Qual})$ except for probability $\frac{1}{2p(k)}$. In the case it is successful, since $\mathsf{SIM}^{(2)}_{h-IG}$ knows also value $b = \log_{\tilde{g}} y_{\mathrm{S}}$ and values $x_i = \log_g y_i$ for all other uncorrupted players, $\mathsf{SIM}^{(2)}_{h-IG}$ knows $\log_g y_i$ or $\log_{\tilde{g}} y_i$ for all values y_i for $P_i \in \mathit{Qual}$, and hence can output a representation a, b of the public output h in bases $g, \tilde{g}$. Since the probability that $\mathsf{SIM}^{(2)}_{h-IG}$ fails to output this representation is the probability that it fails to simulate the ZKPK-DL proofs of P_{S} plus the probability that it fails to extract the needed x_i values from the corrupted players, it follows that for $k \geq \max(k_0, k'_0)$, $\mathsf{SIM}^{(2)}_{h-IG}$ outputs this representation except for probability at most $\frac{1}{p(k)}$. □

Bibliography

[Bac85] Eric Bach. *Analytic Methods in the Analysis and Design of Number-Theoretic Algorithms.* ACM Distiguished Dissertation (1984). MIT Press, Cambridge, MA, 1985.

[BB89] J. Bar-Ilan and D. Beaver. Non-cryptographic fault-tolerant computing in a constant number of rounds. In *Proc. 8th ACM Symp. on Principles of Distributed Computation*, pages 201–209, 1989.

[BCK98] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. A modular approach to the design and analysis of authentication and key exchange protocols (extended abstract). In *Proc. 30th ACM Symp. on Theory of Computing*, pages 419–428, Dallas, 1998. ACM.

[Bea87] Donald Beaver. Oblivious secret computation. Technical Report TR-12-87, Harvard University, December 1987.

[Bea91] Donald Beaver. Foundations of secure interactive computing. In *Proc. CRYPTO 91*, pages 377–391. Springer-Verlag, 1991.

[Bea97] Donald Beaver. Plug and play encryption. In *Proc. CRYPTO 97*, pages 75–89. Springer-Verlag, 1997.

[BF97] Dan Boneh and Matthew Franklin. Efficient generation of shared RSA keys. In *Proc. CRYPTO 97*, pages 425–439, 1997.

[BG92] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In *Proc. CRYPTO 92*, pages 390–420. Springer-Verlag, 1992.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, Chicago, 1988. ACM.

[BH92] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Proc. EUROCRYPT 92*, pages 307–323, Berlin, 1992. Springer-Verlag. Lecture Notes in Computer Science No. 658.

[Blu82] Manuel Blum. Coin flipping by telephone: a protocol for solving impossible problems. In *Proc. 24th IEEE Computer Conference*, pages 133–137, 1982.

[BMR90] Donald Beaver, Silvio Micali, and Phil Rogaway. The round complexity of secure protocols. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 503–513, 1990.

[BOCG91] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation (extended abstract). In *Proc. 25th ACM Symp. on Theory of Computing*, pages 52–61, San Diego, 1991. ACM.

[Boy89] Colin Boyd. Digital multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246, Kingston, Ontario, Canada, May 1989. Claredon Press.

[Bra99] Stefan Brands. Rethinking public-key infrastructures and digital certificates–building in privacy. *Ph.D. dissertation, Technical University of Eindhoven*, 1999.

[BW] E. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent 4,633,470.

[CA89] David Chaum and Hans Van Antwerpen. Undeniable signatures. In *Proc. CRYPTO 89*, pages 212–217. Springer-Verlag, 1989. Lecture Notes in Computer Science No. 435.

[Cam98] Jan Camenisch. Group signature schemes and payment systems based on the discrete logarithm problem. *ETH Series in Information Security and Cryptography, vol.2*, 1998.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[CCD88] D. Chaum, C. Crepeau, and I. Damgård. Multi-party unconditionally secure protocols. In *Proc. 20th ACM Symp. on Theory of Computing*, Chicago, 1988. ACM.

[CD98] Ronald Cramer and Ivan Damgård. Zero-knowledge proof for finite field arithmetics, or: Can zero-knowledge be for free. In *Proc. CRYPTO 98*, pages 424–441. Springer-Verlag, 1998.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient muliparty computations secure against an adaptive adversary. In *Proc. EUROCRYPT 99*, pages 311–326. Springer-Verlag, 1999. Lecture Notes in Computer Science No. 1592.

[CDNO99] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafi Ostrovsky. Deniable encryption. In *Proc. CRYPTO 97*, pages 90–104. Springer-Verlag, 1999. Lecture Notes in Computer Science No. 1294.

[CF85] J. D. Cohen and M. J. Fischer. A robust and verifiable cryptographically secure election scheme. In *Proc. 26th IEEE Symp. on Foundations of Comp. Science*, pages 372–382, Portland, 1985. IEEE.

[CFGN96] Ran Canetti, Uri Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *Proc. 28th ACM Symp. on Theory of Computing*, Philadelphia, 1996. ACM. Fuller version in MIT-LCS-TR # 682, February 1996.

[CG99] Ran Canetti and Shafi Goldwasser. An efficient threshold public key cryptosystem secure aginast adaptive chosen ciphertext attack. In *Proc. EUROCRYPT 99*, pages 90–106. Springer-Verlag, 1999. Lecture Notes in Computer Science No. 1592.

[CGJ+99] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Proc. CRYPTO 99*, pages 98–115. Springer-Verlag, 1999. Lecture Notes in Computer Science No. 1666. Fuller version available at http://theory.lcs.mit.edu/~cis/cis-threshold.html.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *Proc. 26th IEEE Symp. on Foundations of Comp. Science*, pages 383–395, Portland, 1985. IEEE.

[CGS97] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European Transactions on Telecommunications*, 8(5), September 1997. Also prelminary version in Proc. of EUROCRYPT'97. Springer-Verlag. LNCS No. 1233, pp.103-118.

[CH89] R.A. Croft and S.P. Harris. Public-key cryptography and re-usable shared secrets. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 189–201, Kingston, Ontario, Canada, May 1989. Claredon Press.

[CH94] Rand Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In *Proc. CRYPTO 94*, pages 425–438, 1994.

[Cha90] David Chaum. Zero–knowledge undeniable signatures. In *Proc. EUROCRYPT 90*, pages 458–464. Springer-Verlag, 1990.

[CHH97] Ran Canetti, Shai Halevi, and Amir Herberg. Maintaining authenticated communication in the presence of break-ins. In *Proc. 16th ACM Symp. on Principles of Distributed Computation.* ACM, 1997.

[CMI93] M. Cerecedo, T. Matsumoto, and H. Imai. Efficient and secure multiparty generation of digital signatures based on discrete logarithms. *IEICE Trans. Fundamentals*, E76-A(4):532–545, April 1993.

[CS98] Ronald Cramer and Victor Shoup. A practical public-key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Proc. CRYPTO 98.* Springer-Verlag, 1998.

[DDFY94] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *Proc. 26th ACM Symp. on Theory of Computing*, pages 522–533, Montreal, Canada, 1994. ACM.

[DDN91] Danny Dolev, Cynthia Dwork, and Moni Naor. Non-malleable cryptography *(preliminary version)*. In *Proc. 23rd ACM Symp. on Theory of Computing*, 1991.

[Des88] Yvo Desmedt. Society and group oriented cryptography: A new concept. In Carl Pomerance, editor, *Proc. CRYPTO 87*, pages 120–127. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.

[Des94] Yvo Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[DF89] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 307–315. Springer-Verlag, 1989. Lecture Notes in Computer Science No. 435.

[DF91] Yvo Desmedt and Yair Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 457–469. Springer, 1991. Lecture Notes in Computer Science No. 576.

[DJ99] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications, 1999. Available at http:///www.cs.fsu.edu/~desmedt/topics-threshold.html.

[DN00] Ivan Damgard and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. to be published, 2000.

[DNRS99] Cynthia Dwork, Moni Naor, Omer Reingold, and Larry Stockmeyer. Magic functions. In *Proc. 40th IEEE Symp. on Foundations of Comp. Science*, 1999.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Computing*, 12(4), 1983.

[ElG85a] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *IEEE Trans. Info. Theory*, volume 31, 1985.

[ElG85b] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *IEEE Trans. Inform. Theory*, volume 31, pages 469–472, 1985.

[FD92] Yair Frankel and Yvo Desmedt. Parallel reliable threshold multisignature. Technical Report TR-92-04-02, Dept. of EE and CS, U. of Winsconsin, April 1992.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symp. on Foundations of Comp. Science*, pages 427–437, 1987.

[FGMY97a] Yair Frankel, Peter Gemmell, Phil Mackenzie, and Moti Yung. Optimal resilience proactive public-key cryptosystems. In *Proc. 38th IEEE Symp. on Foundations of Comp. Science*, pages 384–393. IEEE, 1997.

[FGMY97b] Yair Frankel, Peter Gemmell, Phil Mackenzie, and Moti Yung. Proactive RSA. In *Proc. CRYPTO 97*, pages 440–454. Springer, 1997. Lecture Notes in Computer Science No. 1294.

[FGY96] Yair Frankel, Peter Gemmell, and Moti Yung. Witness-based cryptographic program checking and robust function sharing. In *Proc. 28th ACM Symp. on Theory of Computing*, pages 499–508, Philadelphia, 1996. ACM.

[FMY99a] Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure distributed threshold public key systems. In *Proceedings of ESA 99*, 1999.

[FMY99b] Yair Frankel, Philip MacKenzie, and Moti Yung. Adaptively-secure optimal-resilience proactive RSA. In *Proc. ASIACRYPT 99*. Springer-Verlag, 1999.

[FMY99c] Yair Frankel, Philip MacKenzie, and Moti Yung. Robust efficient distributed RSA-key generation. In *Proc. 30th ACM Symp. on Theory of Computing*, pages 663–672. ACM, 1999.

[Fra89] Yair Frankel. A practical protocol for large group oriented networks. In *Proc. EUROCRYPT 89*, pages 56–61. Springer, 1989. Lecture Notes in Computer Science No. 434.

[GGJR97] Juan Garay, Rosario Gennaro, C. Jutla, and Tal Rabin. Secure distributed storage and retrieval. In *International Workshop on Distributed Algorithms (WDAG '97)*, pages 275–289. Springer-Verlag, 1997. Lecture Notes in Computer Science No. 1320.

[Gil99] Niv Gilboa. Two party rsa key generation. In *Proc. CRYPTO 99*, pages 116–129, 1999.

[GJKR96a] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. In *Proc. CRYPTO 96*, pages 157–172. Springer-Verlag, 1996. Lecture Notes in Computer Science No. 1109. Extended version available at http://theory.lcs.mit.edu/~cis/cis-threshold.html.

[GJKR96b] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signature (preliminary abstract). In *Proc. EUROCRYPT 96*, pages 354–371. Springer-Verlag, 1996. Lecture Notes in Computer Science No. 1070, to be published in *Journal of Information and Computation*, available at http://theory.lcs.mit.edu/~cis/cis-threshold.html.

[GJKR99] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete–log based cryptosystems. In Jacques Stern, editor, *Proc. EUROCRYPT 99*, pages 295–310. Springer-Verlag, 1999. Lecture Notes in Computer Science No. 1592. Extended version available at http://theory.lcs.mit.edu/~cis/cis-threshold.html.

[GK96] Oded Goldreich and Hugo Krawczyk. On the composition of zero-knowledge proof systems. *SIAM J. Computing*, 25(1), 1996.

[GM82] Shafi Goldwasser and Silvio Micali. Probabilistic encryption and how to play mental poker hiding all partial information. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 365–377, San Francisco, 1982.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[GMR85] Shafi Goldwasser, Silvio Micali, and Charlie Rackoff. The knowledge complexity of interactive proof systems. In *STOC85*, pages 291–304, Providence, May 1985.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charlie Rackoff. The knowledge complexity of interactive proof-systems. *SIAM J. Computing*, 18(1):186–208, February 1989.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game — A completeness theorem for protocols with honest majority. In ACM, editor, *Proc. 19th ACM Symp. on Theory of Computing*, pages 218–229, New York, NY 10036, USA, 1987. ACM Press. (See also [Gol98]).

[Gol95] Oded Goldreich. Foundations of cryptography (fragments of a book). On-line manuscript, February 1995. http://www.wisdom.weizmann.ac.il/~oded.

[Gol98] Oded Goldreich. Secure multi-party computation. On-line manuscript, June 1998. http://www.wisdom.weizmann.ac.il/~oded.

[GRR98] Rosario Gennaro, Michael Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In *Proc. 17th ACM Symp. on Principles of Distributed Computation.* ACM, 1998.

[Hal00] Shai Halevi, February 2000. Personal communication with the author.

[Har94] Lee Harn. Group oriented (t, n) digital signature scheme. *IEE Proceedings. Computers and Digital Techniques*, 141(5), September 1994.

[HJJ+97] Amir Herzberg, Markus Jakobson, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive public–key and signature systems. In *Proceedings of the 4th ACM Conference on Computer and Communication Security*, pages 100–110. ACM, 1997.

[HJKY95] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing, or: How to cope with perpetual leakage. In D. Coppersmith, editor, *Proc. CRYPTO 95*, pages 339–352. Springer-Verlag, 1995. Lecture Notes in Computer Science No. 963.

[HSSD84] Joseph Halpern, Barbara Simons, Raymond Strong, and Danny Dolev. Fault-tolerant clock synchronization. In *Proc. 3th ACM Symp. on Principles of Distributed Computation*, pages 89–102, 1984.

[IS93] T. Itoh and K. Sakurai. On the complexity of constant round zkip of possession of knowledge. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E76-A(1), January 1993.

[JL00] Stanisław Jarecki and Anna Lysyanskaya. Adaptively secure threshold cryptosystems without erasures. Available at theory of cryptography library or at http://theory.lcs.mit.edu/~cis/cis-threshold.html, July 2000.

[Kob94] Neal Koblitz. *A Course in Number Theory and Cryptography*. Springer-Verlag, 2nd edition, 1994.

[Lan95] Susan Langford. Threshold DSS signatures without a trusted party. In D. Coppersmith, editor, *Proc. CRYPTO 95*, pages 397–409. Springer-Verlag, 1995. Lecture Notes in Computer Science No. 963.

[LHL94] C.-H. Li, T. Hwang, and N.-Y. Lee. (t, n)-threshold signature schemes based on discrete logarithm. In *Proc. EUROCRYPT 94*, pages 191–200. Springer-Verlag, 1994. Lecture Notes in Computer Science No. 950.

[LMS85] Leslie Lamport and P. Michael Melliar-Smith. Synchronizing clocks in the presence of faults. *Journal of the ACM*, 32(1):52–78, 1985.

[Lys99] Anna Lysyanskaya. Efficient threshold and proactive cryptography secure against the adaptive adversary. Available at http://theory.lcs.mit.edu/~cis/cis-threshold.html, October 1999.

[Mic92] Silvio Micali. Fair public-key cryptosystems. In Ernest F. Brickell, editor, *Proc. CRYPTO 92*, pages 113–138. Springer-Verlag, 1992. Lecture Notes in Computer Science No. 740.

[Mic99] Silvio Micali. Optimistic exchange of secrets. Manuscript, 1999.

[MR91] Silvio Micali and Phil Rogaway. Secure computation (chapters 1-3). Technical report, Laboratory for Computer Science, MIT, Cambridge, MA 02139, USA, 1991. Preliminary version in Proc. CRYPTO 91, LNCS 576, Springer-Verlag, Berlin 1992, pages 392–404.

[MSNW99] Keith Martin, Rei Safavi-Naini, and Huaxiong Wang. Bounds and techniques for efficient redistribution of secret shares to new access structures. *The Computer J.*, 42(8):638–649, 1999. ISSN 1460 2067.

[NIS91] Digital signature standard (DSS). Technical Report 169. National Institute for Standards and Technology, August 30, 1991.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 33–43, Seattle, 1989. ACM.

[NY90] Moni Naor and Moti Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *Proc. 22nd ACM Symp. on Theory of Computing*, pages 427–437, 1990.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks. In *Proc. 10th ACM Symp. on Principles of Distributed Computation*, pages 51–61, 1991.

[Ped91a] Torben Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proc. CRYPTO 91*, pages 129–140. Springer-Verlag, 1991.

[Ped91b] Torben Pedersen. A threshold cryptosystem without a trusted party. In D. Davies, editor, *Proc. EUROCRYPT 91*, pages 522–526. Springer-Verlag, 1991. Lecture Notes in Computer Science No. 547.

[PK96] C. Park and K. Kurosawa. New ElGamal-type threshold digital signature scheme. *IEICE Trans. Fundamentals*, E79-A(1):86–93, January 1996.

[Rab98] Tal Rabin. A simplified approach to threshold and proactive RSA. In *Proc. CRYPTO 98*, pages 89–104. Springer-Verlag, 1998. Lecture Notes in Computer Science No. 1462.

[RB89] Tal Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 73–85, Seattle, 1989. ACM.

[RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, February 1978.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4:161–174, 1991.

[SG98] Victor Shoup and Rosario Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. In *Proc. EUROCRYPT 98*, pages 1–16. Springer-Verlag, 1998. Lecture Notes in Computer Science No. 1403.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

[Sho99] Victor Shoup. Why chosen ciphertext security matters. *IBM Research Report RZ3076*, 1999.

[Sho00] Victor Shoup. Practical threshold signatures. In *Proc. EUROCRYPT 00*, pages 207–220. Springer-Verlag, 2000.

[Ve00] Personal Communication with a Research Staff at Verisign, April 2000.

[Yao82] Anrew Chi-Chih Yao. Protocols for secure computations. In *Proc. 23rd IEEE Symp. on Foundations of Comp. Science*, pages 160–164, Chicago, 1982. IEEE.

Appendix A

Adaptive Threshold Cramer-Shoup Cryptosystem

To further exemplify the power of our build-block threshold protocols we show how to use them to construct an adaptive erasure-free threshold Cramer-Shoup cryptosystem. We present a threshold key generation protocol and a threshold ciphertext decryption protocol. This construction appeared in [JL00] for the adaptive and erasure-free model. It is based on a previous construction of [CG99] of a threshold Cramer-Shoup protocol secure in the static model.[1]

A.1 The Cramer-Shoup Cryptosystem

Recall the Cramer-Shoup cryptosystem [CS98]. The setting is as follows. We are given a group G_q in which the decisional Diffie-Hellman problem is assumed to be hard, and a universal one-way hash function family (UOWHF) [NY89] $\mathcal{H} : \{0,1\}^* \rightarrow \mathbb{Z}_q$. The secret key consists of five values, a, b, c, d, e, selected from $\mathbb{Z}_q$ uniformly at random. The public key consists of two random bases, $g_1, g_2 \in G_q$, such that the discrete logarithm that relates them is unknown, and the group elements $A = g_1^a g_2^b$, $C = g_1^c g_2^d$ and $E = g_1^e$. To encrypt a message m from a message space M (M is assumed to have an efficiently computable and invertible mapping into G_q, and so we write $m \in G_q$), Alice chooses $r \in \mathbb{Z}_q$ uniformly at random, computes $x = g_1^r$, $y = g_2^r$, $w = E^r m$, $\sigma = \mathcal{H}(x, y, z)$, and $v = A^r C^{r\sigma}$. The ciphertext is the 4-tuple (x, y, w, v).

To decrypt, we use the Canetti-Goldwasser method [CG99]. Bob selects uniformly at random $s \in \mathbb{Z}_q$ and outputs $w/(x^e(v/v')^s)$, where $v' = x^{a+c\sigma} y^{b+d\sigma}$. Note that if $v = v'$, as it should be if the ciphertext is formed via the legal encryption procedure above, then $w/(x^e(v/v')^s) = w/x^e = m$, just like in ElGamal decryption. Recall that assuming that the decisional Diffie-Hellman problem is hard, the Cramer-Shoup cryptosystem is secure

[1] In fact, the two protocols are composed of building blocks for the same arithmetic operations, and the crucial difference between the two constructions is the use of statically-secure building-block protocols for these arithmetic operations in [CG99], and the substitution of these protocols by their adaptively-secure counterparts in [JL00].

against adaptive chosen ciphertext attack, which is the strongest notion of security known for public-key cryptosystems [CS98] (see also [Sho99]).

A.2 Threshold Cramer-Shoup Key Generation

We present the Cramer-Shoup distributed key generation protocol CS-DKG in Figure A-1. We assume that in addition to the Pedersen commitment instance p, q, g, h, a universal one-way hash function $\mathcal{H}$ has been randomly chosen. In known UOWHF constructions, picking a random instance $\mathcal{H}$ of a UOWHF family is equivalent to picking a random long-enough string (e.g. see the recent result of [Sho00] and the references therein), which in our threshold setting can be achieved via the distributed coin-flipping protocol RVSS.

In the CS-DKG protocol we first generate the two random public bases $g_1, g_2 \in G_q$. Since they are random, we generate them by generating two random coins r_1 and r_2 in $\mathbb{Z}_q$ with two (parallel) runs of a distributed coin-flip protocol RVSS followed by two (parallel) runs of the exponentiation protocol Ad-Exp on public input g and the generated sharings RVSS-data$[r_1]$ and RVSS-data$[r_2]$ to obtain two random elements in G_q: $g_1 = g^{r_1}$ and $g_2 = g^{r_2}$. This part of the protocol is secure because RVSS followed by Ad-Exp are secure under parallel composition. The adversary cannot skew this process unless he knows how to compute a discrete logarithm, and by the simulatability of (two parallel instances of) Ad-Exp (Lemma 29), this part of the protocol can be simulated with a single inconsistent player to output any values g_1 and g_2.

Since the private keys a, b, c, d, e are all random in $\mathbb{Z}_q$, we generate them with five (parallel) runs of the coin-flip protocol RVSS. Generation of values A, C, and E, is a variant of the adaptive exponentiation protocol Ad-Exp. To generate $E = g_1^e$ we use the Ad-Exp protocol itself, on inputs RVSS-data$[e]$ and g_1. To generate $A = g_1^a g_2^b$ we follow a protocol very similar to Ad-Exp. Each player uses its additive shares a_i and b_i and publishes $A_i = g_1^{a_i} g_2^{b_i}$. To check that this value is generated correctly, the players can use the verification values $F_{a_i}(0) = g^{a_i} h^{\hat{a}_i}$ and $F_{b_i}(0) = g^{b_i} h^{\hat{b}_i}$ from RVSS-data$[a]$ and RVSS-data$[b]$. Each player P_i should prove that it knows values $a_i, \hat{a}_i, b_i, \hat{b}_i$ which satisfy the above three constraints. This can be formulated as knowledge of an equal representation of values A_i, $F_{a_i}(0)$, and $F_{b_i}(0)$ in appropriate bases made of elements g_1, g_2, g, and h (see Step 3a in Figure A-1), and thus to prove it the players run a simultaneous proof protocol using the ZKPK for equal representations which we include in Figure F-2, page 219. Value $C = g_1^c g_2^d$ is generated in the same way. In fact, all three values A, C, and E can be generated in parallel. As in the exponentiation protocol Ad-Exp, the simultaneous proof can be adaptively simulated with a single inconsistent player.

It follows that CS-DKG protocol is single-inconsistent-player-simulatable. It is also robust in the presence of an $n/2$-threshold adaptive adversary who cannot compute $\log_g h$. As we have argued above, Steps 1-2 of CS-DKG are two executions of Ad-Exp, and the algorithm of Steps 3-4 is a close variant of a parallel execution of three instances of Ad-Exp. Therefore the proof of robustness of CS-DKG is an easy extension of the robustness argument for Ad-Exp we give in Section 5.2.1.

Adaptively Secure Cramer-Shoup Key Generation CS-DKG

Inputs: Pedersen commitment instance p, q, g, h, universal hash function $\mathcal{H}$
Outputs: Sharing of private key elements RVSS-data$[a, b, c, d, e]$
and the corresponding public key elements (g_1, g_2, A, C, E)

1. Run in parallel two RVSS instances to obtain RVSS-data$[r_1, r_2]$ and two instances of Ad-Exp to compute $g_1 = g^{r_1}$ and $g_2 = g^{r_2}$
2. Run RVSS five times in parallel to obtain RVSS-data$[a, b, c, d, e]$.
3. Each P_i publishes $A_i = g_1^{a_i} g_2^{b_i}$, $C_i = g_1^{c_i} g_2^{d_i}$, and $E_i = g_1^{e_i}$, and all players perform a simultaneous proof protocol using ZKPK of equality of representations (Figure F-2, Appendix F) to prove that
   (a) Values A_i, $F_{a_i}(0)$, and $F_{b_i}(0)$ have equal reps. in bases $(g_1, g_2, 1, 1)$, $(g, 1, h, 1)$, and $(1, g, 1, h)$
   (b) Values C_i, $F_{c_i}(0)$, and $F_{d_i}(0)$ have equal reps. in bases $(g_1, g_2, 1, 1)$, $(g, 1, h, 1)$, $(1, g, 1, h)$
   (c) Values E_i and $F_{e_i}(0)$ have equal reps. in bases $(g_1, g_2, 1)$ and $(g, 1, h)$.
4. If any player $P_i \in \mathit{Qual}$ fails, their secret values a_i, b_i, c_i, d_i, and e_i are publicly reconstructed using verification information F_{a_i}, F_{b_i}, F_{c_i}, F_{d_i} and F_{e_i}, and values A_i, C_i, E_i are publicly computed.
   Finally, $A = \prod_{P_i \in \mathit{Qual}} A_i$, $C = \prod_{P_i \in \mathit{Qual}} C_i$, and $E = \prod_{P_i \in \mathit{Qual}} E_i$.

Simulation: (on SIM's additional inputs $\sigma = \log_g h$, random elements $g_1^*, g_2^*, A^*, C^*, E^*$ in G_q, and an identity P_S of an inconsistent player in *Good*)

1. SIM follows the RVSS protocols of Step 1, and simulates the exponentiation protocols to output g_1^* and g_2^*.
2. SIM follows Step 2 of the protocol.
3. SIM acts similarly to the simulator of the exponentiation protocol in Figure 5-3: It interpolates the secret-sharing polynomials to find the values a_i, b_i, c_i, d_i, e_i held by the corrupted players, computes their values A_i, C_i, E_i, and then broadcasts correct values A_i, C_i, E_i for all *uncorrupted* players P_i, except for P_S, for whom SIM broadcasts $A_S^* = A^* / \prod_{P_j \in \mathit{Qual} \setminus \{P_S\}} A_j$, and similarly computed C_S^* and E_S^*. SIM then simulates the simultaneous proof protocol without knowing the correct witnesses for only one player P_S.
4. SIM follows the protocol of Step 4. Unless the adversary can compute discrete-logarithms the output public key is $g_1^*, g_2^*, A^*, C^*, E^*$.

Figure A-1: CS-DKG: Threshold Cramer-Shoup Key Generation

A.3 Threshold Cramer-Shoup Decryption

Adaptively Secure Cramer-Shoup Decryption CS-TDec

Input: Sharings RVSS-data[a,b,c,d,e], key (g_1, g_2, A, C, E), ciphertext (x, y, w, v, σ)
Output: Cleartext $m = wv^{-s}x^{s(a+c\sigma)-e}y^{s(b+d\sigma)}$

1. Players perform RVSS protocol to obtain RVSS-data[s].
2. Each player locally obtains its part of RVSS-data[$a + c\sigma$] and RVSS-data[$b + d\sigma$] from RVSS-data[a, b, c, d, s] and σ.
3. Let $r = s(a + c\sigma)$ and $z = s(b + d\sigma)$. Players perform two parallel Mult-opt instances to get RVSS-data[r] from RVSS-data[$s, a + c\sigma$] and RVSS-data[z] from RVSS-data[$s, b + d\sigma$]
4. Each P_i in *Part* broadcasts $m_i = wv^{-s_i}x^{r_i-e_i}y^{z_i}$. The players perform a simultaneous proof using a ZKPK proof of Figure F-2, page 219, to let each P_i prove the knowledge of equal representation of values m_i/w, $F_{s_i}(0)$, $F_{r_i}(0)/F_{e_i}(0)$, and $F_{z_i}(0)$ in bases $(v^{-1}, x, y, 1, 1, 1)$, $(g, 1, 1, h, 1, 1)$, $(1, g, 1, 1, h, 1)$, and $(1, 1, g, 1, 1, h)$.
5. Value $m = \prod_{P_i \in Part} m_i$ is publicly computed. If any player fails, its secret values are reconstructed and the protocol proceeds.

Simulation: (on SIM's additional inputs $\sigma = \log_g h$, an identity P_S of an inconsistent player in *Good*, and cleartext m^* provided by the oracle that implements an underlying instance of the Cramer-Shoup cryptosystem. See the discussion of the distribution of m^* in this section.)

1.-3. SIM follows Steps 1-3 of the protocol.

4. The simulation is similar to that of the exponentiation protocol (Figure 5-3). SIM interpolates the secret-sharing polynomials to find the values s_i, r_i, e_i, z_i held by the corrupted players, computes their values m_i, and then broadcasts correct values m_i for all *uncorrupted* players P_i, except for P_S, for whom it broadcasts $m_S^* = m^* / \prod_{P_j \in Qual \setminus \{P_S\}} m_j$. SIM then simulates the simultaneous proof protocol without knowing the correct witnesses for only one player P_S.
5. Unless the adversary can compute discrete logarithms, the output cleartext is m^*.

Figure A-2: CS-TDec: Threshold Cramer-Shoup Decryption Protocol

As we mentioned before, our threshold Cramer-Shoup decryption protocol CS-TDec uses the Canetti-Goldwasser method [CG99], which was originally presented in the static adversary model. In the threshold key generation protocol CS-DKG, the players created secret-sharing data-structure RVSS-data for secret keys a, b, c, d, e. To decrypt ciphertext (x, y, w, v), the players perform a distributed coin-flip protocol to select a random secret

$s \in \mathbb{Z}_q$. Now the players need to generate a public output $m = w/(x^e(v/v')^s)$, where $v' = x^{a+c\sigma}y^{b+d\sigma}$, i.e. they need to *extract* value $m = wv^{-s}x^{s(a+c\sigma)-e}y^{s(b+d\sigma)}$ from the sharings of a, b, c, d, e, and s. This extraction is done via the shared multiplication protocol and a variant of the shared exponentiation protocol.

As in the sharing of a refresh polynomial (see Section 4.3.2), from the secret-sharing structures RVSS-data$[a]$ and RVSS-data$[b]$, each player can locally obtain a secret-sharing structure RVSS-data$[k_1a + k_2b]$ for any constants $k_1, k_2 \in \mathbb{Z}_q$ (compare also Figure 4-15). The players need to multiply their secret information in RVSS-data$[a]$ by k_1 and the secret information in RVSS-data$[c]$ by k_2, and add them. The corresponding operations on the public information are an exponentiation to k_1 or k_2 and then a multiplication. In CS-TDec each players does that in Step 2 to get a sharing of $a + c\sigma$ and $b + d\sigma$ from the sharings of a, c, b, d. Next the players employ two parallel instances of the multiplication protocol to get the sharing of $r = s(a + c\sigma)$ and $z = s(b + d\sigma)$ from the existing sharings of the factors s, $(a + c\sigma)$, and $(b + d\sigma)$. Now the players hold the sharings of $-s$, $r = s(a + c\sigma)$, e, and $z = s(b + d\sigma)$, and they need to compute $m = w^1v^{-s}x^rx^ey^z$. This can be done by another variant of the exponentiation protocol, just as values $A = g_1^a g_2^b$ were computed from the sharings of a and b in the Cramer-Shoup distributed key generation protocol CS-DKG. This time each player uses its additive shares s_i, r_i, e_i, z_i and publishes its part $m_i = w^1v^{-s_i}x^{r_i}x^{e_i}y^{z_i}$ of m. The proof that each P_i publishes the correct value is again accomplished via a simultaneous proof protocol using the ZKPK proof system of equal representation, because shares s_i, r_i, e_i, z_i all have corresponding public verification values $F_{s_i}(0)$, $F_{r_i}(0)$, $F_{e_i}(0)$, and $F_{z_i}(0)$ (created via RVSS for s_i's and e_i's and via Mult-opt for r_i's and z_i's).

The simulation algorithm which shows that this protocol is single-inconsistent-player-simulatable follows the protocol during the secret-sharing of s and during the multiplication step which creates the sharing of $r = s(a+c\sigma)$ and $z = s(b+d\sigma)$. The "extraction" step when the players generate $m = wv^{-s}x^{r-e}y^z$ is simulated in a way very similar to the simulation of the exponentiation protocol Ad-Exp. Note that in the proof of security of the resulting threshold Cramer-Shoup cryptosystem, the simulator first simulates the threshold Cramer-Shoup key generation algorithm CS-DKG on input (g_1, g_2, A, C, E), a random Cramer-Shoup public key given by a decryption oracle that implements an instance of the underlying Cramer-Shoup cryptosystem. These keys are then passed as an input to the simulation of the threshold Cramer-Shoup decryption protocol CS-TDec, together with input ciphertext (x, y, w, v, σ) given by the adversary, and the decryption m^* received from the decryption oracle. This m^* is distributed as a random number in G_q if $v \neq v' = x^{a+c\sigma}y^{b+d\sigma}$, or if $v = v'$ then $m^* = w/x^{e^*}$ where $e^* = \log_{g_1} E$ (we denote $\log_{g_1} E$ as e^* rather than e because it is unknown to the simulator).

The public Cramer-Shoup key correspond to some random secret key $(a^*, b^*, c^*, d^*, e^*)$ unknown to the simulator, and the decryption m^* corresponds to a random value s^* in $\mathbb{Z}_q$. By the adaptive secrecy property of RVSS, the sharing of RVSS-data$[s]$ in the simulation agrees with this value s^*. Similarly by the secrecy property of the multiplication protocol, values $s^*(a^* + c^*\sigma)$ and $s^*(b^* + d^*\sigma)$ agree with Step 3 of the simulation. Finally, as in the proof of secrecy of Ad-Exp, we argue that the last step of the simulation produces the same view as the last step of the protocol. Even though the sets of additive shares $\{r_i\}_{P_i \in Good \setminus \{P_S\}}$ and $\{s_i\}_{P_i \in Good \setminus \{P_S\}}$ are not distributed in the same way in the simulation and in the protocol

(e.g. values r_i in the random execution of the protocol should be products of values $f_s^*(i)$ and $f_b^*(i) + f_d^*(i)\sigma$, while in the simulation they are products of values $f_s(i)$ and $f_b(i) + f_d(i)\sigma$), because values $\{s_i\}_{P_i \in Good \setminus \{P_S\}}$ are uniformly distributed, values $\{m_i\}_{P_i \in Good \setminus \{P_S\}}$ produced in Step 3 of the simulation have the same distribution as in a random execution of the protocol that outputs m^*. It follows that value m_S^* is distributed correctly as well, and that the view of the simultaneous proof in Step 3 is as in the protocol too.

Appendix B

Proactive Security: Extensions to Stronger Adversarial Models

In this appendix we describe an extension of threshold cryptosystems to *proactive* cryptosystems, where the resilience of threshold solutions is enhanced by several self-healing protocols that counteract the potential advantage which the adversary might gain by corrupting servers. When a threshold cryptosystem employs such proactive protocols then the attacker can break the system only if he breaks into more than a tolerated threshold of locations in either a short period of time or in such a way that his break-ins remain undetected.

To explain the level of protection offered by proactive cryptosystems, we consider two models of adversarial behavior, both of which extend the capabilities of the threshold adversary, the *mobile* and the *creeping* adversarial models.The threshold adversary we consider in Chapters 4-6 can corrupt no more than t servers *throughout the system operation.* Both the mobile and the creeping adversaries model strictly stronger attacks against a group of servers. Formally, in the *mobile* model time is divided into time-periods of any duration, e.g. an hour, a day, or a week. The adversary can then choose a different subset of t corrupted servers in every time period. In the *creeping* model, the adversary can corrupt all players except of two, but we assume that each adversarial corruption is detected by the remaining uncorrupted players, and that the time between corruptions is long enough to allow these players to complete certain proactive "self-healing" protocols.[1] In the creeping adversarial model we also assume that the human operators can add a new participating server, for example because the adversary has been purged from a corrupted server, or simply because a corrupted server has been replaced with a new server. The adversary should then be able to corrupt such server as well, and the threshold scheme must remain secure as long as there are any two non-corrupted servers in operation. We assume that when the human operators add a new server, they also establish authenticated communication channels between this new server and the rest. Note that we assume a similar establishment of authenticated

[1] As stated above, the creeping model is not strictly stronger than a threshold model, because in the latter we make no conditions on the detectability of corruptions or the length of a delay between one corruption and the next. We can say formally that in a creeping model the adversary makes either t corruptions with no conditions attached, or he makes more corruptions but they have to obey the above conditions.

channels during the *initialization* of a threshold scheme.

We call schemes that are secure in either of these two adversarial models "*proactive*".[2] Such schemes provide better security especially for long-lived systems, where the adversary might have enough time to eventually corrupt almost all participating players. However, as long as these attacks are detectable and/or do not happen all at once – which is modeled, albeit differently, by both the mobile and the creeping adversarial model – the currently uncorrupted players can "proactively" perform various self-healing protocols that neutralize the adversary's potential advantage gained from corruptions staged so far. In fact, the proactive defenses used in the mobile and the creeping models should be implemented and used together, so that the resulting scheme is secure against combinations of mobile and creeping adversarial behaviors.

The mobile adversary model was introduced in the work of [OY91] (see also [CH94]). Subsequently, [HJKY95] showed how to share a secret in this model, and [HJJ+97] showed how to perform in this model threshold operations in the discrete-log based setting. We sketch these protocols in Section B.1 below. The main technique which upgrades a threshold cryptosystem and makes it resistant to a mobile attacker is a periodical execution of a protocol that re-randomizes the secret-sharing of the secret key maintained by the threshold cryptosystem, and erases the shares associated with the previous secret-sharing.

The creeping model has not been, to our knowledge, formally addressed before, but the protective measure that we employ to fight such adversary is a threshold-decrease protocol which was independently considered in [DJ99] and [MSNW99]. Such protocol should be executed by the uncorrupted players whenever they detect that some other player has been corrupted. Similarly, when a new server is *added* to the distributed cryptosystem, once the proper means of authentication are established between this new server and the remaining ones, the players should run a protocol that increases the threshold of the underlying secret-sharing and gives this new server its valid share. We sketch both protocols in Section B.2 below.

We stress that a distributed cryptosystem should have the capability to use all the above proactive self-healing protocols, to achieve security in the presence of attacks that combine the adversarial patterns modeled formally by the mobile and the creeping adversaries.

We remark that the essential part of all proactive protocols is that the participating servers *erase* all their local secret data except of the new shares of the secret key. To neutralize the knowledge the adversary gains at time T by corrupting a group of servers $Bad_T \subset \{P_1, ..., P_n\}$, in a subsequent execution of any proactive self-healing protocol, each uncorrupted server must erase the local state it had at time T. Otherwise, assuming that $|Bad_T| = t$, the adversary who corrupts such server afterwards will learn at least as much as he would by corrupting more than t servers in a regular threshold scheme, and the scheme would be broken. We can therefore offer resistance against mobile and creeping adversaries only in the *erasure-enabled* model of computation.

[2] As we mentioned in the introduction, when proactive systems were introduced in [OY91, CH94, HJKY95], this term was used to designate resistance solely to the mobile adversary, and this is also how we use this term in the research review in Section 1.4.

B.1 Measures against the Mobile Adversary

To upgrade resistance from threshold to mobile adversary, the time is divided into time periods of an arbitrary length, and the participating servers invoke a secret-sharing re-randomization protocol Update at the beginning of every such time period. This protocol guarantees security against a mobile adversary given that all corruptions are passive, i.e. the adversary only looks at the memory of a corrupted player, but does not change its algorithm. To extend the resistance also to the malicious faults, the players must also run a protocol that detects a loss or a corruption of local shares, and a reconstruction protocol that reestablishes their proper shares. Each player can detect that its share has been corrupted, by verifying it against the public information. (Recall that in the Pedersen secret-sharing, each local share $\alpha_i, \hat{\alpha}_i$ can be verified against the public verification information $F_a(i)$. See Figure 4-7, page 67.) Share recovery can be performed with protocol Recon of Figure 4-22, page 119.

The Update protocol itself consists of joint sharing of a random *refresh polynomial*, i.e. the "zero-sharing" protocol ZVSS (see Section 4.3.2). If the secret key a of a threshold cryptosystem is kept with data structure RVSS-data[a], an execution of ZVSS creates a data structure ZVSS-data[b], and each server can then compute its part of the new data-structure RVSS-data'[a] which shares the same key a with a new random secret-sharing polynomial. (This update is described in Section 4.3.2.) As we argued in Section 4.3.2 and 5.1.1, such update protocol ZVSS is robust and secret in both the static and the adaptive adversarial models. Indeed, if a threshold signature scheme is upgraded by running such Update protocol at the beginning of every time period, then the simulator of ZVSS can be built into the simulation argument that proves the unforgeability of this threshold scheme (as in Theorem 3, page 106), to prove that the resulting scheme remains unforgeable in the mobile adversary model. Similarly, the knowledge extractor of ZVSS can be built into the extraction argument that proves the robustness of a threshold scheme, to show that the scheme remains robust in the presence of a mobile adversary.[3]

Note that from the robustness properties of ZVSS (Lemma 9) it follows that under the discrete logarithm assumption, the outputs of protocol Refresh on input RVSS-data[a] form a *correct* data-structure RVSS-data'[a] (i.e. data which satisfies the correctness properties of Figure 4-7) except for probability negligible in the security parameter.

B.2 Measures against the Creeping Adversary

To defend the system against a creeping adversary, the uncorrupted players can proactively trigger protocols Reduce and Increase which re-randomize and change the threshold of the underlying secret-sharing. Both protocols can be thought of as variants of the Update protocol discussed above. The Reduce protocol should be employed by the uncorrupted servers when they detect that some other server is corrupted. By re-randomizing the secret-sharing and reducing the degree of the secret-sharing polynomial, the remaining players maintain an optimal resilience to the adversary. Similarly, when the new server is added to the system, instead of just giving it a new valid share, the fault-tolerance is maximized

[3] For an elaboration of this argument we refer the reader to [HJJ+97].

Protocol to Refresh a Secret-Sharing RVSS-data$[a]$

Inputs: secret-sharing data-structure RVSS-data$[a]$
Output: secret-sharing data-structure RVSS-data$'[a]$

1. Players perform ZVSS on inputs an instance (p, q, g, h) in RVSS-data$[a]$ and obtain an ensemble of outputs ZVSS-data$[b]$
2. Each player P_i computes its new shares of a by adding secret shares in RVSS-data$[a]$ to the corresponding secret shares in ZVSS-data$[b]$, i.e.:
   - P_i computes $\alpha'_i = \alpha_i + \beta_i$, $\hat{\alpha}'_i = \hat{\alpha}_i + \hat{\beta}_i$ (but $a'_i = a_i$ and $\hat{a}'_i = \hat{a}_i$). Furthermore, it computes $\alpha'_{ji} = \alpha_{ji} + \beta_{ji}$ and $\hat{\alpha}'_{ji} = \hat{\alpha}_{ji} + \hat{\beta}_{ji}$ for each $P_j \in Qual_a$. (For each P_j in $Qual_a$ but not in $Qual_b$, we take the default constant polynomials $f_{b_j}(z) = f_{\hat{b}_j}(z) = 0$.)
   - Note that this defines $f'_a(z) = f_a(z) + f_b(z)$ and $f'_{\hat{a}}(z) = f_{\hat{a}}(z) + f_{\hat{b}}(z)$, and for each $P_i \in Qual_a$ it defines $f'_{a_i}(z) = f_{a_i}(z) + f_{b_i}(z)$ and $f'_{\hat{a}_i}(z) = f_{\hat{a}_i}(z) + f_{\hat{b}_i}(z)$.
   - The new verification functions are $F'_a(z) = F_a(z) * F_b(z)$, and $F'_{a_i}(z) = F_{a_i}(z) * F_{b_i}(z)$ for each $P_i \in Qual_a$ (where the default for P_i in $Qual_a$ but not in $Qual_b$ is $F_{b_i}(z) = 1$). The set $Qual'_a$ is set as $Qual_a$.

Figure B-1: Refresh: Distributed Refreshment of a Secret-Sharing

if the degree of the secret-sharing polynomial is increased. However, both the Reduce and the Increase protocols should be executed only when the new total number of participating players is odd, so that the resilience threshold t is a maximal integer such that $2t + 1$ is less or equal to n, the current total number of players.

There are many ways to change the degree of the secret-sharing polynomial (compare [DJ99, MSNW99]), but it seems that the simplest solution is to modify the Update protocol discussed in Section B.1 above, by running the following variants of the ZVSS protocol. (We use the notation of Section 4.3.2 and of Figure 4-6, page 65.)

Assume that in the current secret-sharing data structure RVSS-data$_t[a]$ (which shares the secret key a with a polynomial of degree t), player P_i knows its secret-sharing polynomials $f_{a_i}(z) = \sum_{k=0..t} c_k z^k$ and $f_{\hat{a}_i} = \sum_{k=0..t} \hat{c}_k z^k$ (see also Figure 4-7, page 67). Let $Qual$ be the set of players that participated in creation of RVSS-data$_t[a]$, let $P_n \in Qual$ be a newly corrupted player and let $Qual' = Qual \setminus \{P_n\}$. In the Reduce protocol, the players perform ZVSS modified so that each player picks its secret-sharing polynomials $f_{b_i}(z) = \sum_{k=1..t} d_k z^k$ and $f_{\hat{b}_i} = \sum_{k=1..t} \hat{d}_k z^k$ such that $d_t = -c_t$ and $\hat{d}_t = -\hat{c}_t$. This can be publicly checked by verifying that $B_i^{(t)} = (A_i^{(t)})^{-1}$, where $A_i^{(t)} = g^{c_t} h^{\hat{c}_t}$ and $B_i^{(t)} = g^{d_t} h^{\hat{d}_t}$ are a part of the verification information F_{a_i} and F_{b_i} in RVSS-data$_t[a]$ and ZVSS-data$_t[b]$. The players must then also locally update their shares (and the public verification information) by subtracting out their shares of the polynomials $f_{a_n}, f_{\hat{a}_n}$ which had been secret-shared by the corrupted player P_n. The players in $Qual'$ will then hold their appropriate parts of RVSS-data$'_{t-1}[a]$,

a random secret-sharing of the same secret a, but with a polynomial of degree $t-1$. This protocol is as robust as ZVSS, and it can be simulated in the same manner.

The Increase protocol is even simpler. The players run a modified ZVSS in which they share random secret-sharing polynomials of degree $t+1$ (with their free coefficient equal to zero). The resulting secret-sharing polynomial is a random $t+1$ polynomial that shares the original secret. Again, the robustness of secrecy arguments are the same as for the ZVSS protocol.

Appendix C

Adaptive Erasure-Free Threshold RSA

The adaptive threshold techniques of Chapter 5 and the adaptive erasure-free improvements of Chapter 6 can be applied to threshold schemes in other settings than the discrete-log setting of schemes like Cramer-Shoup or DSS. In particular, the single-inconsistent-player simulation technique can be applied to modify the statically secure Threshold RSA solution of [Rab98] to yield adaptive threshold RSA protocols. This application of adaptively-secure threshold techniques appeared in [CGJ+99] and was improved in [JL00]. The resulting protocol is also insecure-channels-enabled, and therefore, using the simultaneously secure encryption of Chapter 6, it can be efficiently executed in the adaptive erasure-free model. This application of the simultaneously secure encryption appeared in [JL00]. We sketch this protocol below.

In the RSA signature scheme and cryptosystem [RSA78], the public key is a pair (N, e), where $N = pq$, p and q are two large random primes, e is a number s.t. $\gcd(e, \phi(N)) = 1$, and $\phi(N) = (p-1)(q-1)$. The secret key is a number d s.t. $de = 1 \bmod \phi(N)$. The RSA signature on message m in $[1, ..., N-1]$ is a value $S_m = m^d \bmod N$.[1]

A threshold signature scheme consists of two protocols: Threshold Key Generation and Threshold Signature Generation. In [Rab98] solution, the first protocol requires a trusted dealer (see Section 2.4 and the discussion of threshold RSA schemes in Section 1.4). The [Rab98] solution requires a public element g_0 of high order in $\mathbb{Z}_p{}^*$, a constant $L = n!$, and an element $g = g_0^{L^2} \bmod N$. In the threshold key generation protocol, the dealer picks a public key (N, e) and a secret key d, and shares d *additively* among all the participating players by giving to each P_i its share d_i, a random number in $[-nN^2..nN^2]$, and broadcasts a public value $d_{public} = d - \sum_{i=1}^{n} d_i$. For robustness, each d_i is also polynomially secret-shared among all the players with Feldman Verifiable Secret Sharing (see Figure 4-1, page 52) modified to share an integer in the $[-nN^2..nN^2]$ range (instead of an element of a prime group $\mathbb{Z}_q$), with the verification values g^{c_i} computed modulo N. (We use the notation of Figure 4-1, page 52. See also the full description of the "Feldman-Z_n-VSS" protocol in [Rab98].)

[1] In this subsection the operations are no longer implicitly carried out modulo p and q. The operations are in the integers unless otherwise indicated.

Note that this threshold RSA protocol employs an "additive sharing" of the secret key d, and thus a threshold RSA signature protocol of [Rab98] is very similar to the "additive exponentiation" protocol Ad-Exp of Section 5.2.1. To sign message m, each player broadcasts $\sigma_i = m^{d_i} \bmod N$, and the signature is computed as $\sigma = \sigma_{public} \prod_{P_i} \sigma_i \bmod N$, where $\sigma_{public} = m^{d_{public}} \bmod N$. To achieve robustness in the signature generation, the dealer publishes in the key generation stage a token partial signature $w_i = g^{d_i} \bmod N$ for each player P_i on message g. In the signature protocol each player P_i proves that $\log_m \sigma_i = \log_g w_i$ with a ZK proof of [FGY96, GJKR96a].

To make this protocol adaptively secure, we first replace Feldman VSS with a perfectly-secure Pedersen VSS. The extension of Pedersen VSS to sharing over integers is the same as the [Rab98] extension of Feldman VSS. Every secret variable a has its associated shadow variable $\hat{a}$ chosen from the same probability distribution, we require another publicly known element $h = h_0^{L^2} \bmod n$ where h_0 is an element of high order, and the verification information is computed as $g^a h^{\hat{a}} \bmod N$ instead of $g^a \bmod N$.

Replacing Feldman's VSS with Pedersen's VSS implies that the publicly available verification information is of a form $D_i = g^{d_i} h^{\hat{d}_i} \bmod N$ instead of $w_i = g^{d_i} \bmod N$. Therefore, the ZK proof used by the players to prove that their partial signatures $\sigma_i = m^{d_i}$ are correct must be modified. The new proof is the proof of knowledge of (equal) representation of D_i in bases (g, h) and of σ_i in bases $(m, 1)$, where all the exponentiation operations are carried out modulo N. The ZKPK system in Figure F-2, page 219, for proving equality of representation, can be adapted to the RSA setting, similarly to the way the ZK proof of equality of discrete logarithms of [CA89, Cha90] was adapted to the RSA setting by [GJKR96a]. The argument that the resulting protocol is insecure-channels-enabled is similar to the corresponding argument for the Ad-Exp protocol (see Lemma 32, page 166).[2]

[2]We note that the adaptively-secure threshold RSA proposed in [CGJ+99] was more cumbersome than the one outlined above. In particular, it required a refreshment of the sharing of the secret key after each execution of a threshold signature protocol. The adaptive erasure-free techniques of [JL00] eliminated the need of such refreshment by improving the single-inconsistent-player simulation technique. For further explanation see footnote 7, page 137.

Appendix D

Threshold Cryptosystems vs. General MPC Protocols

As we discussed in Section 1.1, threshold schemes are implied by general secure multi-party computation (MPC) results. In this section we explain how a (t, n)-threshold scheme can be implemented with a general (t, n)-threshold MPC protocol. We also lower-bound the costs of the resulting protocols for threshold DSS, Cramer-Shoup, and RSA cryptosystems.

A (t, n)-threshold *secure multi-party computation* protocol is a distributed protocol that allows n players $P_1, ...P_n$ to securely evaluate any randomized function. Such function can be encoded as a Boolean or arithmetic circuit, which takes private inputs x_i of each player P_i, public input X, and a random string r, and outputs $(y_1, ..., y_n;\ Y) = f(x_1, ..., x_n;\ X;\ r)$, where y_i is a private output of player P_i and Y is a public output. By *secure* computation we mean that as long as no more than t of the participating players are corrupted, the protocol maintains the following two properties: (1. Secrecy): The dishonest parties do not learn anything about the private inputs of the honest parties apart of what is revealed to them by the output of the circuit; and (2. Robustness): The dishonest parties cannot disable the honest ones from evaluating circuit f on their inputs x_i, the public input X, and a true random bit string r. The most that the corrupt players can do is to arbitrarily choose their own inputs x_i. Such secure computation protocols were introduced in the work of [Yao82, GMW87].

It is easy to see that any threshold signature scheme, as well as any threshold cryptosystem, can be implemented using secure computation protocols. First we must recall the notion of *secret sharing*. Let S be a function mapping k-bit strings to an ensemble of private and public data of n players. We say that for every x and every $s \in S(x)$, $s = (x_1, ..., x_n;\ X)$ is a *secret-sharing* of x if the following two properties are met: (1) (*Secrecy*) Any t dishonest players learn nothing from their shares x_i and the public info X about x; and (2) (*Robustness*) x is uniquely reconstructible from s. More formally, there exists a polynomial-size circuit f such that for any x and any $s = (x_1, ..., x_n;\ X)$ in $S(x)$, if no more than t dishonest players substitute their inputs x_i with arbitrarily chosen values x_i', then f evaluated on public input X and private inputs x_i for each honest player P_i and x_i' for each dishonest P_i, outputs x. Both these properties can be formalized relative to

either an all-powerful or a computationally bounded adversary.[1] For example, in Shamir's secret sharing scheme [Sha79] which we use in this thesis, some k-bit prime q is fixed, and then $S^{(q)}_{\text{Shamir}}(x)$ is a set of strings $(f(1), ..., f(n);\ \emptyset)$ where $f(z)$ is any t-degree polynomial modulo some q s.t. $f(0) = x$. Secrecy is satisfied if secrets x are uniformly distributed numbers modulo q. If $t < n/3$ then robustness can be guaranteed with any error-correcting codes, for example using Berlekamp-Welch decoder [BW].[2]

To see how a threshold scheme can be implemented with general secure computation protocols and any secret-sharing scheme, take for example any signature scheme Gen, Sig, Ver, where Gen is a key-generation algorithm, Sig is a signing algorithm, and Ver is a verification algorithm. Since these must be probabilistic polynomial-time algorithms, each can be computed by a poly-size randomized circuit of the kind we discussed above. The threshold version of this signature scheme would be composed of n-party protocols TGen and TSig and the same verification algorithm Ver. Protocol TGen, which allows the n players to generate a key for a threshold scheme, is a secure computation protocol for function $f_1(\emptyset, ..., \emptyset;\ \emptyset;\ r_1) = (x_1, ..., x_n;\ (X, y))$ where (x, y) is a random private/public key pair generated by Gen on randomness r_1, and $(x_1, ..., x_n;\ X)$ is any element in $S(x)$. Protocol TSig, which allows n players to sign any message m, is a secure computation protocol for function $f_2(x_1, ...x_n;\ (X, y, m);\ r_2) = (\emptyset, ..., \emptyset;\ u)$, where u is a signature on m generated by Sig on randomness r_2, on public key y, and on a private key x which is secret-shared via $(x_1, ..., x_n;\ X)$. The robustness and secrecy properties of secure computation guarantee that the proper public keys and proper signatures are always output, and that the adversary does not learn to forge signatures because he does not learn anything from the multi-party protocol apart of the final signatures $u^{(i)}$ on messages $m^{(i)}$ submitted for signing, of the public key y, and of the public information X and the t shares x_i of x, from which he learns nothing about x by the secrecy property of the secret-sharing scheme.

General MPC protocols for n players appeared, among other works, in [GMW87, BGW88, CCD88, RB89, BMR90, MR91, Bea91, BH92, BOCG91, CFGN96, CDD+99, Can00]. These solutions were increasingly more efficient, and provably secure in increasingly stronger computational, communicational and adversarial models. All the above works are based on protocols that compute a single arithmetic or Boolean gate in a threshold setting. Such protocols can be composed to form a protocol that securely computes any arithmetic or binary circuit. However, the generality of these results implies certain inefficiencies when such pro-

[1] Note that the robustness property implies that as long as t is below certain threshold then there exists a secure computation protocol that reconstructs x on public input X and the private inputs x_i of honest parties, given that $(x_1, ..., x_n;\ X) \in S(x)$. For example, using protocols of [GMW87], such protocols exist for $t < n/2$ in a similar computational, communicational, and adversarial model as we describe in Section 2.1. We skim the exact model of [GMW87] here, so that to concentrate on the main idea of secure computation protocols.

[2] In Section 4.2.3 we further discuss Shamir's secret-sharing. We also introduce there Feldman's *verifiable* secret sharing [VSS]. To clarify the notation we use here, we can describe Feldman's VSS as follows. Let p and q be two primes s.t. q divides $p-1$ and $|q| = t$. Let g be a generator of a q-order subgroup $G_q = \{g^0, ...g^{q-1}\}$ of the multiplicative group Z^*_p. Elements s in $S^{(p,q,g)}_{\text{Feldman}}(x)$ look like $s = (x_1, ..., x_n;\ (p, q, g, g^{a_0}, ..., g^{a_t}))$, where each $x_i = f(i) \bmod q$ for some t-degree polynomial $f(z) = \sum_{j=0}^{t} a_j z^j \bmod q$ such that $f(0) = a_0 = x \bmod q$, and all the exponentiations $g^{a_0}, ..., g^{a_t}$ are modulo p. Robustness is guaranteed as long as $t < n/2$, and secrecy is preserved in the sense that t dishonest players learn nothing about x beyond what is revealed by $g^{a_0} = g^x \bmod p$.

tocols are used to implement threshold cryptosystems for schemes like DSS, Cramer-Shoup, or RSA, which all have complex circuit representations.

In particular, all the above results except of [BMR90] yield protocols with the number of communication rounds lower-bounded by the depth of an *arithmetic* circuit that implements the computed function, while the [BMR90] protocol requires every player to perform a cryptographic operation per each gate of a *binary* circuit that implements the computed function. Public-key cryptosystems like DSS, RSA, an Cramer-Shoup, all involve a modular exponentiation operation, which to the best of our knowledge requires an arithmetic circuit of $\Omega(k)$ depth, or a binary circuit of $\Omega(k^2)$ depth, where k is the security parameter. Thus using the above general MPC protocols to compute such operations in a threshold setting requires either $\Omega(k)$ communication rounds or $\Omega(k^2)$ cryptographic operations per player. Such costs are prohibitive for the applications of threshold cryptosystems that we consider. In contrast, the threshold cryptosystems we present in Chapters 4-6 all have small constant number of rounds, and each player performs at most $cn(k + n\min(n\log n, k))$ modular multiplications, where n is the number of participating players, and c is a small constant.

Appendix E

Insecure Variants of the Joint-Feldman Protocol

In Section 4.2.3 we described the basic Joint-Feldman protocol and explained that it does not make a secure DKG protocol. In fact, many variants of this protocol have appeared in literature with the claim that they form a secure DKG protocol. Here we describe several such variants, which extended the basic Joint-Feldman protocol by: Signatures on shares; Commitments to y_i; Committing encryption on the broadcast channel; Committing encryption with reconstruction; "Stop, kill and rewind" methodology. We show that all of them fail to achieve the correctness property (C3) and the secrecy (or simulatability) requirement of the definition of a secure DKG scheme (see Section 4.2.2).

Signatures in Share Distribution. In the original protocol proposed in [Ped91b] each player (acting as dealer) signs the shares he distributes in Step a. This was supposed to aid the honest parties in disqualifying dishonest dealers, because a party which receives an incorrect share (i.e. not satisfying the verification equation) could prove that the dealer is dishonest by broadcasting this share with the dealer's signature. Indeed, the original protocol from [Ped91b] uses this simplified procedure in Step c for disqualifying dishonest dealers: a player is disqualified if one valid complaint (i.e. incorrect share with valid signature) is broadcast against him.

We note first of all that with this modification even a single execution of Feldman's protocol fails to be a VSS. Indeed if a dealer gives neither a correct share nor a signature to some player, the player cannot prove the dealer wrong by the above method.

Thus the joint execution of n such protocols may fail to produce a correct sharing at all. Also notice that the basic idea of the attack in Section 4.2.3 still works (P_1 can give to P_2 two signed shares, a correct one and an incorrect one and all other players consistent shares with valid signatures, and then using only P_2 decide if he wants to be disqualified or not).

Initial Commitment Stage. Another difference between the original protocol presented in [Ped91b] and Joint-Feldman of Figure 4-2 is the use of an initial commitment stage in [Ped91b], so that the modified protocol looks as follows:

1. Each P_i chooses $x_i \in Z_q$ uniformly at random, computes $y_i = g^{x_i} \bmod p$, chooses a random string r_i, and broadcasts a commitment $C_i = C(y_i, r_i)$ to all members.

2. Each P_i opens the commitment C_i by broadcasting proper y_i and r_i. A player who fails to do so is disqualified.

3. The players now follow the Joint-Feldman protocol of Figure 4-2 among the non-disqualified players, using the above picked x_i as inputs to be shared.

This initial commitment stage is supposed to force the players to choose their random secrets x_i independently from one another in order to ensure the uniform distribution of the final sum x, but it fails to do that. Indeed the attack in Section 4.2.3 is not based on how dishonest players choose their contribution, but rather on their ability to pull such contribution out of the lot after seeing the honest players' contributions. Thus the dishonest players can follow the protocol, including the extra commitment stages, and still nothing stops them from carrying out the attack described in Section 4.2.3. If P_i decides to do so, he can get his value disqualified and the remaining players will not add y_i to the public key y even if it matches the initial commitment C_i.

Committing Encryption on a Broadcast Channel. The attack described in Section 4.2.3 seems to rely on the assumption that each pair of players is connected by a private channel which allows P_1 and P_2 to "lie" to the other players about the share P_1 sent to P_2. This is possible in several implementations of private channels, for example physically untappable ones (e.g. lead pipes) or private channels built out of one-time pad encryption.

It would seem that using some form of encryption to implement private channels may help in thwarting the attack, since a complaining player may be required to "open" the encrypted message he received to prove that the incorrect share really came from the dealer. This strategy was followed in a modification of Joint-Feldman used for proactive secret sharing [HJKY95, HJJ+97]). However we prove now that this is not the case. The attack can still be carried out even if players exchange messages using encryption.

Let's assume that the players communicate simply via the broadcast channels. Private communication is achieved via a public key "committing" encryption scheme, i.e. an encryption scheme that not only ensures the confidentiality of a message but also commits the sender to the message being sent[1]. In other words, to send a message m secretly to recipient R, the sender picks a random vector r and broadcasts a ciphertext $E = ENC_R^r(m)$. We assume that the recipient using the secret key of ENC_R can recover both m and r, and so he is able to prove that $E = ENC_R^r(m)$ to all other players by revealing m, r.[2]

The Joint-Feldman variant using such committing encryption introduces the following modifications to the protocol of Figure 4-2: Players send the shares α_{ij} in Step a by broadcasting their committing encryptions $E_{ij} = ENC_{P_j}^{r_{ij}}(\alpha_{ij})$. Then in Step b a valid complaint has to consist of a pair (α_{ij}, r_{ij}) such that s_{ij} is an incorrect share and $E_{ij} = ENC_{P_j}^{r_{ij}}(\alpha_{ij})$. In Step c we disqualify a player if there is a single valid complaint against him.

Unfortunately, the protocol modified in such a way is still insecure. P_1 sends to P_2 a bad share α_{12}^* via the committing encryption in Step b. At the same time however it chooses

[1]Contrast this with the *deniable encryption* [CDNO99] where the sender or the receiver may lie about the content of encrypted messages.

[2]Not all encryption schemes allow the receiver to recover both the message m and the random vector r. However what we are arguing here is that even if the committing encryption provided the random vector recovery, this Joint-Feldman variant still cannot be made secure.

the sharing polynomial so that the correct share α_{12} is some fixed value on which P_1 and P_2 agreed earlier (or in other words think of P_1 and P_2 as the same entity, the adversary). If the adversary wants P_1's contribution to be "in", P_2 will not complain in Step c, but will use α_{12} in all further computation. If the adversary wants P_1 disqualified, P_2 broadcasts (α^*_{12}, r_{ij}).

Committing Encryption with Reconstruction. In the Joint-Feldman with committing encryption variant described above, we are following the policy of disqualifying a player P_i as soon as a single valid complaint is filed. Yet, it seems that as the values A_{ik}, $0 \leq k \leq t$ are broadcast by player P_i in Step a of Joint-Feldman and there is a fixed polynomial and a fixed secret, if we have enough shares to reconstruct it we would not need to disqualify the dealer based on a single complaint. We could follow this alternative policy instead: If a valid complaint is filed against P_i, all other players open the encrypted shares P_i sent them: If more than $t+1$ of them match the verification equation then publicly reconstruct x_i, otherwise disqualify P_i.

But even for this policy we show a strategy for the adversary to decide if (say) P_1 is disqualified or not, at a stage where he can influence the distributions. Assume $n = 2t+1$, P_1 sends correct encrypted shares to all the players (honest and dishonest) except for one honest player. This player will complain and everybody is required to open their encrypted shares. The honest players have only t matching shares, thus an additional one is required in order to incorporate P_1's value into the computation. Thus, the decision of whether the secret will be considered is again left in the hands of the adversary.

"Stop, Kill and Rewind" Procedure. Notice that in some of the above attacks against Joint-Feldman and its variants, the adversarial strategy involves a behavior on the part of some of the players which can be publicly identified as faulty. We could modify the Joint-Feldman protocol so that whenever some party is clearly faulty, the protocol stops, that party is excluded from the set of players, and the protocol is started from scratch in the smaller group of players. Note that this can happen at most t times.

However, the adversary can still skew the distribution of the outputs. He just follows the same strategy as in Section 4.2.3. The event that the protocol is repeated is conditioned on the fact that the adversary made a dishonest player visibly faulty. Thus the distribution of the final output is not necessarily uniform, even if the second repetition has a uniformly distributed output.

For the example in Section 4.2.3, the final output y ends with 0 if either

- α ended with 0 and P_1 was not disqualified (prob. 1/2).
- P_1 was disqualified (i.e. α ended with 1, which happens with prob. 1/2) and the output of the second run ends with 0 (prob. 1/2).

Thus the probability that in the presence of such an adversary the protocol outputs a value y which ends with 0 remains $1/2 + 1/2 * 1/2 = 3/4$.

The "stop, kill and rewind" procedure was employed in other variants of Joint-Feldman. For example, it was used in [HJKY95, HJJ+97] in the committing encryption variant. In each case a similar argument shows that the adversary can force the distribution of the resulting y not to be uniform.

Appendix F

3-Round Honest-verifier Public-coin Zero-knowledge Proofs of Knowledge

For completeness, we provide three known Three-round Honest-verifier Public-coin Zero-knowledge Proof of knowledge systems (THPZP's) used in our protocols. Each THPZP proof system is a two-party protocol which allows one party to prove to the other a knowledge of some special element. In this section we first define the THPZP proof system formally, and then present three THPZP systems used in the threshold protocols discussed in this thesis (see Section 4.4.1), namely the [Sch91] proof system for proving knowledge of discrete logarithm, a proof system for proving knowledge of representations and of equality of representations[1] (see the work of Brands [Bra99] or Camenisch [Cam98] and the references therein), and a [CD98] proof system for proving knowledge of committed values which are in a multiplicative relation.

Consider a polynomial-time computable relation $\mathcal{R}$, i.e. a set of pairs (y, w), where the length of w is bounded by a fixed polynomial in the length of y, for every $(y, w) \in \mathcal{R}$. Furthermore, there exists a polynomial-time algorithm which given a pair (y, w) decides if it belongs to $\mathcal{R}$. If $(y, w) \in \mathcal{R}$ then we call the element y a "public value", and w a "witness" of y in $\mathcal{R}$. A proof of knowledge for relation $\mathcal{R}$ is a protocol between two parties, a prover and a verifier, in which on the common input some public value y, the verifier decides if the prover knows the appropriate witness w for the value y in $\mathcal{R}$. If the prover has a noticeable probability of convincing the verifier to accept the proof then the witness w can be efficiently extracted from an interaction with such prover. (We provide a formal definition of a proof of knowledge in Definitions 27-28 below.) For example, in a proof of knowledge of discrete–logarithm, the relation $\mathcal{R}$ consists of pairs $((p, g, g^x \bmod p), x)$ where p is a prime, g is a generator of a multiplicative group ${\mathbb{Z}_p}^*$, and x is an element in $\mathbb{Z}_p$. Proofs of knowledge are related to proofs of language membership. If $\mathcal{R}$ is a polynomial-time computable relation then language $L_{\mathcal{R}} = \{y \mid \exists w \text{ s.t. } (y, w) \in \mathcal{R}\}$ is in NP. An element w s.t. $(y, w) \in \mathcal{R}$ is a witness of the membership of y in $L_{\mathcal{R}}$. In fact, every (zero-knowledge)

[1] Representation is defined in Section 3.2.

proof of knowledge for relation $\mathcal{R}$ is also a (zero-knowledge) proof of membership in $L_{\mathcal{R}}$.

A three-round honest-verifier public-coin zero-knowledge proof of knowledge system (THPZP)[2] for a polynomial-time computable relation $\mathcal{R}$ is specified by a triple of algorithms $P^{(1)}, P^{(2)}, V$ and a function $\mathcal{D} : \{0,1\}^* \to \{0,1\}^*$, which given a public value $y \in L_{\mathcal{R}}$ describes a distribution according to which the public coin is chosen by an honest verifier. It must be possible to sample the distribution $\mathcal{D}(y)$ in time polynomial in $|y|$. The prover, on public input a public value y, and on his private input a witness w s.t. $(y,w) \in \mathcal{R}$, picks some random bits r, and sends to the verifier a "commitment" message $M = P^{(1)}(y,w,r)$. The verifier then sends to the prover a random string R, which we call a "challenge", chosen uniformly in distribution $\mathcal{D}(y)$. The prover then sends back to the verifier its second message $m = P^{(2)}(y,w,r,M,R)$, called "response". Finally, the verifier applies a test $V(y,M,R,m)$ to determine whether to accept the proof.

All the THPZP proof systems we consider meet also the following further restriction:

Definition 26 (Discrete–Log Based THPZP) *We call a THPZP proof system* $(P^{(1)}, P^{(2)}, V, \mathcal{D})$ *for polynomial-time computable relation* $\mathcal{R}$ discrete–log based, *if every* $y \in L_{\mathcal{R}}$ *includes a description of a pair of primes* p,q *s.t.* q *divides* $p-1$ *and the length of* p *is bounded by a fixed polynomial in the length of* $|q|$, *and if the probability distribution* $\mathcal{D}(y)$ *from which the public coin is chosen is a uniform distribution over* $\mathbb{Z}_q$, *where* q *is included in* y.

We define the *proof of knowledge* proof system (Definitions 27-28) and the *honest-verifier public-coin zero-knowledge* property of a proof system (Definition 30). A THPZP is a proof system which satisfies both definitions. We then proceed to give three examples of THPZP's.

Intuitively, Definition 27 below, taken from [BG92, Gol95], says that first of all the proof system must be correct, i.e. the verifier V always accepts a proof if the prover follows the protocol and its private input is a correct witness w for the public value y. Secondly, if any algorithm P*, called the prover, can with some probability p convince the verifier V to accept a proof on common input y, and if this probability p is higher than certain "knowledge error" value $\mathcal{K}$, then we require that the machine P* "knows" the witness w s.t. $(y,w) \in \mathcal{R}$, and we formalize the notion of a Turing Machine knowing some value, by requiring that there exists an efficient algorithm $\mathcal{E}$, called "extractor", which can compute w, in expected time polynomially related to $1/(p-\mathcal{K})$, if $\mathcal{E}$ is given an oracle access to P*. The role of the knowledge error $\mathcal{K}$ is to account for the fact that in some proof of knowledge systems the prover P* might guess the appropriate answers in an interaction with the verifier V even without "knowing" the witness w. In particular, if P*'s probability of passing the proof is smaller or equal to $\mathcal{K}$ then there might be no extractor $\mathcal{E}$ that runs in expected time which is polynomially related to $1/p$. (See Lemma 37 for a simple example of a proof of knowledge system with the non-zero knowledge error.) Since P* is a TM machine, apart of the public input (i,y) it has some private input x and random input r. By an oracle access to P* we mean an oracle access to $\mathsf{P}^*_{(y,x,r)}$ for some fixed values x and r. Note also that the definition of a proof of knowledge does not restrict the cheating prover P* algorithm to execute in a (probabilistic) polynomial-time.

[2] See [BG92] for a history of research on proof of knowledge systems.

Definition 27 (Proof of Knowledge proof system) *A pair of interactive PPT algorithms* (P,V) *specifies a* proof of knowledge *proof system (with error* $\mathcal{K}(\cdot)$*) for a polynomial-time computable relation* $\mathcal{R}$ *if:*

1. *For every* $(y,w) \in \mathcal{R}$*, the verifier* V *always accepts in an interaction* (P,V) *on common input* y *and on* P*'s private input the witness* w*.*

2. *There exists a probabilistic algorithm* $\mathcal{E}$*, called "extractor", and a polynomial* $q(z)$*, such that for every* y *s.t.* $(y,w) \in \mathcal{R}$ *for some* w*, for every interactive algorithm* P* *and every private input* $x \in \{0,1\}^*$ *and random input* $r \in \{0,1\}^*$ *of* P*, *if the probability that verifier* V *accepts in an interaction* (P*,V) *on common input* y *and* P*'s *inputs* (x,r) *is equal to* $p(y,x,r)$ *and* $p(y,x,r) > \mathcal{K}(|y|)$*, where the probability is taken over the random coins of* V*, then* $\mathcal{E}$ *on input* y*, having an oracle access to* $\mathsf{P}^*_{(y,x,r)}$*, i.e. to* P* *running on fixed inputs* (y,x,r)*, outputs a witness* w *s.t.* $(y,w) \in \mathcal{R}$*, with the expected number of steps at most*

$$\frac{q(|y|)}{p(y,x,r) - \mathcal{K}(|y|)}$$

The above definition defines the complexity of the extractor in terms of expected number of steps. However, all the arguments for robustness of the threshold protocols we present in this thesis are in terms of probabilistic polynomial-time (PPT) algorithms. Therefore, it is technically easier for us if the complexity of the extractor in the proof of knowledge systems presented in this appendix is specified in terms of algorithms which have strictly upper bounded running time but are allowed to err with a negligible probability. Namely, we find the following definition useful:[3]

Definition 28 (Proof of Knowledge proof system (stronger formulation)) *A pair of interactive PPT algorithms* (P,V) *specifies a* (strong) proof of knowledge *proof system (with error* $\mathcal{K}(\cdot)$*) for a polynomial-time computable relation* $\mathcal{R}$ *if it satisfies the following variation of the Definition 27 above: The extractor* $\mathcal{E}$*, on input* y*, having an oracle access to* $\mathsf{P}^*_{(y,x,r)}$*, outputs a correct witness* w *s.t.* $(y,w) \in \mathcal{R}$ *with probability at least* $1 - 2^{-|y|}$*, and its running time is upper bounded by*

$$\frac{q(|y|)}{p(y,x,r) - \mathcal{K}(|y|)}$$

Clearly, a proof system which is a proof of knowledge in the sense of Definition 28 is also a proof of knowledge in the sense of Definition 27. Throughout this thesis we adopt the stronger formulation of a proof of knowledge, i.e. Definition 28.

The THPZP proof of knowledge systems, i.e. proofs of knowledge that have additional properties of being three-round honest-verifier public-coin zero-knowledge, play an important role in the threshold protocols we present in this thesis. Namely, in threshold protocols

[3]The proof-of-knowledge property of THPZP proof systems is used to argue the proof-of-knowledge property of the simultaneous proof protocol (Lemma 20, page 114).

conducted by a group of n servers, each server might need to prove some statement to each other server in the group. In Section 4.4.1 we argue that such n^2 instances of the THPZP proof system can be securely executed in parallel, using a single public coin picked by all the servers via a distributed coin-flip protocol, i.e. the joint secret-sharing and reconstruction protocols RVSS and RVSS-REC of Section 4.2.4. We call such parallel execution of the THPZP proof systems a "simultaneous proof protocol". However, to prove that the THPZP proof systems retain their proof-of-knowledge property when executed in parallel in this fashion, we seem to require the following additional property of the extractor that exhibits the proof-of-knowledge property of a (single execution) of a THPZP proof system. Namely, on input a public value y, the extractor can query the prover oracle $\mathsf{P}^*_{(y,x,r)}$ by sending to it only true random coins, picked according to distribution $\mathcal{D}(y)$ specified by the THPZP proof system. In Definition 29 below we define such extraction as *oblivious extraction*. We note that all the THPZP proof systems we present in this appendix have oblivious extractors.

Definition 29 (Oblivious Extraction) *Let a triple of algorithms $(P^{(1)}, V, P^{(2)})$ and a function $\mathcal{D}\{0,1\}^* \rightarrow \{0,1\}^*$ specify a THPZP proof system for relation $\mathcal{R}$. Let $\mathcal{E}$ be an extractor algorithm which exhibits the proof-of-knowledge property of this proof system. (See Definition 28.) We call this extractor* oblivious *if on input y it interacts with the oracle $\mathsf{P}^*_{(y,x,r)}$ by repeating the following procedure: The oracle sends its first message M, extractor $\mathcal{E}$ responds by choosing a random coin R according to distribution $\mathcal{D}(y)$, and the oracle responds with message m. The extractor collects transcripts $(y, M, R^{(i)}, m^{(i)})$ for each such interaction, and based on his collection of such transcripts, the extractor either creates another transcript or outputs a witness w for y in $\mathcal{R}$.*

Definition 30 below is a straightforward restriction of the zero-knowledge property to proof systems in which the verifier is honest and is only allowed to pick random coins and publish them.

Definition 30 (Honest-Verifier Public-Coin Zero-Knowledge proof system) *A pair of interactive PPT algorithms (P,V) and a function $\mathcal{D} : \{0,1\}^* \rightarrow \{0,1\}^*$, where $\mathcal{D}(y)$ is a probability distribution samplable in time polynomial in $|y|$, specify a* (perfect) honest-verifier public-coin zero-knowledge *proof system for a polynomial-time computable relation $\mathcal{R}$ if:*

- *The verifier algorithm* V *on public input y, proceeds as follows. In every round in which it is the verifier's turn to speak, the verifier picks a fresh coin (i.e. a random string) according to distribution $\mathcal{D}(y)$ and sends it to the prover. When the prover sends its last message to the verifier, the verifier decides based on the transcript of the interaction, whether to accept or to reject it.*
- *There exists an efficient (PPT TM) algorithm* SIM*, called "simulator", such that for every (y, w) in $\mathcal{R}$, the following two variables are distributed identically:*
  - *a transcript of an interaction (P,V) on public input y and on P's private input w, where the probability is taken over the random coins of* P *and* V*.*

– *a transcript produced by the algorithm* SIM *on input* y *alone, where the probability is taken over the random coins of* SIM.

In all the proof systems we consider below, the honest-verifier public-coin zero-knowledge property is exhibited by a simulator which creates a transcript of a proof on public input y by first choosing the random coin R uniformly in $\mathcal{D}(y)$ and then computing the rest of the transcript $(M^*, m^*) = \mathsf{SIM}(y, R)$. In Section 4.4.1 we rely on this additional property of the THPZP proof systems presented in Figures F-1, F-2, and F-3, to argue the security of parallel execution of such proof systems, called a "simultaneous proof", in which every player P_i proves in parallel to all other players the knowledge of a witness to its public value y_i, and where the single public coin R used by all these proofs is picked via a distributed coin-flip protocol RVSS. We formalize this property of a simulation of an honest-verifier public-coin zero-knowledge proof system as follows:

Definition 31 (Coin-first Simulatable proof system) *We call a three-round honest-verifier public-coin zero-knowledge proof system* (P,V) coin-first simulatable *if it is three round and if the simulator* SIM *that exhibits the honest-verifier public-coin zero-knowledge property of this proof system (see Definition 30 above) proceeds on input* y *by first picking a coin* R *according to the distribution* $\mathcal{D}(y)$*, and then computing a pair* $(M^*, m^*) = \mathsf{SIM}(y, R)$*, and outputting the triple of messages* (M^*, R, m^*) *as a simulated transcript of a proof. (I.e.* M^* *represents the prover's message of Round 1,* R *the public coin chosen by the verifier in Round 2, and* m^* *the prover's message of Round 3.)*

We proceed to present three examples of a THPZP proof system, all three used in threshold protocols discussed in this thesis (see Section 4.4.1). In Figure F-1 we show the [Sch91] proof system THPZP-DL for proving knowledge of discrete logarithm. In Figure F-2 we show a proof system THPZP-Rep for proving knowledge of representations and of equality of representations (see [Bra99] and [Cam98] for references). Finally, in Figure F-3 we present the [CD98] proof system THPZP-MULT for proving knowledge of committed values which are in a multiplicative relation.

Lemma 37 (THPZP-DL is a proof of knowledge of discrete logarithm) *Protocol* THPZP-DL *of Figure F-1 is a Proof of Knowledge for relation*

$$\mathcal{R} = \{((p, q, g, y), x) \mid x \in \mathbb{Z}_q, g \in G_q, y = g^x \bmod p \;\; p, q \text{ are primes s.t. } q \text{ divides } p - 1\}$$

with knowledge error $1/q$*. Furthermore, an extractor that exhibits this property is oblivious.*

Proof: Note that the verifier V always accepts an interaction with an honest prover whose private input is a correct witness x for the public input (p, q, g, y), i.e. an element $x = \log_g y \bmod p$. Assume that some prover P* on inputs (p, q, g, y, x, r) has probability Pr of getting V to accept. Let k be the length of the public input, and let $T = 1/(Pr - \frac{1}{q})$. The extractor $\mathcal{E}$ interacts with P* on the common input (p, q, g, y) as follows. It repeats the following interaction with $\mathsf{P}^*_{(p,q,g,y,x,r)}$ for $2(k+1)T$ times. It receives the first message M from P*, sends back $R \in \mathbb{Z}_q$ chosen at random, and receives a response m from P*. Note that since P* acts on fixed inputs, it always sends the same commitment value M. If $\mathcal{E}$ receives

THPZP-DL proof system

Three-round Honest-verifier Public-coin Zero-knowledge Proof of knowledge proof system for relation $\mathcal{R} = \{((p,q,g,y),x) \mid x \in \mathbb{Z}_q, g \in G_q, y = g^x \bmod p$, where p,q are primes s.t. q divides $p-1\}$

Common inputs: discrete-log instance (p,q,g), public value $y \in G_q$
Prover's private input: witness $x \in \mathbb{Z}_q$ such that $y = g^x \bmod p$

Round 1: P $\longrightarrow$ V : Pick r at random in $\mathbb{Z}_q$; send $M = g^r$
Round 2: P $\longleftarrow$ V : Send a random coin R chosen uniformly in $\mathbb{Z}_q$
Round 3: P $\longrightarrow$ V : Send $m = r + xR$
Acceptance: V : Verify that $g^m = y^R M$

Figure F-1: THPZP-DL: proof of knowledge of discrete logarithm

any two accepting transcripts, i.e. values R, m and $I'm'$ s.t. $g^m = y^R M$, $g^{m'} = y^{R'} M$, and $R \neq R'$, then it can extract a witness $x = \log_g y = (m_i - m_j)/(R_j - R_j)$ for (p,q,g,y) in $\mathcal{R}$.

We consider as successful only those runs of $\mathcal{E}$ in which it gets some valid transcript M, R, m in the first $(k+1)T$ trials, and some *other* valid transcript in the remaining $(k+1)T$ trials. Note that if V, interacting with $\mathsf{P}^*_{(p,q,g,y,x,r)}$, accepts with probability Pr, then for any R the probability that V accepts a transcripts containing a *different* random coin R' is at least $(Pr - \frac{1}{q}) = 1/T$. Because the probability Pr that V accepts any transcript can also be lower-bounded by $1/T$, we get that the probability that $\mathcal{E}$ fails in any of the two series of $(k+1)T$ trials is at most $(1-1/T)^{(k+1)T} \leq e^{-(k+1)} < 2^{-(k+1)}$. Therefore the probability that $\mathcal{E}$ fails in both of them is at most 2^{-k}, which proves the lemma. □

Lemma 38 *Proof system* THPZP-DL *of Figure F-1 is (perfect) Honest-verifier Public-coin Zero-knowledge and it is Coin-first Simulatable.*

Proof: The simulator SIM on input (p,q,g,y) works as follows. It picks random $R \in \mathbb{Z}_q$ and random $m^* \in \mathbb{Z}_q$, computes $M^* = g^{m^*}/y^R$, and outputs (M^*, R, m^*). The distribution of such transcript is identical to the distribution of a transcript $(M = g^r, R, m = r + xR)$ of (P,V) on common input (p,q,g,y) and P's input $x = \log_g y$, because when we consider m picked by P in Round 3 as a random variable defined as a function of r picked by P in Round 1 and R picked by V in Round 2 (this function is indexed by x and q), then (1) the distribution of m induced by uniform and independent choice of r and R in $\mathbb{Z}_q$, is itself uniform in $\mathbb{Z}_q$, and (2) m is independent from R. □

Lemma 39 (THPZP-Rep is a proof of knowledge of (equal) representations.) *Protocol* THPZP-Rep *of Figure F-2 is a Proof of Knowledge for relation*

$$\mathcal{R} = \{((p,q,\{y^{(j)}, g_1^{(j)}, ..., g_k^{(j)}\}_{j=1,..,n}), (\alpha_1, ..., \alpha_k)) \mid \forall_{j=1..n} \; y^{(j)} = (g_1^{(j)})^{\alpha_1} \cdot ... \cdot (g_k^{(j)})^{\alpha_k}\}$$

THPZP-Rep **proof system**

Three-round Honest-verifier Public-coin Zero-knowledge Proof-of-knowledge for relation $\mathcal{R} = \{(p, q, \{y^{(j)}, g_1^{(j)}, ..., g_k^{(j)}\}_{j=1,..,n}), (\alpha_1, ..., \alpha_k)\}$ (for some $n \in \mathbb{N}$), where p, q are primes s.t. q divides $p-1$, values $\alpha_1, ..., \alpha_k$ are elements of $\mathbb{Z}_q$, values $g_1^{(j)}, ..., g_k^{(j)}$ for each j are elements of $G_q \subset {\mathbb{Z}_p}^*$, and where $y^{(j)} = (g_1^{(j)})^{\alpha_1} \cdot .... \cdot (g_k^{(j)})^{\alpha_k}$ for all $j = 1, ..., n$.

Common inputs:	primes (p, q) s.t. q divides $p-1$,
	public values $y^{(j)}, g_1^{(j)}, .., g_k^{(j)} \in G_q$ for $j = 1, ..., n$
Prover's private input:	witness string $(\alpha_1, ..., \alpha_k) \in (\mathbb{Z}_q)^k$
	such that $\forall_j,\ y^{(j)} = (g_1^{(j)})^{\alpha_1} \cdot ... \cdot (g_k^{(j)})^{\alpha_k}$

[**Note:** Without index j, this is a proof of knowledge of a representation of y in bases $g_1, ..., g_k$. If there are more y's and their bases, indexed by $j = 1, ..., n$, this is a proof of knowledge of *equal* representation of all these y's in the respective bases.]

Round 1:	P $\longrightarrow$ V :	Pick random $r_1, ..., r_k \in \mathbb{Z}_q$;
		$\forall_j$, send $M^{(j)} = (g_1^{(j)})^{r_1} \cdot ... \cdot (g_k^{(j)})^{r_k}$
Round 2:	P $\longleftarrow$ V :	Send a random coin R chosen uniformly in $\mathbb{Z}_q$
Round 3:	P $\longrightarrow$ V :	Send $m_i = r_i + \alpha_i R$ for $i = 1, ..., k$
Acceptance:	V:	$\forall_j$, verify that $(g_1^{(j)})^{m_1} \cdot ... \cdot (g_k^{(j)})^{m_k} = (y^{(j)})^R M^{(j)}$

Figure F-2: THPZP-Rep: proof of knowledge of (equal) representation

(also p, q are primes s.t. q divides $p-1$, values $\alpha_1, ..., \alpha_k$ are elements of $\mathbb{Z}_q$, values $g_1^{(j)}, ..., g_k^{(j)}$ for each $j = 1, .., n$ are elements of $G_q \subset {\mathbb{Z}_p}^*$, and n is some integer), with knowledge error $1/q$. Furthermore, the extractor that exhibits this property is oblivious.

Proof: The extractor construction is very similar to the one in Lemma 37. Similarly as in the proof of Lemma 37, $\mathcal{E}$ repeats the verifier's protocol and interacts with the prover P* on the fixed public inputs an instance (p, q) and values $y^{(j)}, g_1^{(j)}, ..., g_k^{(j)}$ for $j = 1, .., n$, and fixed private and random inputs of P*. As in the proof of Lemma 37, $\mathcal{E}$ repeats this interaction for $2(k+1)T$ steps where k is the length of the public input and $T = 1/(Pr - \frac{1}{q})$. If $\mathcal{E}$ gets two accepting transcripts on two different coins then it holds P*'s commitment message $\{M^{(j)}\}_{j=1..n}$, two random coins $R \neq R'$, and two valid responses $\vec{m} = (m_1, ..., m_k)$ and $\vec{m}' = (m'_1, ..., m'_k)$ s.t. $(y^{(j)})^R M^{(j)} = (g_1^{(j)})^{m_1} \cdot ... \cdot (g_k^{(j)})^{m_k}$ and $(y^{(j)})^{R'} M^{(j)} = (g_1^{(j)})^{m'_1} \cdot .... \cdot (g_k^{(j)})^{m'_k}$ for all $j = 1..n$. Therefore in such case $\mathcal{E}$ can output a valid witness, i.e. a representation $((m_1 - m'_1)/(R - R'), ..., (m_k - m'_k)/(R - R'))$ of each $y^{(j)}$ in its bases $(g_1^{(j)}, ..., g_k^{(j)})$. As in the proof of Lemma 37, if P*'s probability of passing the proof Pr is bigger than $1/q$, then the probability that $\mathcal{E}$ fails is at most 2^{-k}. □

Lemma 40 *Proof system* THPZP-Rep *of Figure F-2 is (perfect) Honest-verifier Public-coin Zero-knowledge and it is Coin-first Simulatable.*

Proof: Similarly to the construction in the proof of Lemma 38, the simulator SIM picks random $R \in \mathbb{Z}_q$ and random vector of elements $(m_1^*, ..., m_k^*)$ in $(\mathbb{Z}_q)^k$, computes $M^{(j)*} = (g_1^{(j)})^{m_1^*} \cdot ... \cdot (g_k^{(j)})^{m_k^*} (y^{(j)})^{-R}$ for each $j = 1..n$, and outputs $(\{M^{(j)*}\}_{j=1..n}, R, (m_1^*, ..., m_k^*))$ as the transcript of the proof. The distribution of such transcript is identical to the distribution of a transcript of an actual proof, because as in the proof of Lemma 38, when we consider the vector $\vec{m} = (m_1, ..., m_k)$ produced by P in Round 3 of a proof as a random variable defined as a function of $\vec{r} = (r_1, ..., r_k)$ chosen by P in Round 1 and R chosen by V in Round 2 (this function is indexed by vector $(\alpha_1, ..., \alpha_k)$ and by prime q), then (1) the distribution of $\vec{m}$ induced by uniform and independent choice of $\vec{r}$ in $(\mathbb{Z}_q)^k$ and R in $\mathbb{Z}_q$, is itself uniform in $(\mathbb{Z}_q)^k$, and (2) $\vec{m}$ is independent from R. □

THPZP-MULT proof system

Three-round Honest-verifier Public-coin Zero-knowledge Proof-of-knowledge for relation $\mathcal{R} = \{((p, q, g, h, A, B, C), (a, \hat{a}, b, \hat{b}, c, \hat{c})) \mid c = ab, A = g^a h^{\hat{a}}, B = g^b h^{\hat{b}}, C = g^c h^{\hat{c}}\}$ where p, q are primes s.t. q divides $p - 1$, $a, \hat{a}, b, \hat{b}, c, \hat{c}$ are all elements of $\mathbb{Z}_q$, g, h, A, B, C are all elements of G_q, relation $c = ab$ holds modulo q and all exponentiations are computed modulo p.

Common inputs: Pedersen Commitment instance (p, q, g, h), i.e.
primes q, p s.t. q divides $p - 1$, values $g, h \in G_q$ s.t. $g \neq h$
values $A, B, C \in G_q$

Prover's private input: $a, \hat{a}, b, \hat{b}, c, \hat{c}$, s.t. $c = ab$ and $A = g^a h^{\hat{a}}, B = g^b h^{\hat{b}}, C = g^c h^{\hat{c}}$

Round 1: P ⟶ V : Pick random $d, x, s, s_1, s_2 \in \mathbb{Z}_q$;
Send $M = g^d h^s$,
$M_1 = g^x h^{s_1}$,
$M_2 = B^x h^{s_2}$

Round 2: P ⟵ V : Send a random coin R chosen uniformly in $\mathbb{Z}_q$

Round 3: P ⟶ V : Send $y = d + bR$,
$z = x + aR$,
$w = s + \hat{b}R$,
$w_1 = s_1 + \hat{a}R$,
$w_2 = s_2 + (\hat{c} - a\hat{b})R$

Acceptance: V : Verify that $g^y h^w = B^R M$,
$g^z h^{w_1} = A^R M_1$,
$B^z h^{w_2} = C^R M_2$

Figure F-3: THPZP-MULT: proof of knowledge of committed a, b, c s.t. $ab = c$

Lemma 41
(THPZP-MULT is a proof of knowledge of "committed product relationship")
Protocol THPZP-MULT *of Figure F-3 is a Proof of Knowledge for relation*

$$\mathcal{R} = \{((p, q, g, h, A, B, C), (a, \hat{a}, b, \hat{b}, c, \hat{c})) \mid c = ab, A = g^a h^{\hat{a}}, B = g^b h^{\hat{b}}, C = g^c h^{\hat{c}}\}$$

(also p, q are primes s.t. q divides $p-1$, values $a, \hat{a}, b, \hat{b}, c, \hat{c}$ are elements of $\mathbb{Z}_q$, and values g, h, A, B, C are elements of $G_q \subset \mathbb{Z}_p{}^*$), with the knowledge error $1/q$. Furthermore, the extractor that exhibits this property is oblivious.

Proof: The extractor construction is very similar to the extractors for the previous two THPZP's (Lemmas 37 and 39). Similarly to those constructions, the extractor $\mathcal{E}$ interacts with P* by following V's algorithm (on fixed public inputs and fixed P*'s private and random inputs), for $2(k+1)T$ steps where k is the length of the public input and $T = 1/(Pr - \frac{1}{q})$. If $\mathcal{E}$ gets two valid transcripts $(\vec{M}, R, \vec{m})$ and $(\vec{M}, R', \vec{m}')$ where $R \neq R'$, $\vec{M}$ is P*'s commitment message $\vec{M} = (M, M_1, M_2)$, and $\vec{m} = (y, z, w, w_1, w_2)$ and $\vec{m}' = (y', z', w', w_1', w_2')$ are two correct responses of P* to R and R', respectively, then the following relations are satisfied:

$$g^y h^w = B^R M, \quad g^z h^{w_1} = A^R M_1, \quad B^z h^{w_2} = C^R M_2$$

as well as

$$g^{y'} h^{w'} = B^{R'} M, \quad g^{z'} h^{w_1'} = A^{R'} M_1, \quad B^{z'} h^{w_2'} = C^{R'} M_2$$

Therefore extractor $\mathcal{E}$ can compute the following representations

$$\begin{aligned} (a, \hat{a}),\ \text{where } a = (z - z')/(R - R') \text{ and } \hat{a} = (w_1 - w_1')/(R - R') \quad &\text{of} \quad A \text{ in bases } (g, h) \\ (b, \hat{b}),\ \text{where } b = (y - y')/(R - R') \text{ and } \hat{b} = (w - w')/(R - R') \quad &\text{of} \quad B \text{ in bases } (g, h) \\ (a, \hat{x}),\ \text{where } \hat{x} = (w_2 - w_2')/(R - R') \quad &\text{of} \quad C \text{ in bases } (B, h) \end{aligned}$$

Since $B = g^b h^{\hat{b}}$, extractor $\mathcal{E}$ can also compute representation

$$(c, \hat{c}),\ \text{where } c = ab \text{ and } \hat{c} = \hat{x} + a\hat{b} \quad \text{of} \quad C \text{ in bases } (g, h)$$

Therefore $\mathcal{E}$ can output a witness $(a, \hat{a}, b, \hat{b}, c, \hat{c})$ of (A, B, C) in $\mathcal{R}_{(p,q,g,h)}$.

Again, just like in the proofs of Lemmas 37 and 39, if P*'s probability of passing the proof Pr is bigger than $1/q$, then running the interaction with P* for $2(k+1)T = 2(k+1)/(Pr - 1/q)$ times guarantees that the probability that $\mathcal{E}$ fails to get such two different transcripts is at most 2^{-k}, which implies the lemma. □

Lemma 42 *Proof system* THPZP-MULT *of Figure F-3 is (perfect) Honest-verifier Public-coin Zero-knowledge and it is Coin-first Simulatable.*

Proof: The simulator SIM, similarly to the simulators that exhibit Lemmas 38 and 40, first picks random $R \in \mathbb{Z}_q$ and random vector of elements $(y^*, z^*, w^*, w_1^*, w_2^*)$ in $(\mathbb{Z}_q)^5$, computes $M^* = B^{-R} g^{y^*} h^{w^*}$, $M_1^* = A^{-R} g^{z^*} h^{w_1^*}$, and $M_2^* = C^{-R} B^{z^*} h^{w_2^*}$, and outputs $((M^*, M_1^*, M_2^*), R, (y^*, z^*, w^*, w_1^*, w_2^*))$ as the transcript of the proof. The distribution of such transcript is identical to the distribution of a transcript of an actual proof, because as in the proofs of Lemma 38 or Lemma 40, when we consider the vector $\vec{m} = (y, z, w, w_1, w_2)$ produced by an honest P in Round 3 of a proof, as a random variable defined as a function of $\vec{r} = (d, x, s, s_1, s_2)$ chosen in Round 1 by an honest P and R chosen in Round 2 by V (this function is indexed by vector $(a, \hat{a}, b, \hat{b}, \hat{c})$ and by prime q), then (1) the distribution of $\vec{m}$ induced by uniform and independent choice of $\vec{r}$ in $(\mathbb{Z}_q)^5$ and R in $\mathbb{Z}_q$, is itself uniform in $(\mathbb{Z}_q)^5$, and (2) $\vec{m}$ is independent from R. □

Appendix G

Parallelizable Zero-knowledge Proof of Knowledge of Discrete Log

In Figure G-1 we include a known zero-knowledge proof-of-knowledge (ZKPK) proof system ZKPK-DL for proving the knowledge of discrete logarithm [Hal00]. The ZKPK-DL proof system, presented in Figure G-1, maintains its important properties even when performed in parallel with polynomially-many untrusted parties.

ZKPK-DL proof system

Zero-Knowledge Proof of Knowledge system for relation

$$\mathcal{R} = \{((p, q, g, y), x) \mid x \in \mathbb{Z}_q, g \in G_q, y = g^x \bmod p\}$$

Common inputs: discrete-log instance (p, q, g), public value $y \in G_q$
Prover's private input: witness $x \in \mathbb{Z}_q$ such that $y = g^x \bmod p$

Round 1:	P $\longrightarrow$ V :	Choose a trapdoor value $\alpha \in \mathbb{Z}_q$; Send $h = g^\alpha$.
Round 2:	P $\longleftarrow$ V :	Choose a random coin $R \in \mathbb{Z}_q$;
		Commit to R by picking random $\hat{R} \in \mathbb{Z}_q$ and sending $C = g^R h^{\hat{R}}$.
Round 3:	P $\longrightarrow$ V :	Choose $r \in \mathbb{Z}_q$; Send $M = g^r$.
Round 4:	P $\longleftarrow$ V :	Open commitment C by sending R and $\hat{R}$.
Round 5:	P $\longrightarrow$ V :	Verify that $g^R h^{\hat{R}} = C$. If so, send $m = r + xR$ and α.
Acceptance:	V :	Verify that $g^\alpha = h$ and $g^m = y^R M$.

Figure G-1: Parallelizable ZKPK proof of knowledge of discrete logarithm

The ZKPK-DL proof system is (statistical) zero-knowledge, which is a stronger property than the honest-verifier public-coin zero-knowledge defined in Definition 30 of Appendix F. We formulate the statistical zero-knowledge property of ZKPK-DL in Lemma 44 below. Furthermore, it remains zero-knowledge when it is executed in *parallel* by each pair of n

parties. Such parallel execution of ZKPK-DL is a crucial part of a distributed protocol h-IG, presented in Figure 7-3 in Chapter 7. The h-IG protocol itself establishes an instance of the Pedersen commitment scheme which is needed by all threshold protocols proposed in this thesis. The proof of the zero-knowledge property of the parallel execution of ZKPK-DL within h-IG is in Lemma 36 in Chapter 7, and the simulation reasoning used there is a straightforward extension of the proof given below for the zero-knowledge property (of a single execution) of ZKPK-DL.

The ZKPK-DL proof system is also a proof of knowledge (see Definition 27, page 215). However, the running time of the extractor process given by that definition is specified in terms of expected time of execution. This is an inconvenient notion for us to work with when we argue the security of distributed protocols which use ZKPK-DL (namely, protocol h-IG in Figure 7-3), because all notions of distributed protocol security we use are specified in terms of probabilistic machines with fixed time-bounds. On the other hand, the ZKPK-DL does not seem to meet the strong proof-of-knowledge property formulated in Definition 28, page 215. That definition requires that the bound on the running time of the extractor is equal to $q(k)/p^*$ where $q(k)$ is some polynomial in the security parameter k, and $p^* = (p(y,x,r) - \mathcal{K}*)$ where $p(y,x,r)$ is the probability that verifier accepts a proof executed with a (possibly dishonest) prover P^* on its given inputs (x,y,r) (whose length is polynomial in k), and $\mathcal{K}$ is an knowledge-error (refer to Appendix F for the explanation of these terms and the notation). Unfortunately, what we can show for ZKPK-DL is an extractor with a running time bounded by $q(k)/(p^*)^2$, which seems incompatible with Definition 28.

Therefore, in Lemma 43 below we show that the proof system ZKPK-DL attains a weaker "proof-of-knowledge-like" property than the standard definition, but whose formalization is very similar to the one we need for the proof-of-knowledge-like property of the h-IG protocol expressed in Lemma 35 in Chapter 7. As we mentioned above, the essential part of the h-IG protocol is a parallel execution of ZKPK-DL proofs by each pair of players. In Lemma 35 we show that if a corrupted player passes such proof with at least one honest player then an efficient procedure can extract a valid witness from such corrupted player. The extraction argument used to prove Lemma 35 is a straightforward extension of the extraction argument used in the proof of Lemma 43 below.

Lemma 43 (ZKPK-DL is a (weak) proof of knowledge of discrete-log)
Protocol ZKPK-DL *of Figure G-1 is a (weak) proof of knowledge for relation* $\mathcal{R} = \{(p,q,g,y,x) \mid x \in \mathbb{Z}_q, g \in G_q, y = g^x \bmod p\}$ *where* p,q *are primes s.t.* q *divides* $p-1$. *Namely,*

1. *For every discrete-log instance* (p,q,g), *and every* $x \in \mathbb{Z}_q$ *and* $y = g^x \bmod p$, *the verifier* V *always accepts in an interaction with an honest prover* P *on common input* (p,q,g,y) *and on* P*'s private input the witness* x.

2. *For every polynomial* $p(\cdot)$, *there exists an efficient (probabilistic polynomial-time) algorithm* $\mathcal{E}$, *called "extractor", such that for every large enough* k, *for* (p,q,g) *a discrete-log instance of security* k, *for every* $x \in \mathbb{Z}_q$ *and* $y = g^x$, *for every interactive probabilistic algorithm* P^* *with any private input* $\mathsf{ah} \in \{0,1\}^*$, *the following properties hold:*

(a) *$\mathcal{E}$ interacts with* P* *on fixed common inputs* (p, q, g, y), *on* P*'s *private input* ah, *and on* P*'s *random input* r *chosen uniformly at random.* $\mathcal{E}$ *first performs the algorithm of* V *until the end of the* ZKPK-DL *protocol and then, if* P* *passes the proof (i.e. if the* V *algorithm accepts), then* $\mathcal{E}$ *interacts with a copy of* P* *on the same* P*'s *inputs* $(p, q, g, y, \mathsf{ah}, r)$. *In particular, it follows that* P*'s *view of an interaction with* V *is distributed identically to* P*'s *view of an interaction with* $\mathcal{E}$.

(b) *With probability at least* $1 - \frac{1}{p(k)}$, *if* P* *passes the proof then* $\mathcal{E}$ *outputs* $x = \log_g y$.

Proof: As for part (1), note that V always accepts an interaction with an honest prover whose private input is a correct witness $x = \log_g y$ for the public input (p, q, g, y) in $\mathcal{R}$.

Fix inputs $(p, q, g, y, \mathsf{ah}, r)$ of P*. We describe the interaction of $\mathcal{E}$ and P* on the common input (p, q, g, y). $\mathcal{E}$ first follows the algorithm of V till the end of the ZKPK-DL protocol. Let $h, C, M, (R, \hat{R}), (m, \alpha)$ be the transcript of this protocol. If P* passes the proof, i.e. when $g^\alpha = h$ and $g^m = y^R M$, then $\mathcal{E}$ repeats the following interaction with a *copy* of P* (running on the same fixed inputs $p, q, g, y, \mathsf{ah}, r$), for $p'(k)$ times, where $p'(k) = 6p''(k)p(k)$ and $p''(k)$ is some small polynomial s.t. $p''(k) \geq \ln(2p(k))$ for all k large enough. In each loop of an interaction with a copy of P*, $\mathcal{E}$ uses the same randomness in Round 2, and hence in Rounds 1-3 the same messages h, C, M are exchanged. However, in Round 4 the extractor $\mathcal{E}$ sends a random R' s.t. $R' \neq R$ and $\hat{R}'$ s.t. $R' + \alpha\hat{R}' = R + \alpha\hat{R}$, i.e. s.t. $g^{R'} h^{\hat{R}'} = C$. If P* responds with m' s.t. $g^{m'} = y^{R'} M$ (i.e. if "the copy of P* passes the proof"), then $\mathcal{E}$ can output a witness $x = \log_g y = (m_i - m_j)/(R_j - R_j)$ of y in $\mathcal{R}_{(p,q,g)}$. Otherwise, $\mathcal{E}$ loops back, i.e. runs this interaction with a copy of P* again, using fresh random value R' in Round 4. If the copy of P* does not pass the proof in all of the $p'(k)$ loops then $\mathcal{E}$ outputs null.

Note that requirement (2a) is satisfied by algorithm $\mathcal{E}$ described above. It remains to argue point (2b), i.e. that the probability of $\mathcal{E}$ outputting is at most $\frac{1}{p(k)}$. Namely, let's call the event that P* passes the proof in an interaction with $\mathcal{E}$ as "P* passes", and the event that in at least one of the $p'(k)$ loops the *copy* of P* passes the proof as "Copy-P* passes". Let's denote by "P* fails" and "Copy-P* fails" the complements of these events. Then we need to argue that the probability *FAIL* of $\mathcal{E}$ outputting $\mathcal{E}$, i.e.

$$FAIL = \Pr[(\text{P* passes}) \wedge (\text{Copy-P* fails})]$$

is at most $\frac{1}{p(k)}$, where the probability is taken over the coins of $\mathcal{E}$.

Denote by "P* passes on C" the event that P* passes and that the message sent to P* in Round 2 by $\mathcal{E}$ is equal to C. Consider a probability threshold $T = \frac{1}{2p(k)}$, and let's define a set of "good" values C as follows:

$$\text{We call } C \in G_q \text{ "good" iff } \Pr[\text{P* passes on } C] \geq T$$

where the probability is taken over *those* coins $R, \hat{R}$ of $\mathcal{E}$ (or V) s.t. $g^R h^{\hat{R}} = C$. Let p_C denote the following probability:

$$p_C = \Pr[\text{Copy-P* passes on } C \mid C \text{ is good}]$$

It is easy to see that if polynomial $p'(k)$ is sufficiently larger than $1/T$ (i.e. sufficiently larger than $p(k)$) then probability $\overline{p_C} = 1 - p_C$ is negligible. Furthermore, *FAIL* can be

upper-bounded as $\overline{p_C} + T$, and therefore $FAIL < T + \delta < 2T = \frac{1}{p(k)}$, which implies the lemma.

We first show that $\overline{p_C}$ is negligible. Let $T' = T - \frac{1}{q}$. It is easy to see that for any C, if the probability that P* passes on C is at least T, then the probability that a copy of P* passes in a single loop of an interaction with $\mathcal{E}$ on that C is at least T', because when $\mathcal{E}$ interacts with a copy of P*, it chooses its coin R' at uniform in a space of $q-1$ coins in $\mathbb{Z}_q \setminus \{R\}$, where R was the coin picked by $\mathcal{E}$ in an interaction with the "original" P*. Therefore, $\overline{p_C} = 1 - p_C \leq (1-T')^{p'(k)} = \left(1 - \left(\frac{1}{2p(k)} - \frac{1}{q}\right)\right)^{p'(k)} \leq \left(1 - \frac{1}{6p(k)}\right)^{p'(k)}$, if $\frac{1}{3p(k)} \geq \frac{1}{q}$, which, since $|q| = k$, and hence $q > 2^{k-1}$, is assured for k s.t. $2^{k-1} \geq 3p(k)$. Since $p'(k) = 6p''(k)p(k)$, probability $\overline{p_C}$ is therefore smaller or equal to $\left(\frac{1}{e}\right)^{p''(k)}$. Therefore, for k large enough so that $p''(k) \geq \ln(2p(k))$, we have that $\overline{p_C} \leq \frac{1}{2p(k)}$, i.e. $\overline{p_C}$ is a negligible function in k.

Finally, we show that $FAIL$ can be upper-bounded as $\overline{p_C} + T$:

$$\begin{aligned}
FAIL &= \Pr[(\text{P* passes}) \wedge (\text{Copy-P* fails})] \\
&= \Pr[\text{C good}] * \Pr[\text{P* passes} \mid \text{C good}] * \Pr[\text{Copy-P* fails}| \text{ C good}] \\
&\quad + \Pr[\text{C bad}] * \Pr[\text{P* passes} \mid \text{C bad}] * \Pr[\text{Copy-P* fails}| \text{ C bad}] \\
&< 1 * 1 * \overline{p_C} + 1 * T * 1 \\
&\leq \left(\frac{1}{2p(k)} + \frac{1}{2p(k)}\right) \\
&= \frac{1}{p(k)}
\end{aligned}$$

□

Lemma 44 (ZKPK-DL is (statistical) zero-knowledge)
Under the discrete-log intractability assumption, the protocol ZKPK-DL *is statistical zero-knowledge. Namely, under the discrete-log intractability assumption, there exists a (PPT TM) algorithm* SIM*, called "simulator", such that for any cheating verifier (non-uniform family of PPT TMs)* V*, and any DLog instance family DL, for (p,q,g) a DLog instance of security parameter k in DL, and for any $y \in G_q$, the statistical difference between the following two variables is a negligible function of k:*

- V*'s *view of an interaction with the prover* P *on public input (p,q,g,y) and on* P*'s *private input $x = \log_g y$, where the probability is taken over the random coins of* P *and* V*
- V*'s *view of an interaction with* SIM *on public input (p,q,g,y), where the probability is taken over the random coins of* SIM *and* V*.

Proof: Consider a following interaction of V* and a simulator $\text{SIM}^{(0)}$. On common input (p,q,g,y), and on V*'s random input $r_{\text{V*}}$, $\text{SIM}^{(0)}$ sends a message h like P (i.e. chosen uniformly in G_q) in Round 1, receives V*'s message C in Round 2, sends message M like P (i.e. chosen uniformly in G_q) in Round 3, receives V*'s message $(R, \hat{R})$ in Round 4, and then rewinds V* to the beginning of Round 3 (i.e. re-runs V* on the same input and the

same randomness r_{V^*}, and giving it the same message h in Round 1), sends in Round 3 a fresh random value M' chosen uniformly in G_q, and receives a response $(R', \hat{R}')$ from V^* in Round 4. (Note that since V^* is a non-uniform family of PPT TMs, its power does not increase if it is given some private input.) Consider the following probability:

$$pr_{(p,q,g,y)} = \Pr[\mathsf{V}^*, \text{ in interaction with } \mathsf{SIM}^{(0)}, \text{ opens } C \text{ in two different but correct ways}]$$

where the probability is taken over the randomness r_{V^*} and the randomness used by $\mathsf{SIM}^{(0)}$ (in picking messages h, M, M').

Let us build two simulators, $\mathsf{SIM}^{(1)}$ and $\mathsf{SIM}^{(2)}$, that interact with V^* on public input (p, q, g, y) in a way that is similar to $\mathsf{SIM}^{(0)}$ described above. We show that (1) If for some discrete–log instance family DL, for (p, q, g) a discrete-log instance of parameter k in DL, and for some $y \in G_q$, the above probability $pr_{(p,q,g,y)}$ is not a negligible function of k, then $\mathsf{SIM}^{(1)}$ breaks the discrete-log assumption; and (2) If $pr_{(p,q,g,y)}$ is a negligible function of k, than $\mathsf{SIM}^{(2)}$, on input (p, q, g, y), produces to V^* a view which has a negligible statistical difference from V^*'s view of a random run of an interaction with the prover P whose private input is $x = \log_g y$. In other words, $\mathsf{SIM}^{(2)}$ is the simulator SIM required by the statement of the lemma. It will follow that under the discrete-logarithm assumption the ZKPK-DL protocol is statistical zero-knowledge.

Assume that there exists a discrete–log instance family DL, a non-uniform PPT TM family of algorithms $\mathsf{V}^* = (\mathsf{V}_1^*, \mathsf{V}_2^*, ...)$, and a polynomial $p()$, such that for all k_0 there exists $k \geq k_0$, such that for (p, q, g) a discrete–log instance of parameter k in DL, for some element $y \in G_q$, probability $pr_{(p,q,g,y)} \geq \frac{1}{p(k)}$. We will show a non-uniform PPT TM family $\mathsf{SIM}^{(1)} = (\mathsf{SIM}_1^{(1)}, \mathsf{SIM}_2^{(1)}, ...)$ which breaks the discrete–log assumption. Namely, for any k, on input (p, q, g) a discrete–log instance of parameter k in DL, machine $\mathsf{SIM}_k^{(1)}$ has a hard-wired element y in G_q for which the above probability $pr_{(p,q,g,y)} \geq \frac{1}{p(k)}$, and $\mathsf{SIM}_k^{(1)}$, on input (p, q, g) and a random $h \in G_q$ interacts with V^* on common input (p, q, g, y) as follows. $\mathsf{SIM}_k^{(1)}$ sends h to V^*, waits to receive C, and sends back a random M in G_q. If it receives some $(R, \hat{R})$ s.t. $g^R h^{\hat{R}} = C$, then it rewinds V^* to the beginning of Round 3, sends another random M' in G_q, and waits to receive another pair $(R', \hat{R}')$. If $g^{R'} h^{\hat{R}'} = C$ and $R' \neq R$ then $\mathsf{SIM}_k^{(1)}$ computes and outputs $\log_g h$ from the two representations of C in bases (g, h). Note that messages h, M, M' that $\mathsf{SIM}^{(1)}$ sends in this interaction are distributed just in V^*'s interaction with $\mathsf{SIM}^{(0)}$, and hence V^* opens its commitment C in two different ways with probability $pr_{(p,q,g,y)} \geq \frac{1}{p(k)}$. Therefore $\mathsf{SIM}^{(1)}$ outputs $\log_g h$ with probability greater or equal to $\frac{1}{p(k)}$, and hence breaks the discrete–log assumption.

Simulator $\mathsf{SIM}^{(2)}$, interacts with V^* on common input a discrete–log instance (p, q, g) and some $y \in G_q$, as follows. $\mathsf{SIM}^{(2)}$ chooses α uniformly in G_q, sends $h = g^\alpha$ to V^*, waits to receive C, and sends back a message M which is a random element in G_q. If V^* replies with correct R and $\hat{R}$, the simulator rewinds V^* to Round 3, picks a random $m' \in \mathbb{Z}_q$, and sends to V^* a new message $M' = g^{m'} y^{-R}$, prepared so that sending m in Round 5 will be a correct response to challenge R. If V^* opens C in a different way, i.e. with some $(R', \hat{R}')$ which are correct but s.t. $R \neq R'$ then $\mathsf{SIM}^{(2)}$ fails. If on the other hand V^* opens C as $(R, \hat{R})$ again, then $\mathsf{SIM}^{(2)}$ answers with (m', α). (The last possibility is that V^* does not

open C this time correctly at all, in which case $\mathsf{SIM}^{(2)}$ should act like P would, i.e. stop the conversation.)

Notice that the messages h, M, M' that $\mathsf{SIM}^{(2)}$ sends to V^* are distributed uniformly and independently in G_q, and hence the probability that V^* opens its C in a different way after rewinding is equal to $pr_{(p,q,g,y)}$. On the other hand, if V^* opens C as the same R or does not open it correctly at all, then the view the verifier V^* sees, composed of messages $(h, C, M', (R, \hat{R}), (m', \alpha))$ or messages $(h, C, M', \mathsf{null}, \mathsf{null})$, is distributed as in a random interaction with the actual prover P acting on private input $x = \log_g y$. This is because in the above transcript M' is distributed uniformly in G_q, while m' is a proper response to the message of V^*, i.e. either a value s.t. $g^{m'} = y^R M$ or null, as in the interaction with the actual prover. Putting the two facts together, it follows that the statistical difference between V^*'s view of an interaction with $\mathsf{SIM}^{(2)}$ and V^*'s view of an interaction with P (running on its private input $x = \log_g y$), is equal to $pr_{(p,q,g,y)}$.

Therefore, if for all discrete–log instance families DL, for all V^*'s, for all polynomials $p()$, there exists k_0 such that for all $k \geq k_0$, for (p, q, g) a discrete–log instance of security parameter k in DL, for all $y \in G_q$, probability $pr_{(p,q,g,y)} < \frac{1}{p(k)}$, (note that the other case is taken care of a construction $\mathsf{SIM}^{(1)}$ above), then the statistical difference between the two views is negligible in k, and the lemma follows. □