Theory and Practice of Verifiable Secret Sharing

by

Rosario Gennaro

Laurea in Matematica, Università di Catania, Italy (1989)
S.M., Massachusetts Institute of Technology (1993)

Submitted to the

Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1996

© Massachusetts Institute of Technology 1996. All rights reserved.

Author ....
Department of Electrical Engineering and Computer Science
May 10, 1996

Certified by ....
Silvio Micali
Professor, Thesis Supervisor

Accepted by ....
F.R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Theory and Practice of Verifiable Secret Sharing

by

Rosario Gennaro

Submitted to the Department of Electrical Engineering and Computer Science

on May 10, 1996, in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Secret Sharing is a fundamental notion for secure cryptographic design. In a Secret Sharing protocol a dealer shares a secret among n parties. In the so called threshold model, the sharing is done so that subsets of $t + 1$ (or more) parties can later reconstruct the secret, while subsets of t (or less) parties have no information about it. The notion can be generalized by having the dealer specify a family of subsets of the n parties, called the access structure. The dealer shares the secret in such a way that only subsets of players in such family (usually called authorized subsets) can reconstruct the secret, while non-authorized subsets have no information about it. Verifiable Secret Sharing (VSS) protocols achieve the above task in the presence of malicously behaving parties.

In our thesis we present a new and stronger definition of VSS. The novelty of the definition is that it satisfies the composition property of secure protocols. That is VSS protocols satisfying the requirements of our definition, can be proven to remain secure even when used as sub-protocols inside larger protocols. Previous definitions did not enjoy this property.

We present also the first VSS protocols in the access structure model, whose security does not depend on unproven computational assumptions.

One of the most important application of VSS protocols is the implementation of robust shared signature schemes. Such protocols allow a group of servers to sign a document with a secret key that is shared among them. We present efficient threshold signature schemes for the Digital Signature Standard and the RSA Signature Algorithm. The protocols are fully robust, that is they tolerate the presence of a threshold of malicious servers who may try to forge signatures or impede the signature process.

Thesis Supervisor: Silvio Micali

Title: Professor

Acknowledgments

First and foremost I would like to thank my advisor, Silvio Micali. I just cannot imagine a better person to work with. His enthusiasm makes research work always exciting. He is also an extremely supportive person, always ready to pump up your self–esteem when things do not go as well as desired.

Special thanks are due to Shafi Goldwasser and Tal Rabin, for serving in my thesis committee. Shafi introduced me to cryptography and always showed a genuine interest in my research. Tal is not only a great person to work with, but also a special friend.

The Theory Group at the MIT Laboratory for Computer Science has provided me with a supportive and cooperative environment. Special thanks to Ron Rivest, whom I consider as a second advisor. My experience with him, creating and teaching the new course on Computer and Network Security ranks among the best ones I have had in my graduate career. I would also like to thank Bonnie Berger, Albert Meyer and Mike Sipser for their support in various stages of my program. And finally the group would not be such an enjoyable place without its staff members, Be Hubbard above all.

My research has been shaped by endless conversations with the other cryptography students/postdocs whose stay at the Lab overlapped with mine. I would like to thank Mihir Bellare, Ran Canetti, Shai Halevi, Stas Jarecki, Joe Kilian, Daniele Micciancio, Rafi Ostrovsky, Phil Rogaway and Ray Sidney for sharing their knowledge and their friendship with me. For the same reasons I would like to thank Hugo Krawczyk at IBM.

In the six years spent in this group I made a lot of friends. Thanks to Javed Aslam, Scott Decatur, Roberto De Prisco, David Gillman, Trevor Jim, Marcos Kiwi, Arthur Lent, Alex Russell, Mona Singh, Roberto Segala, Mark Smith, Lisa Tucker–Kellogg and Ethan Wolf for being not only good colleagues but also great friends. I would also like to remember among them, the late Anya Pogosyants, who tragically left us this year.

Parts of this thesis have been written during visits to the Dipartimento di Matematica of the Università di Catania in Italy. I would like to thank my undergraduate advisor there Alfredo Ferro for suggesting the opportunity to go abroad for graduate school. Very special thanks to my best friend Giovanni Gallo for his tremendous support in the past six years. Also I would not be here if it wasn't for two special high school math teachers, Raffaele Sergi e Nicola Fallica, who taught me to love and enjoy math.

I would like to thank my family: my dad Giuseppe, my mom Maria Teresa, my sister Titti, her husband Antonio, my brother Salvatore and my newly born niece Emilia. In the past six years, the hardest thing was being so far and away from them. However, their encouragement and support was as close and present as ever.

Bibliographic Note

The material in Chapter 2 appeared in the proceedings of EuroCrypt 1995, in a paper coauthored by Silvio Micali [GM95].

The results in Chapter 4 about the Digital Signature Standard will be presented in the proceedings of EuroCrypt 1996 [GJKRa]. The results in Chapter 4 about the RSA Signature Algorithm will appear in the proceedings of Crypto 1996 [GJKRb]. Both of these papers were coauthored by Stas Jarecki, Hugo Krawczyk and Tal Rabin.

I would like to thank my coauthors for allowing me to reproduce these results in my thesis.

Table of Contents

1 Introduction **9**

1.1 The Security of Information 11

1.1.1 Cryptography vs. Information Security 12

1.1.2 What this Thesis is About: Sharing Information 13

1.2 Secret Sharing 14

1.2.1 Threshold Solutions 14

1.2.2 Other Threshold Schemes 16

1.2.3 Generalizing the Concept: Access Structures 17

1.3 Verifiable Secret Sharing 20

1.4 Summary of the results 22

2 The Definition of Verifiable Secret Sharing **23**

2.1 Computational Model 25

2.2 Secure Function Evaluation 26

2.3 Prior definitions of VSS. 30

2.4 Our definition of threshold VSS 31

2.5 Comparison with previous definitions of VSS 33

2.6 A VSS protocol that satisfies our definition 38

2.6.1 The functions 39

2.6.2 The `Share-Verify` protocol 40

2.6.3 The `Recover` protocol 44

2.7 On achieving fault-tolerance of $\frac{n}{2}$ 48

2.7.1 What we did in Section 2.6.3 49

2.7.2 Modifications to Rabin's protocol 50

3 Verifiable Secret Sharing over Access Structures 51

3.0.3 Previous work 52

3.1 Preliminaries 53

3.1.1 Computational Model 53

3.1.2 Adversaries 55

3.2 Definitions 56

3.2.1 A Note on Efficiency 58

3.2.2 A note on reconstruction 58

3.3 The weak adversary case 59

3.3.1 A sketch of the protocol 59

3.3.2 The Weak Protocol 60

3.4 Strong adversaries: the honest dealer case 63

3.5 Strong adversaries: the full case 66

3.5.1 Semi-Verifiable Secret Sharing 66

3.5.2 The full protocol 70

3.6 Lower Bounds 74

3.7 Future Directions 75

4 Robust Threshold Signature Schemes 77

4.1 Signature Schemes 79

4.2 Robust Threshold Signature Schemes 80

4.2.1 Communication Model 81

4.2.2 The Adversary 81

4.2.3 Definitions 82

4.2.4 The uninteresting solution 86

4.3 Our results 86

4.3.1 Previous results 87

4.4 Existing Tools 88

4.4.1 Feldman's VSS 88

4.4.2 Pedersen's VSS 89

4.4.3 Joint Random Secret Sharing 90

4.5 Robust DSS Threshold Signature Schemes 91

4.5.1 The Digital Signature Standard (DSS) 92

4.5.2 DSS Threshold Key-generation 93

4.5.3 Basic Modules of our Solution 93

4.5.4 Eavesdropping and Halting Adversary 97

4.5.5 Malicious Adversaries 101

4.5.6 Improving the fault-tolerance 104

4.5.7 Efficiency Considerations 107

4.6 Robust RSA Threshold Signature Schemes 107

4.6.1 The RSA signature scheme 108

4.6.2 Outline 108

4.6.3 Threshold Sharing of RSA functions 112

4.7 The RSA Non-Interactive Solution 114

4.7.1 Extensions of Information Checking 115

4.7.2 Key Generation Protocol 116

4.7.3 Signature Protocol 117

4.8 Interactive Robust Threshold RSA Signatures 119

4.8.1 Key Generation Protocol 120

4.8.2 The Signature Protocol 120

4.8.3 Untrusted Dealer 126

Bibliography **130**

Chapter 1

Introduction

Tra la la la la la la la

Mon secret je le garde et le garde bien

— *G. Bizet,* Carmen.

After robbing a ship, Captain Hook and his pirate crew have buried the treasure on a desert island. Hook has drawn a map in order to later be able to recover the secret. Now he faces a dilemma: "Where will he store the map?" He is afraid of losing it, for if he does, he will never find the treasure again. Having only one copy of the map is very risky. The other pirates suggest that he make copies of the map and give a copy to each one of them. Hook is smarter than that, and sees through his "friends"' plan. They just want a copy of the map. True, duplicating the map, will reduce the risk of losing it, but it will allow other people to retrieve the gold themselves. Even if Hook gives copies to only a few of his most trusted friends, you know pirates. There is always a chance that during one of the pirate ship's frequent stops in some Caribbean port, one of them would get drunk and easily robbed of his possessions – including the valuable map! Not to count Peter Pan's constant interference with Hook's plans. He could send Tinkerbell over to use her magic powers and convince some of the guys to give her the map .... Gosh, Hook really does not know what to do.

One of the pirates has an idea and says: "Ok, rip the map in several pieces and give each of us one piece. Now each piece will have no real useful information about the treasure location, so nobody can sneak off and get the gold, and when we want to go get it at the right time, we will all pool together the pieces and recover the map."

Hook thinks about it and realizes that this is a good solution, but not good enough. Indeed, now we are back to the beginning. Just one missing piece will prevent them from reconstructing the map, so that if one of the pieces is lost, then goodbye treasure.

"What we need", continues Hook, "is a method of ripping this damn piece of paper is some form of threshold way. What I mean is that each individual piece (or a small collection of them) should be useless. At the same time, putting together all the pieces (or at least a large fraction of them), one must be able to recover the whole map." Hook sits down, and shakes his head, seeing no solution to the problem he just described is such precise terms ...

The word threshold, however, awakens "Sponge" from his sleep. Sponge is Hook's trusted assistant, a drunk with a prestigious academic past. He is completely wasted, and it is hard to understand what he is saying. Sponge stumbles down to his quarters and reemerges with a book on cryptography, surely a relic from his days of graduate school at MIT. He hands it to Hook and then crashes back into oblivion.

Hook starts reading and (much to his surprise) he finds out that famous scientists have tackled and solved this very problem. There are ways to share a secret into such a threshold way, and even in more general ways. They called them Secret Sharing Protocols. He calls on all the pirates, gives them a briefing about the solution, and says: "Ok here is the map, now I will share it according to one of these schemes."

It is now the pirates' turn to complain and be suspicious. "How do we know you are really sharing the map?" shouts one. Indeed Hook was already planning of sharing a fake map with the pirates in order to keep all the gold to himself. Another guy, who had continued reading from the book, pipes in: "There is another problem. Here they warn you about wrong pieces. That is, if a piece of the so shared map is missing, it is no big deal, but if it is different than what it originally was, we are in big trouble. The reconstructed map could be anything, even directions to the closest county jail! Gee, I can already picture that Peter Pan fellow making fun of us, by changing the pieces and making us go around the world instead of to the treasure island."

The pirates are now all gathered around the book trying to find a solution that is fully satisfactory . Well, all except Sponge who is still snoring in the corner. Again, the conversation must have struck a chord in his brain because Hook hears him mumble "Verifi .... able Secret Sharing."

Hook looks it up, turns to the appropriate page and ... YES! this is exactly what they need! The pirates, finally happy, start celebrating with whiskey and music. Unfortunately they forgot to verifiably share the map before they started the celebrations and while they are busy drinking and dancing, nobody realizes that Tinkerbell is flying away, the map securely in her hand ....

1.1 The Security of Information

Today we live in the electronic information age. Computers have brought about tremendous change in the way society lives and operates, yet they have also brought additional threats and dangers. We rely so much on the information processed, stored and transmitted by computers that a simple software bug can have catastrophic consequences on our lives. Even more worrisome are scenarios which involve criminal minds who actively try to detect weaknesses in information systems and use them to their advantage.

For this reason we need Information Security. But what exactly do we mean by that? Very informally, one can say that the goals of information security are threefold:

secrecy Information should be secret. Only authorized parties should access it.

authenticity Information should be authentic. Only authorized parties should modify it.

availability Information should be available. Authorized parties should always have access it.

Information security is clearly a multidisciplinary area. In order to achieve security we need good hardware (a storage/transmission medium that does not lose information, for example) and good software (no bugs). But even if we have the best hardware and bug-free software we need to deal with people, the users. We must prevent their mistakes before they make them. Users may lose or accidentally erase disk containing important data, forget their passwords, and similar things.

We also need to be prepared to outsmart users who maliciously try to infringe on the security of the system. We have to take into account people who tap communication channels, snoop on other people's files, refuse to participate or try to mess up some communication protocol and so on.

1.1.1 Cryptography vs. Information Security

Cryptography is one of the main tools for information security. Historically it was born out of concern only for the secrecy requirement. We know that Caesar used codes to protect the secrecy of messages sent to his officers [Kah67]. Cryptography has been evolving in time to become a scientific discipline from its empirical roots.

A typical example of how cryptography works is as follows. There exists a permutation f which is hard to compute without a secret k that we call the key. If we let $g = f^{-1}$ and m is the data we want to keep secret, then clearly one can publish $y = g(m)$ and knowledge of the key will allow him to recover $m = f(y)$. One can distinguish cryptographic methods into symmetric ($g = f$) and asymmetric ($g \neq f$), but the basic idea is the same. These methods can be used to protect the secrecy of stored or transmitted information.

There are also cryptographic methods to guarantee the authenticity of information. In this case, the function f is used in the following way. Given some data m, the owner of the data attaches to it the value $s = f(m)$ (called a message authentication code in the symmetric case or a signature in the asymmetric one). Clearly, modification of the data m to a value m' by an non-authorized party (who does not hold the key k) are infeasible since such party cannot compute the proper value $s' = f(m')$.

There are several candidate cryptographic functions (as DES, RSA, etc.) which are supposed to satisfy the hardness requirements sketched above. Interested readers can consult standard cryptography textbooks as [Sch96, Sti95].

Does cryptography suffice to guarantee information security? Unfortunately, the answer is not. Indeed, notice how the entire security of cryptographic systems rely on the security of the keys. This situation is an advantage in many cases, since it reduces the security of a potentially large amount of information to that of a small key. At the same time though, it means that we need a non-cryptographic way of protecting keys.

Also notice that encrypted data is potentially less available, since the data's availability is connected to the availability of the key. Losing a key may have disastrous consequences. In particular, this means that the availability of the keys is a priority as important as their secrecy.

1.1.2 What this Thesis is About: Sharing Information

So what is a good non-cryptographic way of protecting information? One could think about secure hardware, some complicated memory chip that would be unaccessible to non-authorized users. But (besides the fact that one has to specify how the authorized people are going to identify themselves to the chip) we have to keep all our goals in mind.

Storing the information in a single place (no matter how well protected) creates what is called a single point of failure. A break-in in this location would compromise the secrecy and/or authenticity of the information stored there. A simple crash of the storing device would make the stored information unavailable.

The best solution is to share in a special way the information among different users in different locations. Each user will receive a piece of the information. The properties needed from such a sharing are the following:

Properties

P1 In order to recover the original information a significant fraction of the pieces are needed.

P2 The original information should be recovered even if a significant fraction of the pieces is corrupted (i.e. missing or altered in some way)

P3 The individuals who receive a piece should be able to check that the sharing has been performed correctly

The rest of this chapter is devoted to the description of Secret Sharing techniques. These are algorithms to share information in such a way to satisfy property **P1**. There are two basic variants of Secret Sharing protocols, depending on the meaning of "significant fraction" in the informal description given above. We will describe both in Section 1.2.

The rest of thesis instead will be about Verifiable Secret Sharing (VSS) protocols which are algorithms that satisfy all the properties informally outlined above.

A plan for the whole thesis is described in Section 1.4.

1.2 Secret Sharing

In this section we will summarize for the reader two different ways to share a secret among a set of parties. The two algorithms differ in their interpretation of the property **P1** described in the previous section, in particular in their definition of *significant* fraction.

Notation: Let $P_1, \ldots, P_n$ be a set of players and D a special player called the dealer. The dealer owns a secret s from a space of secrets S.

D wants to share s among the players $P_1, \ldots, P_n$ in a way that satisfies property **P1**.

1.2.1 Threshold Solutions

Shamir and Blakley were the first to introduce the topic of secret sharing in [Sha79, Bla79]. In their solutions the meaning of significant fraction is specified by the introduction of a new parameter t. The sharing is conducted is such a way that

- if any $t + 1$ pieces are known they will reconstruct s, while any t of them give no information about s.

If the properties above are satisfied then we say we have a (t,n)-threshold secret sharing scheme.

Shamir's idea is the following. Let p be a prime and $S = Z_p$. The dealer on input the secret s performs the following algorithm:

Shamir's Secret Sharing: Sharing Phase

Input for the dealer: A secret $s \in Z_p$

1. Associate to each player P_i arbitrary distinct values $x_i \in Z_p^*$ (in particular $x_i = i$). Make these value public.
2. Choose a random polynomial f of degree t in $Z_p[x]$ under the condition that $f(0) = s$. That is, choose t random numbers in Z_p, let them be $a_1, \ldots, a_t$ and set $f(x) = a_t x^t + \ldots + a_1 x + a_0$ where $a_0 = s$.
3. Give privately to player P_i the share $s_i = f(x_i) \bmod p$

Figure 1-1: Shamir's protocol to share a secret s

Notation: If $s_1, \ldots, s_n$ are generated using the above algorithm we will denote them with $(s_1, \ldots, s_n) \stackrel{(t,n)}{\longleftrightarrow} s \bmod p$.

As shown in [Sha79] the properties above are satisfied. Indeed $t+1$ players will always be able to reconstruct the secret by performing Lagrange polynomial interpolation.

Notation: Let $\Lambda = \{x_{i1}, \ldots, x_{it+1}\}$. Consider the associated $(t+1) \times (t+1)$ Wandermonde matrix $W_\Lambda = (w_{k,j})$ where $w_{k,j} = {x_{ik}}^j$. We denote with $\lambda_{k,j,\Lambda}$ the elements of the inverse matrix of W_Λ and call them the Lagrange interpolation coefficients associated with Λ. Clearly they are computable given only knowledge of Λ.

The reconstruction algorithm is shown in Figure 1-2.

It is also clear why t players have no information about the secret. Indeed given t shares any secret s' could have been shared using the polynomial $f'(x)$ which satisfies $f'(0) = s'$ and the given t shares.

Shamir's Secret Sharing: Reconstruction Phase

Input for Player P_i: A value $s_i \in Z_p$

1. Each player P_i reveals s_i
2. Let $\{P_{i1}, \ldots, P_{it+1}\}$ be a set of $t+1$ players and $\Lambda = \{x_{k1}, \ldots, x_{kt}\}$ the associated subset of values. Each coefficient of the polynomial f can be computed as
$$a_k = \sum_{j=1}^{t+1} \lambda_{k,j,\Lambda} s_{ij} \tag{1.1}$$
3. Plugging $k = 0$ in Equation (1.1) one can compute the secret $s = a_0$.

Figure 1-2: Shamir's protocol to reconstruct a secret s

1.2.2 Other Threshold Schemes

Independently from Shamir, Blakley proposed a different threshold secret sharing scheme [Bla79]. The basic idea is the following: to achieve a (t,n)-threshold scheme, encode the secret $s \in Z_p$ as a point in the $(t+1)$-dimensional space Z_p^{t+1} (for example let s be the first coordinate and choose all the others at random). Each player will receive as his share a random t-dimensional hyperplane passing through that point. Clearly $t+1$ shares will be enough to reconstruct the secret, while t shares will only identify a line passing through the hyperplanes, yielding no information about the secret s.

Although less efficient than Shamir's scheme, Blakley's has a peculiar property. Assume, in either scheme, that $t+1$ players get together and reconstruct the secret. In Shamir's scheme these players reconstruct the whole sharing polynomial f and so are able to compute all the other shares as well. This is not true instead in Blakley scheme, since the shares have no relationship to each other except for passing through the same point. We will see later how this turns out to be an important issue when it comes to formalizing the task of Verifiable Secret Sharing.

Other threshold schemes have been devised since Shamir's and Blakley's contribution, for example the reader can consult [AB83, KGH83].

1.2.3 Generalizing the Concept: Access Structures

Threshold mechanisms are only one of the ways one can imagine to share information. Consider the following example: the secret to be shared is the code for launching nuclear missiles. The president holds the code, but some backup mechanism should be in place, in the eventuality something happens to her. Suppose the Department of Defense wants to implement the following policy: the vice-president and 2 generals can launch the missiles or 5 generals without the vice-president can. These should be the only combination possible. The sharing mechanism required in this case is not a simple threshold one (although it looks a lot like one, indeed can be easily implemented by simple modifications of any threshold scheme.)

In general given the set of players $\{P_1, \ldots, P_n\}$ one could specify a family $\mathcal{F}$ of subsets of such set and then share the secret in such a way that only subsets in $\mathcal{F}$ can reconstruct the secret, while other combinations of players cannot. $\mathcal{F}$ is usually called the access structure and sets in $\mathcal{F}$ are called authorized subsets. Ito, Saito and Nishizeki introduced this model in [ISN87] and showed that a secret can be shared over *any* access structure $\mathcal{F}$. Benaloh and Leichter [BL88] improve this construction. For a survey of the problem the reader is referred to [Sti92]. In this section we will review the construction of Benaloh and Leichter [BL88].

Let $\{P_1, \ldots, P_n\}$ be the set of players and $\mathcal{F}$ the access structure defined on it by the dealer D. We are only interested in monotone access structures, i.e. families $\mathcal{F}$ with the property that

$$\text{if } R \in \mathcal{F} \text{ and } R' \supseteq R \text{ then } R' \in \mathcal{F}$$

Notation: Denote by $B_{\mathcal{F}}$ the basis of $\mathcal{F}$, i.e. the smallest family of subsets $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\} \subseteq \mathcal{F}$ with the following property

$$\text{for all } R \in \mathcal{F} \text{ there exists } S_m \in B_{\mathcal{F}} \text{ such that } R \supseteq S_m$$

That is the basis is the less redundant representation of the access structure; each authorized subset is a superset of a basis subset. We will refer to players with two different notations according to the need. P_i will be the i^{th} player when we enumerate all of them. With $P_{m,j}$ instead we will denote the j^{th} player in the authorized subset $S_m \in B_{\mathcal{F}}$. Notice that for each player P_i there can be up to ℓ different pairs (m, j) such that $P_i = P_{m,j}$.

The basic idea is to represent access structures via monotone boolean DNF functions. Each variable corresponds to a player in the network and the function evaluates to TRUE if and only if the collection of variables set to TRUE corresponds to an authorized subset of players. If $B_{\mathcal{F}}$ is the basis of the access structure $\mathcal{F}$ then the DNF formula corresponding to $\mathcal{F}$ would be constituted by the OR of a collection of AND's, one for each authorized subset in $B_{\mathcal{F}}$. See figure 1-3.

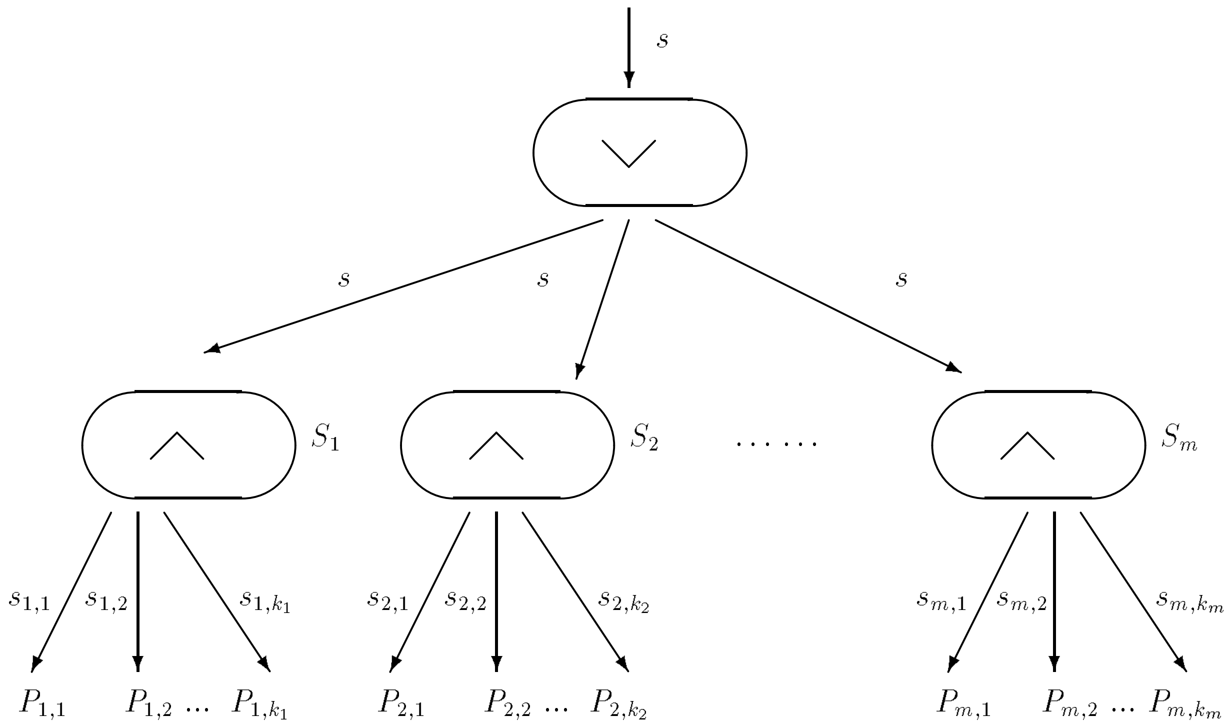


Figure 1-3: The boolean DNF formula associated with $B_{\mathcal{F}}$

As in Shamir's case assume that the secret is a number $s \in Z_p$. The algorithm to share a secret over a general access structure $\mathcal{F}$ [ISN87, BL88] appears in Figure 1-4.

Generalized Secret Sharing: Sharing Phase

Input for the dealer: A secret $s \in Z_p$, the basis of the access structure $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$

1. For each subset $S_m \in B_{\mathcal{F}}$, $S_m = \{P_{m,1}, \ldots, P_{m,k_m}\}$ the dealer:
   (a) chooses random values $s_{m,1}, \ldots, s_{m,k_m} \in Z_p$ such that $s_{m,1} + \ldots + s_{m,k_m} = s \bmod p$
   (b) gives privately to $P_{m,j}$ the value $s_{m,j}$

Figure 1-4: How to share a secret s over a general access structure

Basically the dealer performs ℓ independent $(k_m - 1, k_m)$-threshold sharings of the secret s, one for each subset in the basis. This means that an unauthorized (i.e., not in $\mathcal{F}$) coalition of players has no information about the secret at the end of the sharing phase. Indeed such coalition is going to miss at least one share for any subset in the basis (as the players do not constitute a subset in $\mathcal{F}$.) This means that any secret s' could be possible given the information held by the non-authorized coalition (by setting the missing shares to the appropriate values.) Details can be found in [ISN87, BL88].

Notice that the size of the shares received by each player is related to the size of the secret times the number of subsets in the basis of the access structure. That is, each player may receive up to ℓ values, so an upper bound on the size of the shares is $O(\ell|p|)$.

Benaloh and Leichter in [BL88] show that this approach can be generalized. One can represent $\mathcal{F}$ by any monotone boolean formula composed only by AND and OR gates. The sharing algorithm for the dealer is modified as follows. Start with the secret s on the output wire of the formula and travel down the wire. Whenever it meets an OR gate with fan-in k just replicate the value along the k input wires. Whenever a value goes through an AND gate with fan-in k just perform a $(k-1, k)$

threshold sharing and place the resulting values on the input wires of the gate. The shares of player P_i are the values that end up on the input wires coming out from P_i's gate.

The main advantage of this technique is that in this case the size of the shares will be $O(|\mathcal{F}||p|)$ where with $|\mathcal{F}|$ we denote the size of the smallest monotone formula computing $\mathcal{F}$. This in general could be a more compact representation than the DNF one. Moreover Benaloh and Leichter show that one could also consider (t,k)-THRESHOLD gates in the formula. In this case the dealer should perform a (t,k)-threshold sharing of the value coming through that gate. The representation would be even more compact in this case.

In the following we shall use the Protocol in Figure 1-4 to share the secret using the DNF representation. We will return briefly to the generalized algorithm of Benaloh and Leichter at the end of Chapter 3.

1.3 Verifiable Secret Sharing

We showed that the above protocols satisfy property **P1** for two particular interpretation of the word "significant".

In fact, it is not hard to see that, in the case all players are *honest* (that is they perform their part of the protocol correctly) all 3 properties **P1, P2** and **P3** are satisfied.

However the assumption that the players remain honest is not a realistic one. We should put ourselves in an adversarial position and try to imagine what would happen if some of the players stray away from their instructions and try to take advantage of the others.

In the protocols above there are two main problems that may arise. First, the dealer could share no secret at all and get away with it. For example in Shamir's scheme the shares s_i in Protocol 1-1 could be truly random numbers and only at reconstruction time the players would realize that they do not lie on a polynomial of degree t (thus identifying no secret.)

The other main problem is that while less than t players cannot find out the secret, they can prevent the others from reconstructing it. Again use Shamir's scheme as an example and think of cheating players who contribute a value $\hat{s}_i \neq s_i$ in Protocol 1-2. The interpolation procedure would fail miserably.

Not surprisingly these attacks are basically *denial of service* ones, meaning that they jeopardize the availability of the information. This arises from the fact that the above protocols are not fault-tolerant, that is they are not able to tolerate malicious faults. In short a Verifiable Secret Sharing Protocol is a fault-tolerant secret sharing protocol.

VSS protocols for the threshold model were introduced by Chor, Goldwasser, Micali and Awerbuch in [CGMA85]. Informally a threshold VSS protocol consists of a two-stage protocol. There is a dealer D, n players $P_1, \ldots, P_n$ and we assume that t of them may be *bad* and deviate from their prescribed instructions. The dealer who may also be bad, possesses a value s as a secret input. In the first stage, the dealer commits to a unique value v (in spite of the malicious behavior of the players); moreover, $v = s$ whenever the dealer is honest. In the second stage, the already committed value v will be recovered by all good players (again, no matter what the bad players might do). t is called the fault-tolerance parameter of the protocol and it is basically an upper bound on the number of players that can become corrupted during the protocol.

APPLICATIONS. After the original introduction of the concept of VSS in [CGMA85], several other VSS protocols were proposed motivated by various applications.

In [GMW87, BGW88, CCD88, RB89] VSS protocols were devised as tools for the task of secure multiparty computation. Informally, there are n parties $P_1, \ldots, P_n$ each holding a private input x_i and they want to compute the value $y = f(x_1, \ldots, x_n)$ (where f is a given function) without revealing any information about the inputs. Of particular interest are the protocols of [BGW88, CCD88, Rab94] since they do not rely on cryptographic techniques, so the security of the secret is achieved in an information-theoretic sense. These protocols form the basis of the work in Chapters 2 and 3 of this thesis.

In [Fel87, Ped91, Ped92] VSS protocols were devised for the particular task of sharing secret keys of discrete log based cryptosystems. We will return to these protocols in Chapter 4.

1.4 Summary of the results

The topic of this thesis is to study the theory and practical applications of VSS protocols. The thesis is divided in three parts.

The first part (Chapter 2) recalls some previous work done in the area of VSS protocols in the threshold model. We then present a stronger definition of verifiable secret sharing and exhibit a protocol implementing it. We show that our new notion is preferable to the old ones whenever verifiable secret sharing is used as a tool within larger protocols, rather than being a goal in itself.

In the second part (Chapter 3) we discuss VSS in the access structure model. Strangely enough, the task of fault-tolerant sharing of secrets in the access structure model had not been researched before. We present a formal definition and protocols implementing it. The most important problem we encountered trying to generalize the notion of threshold VSS to the access structure model, was to redefine the notion of fault-tolerance parameter.

Finally the third part of the thesis (Chapter 4) is devoted to the application of VSS protocols to robust threshold signature schemes. In these applications the secret is the secret key of a signature scheme. The task is to augment the signature scheme with

1. a fault-tolerant sharing protocol to share the key among a set of n players
2. a fault-tolerant distributed signature protocol that allows these players to sign messages using their shares of the secret keys, even if at most t of them may be corrupted.

CHAPTER 2

The Definition of Verifiable Secret Sharing

Now each of us take one of the ship's directional crystals. If a day arrives when we may return, any two brought together will point the way home.

— *P. Glass,* The Voyage.

As we saw in the previous chapter, Secret Sharing and Verifiable Secret Sharing (VSS for short) are fundamental notions and tools for information security. Despite the centrality and the maturity of this concept (almost 10 years passed from its original introduction), in this thesis we advocate that stronger and better definitions of VSS are needed.

Until now, VSS was considered only in the threshold model. In this chapter we recall previous work on threshold VSS and present a new stronger definition for such task. For the rest of the chapter, when not explicitly mentioned, we assume implicitly the threshold model.

In the past, several definitions and protocols for VSS have been proposed (E.g., [CGMA85, BGW88, CCD88, FM88, Rab94].) We contend, however, that these notions and these protocols are of *limited use.* In fact, their security concerns "begin when the dealer's secret is committed, and end when it is recovered." Because in many applications running a *single* VSS protocol is exactly what is wanted, these prior definitions and protocols are totally adequate in those scenarios. They are not, as we show in this chapter, adequate in more general scenarios: when VSS is used as a tool towards other ends, that is, when it is used as a **sub-protocol** within a larger protocol. Indeed, and unfortunately, it is by now a well-known phenomenon that protocols that are secure by themselves, cease to be secure when used as a sub-protocols. In these cases the security of the entire protocol must be proven "from scratch" (for instance, this is the case in [FM88] where they use VSS as a tool to reach Byzantine agreement) rather than in a more natural and elegant "modular way."

In this chapter we put forward a definition of VSS that guarantees **reducibility**; that is, security even when VSS is used as a sub-routine in an otherwise secure protocol. A notion of security that guarantees reducibility has been presented by Micali and Rogaway [MR92], for the problem of **function evaluation**. We thus wish to extend reducibility-guaranteeing notions of security to verifiable secret sharing protocols and concretely exhibit VSS protocols that provably satisfy these notions. More precisely, we achieve the following goals:

1. We propose a new definition of VSS based on secure function evaluation

   (This will guarantee the reducibility that characterizes the latter notion.)

2. We compare our new notion with the previously proposed ones, and show that it is *strictly and inherently stronger.*

   (Indeed, though sometimes protocols proved to satisfy weaker properties also satisfy stronger ones, we shall also show that none of the previous VSS protocols can satisfy our new notion.)

3. We show that our new definition is not "empty", meaning we construct VSS protocols satisfying it. Such protocols are obtained by modifications of earlier ones due to Ben-Or, Goldwasser and Wigderson ([BGW88]) and T. Rabin [Rab94].

2.1 Computational Model

We consider n players $P_1, \ldots, P_n$. A protocol is a set of instruction for these players. Each party run his own component of the protocol. The only coordination with other parties is via messages exchanged in an organized fashion.

The communication network is composed by point–to–point private communication channels between players. This means that messages exchanged by players on these channels cannot be read or modified by any other party. We also assume the existence of a broadcast channel. A message sent by player P_i on such a channel is received by all players and all players believe it came from P_i.

The network is synchronous. That is, there is a common clock accessible to all parties. Messages are sent at each clock tick and received by the next tick. The interval of time between two consecutive ticks is called a round. At the beginning of a round a player reads all the messages broadcasted or sent to him, then runs his component of the protocol for that round and computes the messages he will send next. These outgoing messages are computed by the player as a function of the incoming messages and his "history" (i.e. his input, his random coins, and the set of messages previously exchanged.)

We need to model now malicious behavior occurring during the protocol on the part of some of the players. We say that player who does not follow the instructions given to him by the protocol is "bad" or "faulty". We want to put ourselves in the worst possible scenario. In order to do this we assume that the bad players may coordinate their strategies in an optimal way. For this purpose we envision a single external entity called the adversary $\mathcal{A}$. Such an adversary will decide which players to corrupt and will send messages on behalf of the corrupted players during the protocol. We assume that the adversary is all-powerful, i.e., can perform arbitrarily long computations; also we say that $\mathcal{A}$ is completely-informed, i.e., when corrupting a player she finds out all his computational history.

This adversary will be dynamic or adaptive (i.e., decides during the execution of the protocol which processors corrupt); and is also allowed rushing (i.e., in a given round of communication, bad players receive messages before the good ones and, based on those messages, the adversary can decide whom to corrupt next.) In more details here is what happens: at a given round all good players receive the messages sent to them in the previous round and compute the messages they are going to send in the present round. We pessimistically assume that the messages addressed to the currently bad players are delivered immediately. Based on such information the adversary can decide to corrupt a new player and she does so before such player has sent any message to the good players, thus giving $\mathcal{A}$ a chance to change such messages. This is iterated until $\mathcal{A}$ is satisfied and decides not to corrupt anybody else for that round.

We say that an adversary is a t-adversary ($0 \leq t \leq n$) if t is an upper bound on the number of processors she can corrupt (t is also referred to as the **fault-tolerance** of the protocol.)

Notation: In the following we assume that there are $n+1$ players in the network. A special player D called the dealer and n players $P_1, \ldots, P_n$. We say that a player P **broadcasts** a message m, when she puts m on the broadcast channel for everybody to hear it. In particular $\mathcal{A}$, the adversary, can hear the message too. We say that a player P **distributes** a message if she puts m on the private channels connecting her to all the other players. Notice that the adversary $\mathcal{A}$ can see m only if somebody has been corrupted.

2.2 Secure Function Evaluation

In this section we summarize the work of Micali and Rogaway on the definition of secure protocols for function evaluation [MR92].

Informally the problem is the following: n players $P_1, \ldots, P_n$, holding private inputs $x_1, \ldots, x_n$ respectively, want to evaluate a vector-valued function f on their individual secret inputs without revealing them (aside from what is implied by f's output). That is, they want to compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$ such that each player P_i will learn exactly y_i.

This goal is easily achievable if there is an external trusted party, who privately receives all individual inputs and then computes and privately hands out all individual outputs. Of course, even in this *ideal* scenario, the adversary can create some problems. She can corrupt a player P_i before he gives his input x_i to the external party and change it with some other number $\hat{x}_i$. And she can further corrupt players after the function has been evaluated and learn their outputs. These problems should, however, be regarded as *inevitable*. Indeed, following [GMW87], in [MR92] the authors call a protocol for evaluating f secure if it approximates the above ideal scenario "as closely as possible." The nature of this approximation is informally summarized below.

Let us define the following quantities within the context of a protocol P.

Definition 2.1 (Initial Configuration) The initial configuration for P is a vector $\vec{ic}$, whose ith component, $ic_i = (x_i, r_i)$ consists of the private input and the random tape of player P_i.

Definition 2.2 (Traffic) The traffic of player P_i in protocol P at round q, t_i^q, is the set of messages sent and received by P_i up to that round.

Definition 2.3 (Input Function) A *local input function* $\vec{I} = (I_1, \ldots, I_n)$ for P is an n-tuple of functions such that there exists a specific round r such that, by applying I_i to the traffic t_i^r, we get the input player P_i is "contributing to the computation." $\vec{I}(\vec{ic})$ will denote the vector of those values when P is run on initial configuration $\vec{ic}$.

Notice that $I_i(ic_i) = x_i$ only for good players. Corrupted players run with x_i on their input tape, but the contributed input may be different if the adversary so desires.

Definition 2.4 (Output Function) A local output function $\vec{O} = (O_1, \ldots, O_n)$ for P is an n-tuple of functions such that by applying O_i to the final traffic t_i^{final} of player P_i we get his output.

Definition 2.5 (View) The adversary view, $\text{VIEW}^{\mathcal{A}}_{P,Network}$, during protocol P is the probability distribution over the set of computational histories (traffic and coin tosses) of the bad players.

Informally the view represents the knowledge gained by an adversary during the execution of the protocol.

Definition 2.6 (Simulator) A simulator Sim is an algorithm that "plays the role of the good players". The adversary interacts with the simulator as if she was interacting with the network. The simulator tries to create a view for the adversary that is indistinguishable from the real one. He does this without knowing the input of the players, but it is given access to a special oracle called the ideal evaluation oracle. For a protocol P with local input function $\vec{I}$ evaluatable at round r, the rules of the interaction between Sim and the oracle are the following:

- if $\mathcal{A}$ corrupts player P_i before round r Sim gets from the oracle the input x_i and gives it to $\mathcal{A}$.
- at round r Sim gets the output y_i' for all the players corrupted so far, where $(y_1', \ldots, y_n') = f(x_1', \ldots, x_n')$ where $x_i' = x_i$ if P_i is still good, otherwise $x_i' = I_i(t_i^r)$
- if $\mathcal{A}$ corrupts a player P_i after round r, Sim gets from the oracle the pair (x_i, y_i') and gives it to $\mathcal{A}$.

Definition 2.7 (Secure Function Evaluation) Let f be a vector-valued function, P a protocol, Sim a simulator, and $\vec{I}$ and $\vec{O}$ local input and output functions. We say that P securely evaluates the function f with fault-tolerance t if for any t-adversary $\mathcal{A}$

- Correctness: If $\vec{ic}$ is the initial configuration of the network, then

  1. $x_i = I_i(\vec{ic})$ for all good players P_i
  2. with high probability, $\vec{O}(\vec{t}^{final}) = f(\vec{I}(\vec{ic}))$

  (I.e. no matter what the adversary does, the function is evaluated during the protocol on some definite inputs defined by the local input functions over the traffic of the players. These inputs coincide with the original inputs for the good players)

- Privacy: For all initial configurations $\vec{ic}$, if $\text{VIEW}^{\mathcal{A}}_{P,Sim}$ is the adversary view of the simulated execution of the protocol, we have that

$$\text{VIEW}^{\mathcal{A}}_{P,Network} \approx \text{VIEW}^{\mathcal{A}}_{P,Sim}$$

  (I.e., the two views are statistically indistinguishable.)

For completeness we recall the definition of statistical indistinguishability of ensembles of random variables [GMR89].

Definition 2.8 Let $\mathcal{X} = (X_n)_n$ and $\mathcal{Y} = (Y_n)_n$ two countable families of random variables over the same probability space. We say that $\mathcal{X}$ and $\mathcal{Y}$ are statistically indistinguishable and write $\mathcal{X} \approx \mathcal{Y}$ if there exists a negligible function ϵ such that

$$\sum_z |Prob[X_n = z] - Prob[Y_n = z]| = \epsilon(n) \tag{2.1}$$

The definition of negligible function is as follows:

Definition 2.9 (Negligible Function) We say that a function

$$\epsilon \ : \ N \longrightarrow [0,1]$$

is negligible if $\forall\ k$, $\exists\ n_k$ such that $\forall\ n > n_k$,

$$\epsilon(n) \leq \frac{1}{n^k}$$

There are many reasons for which Definition 2.7 captures correctly the notion of a secure computation. In particular, the following one: the definition above allows one to prove formally many desirable properties of secure protocols, the most interesting for us being *reducibility*:

Theorem 2.1 (Reducibility [**MR92**]) *Let f and g be two functions. Suppose there is a protocol P that securely evaluates f in the model of computation in which it can* *perform ideal evaluations of g. Suppose also that there is a protocol Q that securely computes g. Denote with P^Q the protocol in which the code of Q is substituted in P in the places where P ideally computes g. Then P^Q is secure.*

Interested readers are referred to the original paper [MR92] for a proof of this statement and a complete and a formal description of their definition.

Notation: Let P be a protocol computing the n-valued function f and Q be a protocol computing the function g. We say that (P, Q) is a compound protocol if on input $(x_1, \ldots, x_n)$ it computes the function $g(f(x_1, \ldots, x_n))$ in the following manner: P is run first and then Q is run on the output of Q.

2.3 Prior definitions of VSS.

To exactly capture the informal idea of VSS, has proven to be a hard task in itself. The definition reviewed below is that of [FM88], which we will use as a token example of previous VSS definitions in the literature.

Let us first introduce the notion of a fixed event.

Definition 2.10 (Fixed Event) We say that an event $\mathcal{X}$ is fixed at a given round in an execution E of a protocol, if $\mathcal{X}$ occurs in any execution E' of the protocol coinciding with E up to the given round.

Definition 2.11 Let P be a pair of protocols where the second is always executed after the first one, P =(`Share-Verify, Recover`) for the players $P_1, \ldots, P_n, D$. In protocol `Share-Verify`, D has a secret input s; the output of player P_i is a value $verification_i \in \{yes, no\}$. In protocol `Recover`, the input of each player P_i is his computational history at the end of the execution of `Share-Verify`; the output of each P_i is a string σ_i.

We say P is a VSS protocol with fault-tolerance t if the following 3 properties are satisfied:

1. Acceptance of good secrets: In all executions of `Share-Verify` with a t-adversary $\mathcal{A}$ in which the dealer is good, $verification_i = yes$ for all good players P_i.

2. Verifiability: If less than t players output $verification = no$ at the end of `Share-Verify` then at this time a value σ has been fixed and at the end of `Recover` all good players will output the same value σ and moreover if the dealer is good $\sigma = s$ the secret [1]

3. Unpredictability: In a random execution of `Share-Verify` with a good dealer and the secret chosen randomly in a set of cardinality m any t-adversary $\mathcal{A}$ will not be able to predict the secret better than at random i.e. if $\mathcal{A}$ outputs a number a at the end of `Share-Verify` then $Prob[a = s] = \frac{1}{m} + \epsilon(n)$ where ϵ is a negligible function.

4. Unanimity: If a good player outputs yes all good players must output yes at the end of `Share-Verify`.

2.4 Our definition of threshold VSS

In this section we provide a new definition of VSS that guarantees reducibility. The key idea for achieving this property is to cast VSS in terms of secure function evaluation. Accordingly, we shall define two special functions SHAR and REC, and demand that both of them be securely evaluated in the sense of [MR92] (see Section 2.2.)

Let Σ be a set. Consider the n-fold cartesian product Σ^n and the following metric on it: given two vectors $\vec{a}, \vec{b}$ in Σ^n, let us define the distance $d(\vec{a}, \vec{b})$ between them as the number of components in which they differ; that is,

$$d(\vec{a}, \vec{b}) = |\{1 \leq i \leq n, a_i \neq b_i\}|$$

[1] Notice that if we simply ask in the Verifiability condition that "all the good players output the same number σ at the end of the `Recover` phase" it would not be sufficient for our purposes. In fact, we would still allow the adversary to decide during `Recover` what value σ the good players will output. Thus `Share-Verify` would not model a secret commitment as required.

We define the t-disc of $\vec{a}$ as the set of points at distance at most t from $\vec{a}$ i.e.

$$disc_t(\vec{a}) = \{\vec{b} \in \Sigma^n : d(\vec{a}, \vec{b}) \leq t\}$$

We define VSS as a pair of protocols, called `Share-Verify` and `Recover`, that compute, respectively, two functions, SHAR and REC, satisfying the following properties.

SHAR is the function we use to share the secret among the players. It is defined on the entire space for the n players (their private input does not matter in this phase) and on two finite special sets S and R, for the dealer. S is the space of possible secrets, while R is a set of random strings. SHAR is a n-values function. We require that, even after fixing any l ($l \leq t$) components of its output, all secrets are equally likely to generate those components. We call this property t-uniformity (see 2 below.)

Similarly REC is the function used to reconstruct the secret. When computed over the output of SHAR, the function REC returns the secret input of the dealer. In fact, we require to be able to do so even if up to t components of the output of SHAR are arbitrarily changed. We call this property t-robustness of the function REC (see 3 below.)

Definition 2.12 We say that two functions SHAR and REC are a sharing-reconstructing pair with parameter t if they have the following properties:

1. (Domain.)

$$\text{SHAR} : \Sigma^n \times (R \times S) \rightarrow \Sigma^{n+1}$$

and

$$\text{REC} : \Sigma^{n+1} \rightarrow S^{n+1}$$

2. (t-uniformity.) $\forall l \leq t$ there exists an integer m_l such that $\forall\ s_{i_1}, \ldots, s_{i_l} \in \Sigma$ and $\forall\ v_1, \ldots, v_n \in \Sigma$, $\forall\ s \in S$, and $\forall \vec{x} \in \Sigma^{n+1}$ such that $\forall j \in [1, l]\ x_{i_j} = s_{i_j}$, there exist exactly m_l values $r_1, \ldots, r_{m_l} \in R$ such that for $i = 1, \ldots, m_l$,

$$\text{SHAR}(v_1, \ldots, v_n, r_i \circ s) = \vec{x}$$

3. (t-**robustness.**) $\forall\ v_1, \ldots, v_n \in \Sigma$, $\forall\ s \in S$, $\forall\ r \in R$, if

$$\vec{x} \in disc_t(\text{SHAR}(v_1, \ldots v_n, r \circ s))$$

then

$$\text{REC}(\vec{x}) = (s, s, \ldots, s)$$

The definition of a threshold VSS protocol follows. A VSS protocol will be composed of two protocols that *securely* evaluate these two functions; the second being evaluated over the output of the first.

Definition 2.13 *A compound protocol (*`Share-Verify, Recover`*) is a VSS protocol of fault-tolerance t if there exists a sharing-reconstructing pair with parameter t (*SHAR,REC*) such that*

- `Share-Verify` *securely evaluates the function* SHAR*.*
- `Recover` *securely evaluates the function* REC*.*

As we shall see below, by demanding that both components (and particularly the second one) of a share-reconstructing pair be securely evaluated, we are putting an unusually strong requirement on a VSS protocol. But it is exactly this requirement that will guarantee the desired reducibility property.

2.5 Comparison with previous definitions of VSS

Let us compare now Definition 2.13 and Definition 2.11, our token example of prior VSS definitions. To begin with, there is a minor syntactical difference between the two definitions: according to Definition 2.11 when good players find out the dealer is bad they just stop playing and output $verification = no$. In our new definition instead the computation goes on, no matter what. This discrepancy can be eliminated by having protocols in the first definition agree on a default value when the dealer is clearly bad and protocols in the second definition always output $verification = yes$

at the end of Share-Verify (since we are dealing with a secure function evaluation, we are guaranteed that all good players will output a common value). This also takes care of the Unanimity condition in definition 2.11, which is trivially satisfied. With these minor changes we can prove the following:

Theorem 2.2 *If P is a VSS protocol of fault-tolerance t satisfying Definition 2.13, then P is also a protocol of fault-tolerance t satisfying Definition 2.11.*

Proof Let P be a VSS protocol of fault-tolerance t satisfying Definition 2.13. Let $\vec{ic}$ be the initial configuration vector of the network.

P satisfies the Verifiability property of Definition 2.11. Indeed at the end of the Share-Verify phase, let $\sigma = I_D(ic_D)$ i.e., the value contributed by the dealer to the computation. This value is *fixed* at the end of Share-Verify since it is a function of the traffic of the dealer. Moreover because of the t-robustness of the function REC we have the same value σ will be output by all good players at the end of the Recover part. Indeed,

1. because of the t-robustness property it does not matter that t bad players may change their input before computing the function REC.

2. also the Recover protocol is secure, so in particular is correct, meaning it computes the correct output.

Finally, if the dealer is good then $\sigma = I_D(ic_D)$ is equal to the secret s.

P also satisfies the Unpredictability property of Definition 2.11. First notice that the t-uniformity property implies that the output of the function SHAR is composed of t-wise independent uniformly distributed random variables and so it is impossible (in an information-theoretic sense) to predict the secret better than at random for any algorithm that has knowledge of only $l \leq t$ components of such output. However a t-adversary has also her view of the entire Share-Verify protocol (i.e. traffic of bad players and messages broadcasted by good players). Yet, because of the security of the evaluation of the function SHAR the adversary can create the entire view by herself using the simulator, hence the other information is irrelevant. More precisely, let's assume for the sake of contradiction that protocol P does not satisfy the Unpredictability condition. This means that there is a t-adversary $\mathcal{A}$ that

1. does not corrupt the dealer

2. $Prob[\mathcal{A} \text{ guesses } s] = \frac{1}{m} + \delta(n)$

where $m = |S|$ is the cardinality of the space of possible secrets and δ is a non-negligible function.

We will exhibit an algorithm $\mathcal{G}$ (for guesser) that guesses the secret with the same probability while having access only to t components of the output of the function SHAR, thus contradicting the t-uniformity property of SHAR. Since `Share-Verify` is a protocol that securely evaluates the function SHAR it must have a simulator Sim that works according to the specifications of Section 2.2. $\mathcal{G}$ runs Sim with the adversary $\mathcal{A}$. Since the simulated view of the protocol is indistinguishable from the real one, $\mathcal{A}$ will guess the correct secret with probability bigger than $\frac{1}{m} + \delta'(n)$ where δ' is another non-negligible function. Indeed if $\delta' = \epsilon$ was negligible, then the views of the simulated and real executions would not be statistically indistinguishable, as the summation in equation 2.1 would result in something $\geq \delta(n) - \epsilon(n)$ which is non-negligible.

$\mathcal{G}$ uses $\mathcal{A}$'s guess for the secret as his guess. Notice that by running the simulator, $\mathcal{G}$ will only know at most t components of the output of the function SHAR, the ones which will be answered by the oracle in response to request of corruption by $\mathcal{A}$. We have a contradiction, as we found an algorithm that predicts the secret substantially better than at random with knowledge of only t components of the output of SHAR, which contradicts the t-uniformity property. ■

Now the natural question to ask is: Are Definitions 3 and 1 equivalent? That is, if a given VSS protocol P' satisfies Definition 2.11, does it also satisfy Definition 3? The answer to this important question, provided by Theorem 3, is NO. And it better be that way if we want to preserve reducibility of VSS protocols. Indeed as we will see later the formal specifications of Definition 2.11 are not sufficient to guarantee composition of VSS protocols inside larger protocols.

Theorem 2.3 *Definition 2.13 is strictly stronger than Definition 2.11, that is, there are VSS protocols satisfying Definition 2.11, but not Definition 2.13.*

The proof of this theorem is based on the fact that usually VSS protocols (consider for example the one in [BGW88]) reconstruct the secret by just distributing each player's share to all other players. This does satisfy Definition 2.11 since there we do not put any requirement on the secrecy of the shares. But, as the following lemma shows, this does not satisfy Definition 2.13 since doing so we do not compute *securely* the function REC.

Let P_{wrong} be the following 1-round protocol. At round 1 each player distributes his own share.

Lemma 2.1 *Protocol P_{wrong} is not a secure computation of the function* REC.

Proof Consider the following t-adversary $\mathcal{A}_1$. $\mathcal{A}_1$ corrupts 1 player (say P_1) right at the beginning of the protocol. So in the simulated execution the simulator Sim will receive from the oracle the input (share) x_1. Now Sim has to simulate the messages of the good players. Sim does so and distributes $x'_2, \ldots, x'_n$ in the name of the good players. Let s' be the value interpolates by $x_1, x'_2, \ldots, x'_n$. Notice that $\mathcal{A}$ sees all these values x'_i and s' since she receives them through P_1. Now $\mathcal{A}$ corrupts P_2. This means that the oracle will provide Sim and $\mathcal{A}$ with the true input x_2 of P_2 and the true output of the function s. Since Sim had no knowledge of the true x_2 and the secret s when he distributed the simulated values x'_i, only with probability $\leq \frac{1}{|S|}$ we have that $x'_2 = x_2$ and $s = s'$. Thus the simulated execution will not succeed in convincing the adversary that she is talking with the real network, hence P_{wrong} is not secure according to definition 2.7. ■

What is wrong with protocol P_{wrong}? The problem is that the players reveal too much information about their own input share during the protocol. In other words we want that, when we compute $\textsc{rec}(\vec{y})$ over $\vec{y} = \textsc{shar}(v_1, \ldots, v_n, r \circ s)$, no knowledge about $\vec{y}$ should leak except the secret s (including no extra information about the secret s itself.) The rationale for asking this is, again, the fact that we want our VSS protocols to be secure not just by themselves but when used inside subroutines of more complex protocols. Leaking knowledge about the shares (or extra knowledge about the secret) may create problems to the security of the overall protocol. Consider the following example.

Example 2.1 Consider a VSS protocol P, satisfying Definition 2.11, in which the secret is a 3-colorable graph. During the `Recover` protocol the graph is reconstructed together with a 3-coloring of it kindly provided by the dealer. Notice that Definition 2.11 is not violated, but notice also that an adversary gains from the execution of such a protocol some knowledge about the secret she could not obtain by herself. This in turns means that there exists no simulator for this protocol and so that Definition 2.13 cannot be satisfied. And the serious problem with P is that, if used inside a larger protocol in which it is crucial that the knowledge of that particular 3-coloring stays hidden, P, though "secure" as a VSS protocol on its own, jeopardizes the security of the larger protocol.

The problem with the above example is solved by our definition, substituting property 3 (Unpredictability) with a stronger one based on zero-knowledge and simulatability, which is exactly what we do by asking for a secure computation of the functions $\textsc{shar}$ and $\textsc{rec}$. In particular the secure computation of the function $\textsc{rec}$ is the most important difference between the two definitions. In particular we want to stress the importance of maintaining the secrecy of the shares of each individual good player. Indeed one might envision protocols in which the secrecy of the shares is crucial, even after the reconstruction of the secret. So in order for the reducibility property to be preserved, *both* parts of a VSS protocol must be secure on their own. This is what we achieve with Definition 2.13.

Probably one of the reasons this point may appear somewhat moot is that in Shamir's (t, n)-threshold secret sharing scheme [Sha79] (see Section 1.2.1) the shares consist of the value of a polynomial of degree t with free term s. For a t-adversary who corrupts exactly t players, knowing the secret is equivalent to knowing the shares of all players. In fact, knowing the t shares of the corrupted players and the secret at the end of `Recover`, she has $t+1$ points of the t-degree polynomial, and by evaluating the so inferred polynomial at the names of all good players, she easily computes all shares. However, we object that what happens to be true for the VSS protocols based on Shamir's scheme, may not be true for all VSS protocols. For example we saw in Section 1.2.2 that in Blakley's (t, n)-threshold secret sharing scheme [Bla79] at the end of `Recover` a t-adversary has no information about the shares of the good players, even after seeing t shares and the secret. Moreover, even in Shamir-based VSS protocols, if the polynomial has degree strictly larger than the number of corrupted players, then it is no longer true that, for the adversary, knowledge of the secret is equivalent to knowledge of all shares.

The crucial point is that a definition should not incorporate what happens to be true in a specific case. One must look at the task of VSS from a more abstract point of view. The first part `Share-Verify` serves the purpose of sharing the secret, the second part, `Recover`, to reconstruct such secret. It is thus needed that the knowledge gainable by an adversary at the end of a secure VSS protocol exactly coincides with the original secret whenever the dealer is honest.

2.6 A VSS protocol that satisfies our definition

In this section we show that our definition is not empty, that is we show that there are VSS protocols that satisfy it. Notice that we cannot simply invoke the completeness theorems of [BGW88, CCD88, RB89] that say that any function can be computed securely in our model, in particular two functions SHAR and REC constituting a sharing-reconstructing pair. Indeed those results rely on VSS constructions, so we would be just biting our tail. In particular by showing that Definition 2.13 is not empty, we are also showing that the Micali-Rogaway definition of security [MR92] is not empty either.

We show a VSS protocol of fault-tolerance t which is secure according to Definition 2.13 when $n \geq 3t+1$. We obtain it by performing some modifications on a previous protocol due to Ben-Or, Goldwasser, and Wigderson [BGW88]. The modification actually occurs only in the `Recover` part, and uses techniques also developed in [BGW88], but within their "computational protocol" rather than in their VSS protocol.

It is possible to obtain a better fault-tolerance by using the VSS protocol of Tal Rabin [Rab94]. We will sketch this construction in Section 2.7.

Suppose we are dealing with a t-adversary $\mathcal{A}$. Let $n = 3t+1$, $P_1, \ldots, P_n$ the set of players and D the dealer. We will make all our computations modulo a large prime $p > n$. In order to construct our VSS protocol we first have to show a sharing-reconstructing pair of functions with parameter t.

2.6.1 The functions

Let SHAR-A be the following function:

$$\text{SHAR-A}(v_1, v_2, \ldots, v_n, r \circ s) = (s_1, s_2, \ldots, s_n, \epsilon)$$

with $s_i = f(i)$ where $f(x) = s + a_1 x + \ldots + a_t x^t$ and $r = a_1 \circ \ldots \circ a_t$.

It is known from error-correcting codes theory that if one evaluates a polynomial f of degree d over n different points x_i for $i = 1, \ldots, n$ then given the sequence $s_i = f(x_i)$ one can reconstruct the coefficients of f in polynomial time even if up to t elements in the sequence are arbitrarily changed, provided that

$$n \geq 2t + d + 1 \tag{2.2}$$

Indeed this is the Berlekamp-Welch variant of the Reed-Solomon error-correcting code [BW]. For details readers can refer to a standard text like [PW72] or to the Appendix in [Sud96]. The application of error-correcting codes to secret sharing was first noticed by McEliece and Sarwate in [MS81].

Let REC-A be the function

$$\text{REC-A}(s_1, \ldots, s_n, \epsilon) = (s, \ldots, s, s)$$

where s is the result of the Berlekamp-Welch decoder on input $s_1, \ldots, s_n$

Lemma 2.2 *If $n \geq 3t+1$ then* SHAR-A *and* REC-A *constitute a sharing-reconstructing pair with parameter t*

Proof

Domain Immediate once one sets $\Sigma = S = Z_p$ and $R = Z_p^t$.

t-uniformity This is the property first identified by Shamir in [Sha79]. Once one fixes $l \leq t$ shares and the secret s you get an underdetermined system of linear equations in the a_i's. Eliminating the equation $a_0 = s$, the system has t unknowns, but only l equations, with a matrix of full rank (it's a Wandermonde matrix.) This means that we can assign arbitrary values to $t - l$ of the a_i's. For each assignment we get a solution of the system, i.e. a possible r that generated the output of the function. Since the a_i's range in Z_p we have that $m_l = (p+1)^{t-l}$.

t-robustness Immediate once one plugs $d = t$ in Equation (2.2)

■

2.6.2 The `Share-Verify` protocol

We are now ready to describe the VSS protocol that satisfies Definition 2.13. Let K be a security parameter which for simplicity we assume to be a multiple of n. For the sake of completeness, we first show the protocol `Share-Verify`, although it is the same as the one in [BGW88]. The protocol appears in Figure 2-1.

Lemma 2.3 *Protocol* `Share-Verify-A` *securely evaluates the function* SHAR-A *according to Definition 2.7.*

`Share-Verify-A` ([BGW88])

Input for the dealer: A secret $s \in Z_p$.

1. The dealer chooses a random polynomial $f(x)$ in $Z_p[X]$ of degree t with the only condition that $f(0) = s$. Then he sends to player P_i the share $s_i = f(i)$. Moreover he chooses $2K$ random polynomials $f_1, \ldots, f_{2K}$ in $Z_p[X]$ of degree t as well and sends to P_i the values $\sigma_{i,j} = f_j(i)$ for each $j = 1, \ldots, 2K$.
2. Each player P_i broadcasts K/n random bits $\alpha_{(i-1)K/n+j}$ for $j = 1, \ldots, K/n$
3. The dealer broadcasts the polynomials $g_j = f_j + \alpha_j f$ for all $j = 1, \ldots, K$
4. Player P_i checks if
$$\sigma_{i,j} + \alpha_j s_i = g_j(i)$$
If the above equation is not satisfied he broadcasts a complaint.
5. If more than t players complained in the previous round, then each P_i sets $s_i \leftarrow 0$ and STOP.
6. The dealer broadcasts the values $s_k, \sigma_{k,j}\ \forall\ j$ he sent in the first round to the players P_k who complained in round 4.
7. Each player P_i checks that
$$\sigma_{k,j} + \alpha_j s_k = g_j(k)$$
for the players P_k who complained at round 4. If any of the above equations fails, P_i sets $s_i \leftarrow 0$ and STOP.
8. Each player P_i broadcasts K/n random bits $\beta_{(i-1)K/n+j}$ for $j = 1, \ldots, K/n$
9. The dealer broadcasts the polynomials $h_j = f_{K+j} + \beta_j f$ for all $j = 1, \ldots, K$
10. Player P_i checks that
$$\sigma_{i,K+j} + \beta_j s_i = h_j(i)$$
and that
$$\sigma_{k,K+j} + \beta_j s_k = h_j(k)$$
for the players P_k who complained in round 4. If the above equations do not hold, P_i broadcasts a complaint.
11. If more than t players complained in either round 4 or 10 then each player P_i sets $s_i \leftarrow 0$

Figure 2-1: A secure computation of the function SHAR-A

Proof If we look back at Definition 2.7 we see that we have to check that both conditions, correctness and privacy, are satisfied in the *blended* way they interact in the simulation process through the common input and output functions.

First of all, let us define these functions for our protocol. Remember that with t_i we define the traffic of player P_i. Clearly for all players P_i the input function always returns the empty string, $I_i(t_i) = \epsilon$, since the players do not contribute any input during the computation of the function SHAR-A. For the dealer D, the input function is a little bit more complicated. Let us denote with m_i the message the dealer broadcasts to player P_i in round 6 if P_i complained in round 4, or the message the dealer sent to player P_i in round 1 if P_i did not complain. Then $I_D(t_D) = f(0)$ where f is the t-degree polynomial resulting from the Berlekamp-Welch interpolation of the m_i's. The output function is simpler: $O_i(t_i) = m_i$ (where $m_D = \epsilon$). Now we can check both conditions of Definition 2.7.

Correctness: First we have to prove that for all good players P_i, $I_i(t_i)$ is equal to the correct input. In our specific case this has to be checked only for the dealer. If the dealer is good, $m_i = f(i)$ where f is a t-degree polynomial with free term s the secret. So $I_D(t_D) = s$ if the dealer is good. The second correctness condition is that with high probability $\vec{O}(\vec{t}) =$SHAR-A$(\vec{I}(\vec{t}))$. In our case this means that with high probability the values m_i held by good players must be on a single polynomial of degree t. We prove that this is true with probability $\geq 1 - 2^{-\frac{2K}{3}}$ at the end of `Share-Verify-A`. First of all notice that at least $\frac{2K}{3}$ bits are chosen truly randomly by good players in rounds 2 and 8. Each bit represents a "question" that a bad dealer who distributed non-interpolating shares will be able to answer correctly in the following round only with probability $\frac{1}{2}$ (i.e. if he predicted the bit correctly when he distributed the shares). So $2^{-\frac{2K}{3}}$ is a bound on the probability that the dealer guesses the "questions" of the good players. Remember that there are at least $2t + 1$ good players. Now suppose that l good players players complain in step 4. If $l > t$ then all the good players set $s_i \leftarrow 0$ and so all the shares lie on the constant 0-polynomial. If $l \geq t$ then we know that there are more than $t + 1$ players who are satisfied with the dealer messages. This means that with high probability their shares lie on a polynomial of degree t. Now when the dealer broadcasts the shares of the good players who complained, and repeats the question and answer session, the broadcasted shares are checked for being on the same polynomial. So if nobody complains all the good players shares are on a t-degree polynomial.

Privacy: We have to exhibit a simulator for the protocol. We distinguish 2 cases:

Case A: The dealer is corrupted before round 1. The simulator just follows the instructions of the players, with the only exception that it will turn them over to the adversary in case of corruption. Since the players do not contribute any input to the computation this reduces the simulated execution to one of a VSS with a bad dealer. So the simulation will be indistinguishable to the eyes of the adversary.

Case B: The dealer is not corrupted before round 1. Then Sim simulates round 1 by creating a random "fake" secret s' and sharing it among the players according to the protocol instructions using a polynomial f' created by a "fake" string r'. Whenever a player P_i is corrupted the simulator and the adversary will receive the true output s_i. The simulator will "pretend" that $f'(i) = s_i$. If the dealer is not corrupted at all during the protocol then everything will run smoothly since to the eyes of the adversary the execution will look like an ordinary VSS with a good dealer (again this is true because the players do not contribute any input to the computation). If the dealer is corrupted after round 1 however the adversary and the simulator will get from the oracle the true input s and r of the dealer. At that point the simulator turns over the control of the dealer to the adversary, but changes the internal memory of D by bringing up f as the sharing polynomial. This will not affect the view of the adversary so far, since for all players P_i that have been corrupted the simulator pretended that $f'(i) = f(i)$. The simulator changes accordingly the random polynomials f_i used for the zero-knowledge proof to make them consistent with whatever has been broadcast so far. The simulator can always do this since the adversary has at most t points of t-degree polynomials. For the rest of the simulation the simulator uses the polynomial f for the computation of the good players still under his control. In particular when the dealer corrupts one of them, the simulator "pretends" that the share the newly corrupted player received in the simulated execution was computed since from the beginning using f. We claim that this execution is indistinguishable to the adversary from a real one. This is so because the only thing different from a true execution is the fact that the shares the adversary gets before corrupting the dealer are created using a different polynomial than the real one, but $\mathcal{A}$ has too few points to detect the difference. ■

Remark: A completely error-free version of this protocol can be obtained by running a different zero-knowledge proof that the shares lie on a single polynomial. The proof uses a bivariate polynomial. Details can be found in [BGW88, FM88].

2.6.3 The `Recover` protocol

In [BGW88], the `Recover` protocol is performed by having each player distribute the input share s_i. As we have shown in Section 2.5, this is not a secure computation of the function REC against a dynamic adversary. So we need to modify the above `Recover` protocol. The basic idea is that each player will distribute his share to all the other players, but masked up appropriately with some randomness so that no information about the share is revealed but at the same time the secret reconstruction process is not compromised. The protocol appears in Figure 2-2.

Lemma 2.4 *Protocol* `Recover-A` *securely evaluates the function* REC-A *according to Definition 2.7.*

Before embarking in the formal proof of this lemma let us give some intuition of why this is true. For example, notice how the adversary $\mathcal{A}_1$ we described in the proof of Lemma 2.1 has now a simulator with the new protocol `Recover-A`. This is because we are "masking" the true share of player P_i with some randomness provided by the other players with the polynomials p_i. The shares revealed in round 8 have no relationship with the original shares except that they interpolate to polynomials with the same free term (the secret.) This will allow a simulator to simulate the messages at round 8 convincingly without actual knowledge of the input s_i.

Proof As in the proof of Lemma 2.3 we start by defining the input and output functions of our protocol. The input function I_i of player P_i is defined as follow: let

`Recover-A` **(Modified from [BGW88]):**

Input for each player P_i : A value s_i

1. Each player P_i chooses random polynomials $p_i(x), q_{i,1}(x), \ldots, q_{i,2K}(x)$ of degree t and with free term 0. He sends to player P_j the values $p_i(j), q_{i,1}(j), \ldots, q_{i,2K}(j)$
2. Each player P_i broadcasts K random bits $\gamma_{l,(i-1)K/n+m}$ for $l = 1, \ldots, n$ and $m = 1, \ldots, K/n$
3. Each player P_i broadcasts the following polynomials $r_j = q_{i,j} + \gamma_{i,j} p_i$ for each $j = 1, \ldots, K$
4. Each player P_i checks that the information player P_l sent him in round 1 is consistent with what player P_l broadcast in round 3. If there is a mistake or P_l broadcast a polynomial with non-zero free term, P_i broadcasts bad_l. If there are more than t players broadcasting bad_l, player P_l is disqualified. Otherwise P_l broadcasts the information he sent in round 1 to the players who broadcast bad_l
5. Each player P_i broadcasts K random bits $\delta_{l,(i-1)K/n+m}$ for $l = 1, \ldots, n$ and $m = 1, \ldots, K/n$
6. Each player P_i broadcasts the following polynomials $r_j = q_{i,K+j} + \delta_{i,j} p_i$ for each $j = 1, \ldots, K$
7. Each player P_i checks that the information player P_l sent him in round 1 and broadcast in round 4 is consistent with the polynomials player P_l broadcast in round 6. If there is a mistake or P_l broadcast a polynomial with non-zero free term, P_i broadcasts bad_l. If more than t players broadcast bad_l in this round or in round 4, then P_l is disqualified.
8. Let $Good$ be the set of indices of non-disqualified players. Each player P_i distributes the following value $\sigma_i = s_i + \sum_{j \in Good} p_j(i)$. If $l \notin Good$ then the players reset $\sigma_l \leftarrow 0$. The players run the Berlekamp-Welch decoding algorithm on the vector $\sigma_1, \ldots, \sigma_n$ to obtain the polynomial $F(x)$. Output $s = F(0)$.

Figure 2-2: A secure computation of the function REC-A

$m_{i,j}$ be the message P_i receives from P_j in step 1 (i.e. if P_j is honest $m_{i,j} = p_j(i)$). Then the input function is $I(t_i) = \sigma_i - \sum_{j \in Good} m_{i,j}$.

The output function is the following; $O_i(t_i) = F(0) = s$ where F is the t-degree polynomial resulting from the Berlekamp-Welch interpolation of the values $\sigma_1, \ldots, \sigma_n$. That is, $\vec{O}(\vec{t})$ =REC-A$(\sigma_1, \ldots, \sigma_n)$ (see Section 2.6.1.)

Correctness: It is clear that for all good players $I_i(t_i) = s_i$ the correct input share. Then we have to check that at least with high probability $\vec{O}(\vec{t})$ =REC-A$(\vec{I}(\vec{t}))$. In our case this means to prove that

$$\text{REC-A}(\sigma_1, \ldots, \sigma_n, \epsilon) = \text{REC-A}(s_1, \ldots, s_n, \epsilon)$$

Now this equation is not satisfied if and only if the free term of the polynomial interpolated by the σ_i's is different from the free term of the polynomial interpolated by the s_i's. In order to have this eventuality, one of the following things must happen:

- either a bad player P_l succeeds in sharing random "garbage" instead of the values $p_l(j)$ in round 1 (in this case the σ_i's may not interpolate a polynomial at all)
- or P_l does distribute $p_l(j)$ in round 1 but manages to use a polynomial with free term different than zero (in this case the σ_i's will reconstruct a different secret)

Notice that the sharing process of the polynomials p_i is identical to the one of the protocol `Share-Verify`, with the addition that we now check that the free terms of such polynomials is actually zero. A reasoning similar to the one in the proof of Lemma 2.3 shows that P_l succeeds in any of the two cases only with probability $2^{-\frac{2K}{3}}$. Since there are at most $\frac{n}{3}$ bad players, the probability that the protocol computes an incorrect output is at most $\frac{n\, 2^{-\frac{2K}{3}}}{3}$ which for K large enough is exponentially small.

Privacy: We have to exhibit a simulator and then prove that the simulation is indistinguishable from the true network execution. Consider the following simulator Sim_R:

rounds 1-7 During rounds 1 to 7 the simulator just follows plainly the instructions of the players. Since what good players do in these rounds is completely random and not related to their inputs, Sim_R will always be able to create an indistinguishable view.

round 8 Let *Good* be the set of indices of non-disqualified players during the simulated execution. For round 8 we must distinguish two cases:

1. Some player has been corrupted during the previous rounds of the simulated execution. This means that the simulator gets the true output of the computation (i.e., the true secret s) from the oracle. Then at round 8, Sim_R chooses a polynomial g of degree t such that $g(0) = s$ and then for each good player P_i, Sim_R distributes $g(i) + \sum_{j \in Good} p_j(i)$ where p_j is the polynomial shared by player P_j in round 1 of the simulation. The Reed-Solomon interpolation of these values will give as result s. If a player P_l is corrupted at the end of this round, then both Sim_R and $\mathcal{A}$ will learn from the oracle the true input share s_l. If $s_l \neq g(l)$ then Sim_R just changes the value of p_l at the point l so to make the entire sum consistent with what has been broadcasted.

2. No player has been corrupted in the previous rounds of the simulation. The simulator does not know the secret. Sim_R chooses a random value s' and a random t-degree polynomial g, such that $g(0) = s'$. For each good player P_i, Sim_R distributes $g(i) + \sum_{j \in Good} p_j(i)$ where p_j is the polynomial shared by player P_j in round 1 of the simulation. The Reed-Solomon interpolation of this values is now s' but notice that the adversary does not see this value as she has not corrupted anybody yet. If a player P_l is corrupted at the end of this round, then both Sim_R and $\mathcal{A}$ will learn from the oracle the true input share s_l and the true secret s. If $s \neq s'$ the simulator will "adjust" the polynomial g on points that are not seen by the adversary. If $s_l \neq g(l)$ then Sim_R just changes the value of p_l at the point l so to make the entire sum consistent with what broadcasted.

The simulation is indistinguishable from a real execution to the eyes of the adversary. In fact as we already said, in rounds 1-7 all messages are random and unrelated to the input so the simulator can easily play the role of the good players. In round 8 the real share is broadcast "hidden" by some random "garbage", this will allow the simulator to broadcast the message of a good player with the right distribution even without knowing the real input. ■

Remark: Suppose the adversary corrupts exactly t players. Then, at the end of the `Recover` phase the adversary, knows the secret s and the internal state of t corrupted players. This will allow her to compute the share of all the other players, since it will allow her to interpolate all the polynomials p_i (adding the extra condition that they have free term 0.) Does this bother us? No, because this is "intrinsic" in the nature of the function. Indeed this happens also if the function REC is computed in the presence of a trusted evaluation center. However, if the adversary corrupts $l < t$ players, then (assuming the dealer is honest) even knowing l shares and the secret she has no information about the other shares, as it should be since it mirrors an ideal model evaluation of the function REC. This was not true in the original protocol in which each player just distributes his input share.

Given Lemmas 2.2, 2.3 and 2.4, we can state the following:

Theorem 2.4 *Assume $n \geq 3t + 1$. Then the protocol P_A =(*`Share-Verify-A`*,* `Recover-A`*) is a VSS protocol according to Definition 2.13 with fault-tolerance t.*

2.7 On achieving fault-tolerance of $\frac{n}{2}$

In this section we sketch how we can modify Tal Rabin's VSS protocol (see [Rab94]) so that it can fit our stronger definition. We will not go deep into the details of the modifications we are going to use Rabin's protocol as a basis to construct VSS protocols in the access structure model in Chapter 3. The details of the threshold construction can be easily gathered, from the construction in the access structure.

2.7.1 What we did in Section 2.6.3

Another way of looking at protocol `Recover-A` in Figure 2-2 is the following

`Recover-A`

Input for Player P_i: A value s_i.

1. Each player P_i performs the protocol `Share-Verify-A` as the dealer with 0 as his secret, adding to the protocol a verification that the secret is indeed 0. Let $a_i^{(j)}$ the share received by player P_j from P_i's sharing.
2. The players runs the recover protocol from [BGW88] on input $s_i + a_1^{(i)} + \ldots + a_n^{(i)}$, that is
   (a) Each player distributes $s_i + a_1^{(i)} + \ldots + a_n^{(i)}$.
   (b) Each player performs Berlekamp-Welch interpolation on the values received in round (2.a).

Figure 2-3: Another look at protocol `Recover-A`

This was possible because the protocol `Share-Verify-A` from [BGW88] has the following properties:

- it is homomorphic, i.e. given a sharing of a secret a, $(a_1, \ldots, a_n) \stackrel{(t,n)}{\longleftrightarrow} a$ and a sharing of a secret b, $(b_1, \ldots, b_n) \stackrel{(t,n)}{\longleftrightarrow} b$, then we have that

$$(a_1 + b_1, \ldots, a_n + b_n) \stackrel{(t,n)}{\longleftrightarrow} a + b$$

  (see [Ben87] for a discussion of homomorphic secret sharing schemes.)

- it is possible to verify that the secret being shared is indeed 0 by
  - requesting that the test polynomials have all free term 0
  - checking that the broadcasted polynomials in rounds 3 and 9 have all free term 0

In the following we will refer to such protocols as 0-VSS.

2.7.2 Modifications to Rabin's protocol

It turns out that Rabin's protocol [Rab94] has the same properties of the protocol in [BGW88]. In particular this means that we can apply the same transformation technique.

Let `Share-Verify-B` be the sharing protocol in [Rab94]. Let `0-Share-Verify-B` the 0-VSS protocol based on `Share-Verify-B`. In order to make Rabin's VSS protocol secure according to Definition 2.13 we modify the reconstruction protocol as follows

`Recover-B` **(modified from [Rab94])**

Input for Player P_i: A value s_i

1. each player P_i runs `0-Share-Verify-B` as the dealer. Let $a_i^{(j)}$ the share received by player P_j from P_i's sharing.
2. The players run the reconstruction protocol from [Rab94] using $s_i + a_1^{(i)} + \ldots + a_n^{(i)}$ as input.

Figure 2-4:

Rabin's protocol and its modification above achieve fault-tolerance t when $n \geq 2t+1$.

As we already said, this protocol is described in full detail in Chapter 3 although in the more general access structure model. The threshold version can be easily inferred from that description.

CHAPTER 3

Verifiable Secret Sharing over Access Structures

Questa o quella per me pari sono

— *G. Verdi,* Rigoletto.

In Chapter 1 we recalled the notion of Secret Sharing in both the threshold and the access structure model. We also showed that such notions have no **fault-tolerance**, i.e., they are not able to tolerate players who act dishonestly during the protocol.

In order to incorporate fault-tolerance, the notion of Verifiable Secret Sharing was introduced in the literature in [CGMA85] but only for the threshold model. In Chapter 2 we recalled previous works on threshold VSS and pointed out subtleties concerning the definition of VSS protocols.

In this chapter we initiate an investigation of the problem of fault-tolerant sharing of secrets over general access structures. That is, we investigate protocols that guarantee security even when the number of corrupted players is not bounded by a fixed threshold t. We propose definitions and protocols to solve this problem.

The crucial point of our work is to determine how many players can be corrupted by an adversary before the protocol stops being secure. This will turn out to be a function of the specific access structure used in the protocol. In this chapter we will describe two kinds of functions, resulting in two different notions of adversaries being tolerated by our protocols.

We first define a weak adversary that can only corrupt players who belong to a limited number of authorized subsets. The corresponding VSS protocol is very simple and efficient.

Then we define a much stronger adversary that can corrupt players in all but one authorized subsets. The only constraint on this adversary is that at least one authorized subset must remain pure i.e. composed of all uncorrupted players. We present a protocol that achieves the VSS task against such an adversary. We believe this result to be quite surprising, as this means that at reconstruction time we will be able to detect the only one good subset among all the other corrupted ones.

We also present a lower bound on the power of the adversary, that is we prove that against any stronger adversary the task of VSS becomes impossible.

MOTIVATION. We believe that extending the notion of VSS (and in general of fault-tolerant secure computation of which VSS is a building block) to the access structure scenario is of fundamental importance. First of all, by doing so we provide the essential robustness property for the task of secret sharing over general access structures.

Most important, one can argue that the notion of threshold security in fault-tolerant secure computation, is based on the implicit assumption that all nodes of the network participating in the protocol have the same level of security. Assuming that the protocol remains secure unless more than t nodes are corrupted, implies that the corruption of one particular node is as likely as the corruption of any other node in the network. This is not necessarily true. It is actually most often the case that some servers are extremely secure, while others are a very easy target for break-ins. When designing a protocol one a might want to take advantage of this fact and tailor the functioning of the protocol to the actual security features of the network.

By initiating a theory of VSS over general access structure we hope to start reanalyzing all the area of fault-tolerant secure computation over this new paradigm.

3.0.3 Previous work

Up to the writer's knowledge there has been no treatment of Verifiable Secret Sharing in the literature before. In researching this topic we came across to two papers that make passing references to this problem.

In the original paper on generalized secret sharing [BL88], the authors say that it is possible to extend the techniques of Feldman [Fel87] in order to achieve VSS over access structures. In a more recent paper [Sta], Stadler proposes the notion of Publicly VSS in which the correctness of the sharing can be verified by any party in the network and not only by the shareholders. Stadler also makes a passing reference to the possibility of extending his result from the threshold to the access structure model. It is not hard to actually reconstruct the missing details in the above papers.

However both papers achieve security in a *computational* way. That is either the secrecy of the value s at the end of the sharing phase or the correctness of the reconstructed value is guaranteed only against a computationally bounded adversary. So these results are not valid in the model introduced in Section 2.1.

In this chapter, besides giving a complete formal treatment of the problem, we present the first VSS protocols over access structures whose security does not depend on unproven computational assumptions.

3.1 Preliminaries

In this section we discuss the model of computation, recall the notion of access structures and introduce the notions of weak and strong admissible adversaries, that is we define what it means to achieve security not in the threshold sense but rather in the access structure model.

3.1.1 Computational Model

The computational model we use in this chapter is the same one described in Section 2.1. In particular, we assume broadcast and private communication channels.

Access Structures. Recall the notion of access structure from Section 1.2.3. Let $\mathcal{P}$ be the set of players, $\mathcal{P} = \{P_1, \ldots, P_n\}$. Let $\mathcal{F}$ be a family of subsets of $\mathcal{P}$. If the following property is satisfied

$$\text{if } S_1 \in \mathcal{F} \text{ and } S_2 \supseteq S_1 \text{ then } S_2 \in \mathcal{F}$$

then $\mathcal{F}$ is called a (monotone) access structure on $\mathcal{P}$. The subsets in $\mathcal{F}$ are called authorized subsets. Given an access structure $\mathcal{F}$, its *basis* $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\} \subseteq \mathcal{F}$ is defined as the smallest family with the following property

$$\text{for all } R \in \mathcal{F} \text{ there exists } S_m \in B_{\mathcal{F}} \text{ such that } R \supseteq S_m$$

Notation: We will refer to players with two different notations according to the need. P_i will be the i^{th} player when we enumerate all of them. With $P_{m,j}$ instead we denote the j^{th} player in the authorized subset $S_m \in B_{\mathcal{F}}$. Notice that for each player P_i there can be up to ℓ different pairs (m, j) such that $P_i = P_{m,j}$.

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a subset of players. Let $\mathcal{F}$ be an access structure and $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$ its associated basis.

Definition 3.1 *We say that $\mathcal{Q}$ is $\mathcal{F}$-*admissible *if for all $m = 1, \ldots, \ell$ we have that*

$$\mathcal{Q} \cap S_m \subset S_m$$

That is, an admissible subset of players does *not* cover any authorized subset (notice the strict inclusion.)

Definition 3.2 *We say that $\mathcal{Q}$ is $\mathcal{F}$-*weak *if there exists $i_1, \ldots, i_{\ell/2+1}$ such that*

$$\mathcal{Q} \cap S_{i_j} = \emptyset$$

That is, a weak subset of players does not intersect more than half of the subsets in the basis.

Definition 3.3 *We say that $\mathcal{Q}$ is $\mathcal{F}$-*strong *if there exists $m \in \{1, \ldots, \ell\}$ such that*

$$\mathcal{Q} \cap S_m = \emptyset$$

That is, a strong subset of players does not intersect at least one subset S_m in the basis.

Given a set of indices $I \subseteq \{1, 2, \ldots, n\}$ a corresponding set of players $\mathcal{P}_I$ is determined ($P_i \in \mathcal{P}_I$ iff $i \in I$.) We say that a set of indices I is admissible (resp. weak, resp. strong) if the corresponding set of players $\mathcal{P}_I$ is admissible (resp. weak, resp. strong.)

3.1.2 Adversaries

We model the corrupted processors as being coordinated by an *adversary* $\mathcal{A}$. As in Section 2.1, this adversary will be dynamic, all-powerful, completely-informed. He is also allowed rushing (See Section 2.1, for the definitions of the above terms.)

In the previous chapter we dealt with t-adversaries, that is adversaries who were allowed to corrupt at most t players. For example the protocols by Ben-Or, Goldwasser and Wigderson [BGW88] and Chaum, Crepeau and Damgard [CCD88] enforce a bound of $n/3$ corrupted players. Tal Rabin improves this to $n/2$ in [Rab94]. Given a network of players, with an access structure $\mathcal{F}$ defined on it, we need to define what kinds of adversary can be allowed in order to obtain fault-tolerance in the secret sharing process. In our case, we cannot enforce a simple numeric bound since the number of corrupted processors is not a significant value in the access structure model (for example an adversary could corrupt only 3 players and learn the secret or disrupt completely the protocol.) We need to correlate the power of the adversary to the access structure itself. This is what we do in the following.

We say that a subset of players is corrupted if one of its members is corrupted. We say that a subset of players is pure, if it is not corrupted, that is all its members are good throughout the protocol. We say that a subset is fully corrupted if all its members are corrupted.

Let $\mathcal{F}$ be an access structure and $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$ its corresponding basis.

Definition 3.4 *We say that an adversary $\mathcal{A}$ is* admissible *for $\mathcal{F}$ if the set of players corrupted by $\mathcal{A}$ during a protocol is $\mathcal{F}$-admissible.*

This means that for every authorized subset R there exists at least one player in R which is good throughout the protocol, or in other words no authorized subset becomes fully corrupted. The notion of admissibility reflects the need for privacy in the protocol. Indeed in the (t, n) threshold case we assume that an adversary does not corrupt more than t players, otherwise she learns the secret in advance. Similarly in this case we need to prevent adversaries who corrupt a whole authorized subset, since they would learn the secret in advance as well. This requirement is needed even if the adversary is not malicious, that is, limits its activity to gather information from the corrupted players, but let them act correctly during the protocol (so-called gossip adversaries.)

Definition 3.5 *We say that an admissible adversary $\mathcal{A}$ is $\mathcal{F}$-*weak *if the set of players corrupted by $\mathcal{A}$ during a protocol is $\mathcal{F}$-weak.*

This means that more than half of the basis subsets have to remain pure during the protocol, or in other words, we allow the adversary to corrupt at most half of the subsets in the basis. We will see that this definition of weak adversary will yield a simple and intuitive VSS protocol.

This seems to be in some way the equivalent of the threshold case in which we allow the adversary to corrupt only half of the players ($t < n/2$). In fact, the two things are *not* the equivalent of each other. Looking at honest majority in the threshold case from the access structure point of view, one notices that only *one* authorized subset is guaranteed to remain pure, while all the other ones can contain corrupted players. So we formulate the following definition of strong adversary:

Definition 3.6 *We say that an admissible adversary is $\mathcal{F}$-*strong *if the set of players corrupted by $\mathcal{A}$ during a protocol is $\mathcal{F}$-strong.*

Any adversary (weak or strong) can corrupt the dealer.

3.2 Definitions

In this section we adapt the definition by Feldman and Micali [FM88] described in Section 2.3 to the access structure model. As pointed out in Section 2.5 this definition is not completely satisfactory as it does not provide the reducibility property. However in order to establish a definition similar to the one in Section 2.4 we need a notion of secure function evaluation in the access structure scenario. We do not have such a notion as of today, actually we see our work on VSS as a first attempt to devise and analyze such a notion.

So in order to explain better the technical details of our work we will be interested only in stand-alone VSS protocols and stick to this less satisfactory but more intuitive definition. In Section 3.7 we will come back to the problem of secure function evaluation based definitions of VSS over general access structures.

Definition 3.7 Let P be a pair of protocols where the second is always executed after the first one, P =(`Share-Verify, Recover`) for the players $P_1, \ldots, P_n, D$. In protocol `Share-Verify`, D has a secret input s; the output of player P_i is a value $verification_i \in \{yes, no\}$. In protocol `Recover`, the input of each player P_i is his computational history at the end of the previous execution of `Share-Verify`; the output of each P_i is a string σ_i.

We say P is a $\mathcal{F}$-weak (resp. $\mathcal{F}$-strong) VSS protocol if the following 3 properties are satisfied:

1. Acceptance of good secrets: In all executions of `Share-Verify` with a $\mathcal{F}$-weak (resp. $\mathcal{F}$-strong) adversary $\mathcal{A}$ in which the dealer is good, $verification_i = yes$ for all good players P_i.

2. Verifiability/Reconstructability: If at the end of `Share-Verify` the set of players outputting $verification_i = no$ is $\mathcal{F}$-weak (resp. $\mathcal{F}$-strong) then at this time a value σ has been fixed (see def. 2.10) and at the end of `Recover` all good players will output the same value σ. Moreover if the dealer is good $\sigma = s$ the secret. [1]

3. Unpredictability: In a random execution of `Share-Verify` with a good dealer and the secret chosen randomly in a set of cardinality p any $\mathcal{F}$-weak (resp. $\mathcal{F}$-

[1] Notice that if we simply ask in the Verifiability condition that "all the good players output the same number σ at the end of the `Recover` phase" it would not be sufficient for our purposes. In fact, we would still allow the adversary to decide during `Recover` what value σ the good players will output. Thus `Share-Verify` would not model a secret commitment as required.

strong) adversary $\mathcal{A}$ will not be able to predict the secret better than at random i.e. if $\mathcal{A}$ outputs a number a at the end of `Share-Verify` then $Prob[a = s] = \frac{1}{p}$

4. **Unanimity**: If a good player output yes then all good players will output yes at the end of `Share-Verify`.

3.2.1 A Note on Efficiency

As usual we are only interested in **efficient** protocols, that is protocols that do not require too much computational effort from the players. Let $\mathcal{F}$ be an access structure and $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$ it basis. Assume the secret s is an integer $s \leq p$. In our case we require that the protocol needs at most $\text{poly}(n, |p|, \ell, \log \frac{1}{\epsilon})$ computational steps to complete, where ϵ is the probability of error in the protocol (taken over the coin tosses of the players.)

Notice the dependency on ℓ the "size" of the access structure. In general this is inevitable. However there are particular access structures (as the threshold one) for which the dependency on ℓ may not be necessary. We will come back to this point in Section 3.7.

3.2.2 A note on reconstruction

In a secret sharing protocol an authorized subset will be able to reconstruct the secret on its own without cooperating with players outside the subset. This is true in both the access structure model and the threshold model (in which the basis subsets are all the subsets of cardinality $t + 1$.)

However this cannot be true in an adversarial model like ours. Indeed if an adversary can corrupt players in the network and make them behave in any possible way, then the value reconstructed by an authorized subset has no guarantee to be the correct secret. The only way to achieve this guarantee is to use the help of the other players in order to defeat the maliciously behaving players. Once again this is true in both the access structure model and the threshold model. Indeed if one looks at the VSS protocols in Chapter 2 one notices that in order to reconstruct the secret all

n players have to participate in the protocol, although in theory any $t+1$ of them would be enough. It is also true in those protocols that, if we assume that there are no maliciously behaving players, only $t+1$ players are needed to reconstruct the secret since we know that they are going to contribute the correct share.

Our VSS protocols over access structure in this chapter follow the same paradigm. In the presence of maliciously behaving players, we need the cooperation of all n players in order to reconstruct the secret. However an authorized subset composed of all good players get will be able to reconstruct the secret without the help of the other players. Of course there must be some other way to ensure that such subset does not contain a bad player. Indeed the role of the other players in our protocol is to detect and eliminate bad players from the reconstruction phase.

3.3 The weak adversary case

In this section we show a weak VSS protocol, that is a protocol that is guaranteed to work against a weak adversary. The protocol is based on similar techniques as [BGW88, CCD88], but applied to the access structure secret sharing protocol described in Section 1.2.3.

3.3.1 A sketch of the protocol

The basic idea of our weak VSS protocol is based on the cut-and-choose technique introduced in [BGW88, CCD88]. Let K be a security parameter. During `Share-Verify` the dealer will share the secret s and Kn additional auxiliary secrets $t^{(1)}, \ldots, t^{(Kn)}$ according to the scheme in Section 1.2.3. Then, for each $i = 1, \ldots, Kn$ he will be asked to reveal the sharing of either $t^{(i)}$ or of the sum $t^{(i)} + s$, at random. In either case each player checks two things:

- that the shares broadcasted by the dealer match the ones in his possession.
- that the revealed sharing s are proper, i.e. the shares of each authorized subset sum up to the same value.

If this is not the case, the player complains against the dealer. This is followed (as in section 2.6) by a round in which the dealer reveals the information of the complaining players (since either they are bad or the dealer is bad, so no extra information is given to the adversary anyway.) The players repeat the zero-knowledge cut-and-choose proof, checking this time also the revealed information. At the end the players assume that a secret has been shared correctly if the set of complaining players is an $\mathcal{F}$-weak one, (indeed bad players can always complain so we can have a $\mathcal{F}$-weak set of complaining players even in the presence of an honest dealer.) We will show in Theorem 3.1 that, with high probability, at the end of this protocol a value will be fixed as the shared secret among the players.

This implies that all the pure subsets will agree on the same value as the secret (i.e., the sum of the shares in the subset.) Nothing can be said about the value produced by corrupted subsets. However, we know that less than half of the basis subsets are corrupted, so it is sufficient to take as the secret the value that appears most often among the values reconstructed by each basis subset. This is basically an error-correcting mechanism based on the redundancy of the sharing process. The `Recover` part can then be simply implemented by having each player distribute his share among the others.

3.3.2 The Weak Protocol

We are now ready to describe the weak VSS protocol that satisfies Definition 3.7. The protocol appears in Figures 3-1 and 3-2.

Theorem 3.1 *Let $\mathcal{F}$ be an access structure. Protocol* `VSS-1 =(Share-Verify-1,Recover-1)` *is a $\mathcal{F}$-weak Verifiable Secret Sharing protocol according to Definition 3.7.*

Proof The first condition *Acceptance of good secrets* is trivially met, since all good players will output *yes* if the dealer follows the prescribed instructions of Protocol VSS-1.

The last condition *Unpredictability* is satisfied because we assume that the adversary is admissible. Indeed, recall that the secret is shared via a $(k_m - 1, k_m)$

Share-Verify-1

Input for all players: p a prime. $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$ the basis of the access structure $\mathcal{F}$, $S_m = \{P_{m,1}, \ldots, P_{m,k_m}\}$. K a security parameter.
Input for the dealer: $s \in Z_p$, the secret.

1. For each set S_m the dealer chooses random values $s_{m,1}, \ldots, s_{m,k_m} \in Z_p$ such that $s_{m,1} + \ldots + s_{m,k_m} = s \bmod p$. The dealer gives $s_{m,j}$ to player $P_{m,j}$.

2. For each $\alpha = 1, \ldots, 2Kn$ the dealer chooses a random value $t^{(\alpha)} \in Z_p$. He shares it in the same fashion as the original secret s. That is, for each set S_m the dealer chooses random values $t^{(\alpha)}_{m,1}, \ldots, t^{(\alpha)}_{m,k_m} \in Z_p$ such that $t^{(\alpha)}_{m,1} + \ldots + t^{(\alpha)}_{m,k_m} = t^{(\alpha)} \bmod p$. The dealer gives $t^{(\alpha)}_{m,j}$ to player $P_{m,j}$.

3. Each player broadcasts K bits. For ease of notation let us denote them by $b_1, \ldots, b_{Kn}$.

4. For each $\alpha = 1, \ldots, Kn$ the dealer broadcasts the values $\tau^{(\alpha)}_{m,j} = b_\alpha s_{m,j} + t^{(\alpha)}_{m,j} \bmod p$ for all possible m and j.

5. Let $\tau^{(\alpha)}_m = \sum_j \tau^{(\alpha)}_{m,j}$. Each player checks the following two things:
   - there exists a $\tau^{(\alpha)}$ such that $\tau^{(\alpha)}_m = \tau^{(\alpha)}$ for all m
   - $\tau^{(\alpha)}_{m,j} = b_\alpha s_{m,j} + t^{(\alpha)}_{m,j} \bmod p$ for the indices (m, j) relative to him

   If the checks are satisfied then the player output *yes* otherwise he outputs *no*.

6. If the set of players outputting *no* is $\mathcal{F}$-weak then the dealer broadcasts all the values given in rounds 1 and 2 to players who output *no*. Otherwise the dealer is disqualified and players set $s_{m,j} = 0$ and STOP.

7. Each player broadcasts K bits. For ease of notation let us denote them by $\beta_1, \ldots, \beta_{Kn}$.

8. For each $\alpha = 1, \ldots, Kn$ the dealer broadcasts the values $\tau^{(Kn+\alpha)}_{m,j} = \beta_\alpha s_{m,j} + t^{(Kn+\alpha)}_{m,j} \bmod p$ for all possible m and j.

9. Let $\tau^{(Kn+\alpha)}_m = \sum_j \tau^{(Kn+\alpha)}_{m,j}$. Each player checks the following two things:
   - there exists a $\tau^{(Kn+\alpha)}$ such that $\tau^{(Kn+\alpha)}_m = \tau^{(Kn+\alpha)}$ for all m
   - $\tau^{(Kn+\alpha)}_{m,j} = \beta_\alpha s_{m,j} + t^{(Kn+\alpha)}_{m,j} \bmod p$ for the indices (m, j) relative to him and also for the value broadcasted by the dealer in step 6.

   If the checks are satisfied then the player output *yes* otherwise he outputs *no*.

10. If the set of players outputting *no* is $\mathcal{F}$-weak then the players assume a secret has been shared correctly. Otherwise the dealer is disqualified and players set $s_{m,j} = 0$ and STOP.

Figure 3-1: Sharing a secret against weak adversary

`Recover-1`

Input for all players: The history at the end of `Share-Verify-1`

1. Each player P_i distributes the value $s_{m,j}$ for the indices (m,j) relative to him.
2. Each player sets $\sigma_m = \sum_{j=1}^{k_m} s_{m,j}$ and outputs $s = \text{MAJORITY}(\sigma_1, \ldots, \sigma_\ell)$.

Figure 3-2: Reconstructing the secret against a weak adversary

threshold scheme between the members of each authorized subset, and notice that all these sharings are independent. An admissible adversary cannot corrupt at least one player from each authorized subset, so this mean that the shares $s_{m,j}$ give no information to the adversary about the secret. Furthermore shares from different subsets S_m give no information as well since the sharings are independent from each other.

Also we need to prove that the values revealed by the dealer in rounds 4 and 8 yield no information about the secret to the adversary. As the $t^{(i)}$ are randomly chosen and uniformly distributed in Z_p (if the dealer is honest) then we know that the values $\tau^{(i)}$ are random and uniformly distributed in Z_p so they yield no information about s.

Proving that the second condition *Verifiability/Reconstructability* is met needs some more work. Assume that at the end of `Share-Verify-1` the players who output *no* are a $\mathcal{F}$-weak set. The value σ that is fixed at this time is the value that appears more often among the values $s_m = \sum_j s_{m,j}$. Clearly this is the value the good players output at the end of `Recover-1`. We need to prove that there is such a value at least with high probability. It is enough to prove that pure subsets S_m (there is a majority of them) agree on the same value $\sigma_m = \sigma$.

First of all notice that at least K bits are chosen at random in rounds 3 and 7 (as there is at least one good player.) Each bit represents a "question" that a bad dealer who did not perform properly in round 1 will be able to answer correctly only with probability $\frac{1}{2}$ (i.e. if he predicted the bit correctly when he distributed the shares).

So 2^{-K} is a bound on the probability that the dealer guesses the "questions" of the good players.

Assume that the values σ_m determined by pure subsets are not identical. Since the dealer has not been disqualified, we know that all the values $\tau_m^{(i)}$ are identical. This means that some of the values $t^{(i)}$ have not been shared correctly. But, for at least K indices i, this sharing has been revealed randomly with probability 1/2.

Because of these improper sharings, some good players will output *no* at the end of round 5. The dealer broadcasts the values $s_{m,j}$ and $t_{m,j}^{(\alpha)}$ for these players in round 6. These values are checked again in the second questions and answer session, yielding a proper sharing at least for the good players. ■

3.4 Strong adversaries: the honest dealer case

In this section we work against a strong adversary, which means that we can assume that there is only one pure subset in $B_{\mathcal{F}}$ and that for each authorized subset there is at least one good player in it. The full solution in this case is quite complicated and we show it in the next section. For now, just to introduce to the reader the techniques used in the full solution, we also make the additional assumption that the dealer is good.

Under this assumption we do not need to verify that a secret has been shared correctly. The only thing that we must assure is that the secret is reconstructable at recover time in spite of the presence of potentially so many bad players [2].

The basic idea is drawn for Tal Rabin's check vectors technique [Rab94]. To make sure that a player broadcasts the correct share at recover time, the dealer distributes unconditionally secure authentication codes for the shares. When the share is broadcasted only the ones that can be authenticated are accepted as good. The full details of the protocol appear in Figures 3-3 and 3-4.

[2] Indeed rather than VSS this protocol should be called RSS for Reconstructable Secret Sharing

`Share-Verify-HD`

Input for all players: K a security parameter. p a prime larger than 2^K. $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$ the basis of the access structure $\mathcal{F}$, $S_m = \{P_{m,1}, \ldots, P_{m,k_m}\}$.
Input for the dealer: $s \in Z_p$, the secret.

1. For each set S_m the dealer chooses random values $s_{m,1}, \ldots, s_{m,k_m} \in Z_p$ such that $s_{m,1} + \ldots + s_{m,k_m} = s \bmod p$. The dealer gives $s_{m,j}$ to player $P_{m,j}$.
2. For each player P_i and for each pair (m, j), the dealer chooses at random $b^{(i)}_{m,j}, y^{(i)}_{m,j} \in Z_p$ and sets
$$c^{(\alpha)}_{m,j} = b^{(\alpha)}_{m,j} s_{m,j} + y^{(\alpha)}_{m,j} \bmod p \tag{3.1}$$
3. The dealer gives the values $y^{(i)}_{m,j}$, $\forall\ i$, to player $P_{m,j}$
4. The dealer gives the values $b^{(i)}_{m,j}$, $c^{(i)}_{m,j}$, $\forall\ (m, j)$ to player P_i

Figure 3-3: Sharing a secret against a strong adversaries when the dealer is honest

`Recover-HD`

Input for all players: the values received by the dealer during the `Share-Verify-HD` phase.

1. Each player distributes the values $s_{m,j}$ and the values $y^{(i)}_{m,j}$, $\forall\ i$ for the indices (m, j) relative to him.
2. Each player P_i checks for each pair (m, j) that equation (3.1) is satisfied using the values $b^{(i)}_{m,j}$ and $c^{(i)}_{m,j}$ in his possession.
3. If the equation (3.1) is satisfied for the pair (m, j) then P_i broadcasts the value $ACC^{(i)}_{m,j}$, otherwise he broadcasts the value $REJ^{(i)}_{m,j}$.
4. A share $s_{m,j}$, is accepted if the set of players P_i who broadcasted $REJ^{(i)}_{m,j}$ is strong. Let S_m be any set whose shares $s_{m,j}$ have all been accepted. Each player outputs $s = \sum_{j=1}^{k_m} s_{m,j}$.

Figure 3-4: Reconstructing the secret against a strong adversary when the dealer is honest

Theorem 3.2 *Let $\mathcal{F}$ be an access structure. Under the assumption that the dealer is honest, Protocol* `VSS-HD = (Share-Verify-HD,Recover-HD)` *is a $\mathcal{F}$-strong Verifiable Secret Sharing protocol. according to Definition 3.7*

Proof Since we are assuming that the dealer is honest we have to prove only that at the end of `Share-Verify-HD` the adversary has no information about the secret and that at the end of `Recover-HD` she did not prevent the good players from learning the secret.

Since the adversary is admissible, there is no authorized subset which is fully corrupted. This means that the adversary has no information about the secret from the shares $s_{m,j}$ in her possession, since she only has at most $k_m - 1$ shares of different independent $(k_m - 1, k_m)$-threshold sharings. At the same time she cannot get any information from the check vectors since they are unconditionally secure message authentication codes (see [Rab94]) so they leak no information about the missing shares.

In order to prove that the secret is successfully reconstructed we need to prove that the shares accepted at the end of `Recover` are all the good ones and only the good ones.

Let's prove first that all good shares $s_{m,j}$ are accepted at the end of `Recover-HD`. Indeed if $s_{m,j}$ is good then all the other good players will accept it and output $ACC^{(i)}_{m,j}$. So only bad players can output $REJ^{(i)}_{m,j}$ on a good share $s_{m,j}$. Since the adversary is strong, this means that the set of players who outputs $REJ^{(i)}_{m,j}$is strong. By the instruction in round 4 this means that $s_{m,j}$ is accepted.

Now let's prove that the probability that a bad share is accepted at the end of `Recover` is at most $\frac{n^2\ell}{p}$. Indeed as Rabin proves in [Rab94], the probability that a bad share $\hat{s}_{m,j}$ is accepted is equal to the probability that $P_{m,j}$ guesses a value $\hat{y}^{(i)}_{m,j}$ such that the pair $(\hat{s}_{m,j}, \hat{y}^{(i)}_{m,j})$ passes equation (3.1). This probability is exactly $\frac{1}{p}$. Since it might be enough for the bad player that just one among the n players accepts the share, the total probability of accepting a bad share $\hat{s}_{m,j}$ is bounded by $\frac{n}{p}$. Since each player holds up to ℓ shares the probability that any bad player (there are at most n of them) manages to have one of his shares accepted at the end of `Recover` is bounded by $\frac{n^2\ell}{p}$. By choosing the security parameter large enough this probability can be made smaller than any ϵ. ■

3.5 Strong adversaries: the full case

In this section we describe the full VSS protocol for the case of strong adversaries. The protocol is a modification of Rabin's threshold VSS protocol with honest majority [Rab94]. We proceed in two phases: first we describe a protocol which has only some of the desired features of a VSS, we call it a Semi-VSS. Then using Semi-VSS as a tool we construct the full solution.

3.5.1 Semi-Verifiable Secret Sharing

This protocol models a form of semi-commitment in the sense that the dealer has the option to get away with an incorrect sharing that does not represent a secret. However if this is the case the dealer will be discovered and disqualified during the Recover phase. But if the secret is recovered, then it is the secret originally shared by the dealer. So basically what we require from this protocol is that bad players will not be able to

1. change the shared secret at reconstruction time even if they collaborate with the dealer.
2. prevent the reconstruction of the secret when the dealer is honest.

Our protocol is modeled after a similar protocol by Rabin [Rab94]. She calls her protocol WSS for Weak Secret Sharing, but we chose the name of Semi-VSS to avoid confusion with our notion of weak adversary.

The basic idea is that the dealer shares a secret using the protocol in Section 1.2.3. As we said above, we don't care if he is dishonest and the shares do not identify a secret. However we want to prevent the players from changing the value of the secret. In order to do so we use the check vectors again as in the honest dealer case.

Although we do not care about the correctness of the sharing itself, we do care about the correctness of the check information, otherwise a bad dealer and bad players could change the value of the secret at reconstruction time. To this purpose an extra phase is added to allow good players to make sure that the check information held by other players match the share in their possession.

We adapt Rabin's protocol to the access structure scenario. The protocol appears in full detail in figures 3-5.

`Semi-Share-Verify`

Input for all players: K a security parameter. p a prime larger than 2^K. $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$ the basis of the access structure $\mathcal{F}$, where $S_m = \{P_{m,1}, \ldots, P_{m,k_m}\}$. **Input for the dealer:** $s \in Z_p$, the secret.

1. For each set S_m the dealer chooses random values $s_{m,1}, \ldots, s_{m,k_m} \in Z_p$ such that $s_{m,1} + \ldots + s_{m,k_m} = s \bmod p$. The dealer gives $s_{m,j}$ to player $P_{m,j}$.
2. For each player P_i and for each pair (m, j), the dealer chooses at random $\vec{b}^{(i)}_{m,j}, \vec{y}^{(i)}_{m,j} \in Z_p^{2K}$ and sets componentwise
$$\vec{c}^{(i)}_{m,j} = \vec{b}^{(i)}_{m,j} s_{m,j} + \vec{y}^{(i)}_{m,j} \bmod p \tag{3.2}$$
3. The dealer gives the values $\vec{y}^{(i)}_{m,j}$, $\forall i$, to player $P_{m,j}$
4. The dealer gives the values $\vec{b}^{(i)}_{m,j}$, $\vec{c}^{(i)}_{m,j}$, $\forall (m, j)$ to player P_i
5. $P_{m,j}$ chooses K of P_i's check vectors and asks him to broadcast them.
6. P_i broadcasts the required check vectors
7. The dealer either approves the check vectors broadcast by P_i or announces that P_i is faulty.
8. $P_{m,j}$ checks that the broadcasted check vectors (if the dealer did not disqualify P_i) satisfy the required condition (3.2). If not, he asks the dealer to broadcast all his shares $s_{m,j}$ for the indices m, j relative to him.

Figure 3-5: Semi-sharing a secret against a strong adversary

In order to Recover the secret we run a protocol similar to `Recover-HD`. The main difference is that we check if the dealer shared the secret properly. Details in figure

3-6.

Semi-Recover

Input for each player: the values received by the dealer during the `Share-Verify` phase.

1. Each player distributes the values $s_{m,j}$ and the values $y^{(i)}_{m,j}$ $\forall\alpha$ for the indices (m,j) relative to him.
2. Each player P_i checks for each pair (m,j) that equation (3.2) is satisfied for at least one of the unrevealed component using the values $b^{(i)}_{m,j}$ and $c^{(i)}_{m,j}$ in his possession.
3. If the test is satisfied for the pair (m,j) then P_i broadcasts the value $ACC^{(i)}_{m,j}$, otherwise he broadcasts the value $REJ^{(i)}_{m,j}$.
4. A share $s_{m,j}$ is accepted if the set of players P_i who output $REJ^{(i)}_{m,j}$ is $\mathcal{F}$-strong. Let $s_m = \sum_j s_{m,j}$ for all subsets S_m whose shares $s_{m,j}$ have all been accepted. If there exists a value s such that $s_m = s$ for all m, then the players output s otherwise they disqualify the dealer.

Figure 3-6: Reconstructing the a semi-shared secret against a strong adversary

The properties of `Semi-VSS=(Semi-Share-Verify, Semi-Recover)` are stated in the following Lemmas.

Lemma 3.1 *Let $\mathcal{F}$ be an access structure. At the end of* `Semi-Share-Verify` *if the dealer is honest a $\mathcal{F}$-strong adversary cannot predict s better than at random.*

Proof Since the adversary is strong, she is also admissible, thus there is no authorized subset which is fully corrupted. This means that the adversary has no information about the secret from the shares $s_{m,j}$ in her possession, since she only has at most $k_m - 1$ shares of different independent $(k_m - 1, k_m)$-threshold sharings. At the same time she cannot get any information from the check vectors since they are unconditionally secure message authentication codes (see [Rab94]) so they leak no information about the missing shares. So the only way the adversary can get information about a share held by a good player is if such player asks the dealer to reveal his share in round 8 of the protocol. But it is easy to see that such request happens only if either the dealer or the player is bad. ■

Lemma 3.2 *Let $\mathcal{F}$ be an access structure and assume we are in the presence of a $\mathcal{F}$-strong adversary. If the dealer is honest, at the end of* `Semi-VSS` *the good players output the secret s with probability $\geq 1 - \frac{n^2 \ell K}{p}$.*

Proof In order to prove that the secret is successfully reconstructed we need to prove that with high probability the shares accepted at the end of `Semi-Recover` are all the good ones and only the good ones.

Clearly all good shares $s_{m,j}$ are accepted by a reasoning similar to the one in the proof of Theorem 3.2.

Now let's prove that the probability that a bad share is accepted at the end of `Semi-Recover` is at most $\frac{n^2 \ell K}{p}$. Indeed as Rabin proves in [Rab94], the probability that a bad share $\hat{s}_{m,j}$ is accepted is equal to the probability that $P_{m,j}$ guesses a value $\hat{y}^{(i)}_{m,j}$ such that the pair $(\hat{s}_{m,j}, \hat{y}^{(i)}_{m,j})$ passes equation (3.2) for at least one of the K components being checked. This probability is exactly $\frac{K}{p}$. Since it might be enough for the bad player that just one among the n players accepts the share, the total probability of accepting a bad share $\hat{s}_{m,j}$ is bounded by $\frac{nK}{p}$. Since each player holds up to ℓ shares the probability that *any* bad player (there are at most n of them) manages to have one of his shares accepted at the end of `Semi-Recover` is bounded by $\frac{n^2 \ell K}{p}$. ■

Lemma 3.3 *Let $\mathcal{F}$ be an access structure and assume we are in the presence of a $\mathcal{F}$-strong adversary. At the end of* `Share-and-Semi-Verify` *a value σ has been fixed such that with probability $\geq 1 - \frac{n^2 K}{2^K}$ at the end of* `Semi-Recover` *all the honest players will either output σ or disqualify the dealer.*

Proof Consider the shares of the players at the end of `Semi-Share-Verify` (that is for each player, the one held by him or the one broadcasted by the dealer in step 8 if the player made such request.)

These shares either identify a value σ (i.e. their sum for each authorized basis subset is equal to σ) or they don't. In the latter case we know the players disqualify the dealer at the end of `Semi-VSS`. So let's assume that they identify a value σ. Let's prove that the probability that the output of `Semi-Recover` is $\sigma' \neq \sigma$ is at most $\frac{n^2 \ell K}{2^K}$.

Let's assume that the output of `Semi-Recover` is $\sigma' \neq \sigma$. This means that each pure subset (whose shares would sum up to σ and not σ') has been somewhat eliminated. We know there is at least one pure subset S_m so, in order for this to happen it must be that a good share $s_{m,j}$ held by a good player $P_{m,j} \in S_m$ (i.e. not broadcasted by the dealer in round 8 of `Semi-Share-verify`) was not accepted by another good player P_i (otherwise in round 4 the share $s_{m,j}$ would be accepted.)

Let π be the probability that $s_{m,j}$ is not accepted by P_i. Since there are at most n^2 pairs of good players, the total probability is bounded by $n^2\pi$.

So the lemma is proven once we prove that $\pi = \frac{K}{2^K}$. π is the probability that $s_{m,j}$ is not accepted by P_i while $P_{m,j}$ is convinced that P_i would accept it (i.e. does not ask the dealer to broadcast it in round 8.) This means that of the $2K$ components of the vectors in equation (3.2), K must satisfy the equation (the ones chosen by $P_{m,j}$ during the verification) and K must not satisfy it (otherwise P_i would accept). The probability that such event happens is that $P_{m,j}$ chooses the only subset of K correct components at round 5 of `Semi-Share-Verify`. Such probability is is $\frac{1}{\binom{2k}{k}} \leq \frac{k}{2^k}$ ■

3.5.2 The full protocol

Now we are ready to describe the full solution for VSS over general access structure against a strong adversary. This can be considered a modification of Rabin's protocol [Rab94] to work over access structures.

The protocol is loosely based on the ideas behind VSS against a weak adversary. The main difference with the previous protocol is that during the `Share-Verify` phase each player will Semi-VSS his own shares among the other players. So while in the previous protocol bad players could contribute bad shares, here the shares are reconstructed from the Semi-VSS process and so either they are correct or the corresponding player is disqualified. The cut-and-choose zero-knowledge proof for the correctness of the sharing by the dealer is modified accordingly to make sure that players will necessarily do a Semi-VSS of the true shares received from the dealer. Details in Figures 3-7 and 3-8.

Theorem 3.3 *Protocol* VSS-2 *is a Verifiable Secret Sharing protocol against a strong admissible adversary according to Definition 2.11.*

We prove this theorem in a series of Lemmas.

Lemma 3.4 (Acceptance of Good Secrets). *Let $\mathcal{F}$ be an access structure and assume to be in the presence of an $\mathcal{F}$-strong adversary. If the dealer is honest, at the end of* `Share-Verify-2` *all good players output yes.*

Proof Trivial, since if the dealer is honest all the checks of the good players at round 7 are always satisfied (either on the reconstructed or on the broadcasted shares.) ∎

Lemma 3.5 (Unpredictability). *Let $\mathcal{F}$ be an access structure and assume to be in the presence of an $\mathcal{F}$-strong adversary. If the dealer is honest, at the end of* `Share-Verify-2` *the probability that a strong admissible adversary has any information about the secret is at most* $\frac{n^4K^2ell^2}{p}$

Proof As in the proof of Theorem 3.1 one can prove that the cut-and-choose protocol reveals no information to the adversary about s.

Also we know from Lemma 3.1 that performing `Semi-Share-Verify` of the shares of the good people reveals no information about them to the adversary.

So the only way the adversary can learn the share of a good player is that such share is broadcasted by the dealer at round 6. But this happens only if the `Semi-Recover` of any of the values shared by the good player fails. By Lemma 3.2 this happens with probability $\leq \frac{n^2\ell K}{p}$. There are at most n good players, each of them holds at most $\ell K n$ shares (we must include the shares of the test secrets as well) so the total probability of failure is $\leq \frac{n^4K^2\ell^2}{p}$ ∎

`Share-Verify-2`

Input for all players: K a security parameter. p a prime larger than 2^K. $B_{\mathcal{F}} = \{S_1, \ldots, S_\ell\}$ the basis of the access structure $\mathcal{F}$, where $S_m = \{P_{m,1}, \ldots, P_{m,k_m}\}$. **Input for the dealer:** $s \in Z_p$, the secret.

1. For each set S_m the dealer chooses random values $s_{m,1}, \ldots, s_{m,k} \in Z_p$ such that $s_{m,1} + \ldots + s_{m,k} = s \bmod p$. The dealer gives $s_{m,j}$ to player $P_{m,j}$.
2. For each $\alpha = 1, \ldots, Kn$ the dealer chooses a random value $t^{(\alpha)} \in Z_p$. He shares it in the same fashion as the original secret s. That is for each set S_m he chooses random values $t^{(\alpha)}_{m,1}, \ldots, t^{(\alpha)}_{m,k} \in Z_p$ such that $t^{(\alpha)}_{m,1} + \ldots + t^{(\alpha)}_{m,k} = t^{(\alpha)} \bmod p$. The dealer gives $t^{(\alpha)}_{m,j}$ to player $P_{m,j}$.
3. Each player $P_{m,j}$ performs a `Semi-Share-Verify` of the numbers $s_{m,j}$ and $t^{(\alpha)}_{m,j}$. Notice that this way he implicitly shares the number $u^{(\alpha)}_{m,j} = s_{m,j} + t^{(\alpha)}_{m,j}$. For this value he also gives out the appropriate check vectors.
4. Each player broadcasts K bits. For ease of notation let us denote all of them by $b_1, \ldots, b_{Kn}$.
5. For each $\alpha = 1, \ldots, Kn$ the players reconstruct the values $\tau^{(\alpha)}_{m,j} = b_\alpha s_{m,j} + t^{(\alpha)}_{m,j} \bmod p$ for all possible m and j using the `Semi-Recover` protocol.
6. The dealer checks that the values $\tau^{(\alpha)}_{m,j}$ correspond to the ones he gave out in step 1. For each pair (m, j) for which this is not true, the dealer broadcasts *all* the values given to the corresponding player $P_{m,j}$ in rounds 1 and 2. In particular this happens if the reconstruction of the value $\tau^{(\alpha)}_{m,j}$ fails. From now on all references to $P_{m,j}$ pieces will be intended to the pieces broadcasted by the dealer at this time.
7. Let $\tau^{(\alpha)}_m = \sum_{j=1}^{k_m} \tau^{(\alpha)}_{m,j}$. Each player checks that there exists a $\tau^{(\alpha)}$ such that $\tau^{(\alpha)}_m = \tau^{(\alpha)}$ for all m. If the checks are satisfied then the player output *yes* otherwise he outputs *no*.
8. labelssv-end If the set of players outputting *no* is strong then the players assume a secret has been shared, otherwise the dealer is disqualified.

Figure 3-7: Sharing a secret against a strong adversary

`Recover-2`

Input for each player: the history of the previous protocol `Share-Verify-2`

1. For each pair of indices (m, j) the players reconstruct the value $s_{m,j}$ using the `Semi-Recover` protocol. If the reconstruction fails, then the corresponding player is disqualified.
2. Let $s = \sum_j s_{m,j}$ for any subset S_m with no disqualified members. Output s as the secret.

Figure 3-8: Reconstructing the secret against a strong adversary

Lemma 3.6 (Verifiability/Reconstructability). *Let $\mathcal{F}$ be an access structure and assume to be in the presence of an $\mathcal{F}$-strong adversary. With overwhelming probability if at the end of* `Share-Verify-2` *the dealer is not disqualified a value σ has been fixed and at the end of* `Recover-2` *all good players will output the same value σ. Moreover if the dealer is good $\sigma = s$.*

Proof Let $\sigma_{m,j}$ the value fixed at the end of step 3 by the Semi-VSS performed by player $P_{m,j}$ or by the dealer at round 6 if he decides to broadcast the shares of player $P_{m,j}$.

First of all let's prove that these values with high probability define a secret σ (i.e that $\sigma_m = \sum_j \sigma_{m,j} = \sigma$ for all m.)

Indeed assume by contradiction that such values σ_m are not identical. Since the dealer has not been disqualified, we know that all the values $\tau_m^{(i)}$ are identical. This means that some of the values $t^{(i)}$ have not been shared correctly. But since for at least K indices i (there is at least one good player) this sharing has been revealed randomly with probability $1/2$, this event happens with probability $\leq 2^{-K}$.

Let us assume now that σ is well defined. At the end of `Recover-2` each good player will output the value σ unless

1. either the players reconstruct a different $\sigma_{m,j}$ for a pair (m, j)
2. or there is no subset S_m with no disqualified members.

Because of Lemma 3.3 we know that (1) happens with probability $\frac{\leq n^2K}{2^K}$ for each pair (m,j). There are at most $n\ell$ possible pairs (m,j). So the total probability of (1) happening is $\leq \frac{n^3\ell^2K}{2^K}$.

Now let's prove that with high probability there is at least one authorized subset with no disqualified members. Indeed there is at least one pure authorized subset (as we are working against a strong adversary). In order for this subset to end up with a disqualified member we have to witness the disqualification of a good player. A good player is disqualified if one of his Semi-VSS sharings fails to reconstruct. But this happens with probability $\leq \frac{n^2K}{2^K}$ (see Lemma 3.3.) There are n good players each of them performing at most $Kn\ell$ sharings. So the total probability of (2) is $\leq \frac{n^4\ell K^2}{2^K}$.

Finally we prove that if the dealer is honest the probability that $\sigma \neq s$ is $\leq \frac{n^3K\ell^2}{p}$. Indeed if the dealer is honest each $\sigma_{m,j} = s_{m,j}$ with probability $\geq 1 - 2^{-K}$ because if a bad player Semi-VSS something different it will be discovered by the dealer during the cut-and-choose protocol and his shares will be broadcasted. So the only way in which a different value can be output at the end of VSS-2 is if one of the reconstructions of a good $s_{m,j}$ will result in a different value. Because of Lemma 3.2 this happens with probability $\leq \frac{n^2\ell K}{p}$. Since there are at most $n\ell$ shares we get the desired probability.

The lemma is proven by taking the sum of all the failure probabilities and noticing that the total probability can be made as small as desired by an appropriate choice of the security parameters p and K. ■

3.6 Lower Bounds

In this section we prove that strong adversaries are the strongest possible adversaries that can be tolerated in order to obtain VSS protocols over general access structures.

Indeed if the adversary were not admissible this means that there is a subset $\mathcal{Q} \in \mathcal{F}$ fully corrupted during the protocol. Suppose the adversary corrupts these players at the beginning of the VSS protocol and then just listen on to all the conversations. This means that at the end of `Share-Verify` the adversary knows the value of the secret (as she knows all the points on a authorized subset) and this contradicts the

Unpredictability property in Definition 3.7. So the adversary *must* be admissible.

Suppose now that an adversary $\mathcal{A}_1$ can corrupt more players than a strong one. This means that there are no pure subsets in the access structure. Assume for example that $\mathcal{A}_1$ corrupts one player per subset $S_m \in B_{\mathcal{F}}$ at the beginning of the `Recover` phase and just crashes them, i.e. prevents them from participating to such phase. Call $\mathcal{Q}$ the subset of corrupted players. We claim that the remaining set of players $\mathcal{P} - \mathcal{Q}$ will not be able to reconstruct the secret, thus contradicting the Reconstructability property. Indeed assume they were able to do so, i.e. that there is a procedure that on input the shares held by these players can reconstruct the secret s. We prove that this contradicts again the *Unpredictability* requirement of Definition 2.11. Consider the adversary $\mathcal{A}_2$ that corrupts the set of player $\mathcal{P} - \mathcal{Q}$ at the end of `Share-Verify`. Notice that this is an admissible adversary since no authorized subset is fully corrupted. Using the above procedure at the end of `Share-Verify` $\mathcal{A}_2$ can determine the secret, which is a contradiction.

3.7 Future Directions

In this section we discuss some problems left open by the above treatment of VSS protocols over general access structures.

Broadcast channel/Error probability: Both our protocols rely on the existence of a broadcast channel and require a negligible probability of failure. Is this necessary? For example in the threshold case we know that it is possible to achieve VSS without a broadcast channel and in an error-free manner [BGW88, FM88]. Is it possible, at least for the weak adversary case, to devise a VSS protocol that is error-free and does not require a broadcast capability?

Ideal structures: The running time of our protocols depend on ℓ the "size" of the access structure. In general this is inevitable (see [BL88]), however there are access structures for which this is not necessarily the case (for example threshold access structures). Is it possible to classify those "ideal" structures for which there are VSS protocols whose running time does not depend on their size? Do they coincide with the ideal access structures studied in the secret sharing case, (i.e. the structures for which the shares come from the same space as the secret)?

EFFICIENCY: Is it possible to reduce the dependency on ℓ in our protocols?

MONOTONE FORMULA SHARING: In Section 1.2.3 we saw that it is possible to express any access structure via a monotone boolean formula of OR, AND and possibly THRESHOLD gates. Our algorithms use only the DNF formulation of the monotone formula. Is it possible to construct VSS protocols based on the algorithm of Benaloh-Leichter [BL88] for generic boolean monotone circuits? This would result in a more compact representation of the access structure and consequently in more efficient protocols.

SECURE COMPUTATION: Is it possible to define a notion of secure computation over general access structure. As in the threshold case [GMW87, BGW88, CCD88, RB89] the idea would be to have a set of n players and an access structure defined on it. Each player P_i holds an input x_i and we want to compute $f(x_1, \ldots, x_n)$ in such a way that an adversary (as the ones defined in this paper) will not be able to find out the input of a good player or disrupt the computation.

Once such notion is defined, it would be possible to redefine VSS protocols as we did in Chapter 2 casting them as special instances of secure function evaluation.

CHAPTER 4

Robust Threshold Signature Schemes

EDGARDO: *Son tue cifre?? Rispondi!*

LUCIA: *Si ...*

— *G. Donizetti,* Lucia di Lammermoor.

In the previous chapters we discussed the notion of Verifiable Secret Sharing in both the threshold and the access structure model. In this chapter we return to the threshold model to study one of the most interesting applications of VSS protocols: **robust threshold signature schemes**.

The idea of distributed signature schemes is to depart from the single-signer approach in which one person is the sole holder of a secret key, and to allow a group of people to "hold" the key in such a manner that they can, as a group, produce signatures, yet no person on his own can generate a signature. The signature which is generated by the group is the same as if it were generated by a single signer. More in detail: the secret key is shared by a group of n players. In order to produce a valid signature on a given message m, players engage in a communication protocol whose result will be a full signature on m. Informally a distributed signature scheme achieves threshold $t < n$, if no coalition of t (or less) players can produce a new valid signature, even after the system has produced many signatures on different messages. A signature resulting from a threshold signature scheme is the same as if it was produced by a single signer possessing the full secret signature key. In particular, the validity of this signature can be verified by anyone who has the corresponding unique public verification key. In other words, the fact that the signature was produced in a distributed fashion is transparent to the recipient of the signature.

Threshold signatures are motivated both by the need that arises in some organizations to have a group of employees agree on a given message (or a document) before signing it, as well as by the need to protect signature keys from the attack of internal and external adversaries. The latter becomes increasingly important with the actual deployment of public key systems in practice. The signing power of some entities, (e.g., a government agency, a bank, a certification authority) inevitably invites attackers to try and "steal" this power. The goal of a threshold signature scheme is twofold:

1. To increase the availability of the signing agency, by maintaining the service even in the presence of a number of faults in the system
2. at the same time to increase the protection against forgery by making it harder for the adversary to learn the secret signature key.

Indeed, the first goal could be achieved by simply duplicating the signing key among several servers; however such solution would also make forgery easier since now an attacker would have more servers to potentially attack to learn the key. A threshold solution achieves both goals, indeed now one only needs at least $t+1$ active servers to produce a signature, while an attacker has to break into that many servers to be able to forge signatures. It is important, though, that threshold signature schemes have the **robustness** property, i.e. the capacity of tolerating malicious faults in the process.

Notice that in particular, the threshold approach rules out the naive solution based on traditional secret sharing, where the secret key is shared in a group but reconstructed by a *single* player each time that a signature is to be produced. Such protocol would contradict the requirement that no t (or less) players can ever produce a new valid signature. In threshold schemes, multiple signatures are produced without an exposure or an explicit reconstruction of the secret key.

In this chapter we present robust threshold signature schemes for both RSA and DSS, the most widely used signature schemes in the cryptographic community. In designing these protocols we have put as first and foremost priority the **efficiency** of the schemes.

The Chapter continues as follows. In Section 4.1 we recall the notion of signature schemes. In Section 4.2 we define formally what robust threshold signature schemes are: the formalization and the following proofs of the protocols nicely fit with the formalism introduced in Chapter 2 of this thesis. In Section 4.5 and 4.6 we present the results about DSS and RSA respectively.

4.1 Signature Schemes

In this section we recall the definition of signature schemes and their security from [GMR88].

A signature scheme $\mathcal{S}$ is a triple of polynomial time randomized algorithms (`Key-Gen`, `Sig`, `Ver`). Let K be a security parameter.

`Key-Gen` is the **key generation** algorithm. On input 1^K it outputs a pair (y, x), such that y is the **public key** and x is the **secret key** of the signature scheme. y will be also called the **verification key** and x the **signing key**.

$$\texttt{Key-Gen}(1^K) = (y, x)$$

`Sig` is the **signing** algorithm: on input a message m and the secret key x, it outputs sig, a signature of the message m.

$$sig = \texttt{Sig}(m, x)$$

`Ver` is the **verification** algorithm. On input a message m, the public key y, and a string sig, it checks whether sig is a proper signature of m, i.e. generated by the signing algorithm `Sig`.

$$\texttt{Ver}(m, y, sig) = \begin{cases} YES & \text{if } sig = \texttt{Sig}(m, x) \\ NO & \text{otherwise} \end{cases}$$

What we would like from a signature scheme is that it is infeasible to generate valid signatures without knowledge of the private key x. Let F be a forger, that is an efficient algorithm that tries to attack the signature scheme. Various kinds of attacks can be mounted against a signature scheme.

Key-only attack F knows only the public key y of the scheme.

Known message attack F knows y and a number of pairs (m_i, sig_i) where $sig_i = \texttt{Sig}(m_i, x)$, where m_i are not chosen in any particular way (for example they could be random.) If F gets to choose the messages m_i, then we say that this is a **chosen message attack.**

Adaptive chosen message attack In this case F uses the signature algorithm as an **oracle.** That is F chooses one message m_1, gets it signed and receive $sig_1 = \texttt{Sig}(m_1, x)$. Now F chooses m_2 and receives back $sig_2 = \texttt{Sig}(m_2, x)$ and so on ...

What is F goal? We can distinguish them in order of importance

Total break Compute the secret key x.

Universal forgery Compute $sig = \texttt{Sig}(m, x)$ for any message m

Existential forgery Compute $sig = \texttt{Sig}(m, x)$ for a message m whose signature F has not seen before

The strongest signature scheme is clearly one which is existentially unforgeable against adaptive chosen message attack.

4.2 Robust Threshold Signature Schemes

Using a threshold signature scheme, digital signatures can be produced by a group of players rather than by one party. In contrast to the regular signature schemes where the signer is a single entity which holds the secret key, in threshold signature schemes the secret key is shared by a group of n players $P_1, \ldots, P_n$. In order to produce a valid signature on a given message m, individual players produce their **partial signatures** on that message, and then combine them into a full signature on m. A distributed signature scheme achieves threshold $t < n$, if no coalition of t (or less) players can produce a new valid signature, even after the system has produced many signatures on different messages. t is referred as the fault-tolerance parameter of the scheme.

A signature resulting from a threshold signature scheme is the same as if it was produced by a single signer possessing the full secret signature key. In particular, the validity of this signature can be verified by anyone who has the corresponding unique public verification key. In other words, the fact that the signature was produced in a distributed fashion is transparent to the recipient of the signature.

In this section we give formal models and definitions for these ideas.

4.2.1 Communication Model

The communication model is the same as the one described in Section 2.1. That is, we assume that our computation model is composed of a set of n *players* $\{P_1, \ldots, P_n\}$ who can be modeled by polynomial-time randomized Turing machines. They are connected by a complete network of private (i.e. untappable) point-to-point channels. In addition, the players have access to a dedicated broadcast channel; by dedicated we mean that if player P_i broadcasts a message, it will be recognized by the other players as coming from P_i.

4.2.2 The Adversary

The adversary is again an entity $\mathcal{A}$ that coordinates the action of corrupted players. $\mathcal{A}$ is defined as in Section 2.1, however in this case we assume that $\mathcal{A}$ is a computationally bounded machine. That is, $\mathcal{A}$ can perform only randomized polynomial time computations. Also we assume a static adversary, that is the set of corrupted players is decided upon once and for all at the beginning of the protocol. We consider t-adversaries, i.e., adversaries who are allowed to corrupt at most t players.

As we discussed in Section 2.1 corrupted processors may deviate in any way from the instructions given to them for the protocol being performed. In this chapter we will use an extra adjective to refer to this kind of adversary and call it Malicious Adversary. The reason for this is that we would like also to consider two weaker kind of adversaries:

- an Eavesdropping Adversary that learns all the information stored at the corrupted nodes and hears all the broadcasted messages, but will not change the instructions of the players during the protocol.
- a Halting Adversary who is an eavesdropping adversary that may *also* cause corrupted players to stop sending messages during the execution of the protocol (e.g., by crashing or disconnecting a machine).

Our threshold signature schemes tolerate a malicious adversary, so a fortiori they tolerate these weaker adversaries. However, the reason for considering them is that against such adversaries we can achieve a better efficiency of the scheme or a better fault-tolerance. As we said in the introduction to this chapter, these are primary goals for us now and thus, we believe it is worth differentiating among different kinds of adversaries if this results in efficiency gains. Then the most efficient protocol can be used depending on the situation at hand.

Given a protocol P the view of the adversary, is defined as in Section 2.1. That is, the view is the probability distribution (induced by the random coins of the players) on the knowledge of the adversary, namely, the computational history of all the corrupted players, and the public communications and output of the protocol.

4.2.3 Definitions

Let $\mathcal{S}$=(`Key-Gen`, `Sig`, `Ver`) be a signature scheme. A (t, n)-threshold signature scheme $\mathcal{TS}$ for $\mathcal{S}$ is a pair of protocols (`Thresh-Key-Gen`, `Thresh-Sig`) for the set of players $\{P_1, \ldots, P_n\}$.

`Thresh-Key-Gen` is a distributed key generation protocol used by the players to jointly generate a pair (y, x) of public/private keys. At the end of the protocol the private output of player P_i is a value x_i such that the values $(x_1, \ldots, x_n)$ form a (t, n)-threshold secret sharing of x. The public output of the protocol contains the public key y. The pairs (y, x) of public/secret key pairs are produced by `Thresh-Key-Gen` with the same probability distribution as if they were generated by the `Key-Gen` protocol of the regular signature scheme S.

`Thresh-Sig` is the distributed signature protocol. The private input of P_i is the value x_i. The public inputs consist of a message m and the public key y. The output of the protocol is the value $sig = \mathsf{Sig}(m, x)$.

Notice that the verification algorithm is the same as in the regular signature scheme $\mathcal{S}$, modeling the fact that the creation of the signature through a distributed protocol is transparent to the receiver of such a signature.

Security. We need to define what a secure threshold signature scheme is. In order to do so we go back to the definition of secure function evaluation in [MR92].

Consider the following n-input, n-output function KG-FUNC.

$$\text{KG-FUNC}(K \circ r_1 \circ r'_1, \ldots, K \circ r_n \circ r'_n) = (y \circ x_1, \ldots, y \circ x_n)$$

where

$$(y, x) = \texttt{Key-Gen}(1^K, \oplus r_i)$$

($\oplus$ stands for bitwise XOR and the second argument in `Key-Gen` is the random tape of the randomized algorithm `Key-Gen`) and

$$(x_1, \ldots, x_n) = \text{SHAR}(v_1, \ldots, v_n, x \circ \oplus r'_i)$$

where SHAR is the function defined in Section 2.4.

That is KG-FUNC takes as input a security parameter K and some randomness and returns as public output the value y which is a public key of a given signature scheme. The private output of player P_i is a value x_i such that the values $(x_1, \ldots, x_n)$ form a (t, n)-threshold secret sharing of x.

Definition 4.1 *We say that the protocol* `Thresh-Key-Gen` *is a secure threshold key generation protocol if it is a secure computation of the function* KG-FUNC *according to Definition 2.7.*

Consider now the following n-input, n-output function

$$\text{SIG-FUNC}(x_1 \circ \rho_1 \circ y \circ m, \ldots, x_n \circ \rho_n \circ y \circ m) = (sig, \ldots, sig)$$

where

$$(x_1 \circ y, \ldots, x_n \circ y) = \text{KG-FUNC}(K \circ r_1 \circ r'_1, \ldots, K \circ r_n \circ r'_n)$$

and

$$sig = \texttt{Sig}(m, x, \oplus\rho_i)$$

(again the last argument in the `Sig` algorithm represents `Sig` random tape).

Definition 4.2 *We say that the protocol* `Thresh-Key-Gen` *is a secure threshold signature protocol if it is a secure computation of the function* SIG-FUNC *according to Definition 2.7.*

Definition 4.3 *We say that* $\mathcal{TS}$=(`Thresh-Key-Gen, Thresh-Sig`) *is a secure threshold signature scheme if both* `Thresh-Key-Gen` *and* `Thresh-Sig` *are secure.*

INFORMAL LOOK. Recall that Definition 2.7 of secure function evaluation include both correctness and privacy. Using the terminology of the literature on distributed signature schemes, this means that our definition of security includes both robustness and unforgeability.

Robustness means that the protocol will compute a correct output even in the presence of halting or malicious faults. That is at the end of `Thresh-Key-Gen`, no matter what the bad players do, we really have a public key y and shares of the matching secret key x. And, very important, the distribution is the same as if they were generated by a single entity.

Similarly at the end of `Thresh-Sig` we do compute a valid signature for a message m, no matter what bad players may do. In particular we rule out possible denial of service attacks, by players who simply try to prevent the group from signing.

We say that a (t, n)-threshold signature scheme $\mathcal{TS}$ =(`Thresh-Key-Gen`,`Thresh-Sig`) is unforgeable, if no malicious adversary who corrupts at most t players can produce the signature on any new (i.e., previously unsigned) message m, given the view of the protocol Thresh-Key-Gen and of the protocol Thresh-Sig on input messages $m_1, \ldots, m_k$ which the adversary adaptively chose.

This is the analogous to the notion of existential unforgeability under chosen message attack as defined by Goldwasser, Micali, and Rivest [GMR88]. Following [GMR88] one can also define weaker notions of unforgeability (see Section 4.1.)

In order to prove unforgeability we make use of the privacy feature of Definition 2.7. Intuitively, by requiring that the adversary view is simulatable, we ask that the adversary who sees all the information of the corrupted players and the signature of m, could generate by itself all the other public information produced by the protocol `Thresh-Sig`. In other words, the run of the protocol provides no useful information to the adversary other than the final signature on m.

This is actually a stronger property than what we need. Indeed it would be enough for us to say that the executions of the protocols `Thresh-Key-Gen` and `Thresh-Sig` give the adversary no advantage in forging signatures for the scheme $\mathcal{S}$. In other words, we could allow the adversary to gain knowledge provided that such knowledge is useless for forging. However our stronger definition subsumes this specific goal and provides a proof of security for any of the "flavors" of signature security as listed in [GMR88]. Indeed one can prove that if the underlying signature scheme $\mathcal{S}$ is unforgeable (in any of the flavors of [GMR88]) and $\mathcal{TS}$ is simulatable then $\mathcal{TS}$ is unforgeable (with the same flavor as $\mathcal{S}$.)

THE SIMULATORS. The simulators for our protocols work mostly as described in Section 2.2. The only differences are

1. since the adversary is static the simulators receive the inputs of the bad players right at the beginning of the simulation;
2. since the adversary is computationally bounded we allow the simulated view to be computationally indistinguishable (see [GM84]) from the real one.

4.2.4 The uninteresting solution

Casting (as we did) the task of distributed generation of signatures as a secure evaluation of a function in a distributed manner, gives immediately a solution for *any* signature scheme $\mathcal{S}$.

Indeed we know various protocols for the secure computation of any function f in a distributed manner (see for example [GMW87, BGW88, CCD88, RB89].)

However the generality of such protocols involves inefficiency and makes them hopelessly impractical for the task at hand. We need *ad hoc* solutions constructed case by case for the specific signature scheme at hand.

4.3 Our results

In this chapter we present robust threshold signature schemes for both the Digital Signature Standard (DSS) [NIST91] and the RSA signature scheme [RSA78]. The importance of providing threshold solutions for signatures schemes used in practice, is that those systems are the ones that will be deployed in the real world and hence they are the ones that require real protection. The solutions for the two schemes are very different in nature and techniques used, reflecting the different number-theoretic basis of the two underlying signature schemes.

For the case of DSS we present a threshold DSS scheme that tolerates $t = n/4$ maliciously faulty players. If we weaken the power of the adversary then we require $t = n/2$ against an eavesdropping adversary and $t = n/3$ against a halting one. The protocols are very efficient in terms of computation required to the players (comparable to the effort required in signing a single message using DSS) and in the number of messages exchanged during the protocol. The results are described in Section 4.5.

The solution for the RSA case only partially meets the requirements defined in Section 4.2. Indeed it is not known (and a very interesting research problem) how to generate distributively RSA keys by means of a `Thresh-Key-Gen` protocol, *without* recurring to general all-purpose solutions (as the ones referred to in Section 4.2.4.) So we simply assume an external party (a dealer) who actually generates the key and shares it among the players. For the signature part we achieve fault-tolerance of $t = n/2$. But maybe the most important feature of the RSA scheme is that it is **non-interactive**, that is given a message m, each player will simply broadcast a single message which will allow all the other ones to compute the signature for m. We do not have such feature for the DSS protocol. The RSA results are described in Section 4.6.

4.3.1 Previous results

Threshold signatures are part of a general approach known as **threshold cryptography** which was introduced by the works of Boyd [Boy86], Desmedt [Des88], and Desmedt and Frankel [DF90]. This approach has received considerable attention in the literature; we refer the reader to [Des94] for a survey of the work in this area. Particular examples of solutions to threshold signatures can be found in [DF92, DDFY94] for the case of RSA signatures, and [CMI93, Har94, Lan95] for ElGamal-type of signatures. Threshold DSS signatures schemes were recently studied by Langford [Lan95]. DSS signatures turn out to be less amenable to sharing techniques than RSA or even other ElGamal-type of signatures, e.g., see [CMI93, Har94]. Langford has overcome some of these difficulties in the case of DSS, exhibiting a solution which requires a group of $n = t^2 - t + 1$ players in order to tolerate up to t players that might refuse to participate in the signature protocol. In any case all the works cited above did not achieve robust solutions, i.e. they were not able to tolerate a malicious adversary, but only a halting one.

Our results are the first robust threshold signature schemes for DSS. Notice also the dramatic improvement in fault-tolerance (we achieve $t = n/4$ compared to the $t \approx \sqrt{n}$ in [Lan95].)

In a recent and independent work, Frankel, Gemmel, and Yung [FGY96], have extended the notion of *result-checking* introduced by Blum [BK89], to the setting of *witness-based cryptographic checking*. Among the main motivations for that work is the generation of a robust fault-tolerant RSA signature scheme. While [FGY96] provides a more general theoretical framework, our techniques, specifically designed for RSA, result in much more efficient and practical solutions. In particular, our basic protocols involve just a small constant number of modular exponentiations while in [FGY96] a very large number, proportional to the security parameter, of such costly exponentiations is required.

4.4 Existing Tools

In this section we recall a few known techniques that we use in our solutions.

In the following we will make several references to Shamir's Secret Sharing protocol `Shamir-SS` which we described in Section 1.2.1. The reader is referred to that section for details.

4.4.1 Feldman's VSS

In Chapter 2 we recalled various VSS protocols [BGW88, Rab94]. They achieved security against an all-powerful adversary. Since in this chapter we are dealing with a computationally bounded adversary we use a different VSS protocol devised by Feldman [Fel87]. This protocol protects the secrecy of the shared secret only in a computational sense but has the advantage of being more efficient.

Like Shamir's scheme, Feldman's VSS generates for each player P_i a share σ_i, such that

$$(\sigma_1, \ldots, \sigma_n) \stackrel{(t,n)}{\longleftrightarrow} \sigma \bmod q$$

where q is a prime. Let p be a prime of the form $p = kq + 1$ and g an element of order q in Z_p^*. If $f(x) = \sum_j a_j x^j$ is the sharing polynomial then the dealer broadcasts the values $\alpha_j = g^{a_j} \bmod p$. This will allow the players to check that the values σ_i really define a secret by checking that

$$g^{\sigma_i} = \prod \alpha_j^{i^j} \tag{4.1}$$

This step will also allow detection of incorrect shares σ_i' at reconstruction time, as those will not satisfy equation (4.1). Notice that the value of the secret is only computationally secure, e.g., the value $g^{a_0} = g^{\sigma} \bmod p$ is leaked. In the following we will refer to this protocol by `Feldman-VSS`.

4.4.2 Pedersen's VSS

This protocol was presented by Pedersen in [Ped92]. In contrast to Feldman's VSS protocol, this protocol provides information theoretic secrecy for the shared secret as the ones in [BGW88, Rab94]. This is required by some of our techniques in order to achieve provable security. However, in order to achieve security, Pedersen's VSS still uses in a crucial way the fact that the adversary is computationally bounded.

In this implementation the private information of player P_i is the value σ_i such that $(\sigma_1, \ldots, \sigma_n) \stackrel{(t,n)}{\longleftrightarrow} \sigma \bmod q$. The dealer then commits to each share using an unconditionally secure commitment scheme based on the hardness of discrete log (that is the secrecy of the committed value is unconditional, but it is possible to open the commitment in a different way if one is able to solve discrete log.) The commitment has homomorphic properties that allow the players to check that the shares define a secret as in Feldman's VSS. If one assumes that players are not able to open the commitment in different ways, then at reconstruction time bad shares are detected. The scheme tolerates $\frac{n-1}{2}$ malicious faults.

The commitment scheme is as follows: let $p = kq + 1$ be a prime, g an element of order q in Z_p^* and h an element generated by g. Assume nobody knows $DL_g(h)$, the discrete log in base g of h. Then a commitment to a value a can be computed as

$$\textsc{cmt}(a, r) = g^a h^r \bmod p$$

Pedersen shows in [Ped92] that for any $a \in Z_q$ and for randomly uniformly chosen $r \in Z_q$ the value $\textsc{cmt}(a, r)$ is uniformly distributed in Z_p, yielding thus no information about a.

So the dealer generates for each player P_i a share σ_i, such that $(\sigma_1, \ldots, \sigma_n) \stackrel{(t,n)}{\longleftrightarrow} \sigma \bmod q$. If $f(x) = \sum_j a_j x^j$ is the sharing polynomial, then the dealer broadcasts the values $\alpha_j = \textsc{cmt}(a_j, r_j)$ where the r_j's are chosen randomly and uniformly over Z_q. The dealer sends to player P_i the values $\sigma_i = f(i) \bmod p$ and $\tau_i = g(i) \bmod p$, where

$g(x) = \sum_j r_j x^j$.

This will allow the players to check that the values σ_i really define a secret by checking that

$$\text{CMT}(\sigma_i, \tau_i) = \prod \alpha_j^{i^j} \bmod p \tag{4.2}$$

At reconstruction time, each player will be required to reveal both σ_i and τ_i. The other players check that the revealed values satisfy equation (4.2.) This step allows detection of incorrect shares σ_i'. Indeed if P_i reveals $\sigma_i' \neq sigma_i$ and τ_i' such that equation (4.2) is satisfied, then it is easy to see that P_i must be able to compute $DL_g(h)$. So under the assumption that the players cannot compute $DL_g(h)$, the scheme tolerates malicious faults.

In the following we will refer to this protocol as `Pedersen-VSS`.

4.4.3 Joint Random Secret Sharing

In a Joint Random Secret Sharing scheme the players *collectively* choose shares corresponding to a (t, n)-secret sharing of a random value. At the end of such a protocol each player P_i has a share σ_i, where $(\sigma_1, \ldots, \sigma_n) \xleftrightarrow{(t,n)} \sigma$, and σ is uniformly distributed over the interpolation field. As with a regular (t, n)-secret sharing scheme the value σ is kept secret from every player, and even from any coalition of t players. Basically this is a Secret Sharing without a dealer.

It is easy to devise such protocols if one has VSS with the homomorphic property (see Section 2.7.1.) That is, if given a sharing of a secret a, $(a_1, \ldots, a_n) \xleftrightarrow{(t,n)} a$ and and a sharing of a secret b, $(b_1, \ldots, b_n) \xleftrightarrow{(t,n)} b$, then we have that

$$(a_1 + b_1, \ldots, a_n + b_n) \xleftrightarrow{(t,n)} a + b$$

(see [Ben87] for a discussion of homomorphic secret sharing schemes.)

Indeed if this is the case, to realize such a protocol, all players act as dealers of a random local secret that they choose. The final share σ_i ($i = 1, \ldots, n$) is computed as the sum of the shares dealt to P_i by each player. Consequently, the joint secret equals the sum of all dealt secrets. It can be shown that as long as all players correctly share their local secrets and (at least) one of these secrets is chosen randomly then the resultant shares interpolate to a random secret σ. In the cases where active corrupted players may deviate from the protocol, one needs to perform the dealing by each player using a verifiable secret sharing protocol. Shares dealt by players that are detected as corrupted are ignored in the final share computation.

The basic properties of this protocol, namely, the kind of public information it generates, and its fault tolerance are inherited from the underlying secret sharing scheme. In the following we will refer to these protocols as `Joint-Shamir-RSS`, `Joint-Feldman-RSS` or `Joint-Pedersen-RSS` depending which of the secret sharing schemes is used.

Joint Zero Secret Sharing. A particularly interesting case is when the secret jointly generated is known to be a zero. Such a protocol is similar to the above joint random secret sharing protocol but instead of local random secrets each player deals a sharing of the value zero. When verifiability is required each player deals its shares using `Feldman-VSS`. The correct dealing of the value zero is verified by checking that the free coefficient a_0 of each dealing polynomials is 0 (i.e., by checking that $\alpha_0 = g^{a_0} = 1$). We will refer to this protocol as `Joint-Zero-SS`. Notice that by adding such "zero-shares" to existent shares of a secret σ, one obtains a randomization of the shares of σ without changing the secret (first noticed in [BGW88] and this is the property we use in Chapter 2 to protect the shares at reconstruction time.) This is the way we will typically use the `Joint-Zero-SS` protocol.

4.5 Robust DSS Threshold Signature Schemes

In this section we present our results relative to the Digital Signature Standard. The section is structured as follows: in 4.5.1 we recall the Digital Signature Standard; in 4.5.2 we describe the key generation protocol which actually turns out to be a simple application of previous work of Feldman [Fel87] and Pedersen [Ped91]; in 4.5.4 we present a simple and extremely efficient protocol for the case of eavesdropping or halting adversary; the former protocol also serves as introduction to the full result presented in 4.5.5.

4.5.1 The Digital Signature Standard (DSS)

The Digital Signature Standard (DSS) [NIST91] is a signature scheme based on the El-Gamal [Gam85] and Schnorr's [Sch91] signature schemes, which was adopted as the US standard digital signature algorithm. In our description of the DSS protocol we follow the notation introduced in [Lan95], which differs from the original presentation of [NIST91] by switching k and k^{-1}. This change will allow a clearer presentation of our threshold DSS signature protocols.

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 a hash of the message to be signed. 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 checking that $r = (g^{ms^{-1}} y^{rs^{-1}} \bmod p) \bmod q$ where s^{-1} is computed modulo q.

Remark: Later in the simulation of our protocols we will need the value $r^* = g^{k^{-1}} \bmod p$, that is the value r *before* the reduction mod q. We notice that such a value is easily computable from a regular DSS signature pair since

$$r^* = g^{ms^{-1}} y^{rs^{-1}} \bmod p$$

This basically implies that the extra reduction mod q has the only purpose to shorten a DSS signature, but serves no security purpose.

4.5.2 DSS Threshold Key-generation

An instance (p, q, g) of DSS can be generated using a public procedure (e.g., as specified in [NIST91]), or using randomness which is jointly provided by the players.

To generate a pair of public and private keys in a distributed setting *without a trusted party*, we use a *joint verifiable secret sharing* protocol, following another protocol by Pedersen [Ped91]. That is the players run an execution of `Joint-Feldman-RSS` (see Section 4.4). The output of such a protocol is a secret sharing $(x_1, \ldots, x_n) \overset{(t,n)}{\longleftrightarrow} x \bmod q$ of a random value x which in addition reveals $y = g^x \bmod p$. The pair (y, x) is taken to be the public/private key pair.

4.5.3 Basic Modules of our Solution

We start by introducing two building blocks which are central to our solution for threshold DSS signature protocols. The first is an elegant and simple procedure for the shared computation of reciprocals, largely based on a protocol by Bar-Ilan and Beaver [BB89]. This procedure is used in our protocols in the following way: the players first produce a joint sharing of a random k, and then compute from these shares a sharing of the reciprocal k^{-1}; the latter in turn are used to compute $r = g^{k^{-1}}$ (it is essential for the security of our application that information on k is not revealed during this process).

PROBLEM 1: COMPUTING RECIPROCALS. Given a secret $k \bmod q$ which is shared among players $P_1, ...P_n$, generate a sharing of the value $k^{-1} \bmod q$, without revealing information on k and k^{-1}. That is, if $(k_1, \ldots, k_n) \overset{(t,n)}{\longleftrightarrow} k \bmod q$ perform a secure computation of the following function

$$\textsc{inv}(k_1, \ldots, k_n) = (u_1, \ldots, u_n)$$

where $(u_1, \ldots, u_n) \xleftrightarrow{(t,n)} k^{-1} \bmod q$.

The following protocol for the above protocol was reinvented independently by us, but Beaver pointed out to us that it was already appeared in [BB89]. Our version is a simplified one for reasons of efficiency, although it loses in fault-tolerance.

Let q be a prime and k a distributed secret such that each player P_i holds a share k_i and $(k_1, \ldots, k_n) \xleftrightarrow{(t,n)} k \bmod q$. The players compute the sharing of the reciprocal by generating a secret random value a, where each player P_i has a share a_i. This is achieved by carrying out a Distributed Secret Sharing Protocol `Joint-RSS` (Section 4.4). They also execute a `Joint-Zero-SS` generating a polynomial of degree $2t$, to be used in the following multiplication. Such sharing generates a share b_i for player P_i such that $(b_1, \ldots, b_n) \xleftrightarrow{(2t,n)} 0 \bmod q$. They now compute the product of k and a, which maps k to a random point in Z_q. To accomplish this each player broadcasts $k_i a_i + b_i$. The polynomial of degree $2t$ is interpolated using Berlekamp-Welch error correcting protocol [BW]. The result of this interpolation is a random value μ, hence reveals no information on k. Each player locally computes μ^{-1} $[= k^{-1}a^{-1} \bmod q]$, and sets his share u_i, of the secret k^{-1}, to $u_i \triangleq a_i\mu^{-1}$. Details in Figure 4-1.

Lemma 4.1 *Assume $n \geq 4t+1$ and to be in the presence of a malicious t-adversary $\mathcal{A}$. Then the* `Reciprocal Protocol` *is a secure computation of the function* INV.

Proof Indeed it holds that $(u_1, \ldots, u_n) \xleftrightarrow{(t,n)} k^{-1} \bmod q$ since the interpolation polynomial has degree $d = 2t$ and there can be at most t wrong points μ_i. Remember from section 2.6.1 that Berlekamp-Welch works if one has at least $2t + d + 1$ points to interpolate the polynomial. Since $n \geq 4t + 1$ by hypothesis, the protocol is correct.

The protocol achieves also privacy. Indeed we describe a simulator for the protocol. Since the adversary is static we assume that Sim receives the inputs k_i of the corrupted players right at the start of the simulation.

The simulation proceeds as follows:

`Reciprocal Protocol`

Input for each player P_i: A share k_i such that $(k_1, \ldots, k_n) \overset{(t,n)}{\longleftrightarrow} k \bmod q$.

1. The players jointly generate a (t, n) sharing of a *random* element $a \in Z_q$ using `Joint-Feldman-RSS` protocol (Section 4.4). We denote the resulting shares by $a_1, a_2, \ldots, a_n$, i.e., $(a_1, \ldots, a_n) \overset{(t,n)}{\longleftrightarrow} a$.
2. The players execute a $(2t, n)$ Joint-Zero-SS protocol (Section 4.4) after which each player P_i holds a share b_i of the "secret" 0. (The implicit interpolation polynomial is of degree $2t$.)
3. The players reconstruct the value $\mu = ka$ by broadcasting the values $\mu_i = k_i a_i + b_i$, and interpolating the corresponding $2t$-degree polynomial using Berlekamp-Welch error-correcting procedure.
4. Each player computes his share u_i of k^{-1} by setting $u_i \triangleq \mu^{-1} a_i \bmod q$.

Figure 4-1: How to compute shares of the reciprocal of a shared secret (from [BB89])

1. Sim runs a `Feldman-VSS` protocol on a random secret for each good player to simulate the execution of `Join-Feldman-RSS`. Sim also receives the shares of the sharings done by the adversary for the bad players. Notice that this implies that Sim knows the values a_i of the bad players.

2. Sim runs a $(2t, n)$ `Feldman-VSS` protocol on a zero secret for each good player to simulate the execution of `Join-Zero-SS`. Sim also receives the shares of the sharings done by the adversary for the bad players. Notice that this implies that Sim knows the values b_i of the bad players.

3. Sim chooses random k_i for the good players such that all the values μ_i interpolate a polynomial of degree $2t$. Sim can do this because he knows the values μ_i for the bad players (if the adversary decides to broadcast a different ν_i in this round of the simulated execution, it does not matter as the error-correcting mechanism will assure that the result of the interpolation is the same)

The simulation is perfectly indistinguishable from a real execution since the round 1 and 2 are exactly distributed as in the real one. The only difference between round 3 in the simulated and the real execution is that the shares interpolate to different secrets but the adversary has too few points to detect that. ■

Notice that if the adversary is an Eavesdropping or Halting one, then the protocol can be simplified by running simple secret sharings (with no verification) in rounds 1 and 2, and performing regular interpolation in round 3. Notice that the fault tolerance goes up to $n \geq 2t+1$ in the case of eavesdropping adversaries and $n \geq 3t+1$ in the case of halting adversaries.

Problem 2: Multiplication of two secrets. Given two secrets u and v, which are both shared among the players, compute the product uv, while maintaining both of the original values secret (aside from the obvious information which is revealed from the result).

Given that u and v are each shared by a polynomial of degree t, each player can locally multiply his shares of u and v, and the result will be a share of uv on a polynomial of degree $2t$. Consequently, the value uv can still be reconstructed from a set of $2t+1$ correct shares. An additional re-randomization procedure (using the `Joint-Zero-SS` protocol of Section 4.4) is required to protect the secrecy of the multiplied secret; this randomization is essential because a polynomial of degree $2t$ which is the product of two polynomials of degree t is not a random polynomial, and would expose information about u and v.

We note that this solution to the problem of secret multiplication is a simplified version of the the protocols for the same problem presented in [BGW88, CCD88]. In those works extra steps were added in order to reduce the degree of the interpolation polynomial back to t. This involved zero-knowledge proofs and use of error-correction again, with the consequence of losing in efficiency. Even the simplified version of Rabin [Rab95] for this task involves non-trivial zero-knowledge proofs for verifiability.

In contrast to those works, in our case secrets are multiplied only once, thus saving most of the complexity of the solutions in the above works which mainly deal with the problem of repetitive multiplication (indeed this is the main reason it is desirable to reduce the degree of the polynomial back to t.)

4.5.4 Eavesdropping and Halting Adversary

In this section we present our basic protocol for generating a DSS signature in a distributed fashion.

- It is a secure DSS threshold signature scheme in the presence of an Eavesdropping Adversary (Section 4.2.2) when the number of players is $n \geq 2t+1$ where t is the number of faults.
- It is a secure DSS threshold signature scheme in the presence of a Halting Adversary (Section 4.2.2) when the number of players is $n \geq 3t+1$ where t is the number of faults.

In other words, this protocol preserves security (i.e., secrecy and unforgeability) in the presence of less than a half eavesdropping faults. On other hand, this protocol is robust in the presence of an adversary that in addition to eavesdropping can halt the operation of up to a third of the players by, for example, crashing servers, or disconnecting them from the communication lines.

Outline. Initially every player P_i has a share x_i of the secret key x, shared through a polynomial $F(\cdot)$ of degree t, i.e. $(x_1, \ldots, x_n) \overset{(t,n)}{\longleftrightarrow} x \bmod q$. First the players generate distributively a random k (through a random t-degree polynomial $G(\cdot)$) by running the Joint Random Secret Sharing protocol Joint-Shamir-RSS (Section 4.4). To compute $r = g^{k^{-1}} \bmod p$ without revealing k, the players use the Reciprocal Protocol (Section 4.5.3) to compute local shares of k^{-1} and then use "interpolation in the exponent" (see below) to compute $g^{k^{-1}}$. For the generation of the signature's value s, we note that $s = k(m + xr) \bmod q$ corresponds to the constant term of the multiplication polynomial $G(\cdot)(m + rF(\cdot))$. Since the players have shares of both $G(\cdot)$ and $m + rF(\cdot)$, they can compute s by performing the Multiplication Protocol (Section 4.5.3). The full description of protocol `DSS-Thresh-Sig-1` is presented in Figure 4-2. It incorporates the multiplication and reciprocal protocols from Section 4.5.3.

Notation. In the description of `DSS-Thresh-Sig-1` we use the following notation for two share interpolation operations:

- $v = \mathsf{Interpolate}(v_1, \ldots, v_n)$. If $\{v_1, \ldots, v_n\}$ $(n \geq 2t+1)$ is a set of values, such that at most t are *null* and all the remaining ones lie on some t-degree polynomial $F(\cdot)$, then $v \stackrel{\triangle}{=} F(0)$. The polynomial can be computed by standard polynomial interpolation (see Section 1.2.1.)

- $\beta = \mathsf{Exp\text{-}Interpolate}(w_1, \ldots, w_n)$. If $\{w_1, \ldots, w_n\}$ $(n \geq 2t+1)$ is a set of values such that at most t are *null* and the remaining ones are of the form $g^{a_i} \bmod p$ where the a_i's lie on some t-degree polynomial $G(\cdot)$, then $\beta \stackrel{\triangle}{=} g^{G(0)}$. This can be computed by $\beta = \prod_{i \in V} w_i^{\lambda_{0,i,V}} = \prod_{i \in V} (g^{G(i)})^{\lambda_{0,i,V}}$, where V is a $(t+1)$-subset of the non-null w_i's and $\lambda_{0,i,V}$'s are the corresponding Lagrange interpolation coefficients (see Section 1.2.1.)

We prove the security of our protocol in the case of Halting Adversary $(n \geq 3t+1.)$ The proof for the case of an Eavesdropping Adversary can be easily inferred from this one.

Theorem 4.1 *Assume $n \geq 3t+1$ and to be in the presence of an eavesdropping t-adversary $\mathcal{A}$. Then* `DSS-Thresh-Sig-1` *is a secure threshold DSS signature protocol.*

Proof The correctness of the protocol in spite of the possible crashing of up to t servers is immediate, as we always have enough points to interpolate a $2t$ degree polynomial.

In order to prove the unforgeability of our protocol, we need to show a simulator Sim_1. Here is a simulated execution round by round. The input of Sim_1 are the shares x_i for the bad players, the message m being signed. At round 3 Sim_1 also gets from the oracle the value of the signature (r, s) since it is fixed by the traffic at that point.

1. Sim_1 runs a `Shamir-SS` protocol on a random secret for each good player to simulate the execution of `Join-Shamir-RSS`. Sim_1 also receives the shares of

`DSS-Thresh-Sig-1`

Input for each player P_i: A share x_i of the secret key x.

1. **Generate k**
   The players generate a secret random value k, uniformly distributed in Z_q, with a polynomial of degree t, using `Joint-Shamir-RSS` (Section 4.4), which creates shares $(k_1, \ldots, k_n) \overset{(t,n)}{\longleftrightarrow} k \bmod q$.

2. **Generate random polynomials with constant term 0**
   Execute two instances of `Joint-Zero-SS` (using `Shamir-SS` as the underlying secret sharing protocol) with polynomials of degree $2t$. Denote the shares created in these protocols as $(b_1, \ldots, b_n) \overset{(2t,n)}{\longleftrightarrow} 0 \bmod q$ and $(c_1, \ldots, c_n) \overset{(2t,n)}{\longleftrightarrow} 0 \bmod q$

3. **Compute $r = g^{k^{-1}} \bmod p \bmod q$**

   (a) The players generate a random value a, uniformly distributed in Z_q^*, with a polynomial of degree t, using `Joint-Shamir-RSS`, which creates shares $(a_1, \ldots, a_n) \overset{(t,n)}{\longleftrightarrow} a \bmod q$.

   (b) Player P_i broadcasts $v_i = k_i a_i + b_i \bmod q$ and $w_i = g^{a_i} \bmod p$. If P_i does not participate his values are set to *null*. Notice that $(v_1, \ldots, v_n) \overset{(2t,n)}{\longleftrightarrow} ka \bmod q$.

   (c) Player P_i locally computes

   - $\mu \triangleq \mathsf{Interpolate}(v_1, \ldots, v_n) \bmod q \qquad [= ka \bmod q]$
   - $\beta \triangleq \mathsf{Exp\text{-}Interpolate}(w_1, \ldots, w_n) \bmod p \qquad [= g^a \bmod p]$
   - $r \triangleq \beta^{\mu^{-1}} \bmod p \bmod q \qquad [= (g^a)^{\mu^{-1}} = g^{k^{-1}} \bmod p \bmod q]$

4. **Generate $s = k(m + xr) \bmod q$**

   (a) Player P_i broadcasts $s_i = k_i(m + x_i r) + c_i \bmod q$. If P_i does not participate, his value s_i is set to *null*. Notice that $(s_1, \ldots, s_n) \overset{(2t,n)}{\longleftrightarrow} k(m + xr) \bmod q$.

   (b) Each player computes $s \triangleq \mathsf{Interpolate}(s_1, \ldots, s_n) \bmod q$.

5. **Output** (r, s) as the signature for m.

Figure 4-2: DSS Threshold Signature Protocol – Halting ($n \geq 3t + 1$) or Eavesdropping ($n \geq 2t + 1$) Adversary the sharings done by the adversary for the bad players. Notice that this implies that Sim_1 knows the values k_i of the bad players.

2. Sim_1 runs two copies of a $(2t, n)$ `Shamir-SS` protocol on a zero secret for each good player to simulate the execution of `Join-Zero-SS`. Sim_1 also receives the shares of the sharings done by the adversary for the bad players. Notice that this implies that Sim_1 knows the values b_i and c_i of the bad players.

3. At this point Sim_1 knows (r, s). From this he can compute r^*. This round is composed by three subrounds.

   (a) Sim_1 runs a `Shamir-SS` protocol on a random secret for each good player to simulate the execution of `Join-Shamir-RSS`. Sim_1 also receives the shares of the sharings done by the adversary for the bad players. Notice that this implies that Sim_1 knows the values a_i of the bad players.

   Sim_1 chooses a random number $\mu \in Z_q$. Given the values $v_i = a_i k_i + b_i$ of the bad players and the value μ, Sim_1 computes values v_i for the bad players such that $(v_1, \ldots, v_n) \stackrel{(2t,n)}{\longleftrightarrow} \mu$.

   Denote now $g^a = (r^*)^\mu$. (We stress that g^a is only a notation for $(r^*)^\mu$; the value a is never explicitly computed in the simulation). Compute $w_i = g^{a_i}$ for the bad players and from such values and g^a generate w_j for the good players, such that $\textsf{Exp-Interpolate}(w_1, \ldots, w_n) = g^a$. Note that $w_j = g^{\lambda_{j,0,V} a + \sum_i \lambda_{j,i,V} a_i} = (g^a)^{\lambda_{j,0,V}} \prod_i w_i^{\lambda_{j,i,V}}$ for the appropriate Lagrange coefficients λ's.

   (b) Sim_1 broadcasts v_i and w_i for the good players.

   (c) Sim_1 just sits idle.

4. Sim_1 has to broadcasts in the name of the good players values s_i such that $(s_1, \ldots, s_n) \stackrel{(2t,n)}{\longleftrightarrow} s \bmod q$. Notice that Sim_1 knows $s_i = k_i(m + x_i r) + c_i$ for the bad players. He chooses the good players' s_i as randomly chosen values under the constraint that $(s_1, \ldots, s_n) \stackrel{(2t,n)}{\longleftrightarrow} s \bmod q$.

The simulation is statistically indistinguishable from a true one. Clearly rounds 1-2 are exactly the same as in a real execution. In rounds 3-4 Sim_1 is broadcasting different values than the one originally shared by the good players in the simulated rounds 1-2. He does so in order to match the correct signature (r, s). But since the sharings in round 1-2 are unconditionally secure, the adversary does not have enough points to detect the difference. ■

4.5.5 Malicious Adversaries

In this section we present a more robust version of protocol `DSS-Thresh-Sig-1` which remains secure even in the presence of a fully *malicious adversary*. The protocol, `DSS-Thresh-Sig-2` , relies on no assumptions beyond the unforgeability of regular DSS signatures, and can tolerate $\frac{n-1}{4}$malicious faults.

OUTLINE. The protocol is very similar to `DSS-Thresh-Sig-1` . The only difference is that here we need verifiable sharing of secrets since we assume a Malicious Adversary. The random value k is jointly generated by the players using Pedersen's *unconditionally* secure VSS (Section 4.4). This guarantees that absolutely no information is leaked on the values k or k^{-1}. Then the players compute r as in `DSS-Thresh-Sig-1` , with the only difference that now the random value a is jointly generated using Feldman's VSS protocol. As before s is computed from the appropriate shares. Whenever we reconstruct a secret, in order to detect bad shares contributed by malicious players we perform error-correcting using the Berlekamp and Welch decoder [BW]. As before randomization of polynomials (through the joint zero secret sharing protocols) is added in various places in order to hide possible partial information. The full protocol is exhibited in Figure 4-3

Notation. In the protocol, we use the following notation:

$$v = \mathsf{EC\text{-}Interpolate}(v_1, \ldots, v_n)$$

If $\{v_1, \ldots, v_n\}$ $(n = 4t + 1)$ is a set of values, such that at least $3t$ of the values lie on some $2t$-degree polynomial $F(\cdot)$, then $v \triangleq F(0)$. The polynomial can be computed by using the Berlekamp-Welch decoder [BW] (see section 2.6.1.)

Theorem 4.2 *Assume $n \geq 4t+1$ and to be in the presence of a malicious t-adversary $\mathcal{A}$. Then protocol* `DSS-Thresh-Sig-2` *is a secure threshold signature protocol for DSS.*

Proof The correctness of the protocol is due to the error correcting capabilities of polynomial interpolation. Since we are interpolating a polynomial of degree $d = 2t$ and we have t possible errors, using the Berlekamp–Welch bound (see equation 2.2 in Section 2.6.1) we get that the number of points needed in order to correctly interpolate is $d + 2t + 1 = 4t + 1$. Hence, we set $n \geq 4t + 1$.

In order to prove the unforgeability of our protocol, we need to show a simulator Sim_2. Here is a simulated execution round by round. The input of Sim_2 are the shares x_i for the bad players, the message m being signed. At round 4 Sim_2 also gets from the oracle the value of the signature s since it is fixed by the traffic.

1. Sim_2 runs a `Pedersen-VSS` protocol on a random secret for each good player to simulate the execution of `Join-Pedersen-RSS`. Sim_2 also receives the shares of the sharings done by the adversary for the bad players. Notice that this implies that Sim_2 knows the values k_i of the bad players.

2. Sim_2 runs two copies of a $(2t, n)$ `Feldman-VSS` protocol on a zero secret for each good player to simulate the execution of `Join-Zero-SS`. Sim_2 also receives the shares of the sharings done by the adversary for the bad players. Notice that this implies that Sim_2 knows the values b_i and c_i of the bad players.

3. At this point Sim_2 knows (r, s). From this he can compute r^*. This round is composed by three subrounds.

   (a) Sim_2 chooses a random number $\mu \in Z_q$. Sim_2 simulates the execution of Joint-Feldman-VSS in such a way that $g^a = (r^*)^\mu$. (We stress that g^a is only a notation for $(r^*)^\mu$; the value a is never explicitly computed in the simulation). Sim_2 can do this since it is running the execution of $n - t$

DSS-Thresh-Sig-2

Input for all players P_i: a share x_i of the secret key x.

1. **Generate k**
The players generate a secret value k, uniformly distributed in Z_q, by running Joint-Pedersen-RSS with a polynomial of degree t. Notice that this generates $(k_1, \ldots, k_n) \overset{(t,n)}{\longleftrightarrow} k \bmod q$.

2. **Generate random polynomials with constant term 0**
Execute two instances of Joint-Zero-SS with polynomials of degree $2t$ and Feldman-VSS as underlying scheme. Denote the shares created in these protocols as $\{b_i\}_{i\in\{1\ldots n\}}$ and $\{c_i\}_{i\in\{1\ldots n\}}$. Notice that g^{b_i} and g^{c_i} is public information.

3. **Generate $r = g^{k^{-1}} \bmod p \bmod q$**
   (a) Generate a random value a, uniformly distributed in Z_q^*, with a polynomial of degree t, using Joint-Feldman-RSS. This creates shares $(a_1, \ldots, a_n) \overset{(t,n)}{\longleftrightarrow} a \bmod q$. Notice that the public information is g^a and g^{a_i}.
   (b) Player P_i broadcasts $v_i = k_i a_i + b_i \bmod q$. If P_i doesn't broadcast a value set v_i to *null*.
   (c) Player P_i computes locally
      - $\mu \triangleq \mathsf{EC\text{-}Interpolate}(v_1, \ldots, v_n) \bmod q \qquad [= ka \bmod q]$
      - $\mu^{-1} \bmod q \qquad [= k^{-1}a^{-1} \bmod q]$
      - $r \triangleq (g^a)^{\mu^{-1}} \bmod p \bmod q \qquad [= g^{k^{-1}} \bmod p \bmod q]$

4. **Generate $s = k(m + xr) \bmod q$**
Player P_i broadcasts $s_i = k_i(m + x_i r) + c_i \bmod q$. Set $s \triangleq \mathsf{EC\text{-}Interpolate}(s_1, \ldots, s_n)$.

5. **Output** the pair (r, s) as the signature for m

Figure 4-3: DSS Threshold Signature Protocol against a Malicious Adversary, $n \geq 4t + 1$

good players. Notice that Sim_2 also knows the values a_i of the bad players at the end of this simulated round. Given the values $v_i = a_i k_i + b_i$ of the bad players and the value μ, Sim_1 computes values v_i for the bad players such that $(v_1, \ldots, v_n) \overset{(2t,n)}{\longleftrightarrow} \mu$.

(b) Sim_2 broadcasts v_i for the good players.

(c) Sim_2 just sits idle.

4. Sim_2 has to broadcasts in the name of the good players values s_i such that $(s_1, \ldots, s_n) \overset{(2t,n)}{\longleftrightarrow} s \bmod q$. Notice that Sim_2 knows $s_i = k_i(m + x_i r) + c_i$ for the bad players. He chooses the good players' s_i as randomly chosen values under the constraint that $(s_1, \ldots, s_n) \overset{(2t,n)}{\longleftrightarrow} s \bmod q$.

The simulation is statistically indistinguishable from a true one. Clearly rounds 1-2 are exactly the same as in a real execution. In rounds 3-4 Sim_1 is broadcasting different values than the one originally shared by the good players in the simulated rounds 1-2. He does so in order to match the correct signature (r, s). But since the sharings in round 1-2 are unconditionally secure, the adversary does not have enough points to detect the difference. ■

4.5.6 Improving the fault-tolerance

The bound of $\frac{n-1}{4}$ on the fault-tolerance of the previous protocol is basically caused by efficiency reasons. Indeed if one is willing to pay the price for a complex degree-reduction step (as in the general secure multiparty computation protocols of [BGW88]) then one could bring the fault-tolerance to $\frac{n-1}{2}$.

Another way to improve on the fault-tolerance is to avoid using error-correcting codes, but to use some other way to detect and discard incorrect shares. This is indeed possible and it brings the fault tolerance to $\frac{n-1}{3}$ as one still needs $2t$ good shares if degree reduction of the sharing polynomials is not performed. The gain in fault-tolerance however comes at the expenses of an increased amount of computation (e.g. modular exponentiations) required to the players in order to compute a single signature. In this section we simply sketch how to do this.

The previous protocol used the Feldman and Pedersen VSS protocols [Fel87, Ped92] only to assure that secrets where shared correctly. When it came to authenticate the shares broadcasted by players in order to reconstruct a value, however we relied on error-correcting codes. This allowed us to reconstruct the values without further modular exponentiations, but forced us to set the fault-tolerance to $\frac{n-1}{4}$. The protocol sketched in this section will make more extensive use of the properties of Feldman's VSS protocol for the authentication of shares.

In a regular execution of Feldman-VSS the share s_i broadcasted by player P_i can be checked against the publicly known value g^{s_i} for authenticity. Notice that this easily generalizes to the reconstruction of linear combination of secrets. For example, if we want to reconstruct $a + b$, then player P_i broadcasts $a_i + b_i$ which can be checked against $g^{a_i+b_i} = g^{a_i}g^{b_i}$. Things however get more complicated if we want to reconstruct ab because we cannot sieve out bad shares as before, since we do not know how to compute $g^{a_ib_i}$ from g^{a_i} and g^{b_i}.

This is exactly the situation in our previous protocol. The values reconstructed are the product of other shared values. On top of that, one of those shared values (k) has been shared with Uncond-Secure-VSS and not Feldman-VSS. However there is also an advantage: at least one of the values muiltiplied is *random* and has been "recently" shared using a Joint-Feldman-VSS protocol.

This observation led to a clever trick employed in [CMI93]. They show that in the situation described above it is possible to create "authentication pieces" for the resulting shares of the product. In their case both secrets are shared using Feldman-VSS. We show that the trick can be adapted to our case in which one of the values is shared with Uncond-Secure-VSS.

We will now proceed to sketch the method in a general fashion. Let $(a_1, \ldots, a_n) \stackrel{(t,n)}{\longleftrightarrow} a$ be the the sharing obtained from a run of Feldman-VSS. Let $A(x) = A_0 + A_1x + \ldots + A_tx^t$ be the t-degree polynomial used to share the value $a = A_0$. Player P_i holds share $a_i = A(i) \bmod p$ and the values $\alpha_j = g^{A_j} \bmod p$ are publicly know. We also assume that we have performed a Joint-Zero-VSS with a $2t$-degree polynomial $B(x) = B_1x + \ldots + B_{2t}x^{2t}$. Player P_i holds value $b_i = B(i) \bmod p$ and the values $\beta_j = g^{B_j} \bmod p$ are publicly known.

Now assume that the players together generate the sharing of a random value k using Joint-Uncond-Secure-RSS. This means that each player P_i runs one instance of Uncond-Secure-VSS using a random value $k^{(i)}$ as the secret and a random t-degree polynomial $K^{(i)}(x) = K_0^{(i)} + K_1^{(i)}x + \ldots + K_t^{(i)}x^t$. Let $K(x) = \sum_i K^{(i)}(x)$.

Consider now the randomized sharing polynomial of the product secret ar. This the polynomial

$$C(x) = A(x)K(x) + B(x) = B(x) + \sum_i A(x)K^{(i)}(x) = B(x) + \sum_i C^{(i)}(x) \qquad (4.3)$$

if one lets $C^{(i)}(x) = A(x)K^{(i)}(x)$. Let

$$C(x) = C_0 + C_1x + \ldots + C_{2t}x^{2t}$$

and

$$C^{(i)}(x) = C_0^{(i)} + C_1^{(i)}x + \ldots + C_{2t}^{(i)}x^{2t}$$

If we could make the values $\gamma_j = g^{C_j} \bmod p$ public, then we could sieve out bad shares using Feldman's procedure. Using Equation (4.3) we see that such γ_j's values would be known if each player P_i broadcasts the values

$$g_0^{C^{(i)}}, g_1^{C^{(i)}}, \ldots, g_{2t}^{C^{(i)}} \bmod p$$

which he can easily compute from the values α_j and $C_k^{(i)}$. The broadcasted values can be checked by the other players using the usual Feldman's verification procedure (i.e., check that the point on the polynomial they own, does indeed interpolate in the exponent to the values broadcasted by P_i.)

This way when the product ka is reconstructed the t faulty shares can be immediately sieved out and the number of players can be reduced to $3t + 1$ (since we still need at least $2t + 1$ players to reconstruct a polynomial of degree $2t$.)

The application to the above method to our construction of a DSS signature generation protocol is almost immediate. Look at our protocol `DSS-Thresh-Sig-2` . The above method needs to be used when we compute the products ka (round 3c) and kx (round 4) instead of the error-correcting mechanism.

The drawback is that while error-correction is fast, this method requires several extra modular exponentiations.

4.5.7 Efficiency Considerations

As in the case of the generation of regular DSS signatures the most expensive part of our protocols is the computation of r, as it includes all the modular exponentiations and the interactive exchange of messages between players. However (as in the case of regular DSS signatures) such computation can be performed off-line. In this case the signature generation becomes extremely efficient and non-interactive. The protocols are also efficient since they do not require computationally expensive zero-knowledge proofs.

4.6 Robust RSA Threshold Signature Schemes

In this section we present our results relative to the RSA Signature Scheme. The section is structured as follows: in 4.6.1 we recall how RSA works; we outline the basics of our approach in 4.6.2; in 4.6.3 we describe the key generation protocol, as we said in the introduction, we will assume a dealer who generates and shares the key, in contrast to the DSS case in which no such dealer was needed; in 4.7 we present a simple, extremely efficient non-interactive protocol for signature generation; in Sections 4.8 we present a somewhat less efficient interactive signature protocol, however this protocol has some other properties that can make it preferable to the non-interactive one.

4.6.1 The RSA signature scheme

The RSA signature scheme was introduced in [RSA78] and has since become one of the most popular signature schemes in the cryptographic community.

KEY GENERATION. An RSA key is composed of a public key (N, e), and a secret key d, where:

1. N is the product of two large primes $N = p, q$ (nowadays primes are chosen around 500 bits long); let $\phi(N) = (p-1)(q-1)$ the size of the multiplicative group modulo N.

2. e is an integer in $Z^*_{\phi(N)}$.

3. d is the multiplicative inverse of e modulo $\phi(N)$.

SIGNATURE ALGORITHM. Let m be a hash of the message to be signed. The signer computes $S = m^d \bmod N$ as the signature.

VERIFICATION ALGORITHM. A signature S of a message m can be publicly verified by checking that $S^e = m \bmod N$.

4.6.2 Outline

Threshold RSA signatures against eavesdropping or halting adversaries were known in the literature before [DF92, DDFY94]. The basic construction underlying the existing threshold RSA schemes can be roughly described as follows.

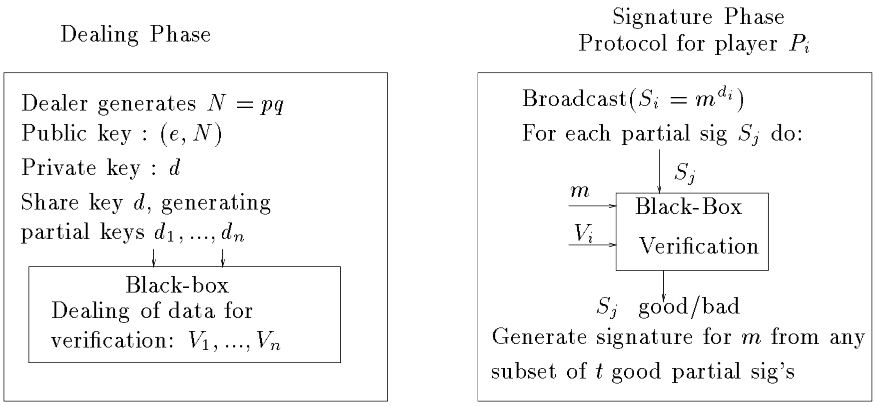
Key Generation Protocol a dealer shares the secret key d, among the n players, such that each player P_i has a "share" d_i of d.

Signature Protocol when a message m is given, any subset of $t+1$ partial signatures $S_i = m^{d_i} \bmod N$ suffice to generate the signature for m.

The best construction is due to De Santis, Desmedt, Frankel and Yung [DDFY94] and it is described in Section 4.6.3

Similarly to non-veriable secret sharing protocols, each of the existing schemes *requires* that all partial signatures, S_i be correct, hence these schemes cannot tolerate faults. If we are to confront failures, we must be able to detect which of the partial signatures are improper. Once the incorrect partial signatures are sieved out, the computation can be carried out in the same manner as in the case where there are no faults.

In this paper we concentrate on providing solutions for the problem of detecting an improper partial signatures. To achieve this goal, additional data $V_1, \ldots, V_n$ is generated in the Dealing Phase, where V_i is known to P_i. This data will be used in the Signature Phase is order to authenticate partial signatures. The following is a schema of a fault-tolerant, threshold RSA signature scheme:



The protocols we present are the "black-boxes" shown in the schema. Note, that the activation of the verification black-box can be avoided under proper operation of the system: each player can try to generate a signature from any subset of $t+1$ broadcasted partial signatures, and then check it using the known public exponent e. If the result is a proper signature, there is no need for the verification of the partial signatures.

In the [DDFY94] solution the shares $d_1, \ldots, d_n$ of the secret key d can be viewed themselves as RSA secret keys. Thus the partial signatures S_i can be viewed as RSA signatures themselves. Yet, for each of these keys the *public exponent*, $e_i = d_i^{-1} \bmod \phi(N)$, is unknown even to the player P_i holding d_i. Furthermore, the public exponent *cannot* be known, as it would expose $\phi(N)$. Hence, we are faced with the problem of checking a partial signature for which we do not have a public exponent. Under the circumstances where the public exponent is not known, there must be some other information that allows the verification of such partial signatures and yet does not reveal any other information, especially information about $\phi(N)$. This is the mathematical challenge in coming up with the data $V_1, ..., V_n$, which is generated in the Dealing Phase.

Different applications which involve fault-tolerant threshold RSA warrant various properties from the RSA protocol. Some applications may require minimal communication between the players during the Signature Phase. Other applications might need that the verification will be "publicly verifiable", namely, that any person, even outside the group of n players, can verify every partial signature.

Thus we have devised two different protocols for verification of partial signatures. The choice of which protocol to use will be motivated by the needs of a specific application. The first protocol has a non-interactive Signature Phase, and requires each player to hold local secret verification data. The second protocol requires interaction in order to generate the signature, yet all the verification data is public. In the Dealing Phase of the second protocol a publicly known sample message and its partial signatures are generated. In the Signature Phase anyone can verify any player's partial signature by interacting with him, and using the sample signature as the basis for the proof. Both solutions are efficient computation-wise (a verification involves only a few RSA exponentiations). Surprisingly, our non-interactive verification is even more efficient in computation than the interactive one. Communication-wise the non-interactive solution is clearly optimal, while the interactive solution requires the exchange of four messages. Both protocols do not leak information that can be used by the players (even malicious ones) to forge other signatures.

THE DEALER. A final comment on the level of trust needed from the dealer, i.e. the entity that shares the secret key and the verification data $V_1, ..., V_N$ among the players. Unfortunately it is not known how to efficiently generate in a distributed manner an RSA key in shared form as we do for El-Gamal like signature schemes (for example the DSS scheme we showed in the previous section.)

Hence, at this stage of knowledge, we need to assume that a single dealer generates the public/private RSA keys, shares it among the players, and also generates the verification data. How much should we trust this dealer?

First of all we are trusting him with the knowledge of the key. The dealer will always be able to go on and sign messages on his own. In practice one can resolve this issue, by implementing the dealer with a trusted device which self-destroys (or simply destroys the content of its memory) once the key-generation and key-sharing tasks are completed. In this scenario one can also trust the dealer with a proper sharing of the key and a proper dealing of the verification data. This is the scenario for the non-interactive solution.

However there are situations in which the dealer might not be trusted with the sharing of the key and the dealing of the verification data. For example consider one-or-threshold signature schemes: assume that a company has one president and n vice-presidents (VPs). Documents signed by the company are valid if signed by the president or by at least t of the VPs. This problem could be solved by having the president choose a key for the company and then share the secret key among the VPs. There is no reason to trust the president to share correctly her signature power among the VPs.

Another important application is key escrow. Although we will not discuss this particular aspect in this thesis, one nice feature of the RSA cryptosystem is that the decryption algorithm is identical to the signature algorithm. This means that sharing the secret key yields not only a threshold signature scheme, but also a threshold key escrow scheme. In this case the shareholders are the law enforcement centers who receive shares of the user's private key. Of course there is no reason to trust the user to actually share the correct decryption key. Again checking on the dealer could be done by running generic all-purpose Verifiable Secret Sharing protocols like the ones in the first chapters of this thesis. However the efficiency cost is too high. Our interactive solution allows us to check the behavior of the dealer quite efficiently. Although in Section 4.8 we present the solution under the assumption that the dealer is honest, in Section 4.8.3 we point out how to add some simple verification steps to check on the dealer's work.

4.6.3 Threshold Sharing of RSA functions

In this section we briefly describe the scheme in [DDFY94] which results in a threshold RSA signature scheme against an eavesdropping or halting adversary. This in turn is the basis of our fully fault-tolerant solutions.

So we want to compute an RSA signature $S = m^d \bmod N$ where d is the secret key. Suppose that we share d among n players using Shamir's (t,n)-threshold scheme [Sha79]. Let d_i be the share of player P_i. We know that d can be written as a linear combination of any set V of $t+1$ shares d_i (see Section 1.2.1):

$$d = \sum_j \lambda_{0,j,V} d_j \bmod \phi(N)$$

where the $\lambda_{0,j,V}$'s are the appropriate Lagrange interpolation coefficients.

So it would be enough for each player P_i to reveal the partial signature $S_i = m^{d_i} \bmod N$ in order to be able to compute

$$S = \prod_j s_j^{\lambda_{0,j,V}} = m^{\sum_j \lambda_{0,j,V} d_j} = m^d \bmod N$$

only from $t+1$ of the shares. However the problem is that the sharing of the key must be done in the ring $Z_{\phi(N)}$ and we need a field in order to compute the Lagrange coefficients.

In [DDFY94] they solve this problem by extending Shamir's scheme to work over any abelian group $(G,+)$. For our purposes let G be $Z_{\phi(N)}$. Let $\pi > n$ be a prime and let u be a root of the cyclotomic polynomial $p(x) = \sum_{j=0}^{q-1} x^j$. Consider $G^{\pi-1}$, this is a module over the the extension of the integers $Z[u] \equiv Z[x]/(p(x))$. Addition in $G^{\pi-1}$ is defined as

$$[g_1, \ldots, g_{\pi-1}] + [g_1', \ldots, g_{\pi-1}'] = [g_1 + g_1', \ldots, g_{\pi-1} + g_{\pi-1}']$$

Scalar multiplication of $Z[u]$ over $G^{\pi-1}$ is defined as

$$b[g_1, \ldots, g_{\pi-1}] = [bg_1, \ldots, bg_{\pi-1}] \text{ for } b \in Z$$

$$u[g_1, \ldots, g_{\pi-1}] = [0, g_1, \ldots, g_{\pi-2}] + [-g_{\pi-1}, \ldots, -g_{\pi-1}]$$

Finally if one defines

$$x_i = \sum_{j=0}^{i-1} u^j$$

it is easy to see that all x_i's and $(x_i - x_j)$'s have (multiplicative) inverses in $Z[u]$

The extended Shamir scheme is then as follows: the dealer chooses a random polynomial R over the module $G^{\pi-1}$ of degree t such that $R(0) = [d, 0, \ldots, 0]$. Give $d_i = R(x_i)$ to trustee i. Because of the properties of the x_i's is then possible to perform polynomial interpolation from at least $t+1$ shares to reconstruct $[d, 0, \ldots, 0] = \sum_j \lambda_{0,j,V} d_j$.

However in our case we don't want to recover d but only $S = F_d(m) = m^d \bmod N$. In order to do that we consider the message space $H = Z_N^*$. We can extend H to a module $H^{\pi-1}$ over $Z[u]$. The module operations are multiplication:

$$[s_1, \ldots, s_{\pi-1}] * [s'_1, \ldots, s'_{\pi-1}] = [s_1 * s'_1, \ldots, s_{\pi-1} * s'_{\pi-1}]$$

and scalar exponentiation:

$$[s_1, \ldots, s_{\pi-1}]^b = [s_1^b, \ldots, s_{\pi-1}^b] \text{ for } b \in Z$$

$$[s_1, \ldots, s_{\pi-1}]^u = [1, s_1, \ldots, s_{\pi-2}] * [s_{\pi-1}^{-1}, \ldots, s_{\pi-1}^{-1}]$$

Now it is clear how to "transport" the threshold scheme from $G^{\pi-1}$ to $H^{\pi-1}$. Partial signatures S_i from a share $d_i = [d_{i,1}, \ldots, d_{i,(\pi-1)}] \in G^{\pi-1}$ are computed simply as

$$S_i = [m^{d_{i,1}}, \ldots, m^{d_{i,(\pi-1)}}]$$

So given a set V of $t+1$ partial signatures we can compute

$$S = \prod_j S_j^{\lambda_{0,j,V}} \tag{4.4}$$

Clearly, the correctness of the [DDFY94] result relies heavily on the correctness of each partial signature.

Observation 1: The partial signature of player P_i in this scheme is just a vector of $\pi - 1$ "regular" RSA signatures. Hence, it will suffice to check each of these signatures on its own. This will allow us in the following to perform operations simply modulo N and not in the algebraic structure described above. Thus, the extension of our protocols to the [DDFY94] construction is straightforward.

Observation 2: We will assume that a partial signatures is good, if it is correct up to a sign. That is, if P_i should broadcast $S_i = m^{d_i} \bmod N$, we will allow him to broadcast either S_i or $-S_i \bmod N$. This might result in the final signature S being "correct up to a sign" i.e., $S = \pm m^d \bmod N$ (easy to see from Equation (4.4)), but this can be easily checked, by testing both signatures using the public exponent e.

4.7 The RSA Non-Interactive Solution

Our non-interactive solution, is enabled by an extension of the Information Checking Protocol (ICP) to a general integer domain. The original ICP [Rab94, RB89] works only over prime fields.

A very rough description of the verification method is the following: assume two players P and V. The prover P holds values d_P in $Z_{\phi(N)}$ (the secret key) and y. The verifier V holds b, c, such that $y = bd_P + c$. Notice that the previous equation must hold over the integers and not modulo $\phi(n)$. Indeed if the equation were modulo $\phi(n)$ it would be enough for P and V to combine their information and compute a multiple of $\phi(N)$ which allows an efficient factorization of N. Given a message m, the prover generates the partial signature $S_P = m^{d_P} \bmod N$, and $Y = m^y \bmod N$. He gives these values to V, who verifies the partial signature by checking whether $(S_P)^b m^c \bmod N = Y$.

Section 4.7.1 exhibits the extensions of ICP. In Section 4.7.2 we describe the dealing of verification information. The verification of partial signatures appears in Section 4.7.3.

4.7.1 Extensions of Information Checking

The following protocol is carried out by three players: a dealer D, who is non-faulty, and two additional players: prover P and verifier V, who can be either faulty or not. In Figure 4-4 we present the ICP-Generation protocol over the integers, carried out by the dealer.

`ICP-Gen-Integers`

Input: RSA composite N, secret value $d_P \in Z_{\phi(N)}$ known to D,
security parameters $0 \leq \delta_1, \delta_2 \leq 1$.

1. Choose $b \in [0 \ldots N^{\delta_1}]$ and $c \in [0 \ldots N^{1+\delta_1+\delta_2}]$ with uniform distribution.
2. Compute $y = c + bd_P$ over the integers.
3. Hand d_P and y to the prover P.
4. Hand b and c to the verifier V.

Figure 4-4: The ICP Generation Protocol

The variables y, d_P and b, c can be later used by players P and V in the following way: when the prover P wants to prove to the verifier V that he holds the value d_P which he received from D, he sends d_P and y to V. Upon receiving values $\hat{d_P}, \hat{y}$ from P, the verifier concludes that $\hat{d_P} = d_P$ only if $\hat{y} = b\hat{d_P} + c$.
For the following we denote $\mathcal{Y} \stackrel{\text{def}}{=} [N^{\delta_1+1}, \ldots, N^{1+\delta_1+\delta_2}]$.

Lemma 4.2 *Given values d_P, and $y \in \mathcal{Y}$, for every possible value of b there is exactly one possible value for $c \in [0 \ldots N^{1+\delta_1+\delta_2}]$ such that $y = bd_P + c$.*

Proof Since the computation is over the integers, there is exactly one value of c for each b, d_P and y. Furthermore, if $b \in [0 \ldots N^{\delta_1}]$ and $y \in [N^{\delta_1+1}, \ldots, N^{1+\delta_1+\delta_2}]$ then value $c = y - bd_P$ is contained in $[0 \ldots N^{1+\delta_1+\delta_2}]$ because

$$1 \leq N^{1+\delta_1} - N^{\delta_1}\phi(N) = y_{min} - b_{max}d_{max} \leq c \leq y_{max} - b_{min}d_{min} = N^{1+\delta_1+\delta_2} - 1$$

which proves the lemma. ∎

Lemma 4.3 $Pr(\, y \notin \mathcal{Y}\,) \leq \frac{1}{N^{\delta_2}}$.

Proof The number of options to choose different pairs of b and c is $N^{1+\delta_1+\delta_2}N^{\delta_1}$. The range $\mathcal{Y}$ is of size $N^{1+\delta_1+\delta_2} - N^{1+\delta_1}$. From Lemma 4.2 it follows that each value y in this range can be generated by N^{δ_1} pairs (b, c). Consequently, the probability of y falling outside of this range is $1 - \frac{N^{\delta_1}(N^{1+\delta_1+\delta_2} - N^{1+\delta_1})}{N^{\delta_1}N^{1+\delta_1+\delta_2}} = \frac{1}{N^{\delta_2}}$. ■

Lemma 4.4 – **ICP over the Integers.**

Completeness. *If P and V follow the protocol then V always accepts d_P.*

Zero-knowledge. *Given b, c the verifier learns no additional information on the value d_P.*

Soundness. *The probability that P generates $\hat{d}_P, \hat{y}$ such that $c + b\hat{d}_P = \hat{y}$ when in fact $\hat{d}_P \neq d_P$ is at most $\frac{1}{n^{\delta_1}} + \frac{1}{n^{\delta_2}}$. We denote this occurrence as OC_1.*

Proof

Completeness. Immediate.

Zero-knowledge. Values b and c are uniformly distributed and randomly chosen without any correlation to d_P, and hence reveal no information of its value.

Soundness. Notice that $\Pr(\, OC_1\,) \leq \Pr(\, OC_1 \mid y \in \mathcal{Y}\,) + \Pr(\, y \notin \mathcal{Y}\,)$. From Lemma 4.2 it follows that if $y \in \mathcal{Y}$ then P will be able to generate values $\hat{d}_P, \hat{y}$ (where $\hat{d}_P \neq d_P$) which satisfy the equation $\hat{y} = b\hat{d}_P + c$ with probability at most $\frac{1}{n^{\delta_1}}$. Lemma 4.3 gives us the probability that y falls out of this range. Combining these probabilities we prove our lemma.■

4.7.2 Key Generation Protocol

We can finally describe the key generation protocol for the non-interactive threshold RSA signature scheme. The dealer will simply run the `DDFY-Sharing` protocol and then add a dealing of the verification data, simply running the `ICP-Gen-Integers` protocol (Figure 4-4) for every pair of players P_i and P_j. All these invocations have as input the same RSA composite N and security parameters δ_1, δ_2. For the invocation where P_i is the prover P, and P_j is the verifier V, the secret key input is d_i, namely P_i's secret partial key.[1]

It will be seen later that a single pair of values b, c suffices for V to verify polynomially many different signatures. This results in an efficient protocol for the dealer, as he invokes a constant number of `ICP-Gen-Integers` protocols, once and for all, regardless of the number of signatures that the system will need to generate.

As a result of the complete key generation protocol, player P_i holds the following values:

1. His share d_i.
2. Auxiliary authentication values $y_{i,1}, \ldots, y_{i,n}$ where $y_{i,j} \in \boldsymbol{Z}$ is used to authenticate his partial signature to P_j.
3. Verification values $b_{1,i}, \ldots, b_{n,i}$ and $c_{1,i}, \ldots, c_{n,i}$ where $b_{j,i} \in [0 \ldots N^{\delta_1}]$ and $c_{j,i} \in [0 \ldots N^{1+\delta_1+\delta_2}]$ are used to verify the correctness of P_j's partial signature.

4.7.3 Signature Protocol

We show the protocol for verification of a partial signature where there are two players, P and V, each holding the data which they received in ICP-Gen-Integers. The protocol appears in Figure 4-5. In the context of the Signature Phase this protocol will be carried out by every pair of players. After executing all these invocations of the Non-interactive Verification Protocol, player P_i will take a subset of $t+1$ shares which he has accepted, and will generate the signature for m.

Lemma 4.5 Non-interactive Verification *Assume that a cheating prover P^* cannot break RSA (in particular, he does not know and cannot compute the factorization of N). Let $N = pq$, where $p < q$, $p = 2p' + 1$, $q = 2q' + 1$, and p, q, p', q' are all prime numbers.*

[1] As pointed out in Section 4.6.3 we can regard d_i as a single element in $Z_\phi(N)$ and not as a vector in $Z_\phi(N)^{\pi-1}$, by carrying out the computations accordingly for each component.

Non-interactive Verification

Input: Player V: $b \in [0 \ldots N^{\delta_1}]$, $c \in [0 \ldots N^{1+\delta_1+\delta_2}]$
Player P: $d_P \in Z_{\phi(N)}$, $y = bd_P + c$
Both players: message $m \in Z_N^*$, RSA composite N

1. P broadcasts the partial signature $S_P = m^{d_P} \bmod N$ and the auxiliary value $Y = m^y \bmod N$
2. V checks if $S_P^b m^c = Y$. If yes, he concludes that $S_P = \pm m^{d_P} \bmod N$ and accepts it.

Figure 4-5: The Non-interactive Verification Protocol

Completeness. *If P and V follow the protocol then V always accepts the partial signature.*

Soundness. *A cheating prover P^* can convince V to accept $S_P \neq \pm m^{d_P} \bmod N$, with probability at most $\frac{1}{p'} + \frac{1}{N^{\delta_1}} + \frac{1}{N^{\delta_2}}$.*

Zero-knowledge. *Any (possibly cheating) verifier V^* interacting with prover P does not learn any information beyond the signature $S_P = m^{d_P} \bmod N$.*

Proof

Completeness. Immediate.

Soundness. First we shall examine the case where $y \in \mathcal{Y}$. Notice that the verifier uses a deterministic procedure to accept or reject the published pair S_P, Y, hence the probability stated in the lemma, is taken over the coin-tosses of the dealer in ICP-Gen-Integers which are consistent with the view of the prover (i.e. the value y). In order for P to convince V to accept S_P, Y, it must hold that $Y = S_P^b m^c \bmod N$. We know from the ICP-Gen-Integers protocol that $y = bd_P + c$, and hence $m^y = (m^{d_P})^b m^c \bmod N$. By dividing these two equations we get that:

$$Ym^{-y} = (S_P m^{-d_P})^b \bmod N \tag{4.5}$$

This means that Ym^{-y} must be in the subgroup $<S_P m^{-d_P}>$. Let k be the minimal value such that $Ym^{-y} = (S_P m^{-d_P})^k \bmod n$. Consequently, Equation (4.5) is satisfied only if: $b = k \bmod \mathsf{ord}(S_P m^{-d_P})$. Since b is chosen at random with uniform distribution from $[0 \ldots N^{\delta_1}]$, the probability that a pair (S_P, Y) satisfies Equation 4.5 is

$$\Pr(\, b = k \bmod \mathsf{ord}(Sm^{-d_P})\,) \leq \frac{\left\lceil \frac{N^{\delta_1}}{\mathsf{ord}(S_P m^{-d_P})} \right\rceil}{N^{\delta_1}} \leq \frac{1}{\mathsf{ord}(S_P m^{-d_P})} + \frac{1}{N^{\delta_1}}$$

There are only four elements of Z_N^* whose order is smaller than p', namely the four roots of unity. If $S_P m^{-d_P} = \pm 1 \bmod N$ then $S_P = \pm m^{d_P} \bmod N$. If the prover finds S_P such that $S_P m^{-d_P}$ is a non-trivial root of unity, then he can factor N. For all other choices of S_P, order $\mathsf{ord}(S_P m^{-d_P}) \geq p'$. This completes the proof for the case where $y \in \mathcal{Y}$. In addition, from Lemma 4.3 we know that the probability that $y \notin \mathcal{Y}$ is at most $\frac{1}{n^{\delta_2}}$, by combining these two probabilities we get the desired probability.

Zero-knowledge. Values b and c are picked independently from d_P, hence they give out no information on d_P. Furthermore, knowing b, c and $S_P = m^{d_P} \bmod n$, the verifier can compute $Y = m^y = S_P^b m^c \bmod N$.$\blacksquare$

4.8 Interactive Robust Threshold RSA Signatures

The basic idea underlying the interactive solution is that it suffices to know a single sample message and its partial signature (w, w^{d_P}) to verify the partial signature of any other message m, under the same key. The sample message will be used in the extension of a protocol by Chaum's and van Antwerpen's [CA90, BCDP91].

In this section we assume that the dealer is trusted throughout the Dealing Phase, in Section 4.8.3 we show how to dispense with this assumption.

4.8.1 Key Generation Protocol

The dealer runs the `DDFY-Sharing` protocol in Section 4.6.3. Then he chooses a public sample message w and generates sample partial signatures $W_i = w^{d_i}$, for each one of the partial keys held by the players.

`RSA-Key-Gen-2`

Input for all players: RSA modulus N.

1. D runs the `DDFY-Sharing` protocol in Section 4.6.3. Let d_i be the share given to player P_i.
2. D chooses a random value w in Z_n^* and broadcasts values $W_i = w^{d_i} \bmod N$ for $i = 1, 2, \ldots, n$.

Figure 4-6: RSA Threshold Key Generation Protocol for the non-interactive case

4.8.2 The Signature Protocol

When it is time to sign a message m each player P_i broadcasts the partial signature $S_i = m^{d_i} \bmod N$. Then he proves in zero-knowledge to all other players that $DL_m(S_i) = DL_w(W_i)$.

The protocol `Composite-EDL` proves equality of discrete logs modulo a composite. It is based on a protocol by Chaum and van Antwerpen [CA90], which was further developed in [BCDP91]. However that protocol is designed to prove such a statement over a prime field Z_p. That protocol and the proof presented there do not work over Z_n^* for a composite n as required here, since they strongly rely on the existence of a generator for the multiplicative group Z_p^*.

For clarity of presentation, we concentrate on two players only, the prover P and the verifier V. The player P has his secret key $d_P \in Z_\phi(N)$ and both players have access to a publicly known sample message w and its partial signature, $W_P = w^{d_P} \bmod N$, under P's key.

`RSA-Thresh-Sig-2`

Input for all players: An RSA modulus N, the values $w, W_1, \ldots, W_n$, a message m.

1. Each player broadcasts $S_i = m^{d_i} \bmod N$
2. Each pair of players P_i, P_j execute the protocol `Composite-EDL` in Figure 4-8, on input S_i, m, W_i, w, n. P_i will run as the prover on private input d_i. P_j will act as the verifier.
3. P_i will accept partial signatures S_j which passes the above proof. Then interpolates a set V of $t+1$ of them to compute $S = \prod_{j \in V} S_j^{\lambda_{0,j,V}}$

Figure 4-7: Interactive RSA Threshold Signature Protocol

`Composite-EDL`(S_P, m, W_P, w, N)

Input for the Prover: a secret $d_P \in Z_\phi(N)$
Common Input: A message m, its claimed signature S_P, a sample message w, its signature W_P, RSA composite N.

1. V chooses $i, j \in [0 \ldots N]$ and computes $R \stackrel{\text{def}}{=} m^i w^j \bmod N$
   $V \longrightarrow P : R$
2. P computes $S_R \stackrel{\text{def}}{=} R^{d_P} \bmod N$
   $P \longrightarrow V : S_R$
3. V verifies that $S_R = S_P^i W_P^j \bmod N$.
   If equality holds then V concludes that $S_P = \pm m^{d_P} \bmod N$ and accepts it.

Figure 4-8: Interactive Verification of Partial Signatures

We state the security properties of the Interactive Verification protocol in Lemma 4.6 below.

Lemma 4.6 Interactive Verification. *Assume that a cheating prover P^* cannot break RSA (in particular, he does not know and cannot compute the factorization of n), and that w was chosen at random. Let $N = pq$, where $p < q$, $p = 2p' + 1$, $q = 2q' + 1$, and p, q, p', q' are all prime numbers.*

Completeness. *If P and V follow the protocol then V always accepts.*

Soundness. *No cheating prover P^* can convince V to accept $S_P \neq \pm m^{d_P} \bmod N$, except for a negligible probability.*

The protocol in figure 4-8 is actually not *zero-knowledge*. For example, a cheating verifier V^* can choose R in a different way than specified and then learn S_R, which V^* could not compute by himself. However, there are well-known techniques [GMW91, BCC88, Gol95] to add the zero-knowledge property to the above protocol using the notion of a *commitment function*: Instead of P sending S_R in Step 2, he sends a commitment $commit(S_R)$, after which V reveals to P the values of i and j. After checking that $R = m^i w^j \bmod N$, P sends S_R to V. The verifier checks that S_R corresponds to the value committed by P and then performs the test of Step 4 above.

The zero-knowledge condition is achieved through the properties of the commitment function, namely, (I) $commit(x)$ reveals no information on x, and (II) P cannot find x' such that $commit(x) = commit(x')$. Commitment functions can be implemented in many ways. For example, in the above protocol $commit(S_R)$ can be implemented as RSA encryption of S_R concatenated with a random string r (the encryption is required to be semantically secure, see [GM84, BR94]), using a public key for which the private key is not known to V (and possibly, not even known to P). To open the commitment, P reveals both S_R and r. This implementation of a commitment function is very efficient as it involves no long exponentiations.

Proof of Lemma 4.6

The proof of the completeness property of the protocol is immediate. The rest of this section is devoted to the proof of soundness.

Lemma 4.7 *The prover's cheating probability in the Interactive Verification Protocol is at most* $\frac{ind(w)}{\mathsf{ord}(S_P/m_P^d)} + 2\frac{N-\phi(N)}{N}$.

Proof Denote by b the quotient $\frac{S_P}{m_P^d}$. If $b = 1$ the signature S_P is correct. If $b \neq 1$, the prover's probability to cheat, i.e., to convince V to accept S_P, is maximized by choosing S_R that passes V 's test (in step 3) with maximal probability (relative to the values i, j chosen by V). Since the prover chooses S_R after having seen the "challenge" R from V (and based on its knowledge of S_P, m, w, d_P and N), then the proof of soundness needs to capture that some information on i, j (at least from the information theoretic point of view) is available to the prover when selecting S_R.

In the actual protocol, V chooses i, j randomly from the set $[0 \ldots n]$; for simplicity of analysis we will assume that these values are chosen from $[0 \ldots \phi(N)]$, and will account for the event that either i or j fall outside of this range in the prover's probability to cheat. The probability of such event (i.e., that i or $j \notin [0 \ldots \phi(N)]$) is at most $2\frac{N-\phi(N)}{N}$, and it appears in the lemma's bound.

Therefore we assume $i, j \in [0 \ldots \phi(N)]$ and define the following two sets $E(R) = \{(i,j) \in [0 \ldots \phi(N)] : R = m^i w^j \bmod N\}$ and $E(R, S_R) = \{(i,j) \in E(R) : S_R = S_P^i {W_P}^j \bmod N\}$. By choosing a value S_R in step 2 of the protocol, the prover P can cheat V to accept S_P only if the pair (i,j) chosen by V in step 1 actually belongs to $E(R, S_R)$. Since the choice of V is uniformly distributed over $E(R)$, then the probability of P to succeed is at most $\frac{|E(R,S)|}{|E(R)|}$. This is true even after P has seen R, and even if P could explicitly compute all the pairs in the set $E(R)$ and $E(R, S)$. By bounding the above expression for all possible values of R and S we get a bound on the prover's probability to cheat. Therefore, the lemma follows from the following bounds (which are proven below) on the size of the above sets for *any* choice of R and S (and any value of m, w, d_P, and S_P):

(A.1) $\quad |E(R)| \geq \phi(N)$

(A.2) $\quad |E(R,S)| \leq \frac{ind(w)\cdot\phi(N)}{\mathsf{ord}(b)}$ ∎

We stress that the lemma holds also for a computationally unbounded cheating prover. Also, the bound in the lemma is tight for such a prover (up to the term $2\frac{N-\phi(N)}{N}$). Next, we apply the lemma to prove the soundness of the protocol in the case that N is chosen with a particular structure, and the (cheating) prover cannot break RSA.

Corollary 4.1 *Assume the prover P cannot break RSA (in particular, it does not know and cannot compute the factorization of N), and that w was chosen at random (or as an element of small index). Let $N = pq$, where $p < q$, $p = 2p' + 1$, $q = 2q' + 1$, and p, q, p', q' are all prime numbers. Then, the prover's probability to convince V of accepting $S_P \neq \pm m^{d_P} \bmod N$ is at most $\frac{O(1)}{p}$.*

Proof The bound in Lemma 4.7 is given in terms of the order of some elements in the group Z_N^*. Thus, in order to establish the exact bound for the above special form of N we need to study the order of elements in this particular group. There are only four elements in Z_N^* whose order is less than p'. is one element of order 1 in Z_N^* (the unit element), 3 of order 2 (-1 and two other non-trivial roots of 1), $4p' + 4q' - 8$ elements of order ranging between p' and $2q'$, and the rest have all order which is at least $p'q'$. (This can be argued based on the special form of p and q and the order of elements modulo these primes, and then using the Chinese Remainder Theorem). In particular, the order of w, which is chosen at random, is at least $p'q'$, with probability $1 - \frac{2(p'+q')}{\phi(N)} \geq 1 - \frac{2}{p'}$, and then $ind(w)$ which equals $\phi(N)/\mathsf{ord}(w)$ is at most 4 (notice that $\phi(N) = 4p'q'$).

As for b, a successful cheating of P happens when it convinces V to accept a value $S_P = bm^{d_P} \bmod N$, for $b \neq \pm 1$. Notice that the prover (who knows d_P) can compute b. This excludes the possibility that b would be one of the non-trivial square roots of 1, since knowledge of such an element would allow the prover to factor N. Therefore, b must be of order at least p'. Finally, the expression $\frac{N-\phi(N)}{N}$ is at most $1/p'$ in this case. The corollary then follows by replacing these values in the bound expression $\frac{ind(w)}{\mathsf{ord}(S_P/m_P^d)} + 2\frac{N-\phi(N)}{N}$. in Lemma 4.7. ∎

Proof of inequalities (A.1) and (A.2). We proceed to prove inequalities (A.1) and (A.2) as we need to complete the proof of Lemma 4.7. We need to define some more sets and prove a few lemmas (we omit the mod N notation when it is clear from the context). We define $I(R) = \{i : \exists j, (i,j) \in E(R)\}$ and $I(R, S_R) = \{i : \exists j, (i,j) \in E(R, S_R)\}$. To prove inequality (A.1) we need to bound the size of the set $E(R)$. We do this by relating the size of $E(R)$ to that of $I(R)$ in the next lemma, and then bounding $|I(R)|$.

Lemma 4.8 *$\forall R$, $|E(R)| = ind(w) \cdot |I(R)|$.*

Proof Fix $i \in I(R)$. The pairs $(i, j_1), (i, j_2) \in E(R)$ iff $R = m^i w^{j_1} = m^i w^{j_2}$ iff $w^{j_1 - j_2} = 1$ iff $\mathsf{ord}(w)$ divides $j_1 - j_2$. Therefore, for each $i \in I(R)$ there exist exactly $ind(w) = \phi(N)/\mathsf{ord}(w)$ j's such that $R = m^i w^j$. ■

Lemma 4.9 *$\forall R$, $|I(R)| = 0$ or $|I(R)| \geq \mathsf{ord}(w)$*

Proof For any given value of R, $i_1, i_2 \in I(R)$ iff there are j_1, j_2 such that $R = m^{i_1} w^{j_1} = m^{i_2} w^{j_2}$. This implies that for any such $i_1, i_2, m^{i_1 - i_2} \in <w>$. On the other hand, if for values Δ and i, $m^\Delta \in <w>$, and $i \in I(R)$ then $i + \Delta \in I(R)$ (because then there exist j, j' such that $R = m^i h^j$ and $m^\Delta = w^{j'}$ from which it follows that $R = m^{i+\Delta} w^{j-j'}$). Therefore, we get that $I(R) = \{i_0 + \Delta : m^\Delta \in <w> \text{ and } \Delta < \phi(N)\}$ where i_0 is the minimal element in $I(R)$. Thus, if $|I(R)| \neq 0$ then $|I(R)| = |D|$, where $D = \{\Delta < \phi(N) : m^\Delta \in <w>\}$. We proceed to bound the size of D. Using standard arguments it is easy to show that if δ is the minimal non-zero element of D then the elements of D are exactly the multiples of δ (smaller than $\phi(N)$). Thus, $|D| = \phi(N)/\delta$. We end the proof by showing $\delta \leq ind(w)$. Let $i_1 < i_2 \leq \delta$. The cosets $m^{i_1} <w>$ and $m^{i_2} <w>$ are disjoint (a common element would imply that $w^{i_2 - i_1} \in <w>$ in contradiction to the minimality of δ). Thus, $m^1 <w>$, $m^2 <w>$, $\cdots m^\delta <w>$, are δ disjoint cosets in Z_N^* each of size $\mathsf{ord}(w)$. We have $\delta \leq \frac{|Z_N^*|}{\mathsf{ord}(w)} = \frac{\phi(N)}{\mathsf{ord}(w)} = ind(w)$. In sum, $|I(R)| = |D| = \phi(N)/\delta \geq \phi(N)/ind(w) = \mathsf{ord}(w)$ ■

Lemma 4.10 *$\forall R, S$, $|E(R,S)| \leq ind(w) \cdot |I(R,S)|$*

Proof Similarly to the proof of Lemma 4.8 one can show that for any $i \in I(R)$, if the pairs $(i, j_1), (i, j_2) \in E(R)$ then $\mathsf{ord}(w)$ divides $j_1 - j_2$. Therefore, for each $i \in I(R)$ there are at most $ind(w) = \phi(N)/\mathsf{ord}(w)$ different j's for which $(i, j) \in E(R)$. ■

Lemma 4.11 $\forall R, S,\ |I(R, S)| \leq \frac{\phi(N)}{\mathsf{ord}(b)}$, *for* $b = \frac{S_P}{m^{d_P}}$.

Proof Let (i_1, j_1), (i_2, j_2) be elements of $E(R, S)$. By the definition of $E(R, S)$, and by putting $\Delta_i \stackrel{\text{def}}{=} i_1 - i_2$ and $\Delta_j \stackrel{\text{def}}{=} j_2 - j_1$, we get $m^{\Delta_i} = w^{\Delta_j}$ and $S_P^{\Delta_i} = W_P^{\Delta_j}$. Then,

$$S_P^{\Delta_i} = W_P^{\Delta_j} = (w^{d_P})^{\Delta_j} = (w^{\Delta_j})^{d_P} = (m^{\Delta_i})^d_P = (m^d_P)^{\Delta_i},$$

which implies $b^{\Delta_i} = (S_P/m^d_P)^{\Delta_i} = 1$, and then $\mathsf{ord}(b)$ divides Δ_i. In other words, we have proved that for any $i_1, i_2 \in I(R, S)$, $\mathsf{ord}(b)$ divides $i_1 - i_2$. Since all elements in $I(R, S)$ are smaller than $\phi(N)$ there can be at most $\frac{\phi(N)}{\mathsf{ord}(b)}$ such elements. ■

Corollary 4.2 *Inequalities (A.1) and (A.2) hold, namely,* $|E(R)| \geq \phi(N)$ *and* $|E(R, S)| \leq \frac{ind(w) \cdot \phi(N)}{\mathsf{ord}(b)}$.

Proof Directly from Lemmas 4.8,4.9, 4.10 and 4.11. ■

4.8.3 Untrusted Dealer

In this section we show how to verify that the dealer performs correctly the sharing of the keys and of the verification data in the interactive protocol of Section 4.8 This will allow us to reduce significantly the amount of trust on the server that initializes the system. One thing to notice though is that we cannot tolerate a malicious dealer D that cooperates with some of the players P_i. Indeed in this case we could not avoid that D communicates to the bad players the factorization of N or the secret key and they would be able to sign on their own. Notice also that the bad players could mount undetected a denial of service attack by giving out incorrect partial signatures, since, as we saw in Section 4.8, the security of the signature protocol is guaranteed only in the case that bad players cannot factor N. However in this case the good players would realize that the secrecy of the factorization of N is lost and revocate the public key.

However we believe that the verification procedure described in this section has still some value. Indeed one can picture a scenario in which the dealer is a black-box that generates RSA keys, their sharings and the verification information. This procedure will allow to check for the correctness of the program inside the black box.

Verification of the parameters. We need to check that the dealer chooses the modulus n of the right form, i.e. $n = pq$ with $p = 2p' + 1$ and $q = 2q' + 1$. Although this choice can be theoretically checked using the general results of [GMW87] on secure computation, the resultant solution would be hardly practical. To alleviate this problem one could have the dealer generate a large set of moduli $n_1, n_2, \cdots$ from which the players collectively choose a random element, say n_i. Next, D shows the factorization into primes of all the other moduli in the set. If all are of the right form then n_i is chosen as the modulus n, otherwise D is disqualified. Also in order to make sure that the sample message w is chosen at random, the players collectively generate it by using some protocol for collective coin toss.

Verification of shares and sample signatures. The following is a procedure by which each player can verify the correct dealing of d into the shares $d_1, \ldots, d_n$, and the correct value of the sample signatures $W_i = w^{d_i}$. Verifying the correctness of the shares means that the values d_i all lie in a unique polynomial of degree t whose free coefficient is d (the secret exponent of the collective signature). The values W_i are correct if they correspond to the partial signatures w^{d_i} for the verified shares d_i. Let $x_1, \ldots, x_n$ be the values used for polynomial evaluation in the `DDFY-Sharing` procedure, and let $x_0 = 0$. Let $a_{ij}, i = 1, 2, \ldots, t+1, \;\; j = 1, 2, \ldots, n$, be *interpolation coefficients* such that for any polynomial p of degree t, and for $j = 0, 1 \ldots, n$ it holds that $p(x_j) = \sum_{i=1}^{t+1} a_{ij} p(x_i)$. Since a set of values $d_1, \ldots, d_n$ is a correct sharing of the value d if and only if there exists a polynomial p of degree t such that $d = p(0)$, and $d_j = p(x_j)$, for $j = 1, 2, \ldots, n$, then we get that a correct sharing is verified by the equations:

$$d = \sum_{i=1}^{t+1} a_{i0} d_i \qquad d_j = \sum_{i=1}^{t+1} a_{ij} d_i \qquad \text{for } j = t+1, \ldots, n \tag{4.6}$$

In our case the explicit values of all the shares d_i are not available to each player, therefore the checking of correct dealing is done using the equivalent of the above equations "in the exponent", namely, each player verifies that (remember that $w_i = w^{d_i}$):

$$w = \prod_{i=1}^{t+1} (w_i^{a_{i0}})^e \qquad w_j = \prod_{i=1}^{T} w_i^{a_{ij}} \qquad \text{for } j = t+1, \ldots, n \tag{4.7}$$

In order to be able to claim that the verification of (4.6) implies the correctness of (4.7) we need to solve two problems. The first is the fact that there may be a value W_i which is not an exponent of w, (this can be the case if both the dealer $\mathcal{D}$ and the player P_i are corrupted). The second problem is that even if the values W_i are all exponents of w, the equality in (4.7) only implies that the equality in (4.6) holds modulo $\mathsf{ord}(w)$, which may be a problem if w is an element of low order.

To verify that the values W_i are indeed exponents of w we use the following sub-protocol. For each $i \in [0 \ldots n]$ the dealer D chooses a value $r \in [0 \ldots \phi(N)]$ and broadcasts $W' = w^r$. The players collectively choose a random bit b. If $b = 0$, D broadcasts the value r, otherwise it broadcasts the value $d_i + r \bmod \phi(N)$. If $W_i \notin <w>$ then the probability that D provides a correct response is $1/2$. By repeating this procedure k times the probability of the dealer to cheat is 2^{-k}.

As for the problem that equalities (4.6) are verified only modulo $\mathsf{ord}(w)$, we point out that because of the assumed form of p and q (i.e., $(p-1)/2$ and $(q-1)/2$ being prime numbers), a random element w has with overwhelming probability order $\phi(N)/2$ or $\phi(N)/4$. In the former case, the order of w is a multiple of the order of all other elements in Z_N^* and then (4.7) implies (4.6). In the case $\mathsf{ord}(w) = \phi(N)/4$, the element $-w$ is of order $\phi(N)/2$. Therefore, one solution to the above problem is to repeat the described process for both w and $-w$.

Although the protocol is a little complicated and loses somewhat in efficiency, it is important to remember that this protocol is executed only once at initialization of the system.

Bibliography

[AB83] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, IT-30:208–210, March 1983.

[BB89] J. Bar-Ilan, and D. Beaver. Non-Cryptographic Fault-Tolerant Computing in a Constant Number of Rounds. In *Proceedings of the ACM Symposium on Principles of Distributed Computation*, pp.201–209, 1989.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 37(2):156–189, 1988.

[BCDP91] J. Boyar, D. Chaum, I. Damgård, and T. Pedersen. Convertible undeniable signatures. In A.J. Menezes and S. A. Vanstone, editors, *Proc. CRYPTO 90*, pages 189–205. Springer-Verlag, 1991. Lecture Notes in Computer Science No. 537.

[Ben87] J. Benaloh. Secret sharing homomorphisms: Keeping shares of a secret sharing. In A. M. Odlyzko, editor, *Proc. CRYPTO 86*. Springer, 1987. Lecture Notes in Computer Science No. 263.

[BK89] M. Blum and S. Kannan. Program correctness checking and the design of programs that check their work. In *Proc. of the 21st ACM Symposium on Theory of Computing*, 1989.

[BL88] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In S. Goldwasser, editor, *Proc. CRYPTO 88*, pages 27–36. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 403.

[Bla79] G. R. Blakley. Safeguarding cryptographic keys. In *Proc. AFIPS 1979 National Computer Conference*, pages 313–317. AFIPS, 1979.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for fault-tolerant distributed computing. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 1–10, Chicago, 1988. ACM.

[Boy86] C. Boyd. Digital Multisignatures. In H. Baker and F. Piper, editors, *Cryptography and Coding*, pages 241–246. Claredon Press, 1986.

[BR94] M. Bellare and P. Rogaway. Optimal asymmetric encryption. In *Euro-Crypt'94*, 1994.

[BW] E. Berlekamp and L. Welch. Error correction of algebraic block codes. US Patent 4,633,470.

[CA90] David Chaum and Hans Van Antwerpen. Undeniable signatures. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 212–217. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *Proc. 20th ACM Symp. on Theory of Computing*, pages 11–19, Chicago, 1988. ACM.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *Proc. 26th IEEE Symp. on Foundations of Comp. Science*, pages 383–395, Portland, 1985. IEEE.

[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.

[DDFY94] Alredo 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, Santa Fe, 1994. IEEE.

[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 G. Desmedt. Threshold cryptography. *European Transactions on Telecommunications*, 5(4):449–457, July 1994.

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In G. Brassard, editor, *Proc. CRYPTO 89*, pages 307–315. Springer-Verlag, 1990. Lecture Notes in Computer Science No. 435.

[DF92] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 457–469. Springer, 1992. Lecture Notes in Computer Science No. 576.

[DH76] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Trans. Inform. Theory*, IT-22:644–654, November 1976.

[Fel87] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symp. on Foundations of Comp. Science*, pages 427–438, Los Angeles, 1987. IEEE.

[FGY96] Y. Frankel, P. Gemmell, and M. Yung. Witness-based Cryptographic Program Checking and Robust Function Sharing. To appear in proceedings of STOC96, 1996.

[FM88] Paul Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. In *20th ACM Symposium on Theory of Computing*, 1988. Final version to appear in the SIAM J. of Computing.

[Gam85] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Inform. Theory*, 31:469–472, 1985.

[GJKRa] Rosario Gennaro, Stas Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold DSS signatures. To appear in EUROCRYPT 1996.

[GJKRb] Rosario Gennaro, Stas Jarecki, Hugo Krawczyk, and Tal Rabin. Robust and efficient sharing of RSA functions. To appear in CRYPTO 1996.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[GM95] Rosario Gennaro and Silvio Micali. Verifiable secret sharing as secure computation. In *EUROCRYPT'95*, volume 921 of *Lecture Notes in Computer Science*, pages 168–182. Springer-Verlag, 1995.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. *SIAM. J. Computing*, 18(1):186–208, February 1989.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *19th ACM Symposium on Theory of Computing*, pages 218–229, 1987.

[GMW91] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.

[Gol95] Oded Goldreich. *Foundation of Cryptography – Fragments of a Book*. Electronic Colloquium on Computational Complexity, February 1995. Available online from `http://www.eccc.uni-trier.de/eccc/`.

[Har94] Lein Harn. Group–oriented (t, n) threshold digital signature scheme and digital multisignature. *IEE Proc. Comput. Digit. Tech.*, 141(5):307–313, 1994.

[ISN87] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings IEEE Globecom '87*, pages 99–102. IEEE, 1987.

[Kah67] D. Kahn. *The Codebreakers*. Macmillian, New York, 1967.

[KGH83] E. Karnin, J. Greene, and M. Hellman. On sharing secret systems. *IEEE Transactions on Information Theory*, IT-29:35–41, 1983.

[Lan95] S. Langford. Threshold DSS signatures without a trusted party. In *Crypto'95*, pages 397–409. Springer-Verlag, 1995. Lecture Notes in Computer Science No. 963.

[MR92] S. Micali and P. Rogaway. Secure computation. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 392–404. Springer, 1992. Lecture Notes in Computer Science No. 576.

[MS81] R. McEliece and D. Sarwate. On sharing secrets and reed-solomon codes. *Communications of the ACM*, 24(9):583–584, September 1981.

[NIST91] National Institute for Standards and Technology. A proposed federal information processing standard for digital signature standard (DSS). Technical Report FIPS PUB XX, National Institute for Standards and Technology, August 1991. DRAFT.

[Ped91] T. Pedersen. Distributed provers with applications to undeniable signatures. In *EuroCrypt'91*, 1991.

[Ped92] T.P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In J. Feigenbaum, editor, *Proc. CRYPTO 91*, pages 129–140. Springer, 1992. Lecture Notes in Computer Science No. 576.

[PW72] W. Peterson and E. Weldon. *Error Correcting Codes*. MIT Press, 1972.

[Rab94] Tal Rabin. Robust sharing of secrets when the dealer is honest or cheating. *Journal of the ACM*, 41(6):1089–1109, 1994.

[Rab95] M. Rabin. A Simplification Approach to Distributed Multiparty Computations. Personal communication to Tal Rabin, 1995.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *21st ACM Symposium on Theory of Computing*, pages 73–85, 1989.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Sch91] C. P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4:161–174, 1991.

[Sch96] Bruce Schneier. *Applied Cryptography (Second Edition)*. John Wiley & Sons, 1996.

[Sha79] A. Shamir. How to share a secret. *Communications of the ACM*, 22:612–613, November 1979.

[Sta] Markus Stadler. Publicly verifiable secret sharing. To appear in EUROCRYPT 1996.

[Sti92] D.R. Stinson. An explication of secret sharing schemes. *Designs, Codes, and Cryptography*, 2:357–390, 1992.

[Sti95] Douglas R. Stinson. *Cryptography: Theory and Practice*. CRC Press, 1995.

[Sud96] Madhu Sudan. *Efficient Checking of Polynomials and Proofs and the Hardness of Approximation Problems*, volume 1001 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.