FUNCTION AND SECRET SHARING EXTENSIONS FOR BLAKLEY AND ASMUTH-BLOOM SECRET SHARING SCHEMES

A THESIS

SUBMITTED TO THE DEPARTMENT OF COMPUTER ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

İlker Nadi Bozkurt

August, 2009

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Ali Aydın Selçuk (Advisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Fazlı Can

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Kağan Gökbayrak

Approved for the Institute of Engineering and Science:

Prof. Dr. Mehmet B. Baray
Director of the Institute

ABSTRACT

FUNCTION AND SECRET SHARING EXTENSIONS FOR BLAKLEY AND ASMUTH-BLOOM SECRET SHARING SCHEMES

İlker Nadi Bozkurt
M.S. in Computer Engineering
Supervisor: Assist. Prof. Dr. Ali Aydın Selçuk
August, 2009

Threshold cryptography deals with situations where the authority to initiate or perform cryptographic operations is distributed amongst a group of individuals. Usually in these situations a secret sharing scheme is used to distribute shares of a highly sensitive secret, such as the private key of a bank, to the involved individuals so that only when a sufficient number of them can reconstruct the secret but smaller coalitions cannot. The secret sharing problem was introduced independently by Blakley and Shamir in 1979. They proposed two different solutions. Both secret sharing schemes (SSS) are examples of linear secret sharing. Many extensions and solutions based on these secret sharing schemes have appeared in the literature, most of them using Shamir SSS. In this thesis, we apply these ideas to Blakley secret sharing scheme.

Many of the standard operations of single-user cryptography have counterparts in threshold cryptography. Function sharing deals with the problem of distribution of the computation of a function (such as decryption or signature) among several parties. The necessary values for the computation are distributed to the participants using a secret sharing scheme. Several function sharing schemes have been proposed in the literature with most of them using Shamir secret sharing as the underlying SSS. In this work, we investigate how function sharing can be achieved using linear secret sharing schemes in general and give solutions of threshold RSA signature, threshold Paillier decryption and threshold DSS signature operations. The threshold RSA scheme we propose is a generalization of Shoup's Shamir-based scheme. It is similarly robust and provably secure under the static adversary model.

In threshold cryptography the authorization of groups of people are decided

simply according to their size. There are also general access structures in which any group can be designed as authorized. Multipartite access structures constitute an example of general access structures in which members of a subset are equivalent to each other and can be interchanged. Multipartite access structures can be used to represent any access structure since all access structures are multipartite. To investigate secret sharing schemes using these access structures, we used Mignotte and Asmuth-Bloom secret sharing schemes which are based on the Chinese remainder theorem (CRT). The question we tried to asnwer was whether one can find a Mignotte or Asmuth-Bloom sequence for an arbitrary access structure. For this purpose, we adapted an algorithm that appeared in the literature to generate these sequences. We also proposed a new SSS which solves the mentioned problem by generating more than one sequence.

Keywords: secret sharing, threshold cryptography, function sharing, multipartite access structures.

ÖZET

BLAKLEY VE ASMUTH-BLOOM ANAHTAR PAYLAŞTIRMA YÖNTEMLERİ İÇİN FONKSİYON VE ANAHTAR PAYLAŞTIRMA EKLENTİLERİ

İlker Nadi Bozkurt
Bilgisayar Mühendisliği, Yüksek Lisans
Tez Yöneticisi: Yrd. Doç. Dr. Ali Aydın Selçuk
Ağustos, 2009

Eşik kriptografisi, kriptografik bir işlemin gerçekleştirilebilmesi için gerekli olan yetkinin birden çok kullanıcı arasında paylaştırılması gereken durumlarla ilgilenir. Böyle durumlarda genellikle, bir bankanın gizli kriptografik anahtarı gibi çok gizli bir bilgi, bir anahtar paylaşım yöntemi kullanarak, belli sayıda katılımcının gizli bilgiye ulaşabileceği; ancak daha az sayıdaki grupların ulaşamayacağı şekilde bir grup insan arasında paylaştırılır. Anahtar paylaşım problemi ve ilk çözümleri 1979 yılında birbirlerinden bağımsız biçimde Shamir ve Blakley tarafından sunulmuştur. Birbirinden farklı olan bu iki anahtar paylaşım yöntemi de lineer bir anahtar paylaşımı yöntemidir. Literatürde anahtar paylaşım yöntemlerine birçok eklenti yapılmış ve bu yöntemlere dayanan birçok çözüm yer almıştır. Literatürdeki pek çok eklenti temel olarak Shamir anahtar paylaşım yöntemini kullanmıştır. Bu çalışmada Shamir anahtar paylaşım yöntemi için önerilmiş olan bazı eklentilerin Blakley anahtar paylaşım yöntemine nasıl uygulanabilecekleri gösterilmiştir.

Standart tek kullanıcılı pek çok kriptografik işlemin eşik kriptografisinde karşılığı vardır. Fonksiyon paylaştırılması problemi, kriptografik bir operasyonun (örneğin şifre çözme veya nitelikli imza atma) hesaplanmasının farklı katılımcılar arasında paylaştırılması ile ilgilidir. Hesaplama için gerekli değerler, uygun bir anahtar paylaşım yöntemi kullanarak taraflara dağıtılır. Daha önce literatürde, pek çoğu Shamir'in anahtar paylaşımını kullanan bir çok fonksiyon paylaşım yöntemi yer almıştır. Bu çalışmada, lineer anahtar paylaşım yöntemleri kullanarak fonksiyon paylaşımının nasıl yapılabileceği incelenmiş ve RSA imzası oluşturma, Pailier şifre çözme ve Sayısal İmza Standardı (DSS) imzası oluşturma için çözümler sunulmuştur. Bu çalışmada önerilen eşik RSA yöntemi Shoup'un

Shamir anahtar paylaşımı temelli yönteminin bir genellemesidir. Bu yöntem, benzer bir şekilde sağlam ve sabit düşman modelinde kanıtlanabilir şekilde güvenlidir.

Eşik kriptografisinde grupların yetkilendirilmesi basitçe sadece grubun büyüklüğü göz önüne alınarak yapılır. Bundan başka, istenen herhangi bir grubun yetkilendirilebildiği genel erişim yapıları vardır. Kullanıcıların gruplara ayrıldığı ve grup içindeki kullanıcıların birbirlerinin dengi olduğu çok bölümlü erişim yapıları genel erişim yapılarının bir örneğini oluştururlar. Bu erişim yapısı herhangi bir erişim yapısını göstermek için kullanılabilir, çünkü bütün erişim yapıları çok bölümlüdür. Bu erişim yapılarını kullanarak anahtar paylaşımı problemini incelemek için Çin kalan teoremine dayanan Mignotte ve Asmuth-Bloom anahtar paylaşım yöntemleri kullanıldı. Cevaplamaya çalıştığımız soru herhangi bir erişim yapısı için Mignotte veya Asmuth-Bloom dizilerinin bulunup bulunamayacağıdır. Bu amaç için literatürde yer alan bir yöntem uyarlanarak bu diziler oluşturulmuştur. Buna ek olarak, bahsedilen problemi birden çok dizi oluşturarak çözen yeni bir anahtar paylaşım yöntemi önerilmiştir.

Anahtar sözcükler: eşik kriptografisi, anahtar paylaşım yöntemleri, fonksiyon paylaşımı, çok kısımlı erişim yapıları.

Acknowledgement

First and foremost I would like to thank my supervisor, Dr. Ali Aydın Selçuk for his patience and guidance. It is a great privilege to work with him, his understanding and kind nature helped me to finish this thesis. I would also like to thank Kamer Kaya for his help and contributions throughout this study.

I would like to thank Dr. Fazlı Can for reading my work and giving feedback. I also want to thank Özgür Bağlıoğlu for reviewing my work, Musa Barış Demiray for his valuable work with Photoshop, İnci Durmaz and Serkan Uzunbaz for their contributions to implementations of some function sharing schemes. Also, I have to express my gratitude to Özgür Özuğur who made it easy for me to work on my thesis. Last but not least, special thanks to Kübra Gökdemir for her much needed support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Secret Sharing Schemes

The secure storage of the private keys of a cryptosystem is an important problem. Possession of a highly sensitive key by an individual may not be desirable as the key can easily be lost or as the individual may not be fully trusted. Giving copies of the key to more than one individual increases the risk of compromise. A solution to this problem is to give shares of the key to several individuals, forcing them to cooperate to find the secret key. This not only reduces the risk of losing the key but also makes compromising the key more difficult. In threshold cryptography, secret sharing deals with this problem, namely, sharing a highly sensitive secret among a group of n users such that only when a sufficient number t of them come together can the secret be reconstructed. More formally, in a secret sharing scheme there is one dealer and n players. The dealer gives a secret to the players, but only when some specific conditions are fulfilled. The dealer accomplishes this by giving each player a share in such a way that any group of t (for threshold) or more players can together reconstruct the secret but no group of fewer than t players can. Such a system is called a (t, n)-threshold scheme (sometimes it is written as an (n, t)-threshold scheme).

The problem of secret sharing and the first solutions were introduced in 1979

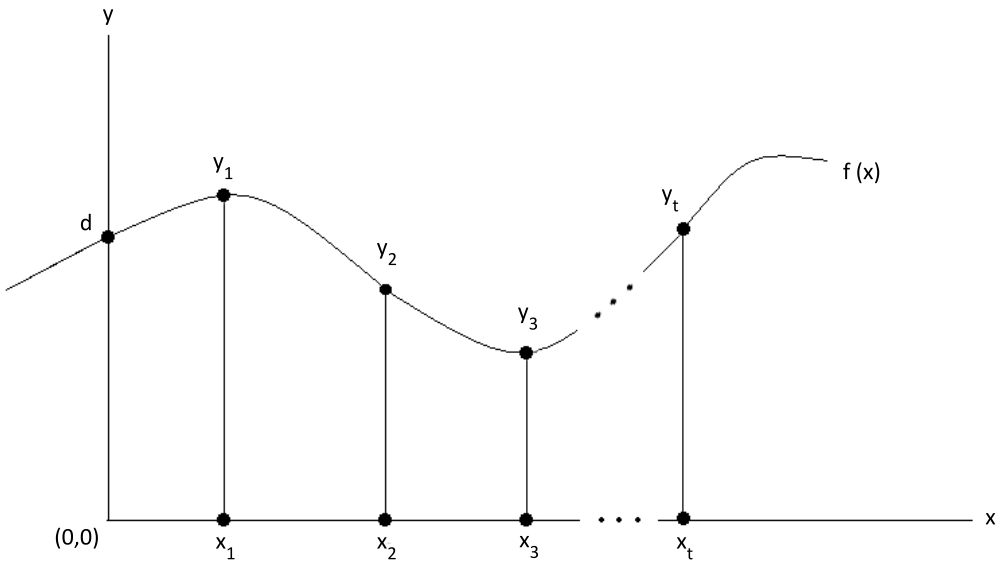

Figure 1.1: Shamir secret sharing scheme

independently by Shamir [48] and Blakley [3]. The approaches of Shamir and Blakley to solving the secret sharing problem were quite different, but the essential notion is the same in both cases. Other secret sharing schemes soon appeared in the literature. Mignotte [40] and Asmuth-Bloom [1] secret sharing schemes are based on Chinese remainder theorem (CRT). They solve exactly the same problem with Shamir and Blakley SSS but the approach is entirely different. Shamir's and Blakley's solutions are also different but their secret sharing schemes are members of the family of linear secret sharing schemes [32]. We will explain in detail Shamir, Blakley and linear secret sharing schemes in this introductory chapter. We will explain Mignotte and Asmuth-Bloom secret sharing schemes in Chapter 4.

1.1.1 Shamir Secret Sharing Scheme

Shamir's solution to the secret sharing problem is based on polynomial interpolation over a finite field $GF(q)$ (Galois field of prime order q). Figure 1.1 shows the basic idea. Given t points in the two dimensional plane, $(x_i, y_i), i = 1, 2, \ldots, t$ there is a unique polynomial $f(x)$ of degree $t-1$, for which $f(x_i) = y_i$ for all i.

If the secret is taken to be an element $d \in GF(q)$, it can be partitioned into n shares as follows. A polynomial $f(x) = \sum_{i=0}^{t-1} a_i x^i$, is generated such that a_0 is set to the secret value d and the coefficients a_1 to a_{t-1} are assigned random values from the Galois field $GF(q)$. The value $d_i = f(i)$ is given to user i.

When t out of n users come together, they can construct the polynomial using Lagrange interpolation. Without loss of generality assume players $1, 2, \ldots, t$ want to obtain the secret d. They compute d as follows:

$$d = \sum_{i=1}^{t} (d_i \cdot \prod_{j \neq i} \frac{x_j}{x_j - x_i}). \tag{1.1}$$

1.1.2 Blakley Secret Sharing Scheme

Blakley secret sharing scheme has a different approach based on hyperplane geometry: To implement a (t, n) threshold scheme, each of the n users is given a hyperplane equation in a t dimensional space over a finite field $GF(q)$ such that each hyperplane passes through a certain point. The intersection point of the hyperplanes is the secret. When t users come together, they can solve the system of equations to find the secret.

Figure 1.2 shows an example realization of Blakley SSS. Here $t = 2$, so each hyperlane equation is actually a line equation in the 2-dimensional space.

Blakley proposed choosing the hyperplanes that pass through the secret point randomly. If q is sufficiently large and t is not large, then the probability that any t of the hyperplanes intersect in some point other than the secret point is close to zero [3]. Thus generally it is possible to find the secret from any t of the n shares. However, it may not be possible to find the intersection point in some cases. In this case the resulting matrix is singular, i.e. the determinant is zero. The probability that a randomly chosen $t \times t$ matrix with elements chosen from the finite field $GF(q)$ is nonsingular can be computed by $\left(1 - \frac{1}{q}\right)\left(1 - \frac{1}{q^2}\right) \ldots \left(1 - \frac{1}{q^t}\right)$ [36]. This follows from the fact that the first column can be anything but the zero vector, the second column can be anything but the multiples of the first column, and in general the k-th column can be any vector not in the linear span of the

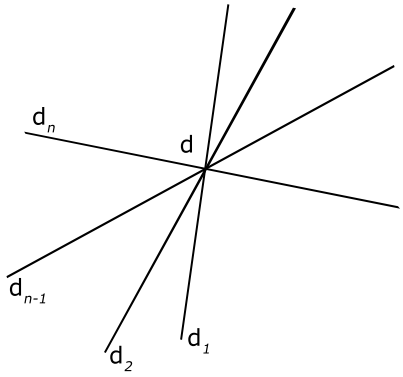

Figure 1.2: Blakley secret sharing scheme for $t = 2$

first $k-1$ columns. When the prime q is large, this probability is high enough to insure that the matrix will be invertible.

1.1.3 Linear Secret Sharing Schemes

Both Shamir and Blakley are *linear* threshold secret sharing schemes: As Karnin et al. [32] observed, Shamir SSS is a subclass of a broader class of linear secret sharing schemes. The polynomial share computation can be represented as a matrix multiplication by using a Vandermonde matrix. Similarly, the secret and the shares of the Blakley SSS can be represented as a linear system $Ax = y$ where the matrix A and the vector y are obtained from the hyperplane equations.

More formally, a linear (t, n) threshold secret sharing scheme (LSSS) can be defined as follows: Let $\mathcal{F}$ be a finite field and let A be a full-rank public $n \times t$ matrix with entries chosen from $\mathcal{F}$. Let $x = (x_1, x_2, \ldots, x_t)^T$ be a secret vector from $\mathcal{F}^t$. Let a_{ij} denote the entry at the ith row and jth column of the matrix A.

1.1.3.1 Dealing Phase

The dealer chooses a secret vector $x \in \mathcal{F}^t$ where the first entry x_1 is set to the secret value and the values of the other coordinates are set randomly from the field $\mathcal{F}$. The ith user will get a his share $y_i \in \mathcal{F}$,

$$y_i = a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{it}x_t. \tag{1.2}$$

For a (t, n) threshold scheme there will be n such shares, and hence we will have an $n \times t$ linear system

$$Ax = y. \tag{1.3}$$

The dealer then sends the secret value of y_i to user i for $1 \leq i \leq n$ and makes the matrix A public.

1.1.3.2 Share Combining Phase

Share combining step is simply finding the solution of a linear system of equations. Suppose that a coalition $\mathcal{S} = \{i_1, \ldots, i_t\}$ of users come together. They form a matrix $A_{\mathcal{S}}$ using their equations and solve

$$A_{\mathcal{S}}x = y_{\mathcal{S}}, \tag{1.4}$$

where $y_{\mathcal{S}}$ is the vector of the secret shares of the users. The secret is found as the first coordinate of the solution.

Most of the proposed secret sharing schemes are linear (the exceptions Mignotte and Asmuth-Bloom secret sharing schemes will be examined in Chapter 4), but the concept of an LSSS was first considered in its full generality by Karchmer and Wigderson [31] who introduced the notion of Monotone Span Programs.

Definition 1. A Monotone Span Program $\mathcal{M}$ is a triple (K, M, ψ), where K is a finite field, M is a matrix (with n rows and $m \leq n$ columns) over K, and $\psi : \{1, \ldots, m\} \to \{1, \ldots, n\}$ is a surjective (onto) function. The size of $\mathcal{M}$ is the number of rows (m).

ψ labels each row with a number from $1, \ldots, n$ corresponding to a player, so we can think each player as being the owner of one or more rows.

MSP's and LSSS's are in natural 1-1 correspondence as mentioned by Karchmer and Wigderson.

1.2 Properties of Secret Sharing Schemes

An important concept related to secret sharing schemes is information rate which compares the sizes of the shares to the size of the secret. It is introduced by Brickell [6] and is defined as follows:

$$\rho = \frac{\log_2(|d_{shares}|)}{\log_2(|d|)}. \tag{1.5}$$

Obviously having a high information rate is a desirable feature in secret sharing schemes. Secret sharing schemes with information rate equal to 1 are called ideal by Brickell.

Another important concept is perfectness. A secret sharing scheme is said to be perfect, when coalitions of size less than the threshold cannot obtain additional information about the secret compared to someone who does not have any shares or information about the secret. That is, until there are t players in the coalition, all values of the secret should be equally likely.

Shamir's scheme is perfect and ideal. The size of the shares is equal to the size of the secret. So, it is ideal. Also, for a given polynomial of degree $t-1$ and $t-1$ points on the polynomial, we can choose any point to be on the polynomial and for each different point, the value of the polynomial at 0 will differ. So all values are equally likely to be secret. Hence, Shamir's scheme is also perfect.

Blakley's secret sharing scheme is not ideal. Every user is given a hyperplane equation, instead of a number the same size as the secret, lowering the information rate below to 1. If we take the secret as the intersection point itself, then the scheme is not perfect. Because each hyperplane equation narrows down the

possibilities and when there are $t-1$ hyperplane equations it is guaranteed that the secret point lies in the intersection of all $t-1$ hyperplanes, which is a line. However, by choosing the secret as one of the coordinates (we choose the first coordinate), Blakley's scheme can be made perfect.

1.3 Extensions to Secret Sharing

In this section we will discuss several extensions to secret sharing schemes and cover the secret sharing literature that dealt with these extensions. In the next chapter, Blakley secret sharing will be enhanced with some of these extensions.

The presented secret sharing schemes solve the problem of making the secret available to sufficiently large coalitions. But the coalitions can actually recover the secret only if the following two conditions are met:

1. The dealer is honest and distributes consistent shares to each user
2. The users who participate in the secret recovery phase are honest and send correct shares.

Obviously if an adversary corrupts the dealer and/or some participants, he can prevent the successful recovery of the secret. Moreover, if he can find corrupt enough users he may be able to destroy the secret. For example, if $n = 2t-1$ and the adversary corrupts more than $t-1$ participants, then the secret is lost. Verifiability extension deals with these problems. In a *verifiable* secret sharing scheme, the users are able to verify that their shares are consistent.

The case of a possible dishonest dealer has been discussed for the first time by Chor, Goldwasser, Micali, and Awerbuch [8], who introduced the notion of verifiable secret sharing schemes in which every user can verify that he has received a valid share. After Chor et al. more efficient non-interactive verifiable secret sharing schemes were proposed by Feldman [17] and Pedersen [45]. The security of

the Feldman's scheme depends on the hardness of the discrete logarithm problem whereas Pedersen's scheme is information theoretically secure.

The problem of cheating in the reconstruction phase has been discussed by McEliece and Sarwate [38], and later on, by Tompa and Wool [50]. As Schoenmakers has remarked in [47], verifiable secret sharing can also be seen as a solution for the problem of cheating — the shares presented in the reconstruction phase may be verified with respect to the distribution phase.

A further extension to secret sharing is *public verifiability*. In a publicly verifiable secret sharing scheme, the users are able to verify that distributed shares are consistent with each other. This property is included in the seminal paper of Chor et al. on verifiable secret sharing schemes. However, both Feldman and Pedersen's schemes do not support public verifiability.

As described above, the secrecy of the secret is protected if less than t players are corrupted. However, if the secret sharing scheme and therefore the secret is long lived, an adversary may corrupt enough players in this long time period. To prevent this problem, *proactive* secret sharing schemes are proposed. In [25], Herzberg et al. proposed a proactive secret sharing scheme where the shares are renewed periodically without changing the secret. Because of the proactivity property, an adversary has to corrupt t users in a specific time period (share update period) e.g., a day, a week or a month.

1.4 Function Sharing Schemes

A shortcoming of secret sharing schemes is the need to reveal the secret shares during the reconstruction phase. The system would be more secure if the subject function can be computed without revealing the secret shares or reconstructing the secret. This is known as the function sharing problem. A function sharing scheme (FSS) requires distributing the function's computation according to the underlying SSS such that each part of the computation can be carried out by a different user and then the partial results can be combined to yield the function's

value without disclosing the individual secrets.

FSSs are typically used to distribute the private key operations in a public key cryptosystem (i.e. the decryption and signature operations) among several parties. Sharing a private key operation in a threshold fashion requires first choosing a suitable SSS to share the private key. Then the subject function must be arranged according to this SSS such that combining the partial results from any t parties will yield the operation's result correctly. This is usually a challenging task and requires some ingenious techniques.

Function sharing problem is formally introduced by Desmedt and Frankel in 1989 [12]. They also proposed non-interactive and practical threshold function sharing schemes for ElGamal encryption scheme. The solutions they proposed were based on Shamir and Blakley SSS.

After Desmedt and Frankel's work, the function sharing problem for RSA public-key cryptosystem was investigated by several researchers where Shamir SSS was the main tool. The additive nature of the Lagrange's interpolation formula used in the combining phase of Shamir's scheme makes it an attractive choice for function sharing, but it also provides several challenges. One of the most significant challenges is the computation of inverses in $\mathbb{Z}_{\phi(N)}$ for the division operations in Lagrange's formula, while $\phi(N)$ should not be known by the users. There are two main difficulties in this respect:

1. An inverse x^{-1} will not exist modulo $\phi(N)$ if $\gcd(x, \phi(N)) \neq 1$.
2. Even when x^{-1} exists it should not be computable by a user, since that would enable computing $\phi(N)$.

The first solution to this problem was proposed by Desmedt and Frankel [12], which solved the problem by making the dealer compute all potentially needed inverses at the setup time and distribute them to users along with the shares. A more elegant solution was found three years later by DeSantis et al. [46]. They carried the arithmetic into a cyclomatic extension of $\mathbb{Z}$, which enabled computing inverses without knowing $\phi(N)$. Finally, an ingenious solution was given by

Shoup [49] where he removed the need of taking inverses in Lagrange interpolation altogether.

Shoup's practical RSA scheme has inspired similar works on different cryptosystems. Fouque et al. [18] proposed a similar Shamir-based threshold solution for the Paillier cryptosystem and used it in e-voting and lottery protocols. Later, Lysyanskaya and Peikert [37] improved this work and obtained a threshold Paillier encryption scheme secure under the adaptive adversary model. The threshold RSA signatures we present in Chapter 3 are also inspired by Shoup's work.

Although using Shamir SSS for sharing the ElGamal signature and decryption functions has its own unique problems, the computation of inverses in the exponent is relatively easier than that in RSA since all of the operations are done modulo p where p is a public prime and hence $\phi(p) = p - 1$ is also public. As mentioned above, Desmedt and Frankel solved the function sharing problem in 1989 for the ElGamal decryption function. However, an ElGamal based threshold signature was not proposed until 1996. In [22], Gennaro et al. proposed the first efficient threshold scheme for the Digital Signature Standard (DSS).

Since functions sharing schemes are based on secret sharing schemes, the extensions on secret sharing schemes can also be defined for function sharing schemes. For example, the *robustness* extension is similar to the verifiability extension for SSSs. We say that a function sharing scheme is *robust* if it can withstand participation of corrupt users in the function evaluation phase. The general approach to achieve robustness in function sharing schemes is sending more information along with the partial result. In that approach, each user in the coalition sends a proof of correctness of his partial result. In robust FSS schemes, a valid proof of correctness cannot be generated by a user unless he has the correct share and he provides the correct partial. Gennaro et al. proposed a robust threshold RSA scheme [21] and a robust DSS signature scheme [22, 23]. The threshold RSA signature we describe in Chapter 3 is also a robust FSS.

In summary, several solutions for sharing the RSA, ElGamal, and Paillier private key operations have been proposed in the literature [11–14, 18, 34, 37, 46, 49]. Almost all of these schemes have been based on the Shamir SSS with the

exceptions being [34] based on Asmuth-Bloom and [12] giving a Blakley based ElGamal.

1.5 Secret Sharing in General Access Structures

The (authorized) *access structure* of a secret sharing scheme is the set of all groups which are designed to reconstruct the secret. We will denote the access structure of a secret sharing scheme with Γ. The elements of the access structure are referred to as authorized groups (sets) and the rest are called unauthorized groups (sets). The set of all unauthorized groups is called the *adversary structure*. The adversary structure will be denoted by $\overline{\Gamma}$.

In (t, n) threshold systems, the groups of people who can recover the secret, i.e. the access structure, are decided simply according to the cardinality of the group. So the definition of (t, n) threshold access structures can be given as follows:

$$\Gamma = \{A \in \mathcal{P}(\{1, 2, \ldots, n\}) : \ |A| \geq t\}. \tag{1.6}$$

The adversary structure is obviously:

$$\overline{\Gamma} = \{A \in \mathcal{P}(\{1, 2, \ldots, n\}) : \ |A| < t\}. \tag{1.7}$$

Threshold access structures may be inadequate in some situations. In practice, there may be situations in which every authorized subset has to contain participants from a certain subset or in which an authorized subset of players cannot contain a certain subset of players. Some researchers investigated this problem, i.e. constructing secret sharing methods that allow more general access structures than threshold ones. In general access structures, any group can be designed as authorized i.e. eligible for recovering the secret.

Ito, Saito and Nishizeki did the first work on secret sharing schemes with general access structures [29]. They remarked that any access structure has to satisfy the following condition:

$$(\forall B \in \mathcal{P}(\{1, 2, \ldots, n\}))((\exists A \in \Gamma)(A \subseteq B) \Rightarrow B \in \Gamma). \tag{1.8}$$

Intuitively, this means if a group can recover the secret, so can a larger group (containing the group that can recover the secret). Such access structures are called monotone access structures by Benaloh and Leichter in [2]. We will be interested only in monotone access structures. This monotonicity property implies a dual property for the adversary structure:

$$(\forall B \in \mathcal{P}(\{1, 2, \ldots, n\}))((\exists A \in \overline{\Gamma})(B \subseteq A) \Rightarrow B \in \overline{\Gamma}). \tag{1.9}$$

This means that if a group of players cannot recover the secret, neither can a smaller group.

In [29], Ito et al. proposed a multiple assignment method in which one or more shares of the (t, n) threshold scheme are allocated to each member. Ito et al. proved that, by distributing one or more shares to each member, it is possible to implement any form of access structure. However, with their method each of the n participants may have to hold on the order of 2^n shares in the worst case. The paper of Benaloh and Leichter [2] give a far simpler and more efficient method of developing a secret sharing scheme for any monotone access structure. The idea that they utilized is to translate the access structure into a monotone formula. Each variable in the formula is associated with a participant of the secret sharing scheme, and the value of the formula is *true* if and only if the set of variables which are *true* corresponds to a subset of the players that is in the access structure. This formula is then used as a template to describe how a secret is to be divided into shares.

Multipartite access structures constitute an example of general access structures in which members of certain groups are equivalent to each other and can be interchanged without changing the authorization of the group. Multipartite access structures were introduced in [43]. In the same work, the authors completely characterized ideal bipartite structures. Furthermore, the information rate of non-ideal structures is bounded and studied. Multipartite access structures can be considered general access structures since they can be used to represent any access structure as shown by [24]. We will describe multipartite access structures in detail in Chapter 4.

Chapter 2

Extensions to Blakley Secret Sharing Scheme

The secret sharing schemes given in the introductory chapter presents solutions to the secret sharing problem. However, several extensions to these schemes are possible and, moreover these extensions are necessary for a SSS to be used effectively in practical examples. In this chapter, we will discuss some extensions for secret sharing schemes that appeared in literature. Most of the extensions are given for Shamir SSS. Here, we will enhance Blakley's secret sharing scheme with these extensions. In [27] and [33], how Asmuth Bloom secret sharing scheme [1] can be enhanced with these properties is discussed. Before moving on to these extensions, it is useful to discuss a particular property of secret sharing schemes, namely homomorphism, which will be used in several ways, not only in these secret sharing extensions but also in function sharing schemes.

2.1 Homomorphic Properties of Blakley Secret Sharing

Homomorphism is a concept related to functions. A function f is said to be $(\oplus, \otimes)$ homomorphic, if f satisfies $f(x \oplus y) = f(x) \otimes f(y)$ for operations $\otimes$ and $\oplus$. A secret sharing scheme is a function which maps secrets to the distributed shares, so we can talk about homomorphism in the context of secret sharing schemes. The definition of homomorphism for a secret sharing scheme was given in [9].

Definition 2. Let $\oplus$ and $\otimes$ be binary functions on elements of the secret domain S and of the share domain T, respectively. We say that a (t, n) threshold scheme has the $(\oplus, \otimes)$-homomorphism property (or is $(\oplus, \otimes)$-homomorphic) if for all $\mathcal{S}$, whenever

$$d = F_{\mathcal{S}}(d_{i_1}, \ldots, d_{i_t})$$

and

$$d^{'} = F_{\mathcal{S}}(d^{'}_{i_1}, \ldots, d^{'}_{i_t}),$$

then

$$d \oplus d^{'} = F_{\mathcal{S}}(d_{i_1} \otimes d^{'}_{i_1}, \ldots, d_{i_t} \otimes d^{'}_{i_t}),$$

where d and $d^{'}$ denote shared secrets, $d_i \otimes d^{'}_i$ are shares of user i for secrets d and $d^{'}$ respectively, $\mathcal{S}$ is the coalition and $F_{\mathcal{S}}$ is the function used by coalition $\mathcal{S}$ for recovering the secret from their shares.

Homomorphism property implies that the composition of the shares are the shares of the composition. For a secret sharing scheme homomorphism is a useful property, and in some cases it is also a necessary property. For example, the threshold Digital Signature Standard (DSS) signature scheme given in Chapter 3 requires that the underlying linear SSS is $(\times, \times)$-homomorphic. Another example is related to joint random secret sharing. The joint random secret sharing schemes given in this chapter make use of the $(+, +)$-homomorphism properties of Shamir and Blakley secret sharing schemes. We also use $(+, +)$-homomorphism property for proactivity property.

We are interested in using 3 operations with Blakley SSS for homomorphism: multiplication by a scalar, addition and multiplication. Multiplication by a scalar and addition are linear operations and Blakley SSS is a linear SSS. It is no surprise that these operations provide homomorphism property without changing the threshold. Multiplication on the other hand is not a linear operation and increases the threshold (to t^2).

First, we show that Blakley SSS is $(+,+)$-homomorphic. Let A be an $(n \times t)$ matrix, x_1, x_2 be column vectors of length t and y_1, y_2 be column vectors of length n. First elements of x_1 and x_2 contain the secret values s_1 and s_2 respectively, i.e. if we denote ith element of a vector v with $v[i]$, then $s_1 = x_1[1]$ and $s_2 = x_2[1]$. The elements of y_1 and y_2 hold the shares corresponding to the secrets s_1 and s_2. Then, $Ax_1 = y_1$ and $Ax_2 = y_2$ implies

$$A(x_1 + x_2) = y_1 + y_2 \tag{2.1}$$

The above equation means that by summing the shares they have for the secrets s_1 and s_2, the players can find shares for the secret $s_1 + s_2$. The resulting secret sharing scheme is (t, n) as the original secret sharing schemes.

Multiplication by a scalar case is also easy to show. Let A again be an $(n \times t)$ matrix, x be a column vector of length t, y be a column vector of length n and c be a scalar value.

$$Ax = y \Rightarrow A(cx) = cy. \tag{2.2}$$

So, when every user multiplies his share d_i of a secret d with a scalar c, they obtain a share of the secret cd. The new secret is also (t, n) as the original one.

Now, we show that Blakley SSS is $(\times,\times)$-homomorphic. This problem is investigated by Cramer, Damgard and Maurer [10] in the context of Monotone Span Programs (MSP) (see Section 1.1.3). Cramer et al. defined multiplicative MSPs and showed it is possible to find an algorithm to convert any MSP to a multiplicative MSP of size at most twice the original MSP. We will not phrase the definition of multiplicative MSPs here but they are merely the equivalent

of $(\times, \times)$-homomorphic LSSSs. Cramer et al. showed the existence of such algorithms but it was Nikov et al. [42] who showed the full characterization of multiplicative MSPs. They proved that using two (different) MSPs to compute their resulting MSP is more efficient than building a multiplicative MSP. We will follow Nikov et al.'s approach for $(\times, \times)$-homomorphism.

To show Blakley SSS is $(\times, \times)$-homomorphic, let's first define the diamond $\diamond$ operation for vectors and matrices from Nikov et al. [42]. Here, we give the definition as given in Nikov et al.'s paper which because of working with monotone span programs, players are allowed to have more than one row. Diamond $\diamond$ operation is defined in terms of Kronecker product. For definition and some properties of Kronecker product the reader may consult [51]. In the following, $\otimes$ is used to denote the Kronecker product operator, which is the common usage, in contrast to the above usage as a symbol for an operator in general. For vectors, the diamond $\diamond$ operation is defined as:

$$x \diamond y = (\overline{x}_1 \otimes \overline{y}_1, \ldots, \overline{x}_n \otimes \overline{y}_n),$$

where the subvector $\overline{x}_i$ consists of elements of x belonging to player i. If each of the n players have k entries, then length of x and y is kn, whereas the length of $x \diamond y$ is $k^2 n$. When each player has one element, then $x \diamond y$ is just the element-wise multiplication of x and y. For matrices the definition is parallel to the vector case. Let A_k denote the matrix composed of rows of player k in matrix A. Then, the definition of $\diamond$ operation for matrices is as follows:

$$A \diamond B = \begin{pmatrix} A_1 \otimes B_1 \\ A_2 \otimes B_2 \\ \vdots \\ A_t \otimes B_t \end{pmatrix}.$$

Now let A be the public matrix in Blakley SSS, y_1 be the share vector of secret s_1, and y_2 be the share vector of secret s_2, i.e.

$$Ax_1 = y_1 \tag{2.3}$$

$$Ax_2 = y_2, \tag{2.4}$$

where first coordinates of x_1 and x_2 are s_1 and s_2 respectively. Then according to Lemma 3 in [42], $y = y_1 \diamond y_2$ is the share vector of secret $s_1 s_2$ corresponding to matrix $A \diamond A$, i.e.

$$(A \diamond A)(x_1 \otimes x_2) = y_1 \diamond y_2. \tag{2.5}$$

The first coordinate of $(x_1 \otimes x_2)$ is $s_1 s_2$ as desired. Obviously, if A is an $n \times t$ matrix, then $A \diamond A$ is an $n \times t^2$ matrix and to be able to recover the new secret, which is the multiplication of the original secrets, a coalition of at least t^2 players is needed. Hence, we can talk about $(\times, \times)$-homomorphism only if $n \geq t^2$.

2.2 Joint Random Secret Sharing

In this section, we present joint random secret sharing in which certain secret sharing schemes can be configured without the presence of a dealer. In this context, Jackson, Martin and O'Keefe [30] made the distinction of implicit and explicit secrets. By their definition, an *explicit* secret is a fixed value that is predetermined by factors outside the secret sharing scheme design. On the other hand, a secret is said to be *implicit* if it does not take a predetermined value. The secret sharing scheme has to protect the secret, but it can take any value from a specified domain. In the case of dealer free secret sharing schemes, the secret will be considered implicit.

For threshold access structures, dealer-free secret sharing was first discussed by Meadows [39]. In Meadows' scheme, the first t users generate their own shares randomly. However, to generate the shares of the remaining $n-t$ players, a black box is required. This black box is trusted with all the shares and the value of the implicit secret so it plays the role of a mutually trusted authority as Jackson et al. observed [30]. So the presented scheme is not really a dealer-free secret sharing scheme. However, it is the first paper discussing the concept of secret sharing without a dealer.

Ingemarsson and Simmons proposed an elegant scheme for dealer-free threshold secret sharing in [28]. In this scheme, the ith user first chooses an arbitrary

element d_i that will be the share of some secret d with respect to a unanimous secret sharing scheme of rank n (defined in the next paragraph) and then the element d_i is shared among the rest of users.

As it has been remarked in [30], joint random secret sharing can be achieved using a unanimous (n, n)-threshold scheme (unanimous consent structure of rank n). Let $m \geq 2$ be a fixed positive integer.

- Every participant chooses his share d_i as a random number from Z_m;
- The secret d is generated (and can be reconstructed) as $d = \sum_{i=1}^{n} d_i \bmod m$.

In the same paper, Jackson, Martin, and O'Keefe have remarked that any $(\otimes, \oplus)$-homomorphic secret sharing scheme can be used to construct a dealer-free secret sharing scheme.

- The ith participant chooses an element d_i and constructs, using a $(\otimes, \oplus)$-homomorphic secret sharing scheme, the shares $d_{i1}, \ldots, d_{in}$ corresponding to the secret d_i and securely distributes d_{ij} to the jth participant, for all $1 \leq j \leq n, j \neq i$;
- The secret d will be $d = \sum_{i=1}^{n} d_i$;
- Each participant computes his share as $d_i = \sum_{j=1}^{n} d_{ji}, 1 \leq i \leq n$.

Since Blakley secret sharing scheme is $(+, +)$-homomorphic, joint random secret sharing using Blakley SSS can be done as follows:

- The first player generates and broadcasts a full rank $n \times t$ matrix A.
- Each player i chooses d_i randomly as a secret and shares it using the matrix A. That is, player i sends y_{ij} to player j for $j \neq i$.
- Player j sums the shares it receives to construct his share : $y_j = \sum_{i=1}^{n} y_{ij}$. y_j's are shares of the secret $d = \sum_{i=1}^{n} d_i$.

2.3 Verifiable Secret Sharing

The secret sharing schemes presented in the introductory chapter assume that the parties involved behave honestly. In this section, we discuss some solutions for the case in which the dealer or some users may behave maliciously.

2.3.1 Feldman's Scheme

The scheme of Chor, Goldwasser, Micali, and Awerbuch [8] has a great disadvantage - it is interactive, i.e., some interaction between users is required in order to verify the consistency of the shares. Moreover, the communication complexity in their scheme is exponential. Feldman [17] has proposed a non-interactive scheme for achieving verifiability in Shamir's threshold secret sharing scheme. The main idea is to use a homomorphic one-way function f which satisfies $f(x+y) = f(x) \cdot f(y)$ and to broadcast $f(a_0), \ldots, f(a_{k-1})$, where $P(x) = a_0 + a_1 x + \ldots + a_{k-1}x^{k-1}$ is the polynomial used in Shamir's scheme. The consistency of the share $d_i = P(i)$ can be tested by verifying that

$$f(d_i) \stackrel{?}{=} f(a_0)f(a_1)^i \ldots f(a_{k-1})^{i^{k-1}}. \tag{2.6}$$

Indeed, by the homomorphic property of the function f,

$$f(a_0 + a_1 i + \ldots + a_{k-1}i^{k-1}) = f(a_0)f(a_1)^i \ldots f(a_{k-1})^{i^{k-1}}. \tag{2.7}$$

A good candidate for the function f is $f : \mathbb{Z}_q \to \mathbb{Z}_p$, $f(x) = g^x \bmod p$, where p and q are odd primes such that $q|(p-1)$, and $g \in \mathbb{Z}_p^*$ an element of order q. In this case we obtain the following scheme:

- The prime numbers p and q are generated such that $q|(p-1)$, and $g \in \mathbb{Z}_p^*$ an element of order q. All these numbers are public;
- The dealer generates the polynomial $P(x) = a_0 + a_1 x + \ldots + a_{k-1}x^{k-1}$ over $\mathbb{Z}_q$ such that $a_0 = d$ and makes $g_i = g^{a_i} \bmod p$ public, for all $0 \leq i \leq k-1$;

- The dealer securely distributes the share $d_i = P(i)$ to the ith user, for all $1 \leq i \leq n$;
- Each user can verify the correctness of the received share d_i by testing

$$g^{d_i} \bmod p \stackrel{?}{=} \prod_{j=0}^{k-1} g_j^{i^j} \bmod p.$$

2.3.2 Feldman's Scheme with Blakley

Blakley's SSS can be enhanced with verifiability property. Applying Feldman's idea for Shamir's SSS to Blakley's SSS we can obtain a verifiable Blakley SSS.

During dealing phase of Blakley's SSS, the ith user will get his share $d_i \in \mathcal{F}$,

$$d_i = a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{it}x_t. \tag{2.8}$$

as a hyperplane equation. As in Feldman's extension to Shamr SSS we use a homomorphic one-way function f which satisfies $f(x+y) = f(x) \cdot f(y)$. For verification, the values of $f(x_1), f(x_2), \ldots, f(x_t)$ are broadcasted by the dealer. The consistency of the share d_i can be checked by verifying that

$$f(d_i) \stackrel{?}{=} \prod_{j=1}^{t} f(x_j)^{a_{ij}}. \tag{2.9}$$

This easily follows from the homomorphic property of function f, since

$$f(y_i) = f\left(\sum_{j=1}^{t} a_{ij}x_j\right) = \prod_{j=1}^{t} f(x_j)^{a_{ij}} \tag{2.10}$$

Note that $f(x+y) = f(x)f(y)$ implies $f(xy) = (f(x))^y = (f(y))^x$.

Again the function f can be chosen as $f : \mathbb{Z}_q \to \mathbb{Z}_p$, $f(x) = g^x \bmod p$, where p and q are odd primes such that $q|(p-1)$, and $g \in \mathbb{Z}_p^*$ an element of order q. Putting up everything together, we obtain the following scheme:

- The dealer securely distributes the share d_i to the ith user as in equation 2.8, for all $1 \leq i \leq n$;
- The prime numbers p and q are generated such that $q|(p-1)$, and $g \in \mathbb{Z}_p^*$ an element of order q. All these numbers are public;
- The dealer broadcasts $f_i = g^{x_i} \bmod p$, for all $1 \leq i \leq t$;
- Each user can verify the correctness of the received share d_i by testing

$$g^{d_i} \bmod p \stackrel{?}{=} \prod_{j=1}^{t} f_j^{a_{ij}} \bmod p.$$

2.3.3 Pedersen's Scheme

Feldmans scheme has the limitation that $f(d)$ is broadcasted and, thus, the privacy of the secret depends on a computational assumption, on the hardness of inverting function f in particular. Pedersen [45] has proposed the following non-interactive and information-theoretically secure verifiable variant of Shamir's threshold secret sharing scheme for sharing a secret d:

- The primes p and q, and integers g and h are generated such that $q|(p-1)$, and $g, h \in \mathbb{Z}_p^*$ are elements of order q. All these numbers are public;
- The dealer chooses $r \in \mathbb{Z}_q$ randomly
- The dealer generates the polynomials $P(x) = d + P_1 x + \ldots + + P_{k-1} x^{t-1}$ and $Q(x) = r + Q_1 x + \ldots + Q_{k-1} x^{t-1}$ over Z_q and broadcasts $f_i = f(P_i, Q_i) = g^{P_i} h^{Q_i} \bmod p$, for all $1 \leq i \leq t-1$;
- The dealer also broadcasts $f_0 = f(d, r) = g^d h^r \bmod p$
- The dealer securely distributes $d_i = (P(i), Q(i))$ to the ith user, for all $1 \leq i \leq n$

- Each user can verify the correctness of the received share $d_i = (s_i, t_i)$ by testing

$$g^{s_i} h^{t_i} \bmod p \stackrel{?}{=} \prod_{j=0}^{t-1} f_j^{i^j} \bmod p.$$

So, with Pedersen's verifiable secret sharing scheme the security of the scheme do not depend on a computational assumption, since the power of the secret value is masked with another value. This approach however has its own limitation. If the dealer can solve the discrete logarithm problem, he can distribute incorrect shares.

2.3.4 Pedersen's Scheme with Blakley

We can apply Pedersen's idea to Blakley's SSS similar to Feldman's case and enhance Blakley's SSS with verifiability property in another way.

Pedersen generates an additional polynomial in Shamir's SSS to add verifiability. For Blakley's SSS, we can do the same thing by choosing a random point and distribute shares of this random point. To avoid problems of Feldman's verifiable SSS, a bivariate homomorphic function f is chosen.

So, during the dealing phase , the ith user will get his share $y_i \in \mathcal{F}$,

$$y_i = a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{it}x_t \tag{2.11}$$

of the secret point and $w_i \in \mathcal{F}$,

$$w_i = a_{i1}r_1 + a_{i2}r_2 + \ldots + a_{it}r_t \tag{2.12}$$

of the randomly chosen point $r = (r_1, r_2, \ldots, r_t)$. Unlike Feldman's verifiable SSS, we use a bivariate one-way function f which is homomorphic in the following sense: $f(x_1 + x_2, y_1 + y_2) = f(x_1, y_1)f(x_2, y_2)$. For verification, the values of $f(x_1, r_1), f(x_2, r_2), \ldots, f(x_t, r_t)$ are broadcasted by the dealer. Here the fact that the value of the function of f depends not only on the coordinates of the secret

point but also the coordinates of the random point, makes this scheme information theoretically secure as opposed to Feldman's scheme. A suitable choice for function f is

$$f(x, y) = g^x h^y \bmod p,$$

where the choice of the parameters g, h, p and q are the same as in the previous section. The whole protocol is as follows:

- The dealer securely distributes the share y_i to the ith user as in equation (2.11), and share r_i as in equation (2.12) for all $1 \leq i \leq n$;
- The prime numbers p and q are generated such that $q|(p-1)$, and $g, h \in \mathbb{Z}_p^*$ are elements of order q. All these numbers are public;
- The dealer broadcasts $f_i = g^{x_i} h^{r_i} \bmod p$, for all $1 \leq i \leq t$;
- Each user can verify the correctness of the received share y_i by testing

$$g^{y_i} h^{r_i} \bmod p \stackrel{?}{=} \prod_{j=1}^{t} f_j^{a_{ij}} \bmod p.$$

2.4 Proactive Secret Sharing

Secret sharing schemes assume long lived shares. However, over a long period of time, the protection provided by a secret sharing scheme may be insufficient. The security in a system that is exposed to attacks and break-ins might become exhausted; several faults may occur:

- Secrets can be revealed
- Shares can gradually be corrupted/compromised
- Hardware failures may occur, resulting in losing shares.

The goal of proactive security scheme is to prevent the adversary from learning the secret or from destroying it. In particular any group of t non-faulty shareholders should be able to reconstruct the secret whenever it is necessary.

The core properties of a proactive secret sharing scheme is as follows:

- It renews existing shares without changing the secret, so that previous exposures of shares will not damage the secret.
- It recover lost or corrupted shares without compromising the secrecy of the shares.

In this thesis we will content ourselves with renewing the shares.

After an update/renewal of the shares without changing the secret, all of the non-updated shares the attacker has accumulated become useless. An attacker can only recover the secret if he can find enough non-updated shares to reach the threshold. This situation should not happen because the players should have deleted their old shares. Additionally, an attacker cannot recover any information about the original secret from the update information, because they contain only random information.

2.4.1 Share Renewal with Dealer

Papers that deal with proactivity didn't assume the existence of a dealer. Since the share renewal idea is the same regardless of dealer's presence, we give the following for giving the basic idea of share renewal with the help of a dealer.

2.4.1.1 Share Renewal with Shamir SSS

If the dealer is still in place, share renewal is easy: For Shamir secret sharing scheme, the dealer generates a new random polynomial with constant term 0 and calculates for each remaining player a new ordered pair, where the x-coordinates

of the old and new pairs are the same. Each player then adds the old and new y-coordinates to each other and keeps the result as the new y-coordinate of the secret.

2.4.1.2 Share Renewal with Blakley SSS

We know that Blakley SSS is (+,+)-homomorphic. Thus, we can apply the same sharing zero idea to Blakley SSS: The dealer creates a random vector x, with its first coordinate set to 0. Then, he shares this secret point using the same matrix that is used to share the original secret and sends the new shares to the players. The players add their old and new shares and obtain a new share for the original secret.

2.4.2 Share Renewal without Dealer

If the dealer does not exist at the time of the update, the players have to generate the updates themselves. Since there is no dealer, the players take turns being the dealer and each player shares the (non)secret 0. Assuming the nonexistence of the dealer is more realistic, as the dealer may not be as long lived as the secrets are. The protocols given below are described by Herzberg et al. [25] for Shamir SSS.

2.4.2.1 Against Passive Attackers

With Shamir SSS, each player generates a random polynomial of degree $t-1$ passing through $(0,0)$. Then he sends the shares of this polynomial to other players. Each player, sums his non-updated share with the shares he received from other players and his share of his polynomial. Since each update polynomial passes from $(0,0)$, then after the update the new polynomial passes from the original secret point.

With Blakley SSS, we use the above approach. Each player generates a random point with first coordinate set to zero. Then he sends the shares of this point to every other player. Players finds their updated shares by summing all the received shares with the non-updated share.

2.4.2.2 Against Active Attackers

The above share-renewal protocols will not work if there is an active attacker among the players. An active attacker can destroy the secret in the above schemes, by sharing a polynomial not passing from $(0, 0)$ in Shamir SSS and by sharing a secret point with a non-zero first coordinate in Blakley SSS. Also, he can prevent inconsistent shares to destroy the secret. To prevent this situation, we need verifiability. In the Verifiable Secret Sharing Section, it is shown that how verifiability can be achieved with Shamir and Blakley secret sharing schemes.

The above protocols are modified as follows. Each player shares 0 verifiably. When a player receives a share from another player, he first checks the consistency of the received share. If the share is found to be inconsistent or the share is not a share of the secret 0, the player accuses the sender of the share and notifies other players. If no accusations occur, the players sum their received updates with their old shares to obtain their updated shares.

Below we give with Blakley SSS and Feldman's VSS how share renewal can be achieved. We assume that the matrix A is public. Assume f is a function chosen as $f : \mathbb{Z}_q \rightarrow \mathbb{Z}_p$, $f(x) = g^x \bmod p$, where p and q are odd primes such that $q|(p-1)$, and $g \in \mathbb{Z}_p^*$ an element of order q. Let d denote the secret, $d_i^{(k)}$ denote the share of player i after update k. Let y_i be the share vector of player i. Let $v[j]$ denote the jth element of vector v.

- Each player i plays the role of the dealer.
- Player i generates a random point $r_i = (r_{i1}, r_{i2}, \ldots, r_{it})$ where $r_{i1} = 0$ and computes

$$Ar_i = y_i. \tag{2.13}$$

- Player i broadcasts $f_{ij} = g^{r_{ij}} \bmod p$ for $1 \leq j \leq t$ and sends $y_i[j]$ to player j.
- If a player j is not blamed as a corrupt dealer each player i has a share $y_j[i]$ from player j.
- Let U be the set of uncorrupt players.
- Each player i updates his own share by performing

$$d_i^{(k)} = d_i^{(k-1)} + \sum_{j \in U} y_j[i]. \tag{2.14}$$

- The new verification values are set

$$f_i^{(k)} = f_i^{(k-1)} \prod_{j \in U} f_{ij} \tag{2.15}$$

  for all $1 \leq i \leq t$.

Chapter 3

Threshold RSA Signatures with Linear Secret Sharing Schemes

3.1 Introduction

In this chapter, we show how to generalize Shoup's ideas [49] to do function sharing with any linear SSS, and we give a robust threshold RSA signature scheme. A linear SSS, where the solution is based on solving a linear system, naturally requires computing inverses for reconstructing the secret. We show how to utilize such a system for function sharing while avoiding computation of inverses modulo $\phi(N)$ completely, where N is the RSA modulus.

We also discuss how this approach can be applied to other public key cryptosystems and show an example on the Paillier decryption function.

3.2 Sharing RSA Signature Computation

In this section, we describe our threshold RSA signature scheme which works with any linear SSS in general.

3.2.1 Setup

In the RSA setup phase, the RSA primes p and q are chosen as $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are large primes. The RSA modulus is computed as $N = pq$. Let $m = p'q'$. The public key e is chosen as a prime number, details of which will be explained in the next section. After choosing e, the private key d is computed such that $ed \equiv 1 \pmod{m}$. Then the dealer shares the private key d among n users using a linear threshold SSS described in Section 1.1.3.

The dealer also chooses v as a generator of Q_N, where Q_N is the subgroup of squares in $\mathbb{Z}_N^*$. He computes and broadcasts

$$v_i = v^{y_i} \in Q_N, \tag{3.1}$$

for $1 \leq i \leq n$, which are the verification keys to be used in the proofs of correctness of the partial signatures, where y_i is the secret share of user i.

3.2.2 Signing

Let $H(.)$ be a hash function mapping input messages to $\mathbb{Z}_N^*$ and let $w = H(M) \in \mathbb{Z}_N^*$ be the hashed message to be signed. Assume a coalition $\mathcal{S}$ of size t wants to obtain the signature $s = w^d \bmod N$.

3.2.2.1 Generating partial signatures

Let $\mathcal{S} = \{i_1, \ldots, i_t\}$ be the coalition of t users, forming the linear system

$$A_{\mathcal{S}} x = y_{\mathcal{S}}.$$

Let c_{ij} be the ij-th cofactor of matrix $A_{\mathcal{S}}$ and let $C_{\mathcal{S}}$ be the adjugate matrix,

$$C_{\mathcal{S}} = \begin{pmatrix} c_{11} & c_{21} & \ldots & c_{t1} \\ c_{12} & c_{22} & \ldots & c_{t2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1t} & c_{2t} & \ldots & c_{tt} \end{pmatrix}.$$

If we denote the determinant of $A_{\mathcal{S}}$ by $\Delta_{\mathcal{S}}$ we have,

$$A_{\mathcal{S}}C_{\mathcal{S}} = C_{\mathcal{S}}A_{\mathcal{S}} = \Delta_{\mathcal{S}}I_t, \tag{3.2}$$

where I_t denotes the $t \times t$ identity matrix.

For our scheme, each user $i \in \mathcal{S}$ computes his partial signature as

$$s_i = w^{2c_{i1}y_i} \bmod N. \tag{3.3}$$

3.2.2.2 Verifying partial signatures

Each user computes and publishes a proof of correctness for the verification of his partial signature. The proof of correctness of the partial signature of user i is a proof that the discrete logarithm of s_i^2 to the base

$$\tilde{s}_i = w^{4c_{i1}} \bmod N \tag{3.4}$$

is the same as the discrete logarithm of v_i to the base v. To prove this, a protocol by Shoup [49] which is a non-interactive version of Chaum and Pedersen's [7] interactive protocol is used:

Let $L(n)$ be the bit-length of n. Let H' be a hash function, whose output is an L_1-bit integer, where L_1 is a secondary security parameter. To construct the proof of correctness, user i chooses a random number $r \in \{0, 1, \ldots, 2^{L(N)+2L_1} - 1\}$, computes

$$\begin{aligned} v' &= v^r \bmod N, \\ s' &= \tilde{s}_i^r \bmod N, \\ D &= H'(v, \tilde{s}_i, v_i, s_i^2, v', s'), \\ \sigma &= y_i D + r. \end{aligned}$$

Then user i publishes his proof of correctness as (σ, D).

To verify this proof of correctness, one checks whether

$$D \stackrel{?}{=} H'(v, \tilde{s}, v_i, s_i^2, v^\sigma v_i^{-D}, \tilde{s}_i^\sigma s_i^{-2D}).$$

3.2.2.3 Combining partial signatures

To combine the partial signatures, we simply compute

$$\overline{s} = \prod_{i \in \mathcal{S}} s_i \bmod N. \tag{3.5}$$

Note that, by equation (3.2), we have

$$\overline{s} = w^{d\,\delta} \bmod N, \tag{3.6}$$

where

$$\delta = 2\,\Delta_{\mathcal{S}}. \tag{3.7}$$

Given that e is a prime number relatively prime to $\Delta_{\mathcal{S}}$, it is easy to compute the signature $s = w^d \bmod N$ from $\overline{s}$. Take

$$s = \overline{s}^a w^b \bmod N, \tag{3.8}$$

where a and b are integers such that

$$\delta a + eb = 1, \tag{3.9}$$

which can be obtained by the extended Euclidean algorithm on δ and e.

3.3 Solution of the Linear System

In a linear SSS, the private key is found by the solution of the linear system $A_{\mathcal{S}} x = y_{\mathcal{S}}$. However, this system may not have a unique solution over $\mathbb{Z}_{\phi(N)}$. If $\gcd(\Delta_{\mathcal{S}}, \phi(N)) > 1$, the matrix $A_{\mathcal{S}}$ will not have an inverse modulo $\phi(N)$, and the linear system will have many different solutions. Interestingly, our threshold signature scheme computes the correct signature in this case as well.

When $\gcd(\Delta_{\mathcal{S}}, \phi(N)) > 1$ and the linear system yields many different solutions for d, note that the value $\Delta_{\mathcal{S}} d$ is a fixed number for all these possible solutions, and is equal to

$$\Delta_{\mathcal{S}} d = \Sigma_i 2 c_{i1} y_i.$$

Hence, the incomplete signature

$$\begin{aligned} \overline{s} &= w^{\Sigma_i 2 c_{i1} y_i} \bmod N \\ &= w^{2\Delta_{\mathcal{S}} d} \bmod N \end{aligned}$$

is the same for every solution of the system $A_{\mathcal{S}} x = y_{\mathcal{S}}$.

Then the signature s is obtained from $\overline{s}$ as

$$s = \overline{s}^a w^b \bmod N,$$

where a and b are the integer solutions of $2\Delta_{\mathcal{S}} a + eb = 1$. Hence, the signature s is $w^d \bmod N$ for the right d value, computed according to the public key e.

3.4 Choosing e

The choice of e is critical in the setup phase because the solution depends on e and $\Delta_{\mathcal{S}}$ being relatively prime. To achieve this, we can either choose a special matrix whose determinant is known to be relatively prime to e, or choose e as a sufficiently large prime according to t and n so that the probability that $\Delta_{\mathcal{S}}$ is divisible by e will be negligible for any coalition $\mathcal{S}$.

3.4.1 Choosing e probabilistically

We can use a probabilistic approach for choosing e. The chosen value will depend on the value of t and n.

We want to calculate the probability that determinant of none of $A_{\mathcal{S}}$ matrices

will be divisible by e. We have

$$P\left(\begin{array}{l}\text{determinant of none}\\ \text{of } A_{\mathcal{S}} \text{ is divisible by } e\end{array}\right) = 1 - P\left(\begin{array}{l}\text{determinant of at least}\\ \text{one of } A_{\mathcal{S}} \text{ is divisible by } e\end{array}\right). \quad (3.10)$$

We also have

$$P\left(\begin{array}{l}\text{determinant of at least one}\\ \text{of } A_{\mathcal{S}} \text{ is divisible by } e\end{array}\right) = P\left(\bigcup_{\mathcal{S}} |A_{\mathcal{S}}| \text{ is divisible by } e\right). \quad (3.11)$$

For coalitions having common players, the events in the right hand side of equation (3.11) are not independent. We can use union bound (Boole's inequality) to bound the right hand side of equation of (3.11).

$$P\left(\bigcup_{\mathcal{S}} |A_{\mathcal{S}}| \text{ is divisible by } e\right) \leq \sum_{\mathcal{S}} P\left(|A_{\mathcal{S}}| \text{ is divisible by } e\right) \quad (3.12)$$

The probability that a certain random integer is divisible by a prime number e is $1/e$. In a (t, n) threshold scheme, there are $\binom{n}{t}$ different possible coalitions $\mathcal{S}$ of size t. By combining these facts with equations (3.10),(3.11) and inequality (3.12), we obtain

$$P\left(\begin{array}{l}\text{determinant of none}\\ \text{of } A_{\mathcal{S}} \text{ is divisible by } e\end{array}\right) \geq 1 - \frac{\binom{n}{t}}{e}. \quad (3.13)$$

If we take $e \gg \binom{n}{t}$, we have

$$P\left(\begin{array}{l}\text{determinant of none}\\ \text{of } A_{\mathcal{S}} \text{ is divisible by } e\end{array}\right) \approx 1. \quad (3.14)$$

For example, if we take $(t, n) = (10, 20)$ and take a 50 bit length prime e, the probability of any of the determinants not being relatively prime to e will be negligible. If we want to be certain about this, the dealer can check all $\binom{n}{t}$ determinants against e and choose another one if any of the determinants is not relatively prime to e. This is time consuming but will be done only once as a precomputation step by the dealer.

3.4.2 Bounding the determinant

Let a_{max} denote the maximum value in the matrix A. Then $t! \cdot a_{max}^t$ is clearly an upper bound on $|A_{\mathcal{S}}|$. We want

$$\phi(N) > e > \frac{t! \cdot a_{max}^t}{2} > |A_{\mathcal{S}}| \tag{3.15}$$

From this we obtain

$$\sqrt[t]{\frac{2e}{t!}} > a_{max} \tag{3.16}$$

For example if we take e as a 100 bit number and work with $(t, n) = (10, 20)$ we find that a_{max} should have about 8 bits. With $2^8 = 256$ possible values for the values of the matrix A we can find plenty of $n \times t$ matrices of rank t. With a slight increase in size of e, the value of a_{max} and the number of different matrices to choose from can be increased considerably.

3.4.3 Choosing a Vandermonde matrix as the coefficient matrix

A simple choice for the matrix A that enables us to guarantee that e will be relatively prime to the determinant of the coefficient matrix is to choose the rows of the matrix A as the rows of a Vandermonde matrix. Note that this is exactly the case for Shamir secret sharing. Then $A_{\mathcal{S}}$ will have the following form for a coalition $\mathcal{S}$ of size t:

$$A_{\mathcal{S}} = \begin{pmatrix} 1 & a_1 & a_1^2 & \dots & a_1^{t-1} \\ 1 & a_2 & a_2^2 & \dots & a_2^{t-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & a_t & a_t^2 & \dots & a_t^{t-1} \end{pmatrix}$$

The determinant of the Vandermonde matrix is nonzero, provided that no two rows are identical, and is given by the following formula:

$$|A_{\mathcal{S}}| = \prod_{i,j=1, i<j}^{t} (a_i - a_j) \tag{3.17}$$

Without loss of generality take $(a_1, a_2, \ldots, a_n) = (n, n-1, \ldots, 1)$. Obviously,

$$\prod_{i,j=1,i<j}^{t} (a_i - a_j) \mid \prod_{i,j=1,i<j}^{n} (a_i - a_j).$$

We also have,

$$\prod_{i,j=1,i<j}^{n} (a_i - a_j) = 1^{\alpha_1} 2^{\alpha_2} \ldots (n-1)^{\alpha_{n-1}} \tag{3.18}$$

for some $\alpha_1, \alpha_2, \ldots, \alpha_{n-1}$. Hence by choosing e as a prime greater than or equal to n we can guarantee that the determinant of any $A_{\mathcal{S}}$ will be relatively prime to e.

3.5 Security Analysis

Now we will prove that the proposed threshold RSA signature scheme is secure provided that the standard RSA signature is secure. We assume a static adversary model in the sense that the adversary controls exactly $t-1$ users and chooses them at the beginning of the attack. The adversary obtains all secret information of the corrupted users along with the public parameters of the system. She can control the actions of the corrupted users, asking for partial signatures of messages of her choice but cannot corrupt any other user in due course.

First we will analyze the proof of correctness. Then using this analysis we will prove that the proposed threshold signature scheme is secure.

3.5.1 Analysis of the Proof of Correctness

For generating and verifying the proof of correctness, the following properties hold:

3.5.1.1 Completeness

If the ith user is honest then the proof succeeds since

$$v^{\sigma} v_i^{-D} = v^{y_i D} v^r v_i^{-D} = v^r = v'$$

and

$$\tilde{s}_i^{\sigma} s_i^{-2D} = w^{4c_{i1}(y_i D + r)} w^{-4c_{i1} y_i D} = s^r = s'.$$

3.5.1.2 Soundness

To prove the soundness of the proof of correctness, we have to show that the adversary cannot construct a valid proof of correctness for an incorrect share, except with negligible probability. Let (σ, D) be a valid proof of correctness for a message w and partial signature s_i. We have $D = H'(v, \tilde{s}_i, v_i, s_i^2, v', s')$, where

$$\tilde{s}_i = w^{4c_{i1}}, v' = v^{\sigma} v_i^{-D}, s' = \tilde{s}_i^{\sigma} s_i^{-2D}.$$

Obviously $\tilde{s}_i$, v_i, s_i^2, v' and s' all lie in Q_n and we know that v is a generator of Q_n. So we have

$$\tilde{s}_i = v^{\alpha}, v_i = v^{y_i}, s_i^2 = v^{\beta}, v^{'} = v^{\gamma}, s' = v^{\mu},$$

for some integers $\alpha, \beta, \gamma, \mu$. From this we have,

$$\sigma - D y_i \equiv \gamma \pmod{m} \tag{3.19}$$

$$\sigma\alpha - D\beta \equiv \mu \pmod{m}. \tag{3.20}$$

From equations (3.19) and (3.20) we get,

$$D(\beta - y_i\alpha) \equiv \alpha\gamma - \mu \pmod{m}. \tag{3.21}$$

A share is correct, if and only if,

$$\beta \equiv y_i\alpha \pmod{m}. \tag{3.22}$$

If (3.22) does not hold, then it does not hold either mod p' or mod q' and so (3.21) uniquely determines D mod p' or D mod q'. But the distribution of D is uniform in the random oracle model, so this happens with negligible probability.

3.5.1.3 Zero Knowledge Simulatability

To prove zero knowledge simulatability, we will use the random oracle model for the hash function and construct a simple simulator that simulates the adversary's view without knowing the value y_i. When an uncorrupted user wants to create a proof (σ, D) for a message w and partial signature s_i, the simulator chooses $D \in \{0, \ldots, 2^{L_1} - 1\}$ and $\sigma \in \{0, \ldots, 2^{L(N)+2L_1} - 1\}$ at random and defines the value of the random oracle at $(v, \tilde{s}_i, v_i, s_i^2, v^\sigma v_i^{-D}, \tilde{s}_i^\sigma s_i^{-2D})$ to be D. Note that, the value of the random oracle is not defined at this point with all but negligible probability. When the adversary queries the oracle, if the value of the oracle was already set the simulator returns that value, otherwise it returns a random value. It is obvious that the output of this simulator is statistically indistinguishable from real output.

3.5.2 Security of the Proposed Signature Scheme

To reduce the problem of the security of the proposed threshold signature scheme to that of the standard RSA signature, the following proof constructs another simulator.

Theorem 1. *In the random oracle model for H', the proposed threshold signature scheme is a secure threshold signature scheme (robust and non-forgeable) under the static adversary model given that the standard RSA signature scheme is secure.*

Proof. We will simulate the threshold protocol with no information on the secret where the output of the simulator is indistinguishable in the adversary's view. Afterwards, we will show that the secrecy of the private key d is not disrupted by the values obtained by the adversary. Thus, if the threshold RSA scheme is not secure, i.e. an adversary who controls $t - 1$ users can forge signatures in the threshold scheme, one can use this simulator to forge a signature in the standard RSA signature scheme.

Let $i_1, \ldots, i_{t-1}$ be the set of corrupted players. To simulate the adversary's

view, we simply choose the y_{i_j} values belonging to the set of corrupted players at random from the set $\{0, \ldots, \lfloor N/4 \rfloor - 1\}$. The corrupted players' secret key shares are random numbers in the set $\{0, \ldots, m-1\}$. Once these values are chosen, the values y_i for the uncorrupted players are completely determined modulo m, but cannot easily be computed. However, given $w, s \in \mathbb{Z}_N^*$ with $s^e = w$, we can easily compute s_{i_t} for an uncorrupted user i_t as

$$s_{i_t} = w^{2c_{t1}y_{i_t}} = s^{2\Delta_{\mathcal{S}}} w^{-2\sum_{j=1}^{t-1} c_{j1}y_{i_j}}. \tag{3.23}$$

Note the dependence of $\Delta_{\mathcal{S}}$ and c_{j1} values on the coalition $\{i_1, \ldots, i_{t-1}, i_t\}$.

Using this technique, we can generate the values $v, v_1, \ldots, v_n$, and also generate any share s_i of a signature, given the standard RSA signature. These values produced by the simulator and the proof of correctness given in this section are computationally indistinguishable from the real ones. Hence, the threshold RSA signature scheme based on a linear SSS is secure given that the standard RSA signature scheme is secure. □

3.6 Application to Other Public Key Cryptosystems

So far, we investigated only how to share the RSA signature function by using a linear SSS. The same approach can also be used to share the RSA decryption function since the signature and decryption functions are mostly identical. Besides RSA, the proposed approach can also be used to share other public key cryptosystems (PKC) where the private key is used in the exponent, such as the ElGamal [16], Naccache-Stern [41] and the Paillier [44] decryption functions.

Below, as an example, we describe how our approach can be utilized for sharing the Paillier decryption and DSS signature functions. The Paillier decryption scheme works along the same lines as Fouque et al.'s extension [18] of Shoup's work to the Paillier cryptosystem. They used this threshold Paillier cryptosystem in e-voting and lottery schemes. Later, Lysanskaya and Peikert [37] improved

this work and obtained a threshold Paillier encryption scheme secure under the adaptive security model. Besides these Shamir based Paillier threshold systems, a provably secure threshold Paillier cryptosystem based on Asmuth-Bloom SSS was given in [34].

3.6.1 The Paillier Cryptosystem

The Paillier PKC is based on the properties of Carmichael function over $\mathbb{Z}_{N^2}$ where N is an RSA composite. Security of the cryptosystem is based on the intractability of computing discrete logarithms in $\mathbb{Z}_{N^2}$ without the Carmichael number $\lambda(N)$.

3.6.1.1 Key Generation

Let $N = pq$ where p and q are large prime integers. Let g be an arbitrary element from $\mathbb{Z}_{N^2}$ such that its order is a multiple of N. Let $\lambda = (p-1)(q-1)/2$ denote the the Carmichael function for N. The public and private keys are (N, g) and λ, respectively.

3.6.1.2 Encryption

Let w be the message to be encrypted. Choose a random $r \in \mathbb{Z}_{N^2}$ and compute the ciphertext as

$$s = g^w r^N \bmod N^2.$$

3.6.1.3 Decryption

The plaintext is obtained by

$$w = \frac{L(s^\lambda \bmod \mathbb{Z}_{N^2})}{L(g^\lambda \bmod \mathbb{Z}_{N^2})} \bmod N$$

where $L(x) = (x-1)/N$ for $x \equiv 1 \pmod N$.

Paillier proved that this scheme is semantically secure under the assumption that it is hard to detect whether a given random element in $\mathbb{Z}_{N^2}$ is an N-residue. The cryptosystem possesses the following homomorphic properties:

$$E(w_1 + w_2) = E(w_1).E(w_2)$$
$$E(k.w) = E(w)^k.$$

3.6.2 Sharing the Paillier Decryption Function

Since $\lambda(N)$ must be kept secret, the inverse computation problem is similar to the one we encountered while sharing the RSA signature function. Our threshold Paillier scheme is given below:

3.6.2.1 Key Generation

In the Paillier setup phase, choose two safe primes $p = 2p' + 1$ and $q = 2q' + 1$, where p' and q' are large primes and $\gcd(N, \varphi(N)) = 1$ for $N = pq$. Let $m = p'q'$. Let $\beta \in_R \mathbb{Z}_N^*$ and $(a, b) \in_R \mathbb{Z}_N \times \mathbb{Z}_N^*$. Compute

$$g = (1 + N)^a \times b^N \bmod N^2.$$

Share the private key $d = \beta m$ among n users with modulo Nm by using the linear SSS. Let

$$\theta = L\left(g^{\beta m}\right) = a\beta m \bmod N.$$

Set the public key as (g, N, θ). Choose v as a generator of Q_{N^2}, where Q_{N^2} is the cyclic group of squares in $\mathbb{Z}_{N^2}$. Compute the verification keys

$$v_i = v^{y_i} \in Q_{N^2}$$

for $1 \leq i \leq n$ as before.

3.6.2.2 Encryption

Let w be the message to be encrypted. Choose a random $r \in \mathbb{Z}_{N^2}$ and compute the ciphertext as $s = g^w r^N \bmod N^2$. Let $m = p'q'$.

3.6.2.3 Decryption

Let s be the ciphertext to be decrypted and $\mathcal{S} = \{i_1, \ldots, i_t\}$ denote a coalition of t users that will compute the plaintext together. Let $A_{\mathcal{S}}$ be the coalition matrix and $C_{\mathcal{S}}$ be the corresponding adjugate matrix, respectively, as in Section 3.2. Each member $i \in \mathcal{S}$ computes his partial value as

$$s_i = s^{2c_{i1}y_i} \bmod N^2$$

where c_{i1} is the ith element of the first row of $C_{\mathcal{S}}$. He also generates a proof of correctness which is used to prove that the discrete logarithm of s_i^2 to the base $\tilde{s} = w^{4c_{i1}}$ is the same as the discrete logarithm of v_i to the base v. Note that the proof is now working on a cyclic group of unknown order mN.

After the partial decryptions are obtained, the combining algorithm computes the plaintext

$$w = \frac{L\left(\prod_{i\in\mathcal{S}} s_i \bmod N^2\right)}{2\Delta_{\mathcal{S}}\theta} \bmod N.$$

Note that

$$\begin{aligned}
\prod_{i\in\mathcal{S}} s_i &\equiv s^{2\Delta_{\mathcal{S}}\beta m} \\
&\equiv g^{2\Delta_{\mathcal{S}}\beta m w} \\
&\equiv (1+N)^{2\Delta_{\mathcal{S}} a\beta m w} \\
&\equiv 1 + 2\Delta_{\mathcal{S}} a\beta m w N \\
&\equiv 1 + 2\Delta_{\mathcal{S}}\theta w N \pmod{N^2}.
\end{aligned}$$

3.6.3 Digital Signature Standard

The DSS is summarized below:

- *Key Generation Phase*: Let p and q be large prime numbers where $q|p-1$ and $g \in \mathbb{Z}_p^*$ be an element of order q. The private key $\alpha \in_R \mathbb{Z}_q^*$ is chosen randomly and the public key $\beta = g^{\alpha} \pmod p$ is computed.

- *Signing Phase*: The signer first chooses a random ephemeral key $k \in_R \mathbb{Z}_q^*$ and then computes the signature (r, s) where

$$r = (g^{k^{-1}} \bmod p) \bmod q$$

$$s = k(w + \alpha r) \bmod q$$

  for a hashed message $w \in \mathbb{Z}_q$.

- *Verification Phase*: The signature (r, s) is verified by checking

$$r \stackrel{?}{=} (g^{ws^{-1}} \beta^{rs^{-1}} \bmod p) \bmod q$$

  where s^{-1} is computed in $\mathbb{Z}_q^*$.

3.6.4 Sharing the DSS Signature Function

To obtain a threshold DSS scheme the dealer first generates the private key α and shares it among the users by (t, n). To obtain a DSS signature a random ephemeral key k has to be generated as well. This value will be generated using Joint Random Secret Sharing described in Chapter 2. Also, we have to use the $(\times, \times)$-homomorphism property of Blakley SSS (or linear SSS more generally) since the signing equations require the multiplications of the shares. Note that anyone can forge signatures if he knows k for a valid signature (r, s). Hence, the signing equations should be computed such that noone obtains k.

3.6.4.1 Key Generation

The parameters of DSS p, q, g, α and β are generated as described in the previous section. p, q, g and β are made public. The value of α is shared (t, n) using a linear SSS by the dealer. In the protocol description below, α_i denotes the share of player i for the private key α. The $(n \times t)$ matrix A of the LSSS is also made public by the dealer.

3.6.4.2 Computing Partial Signatures

Suppose a coalition $\mathcal{S}$ of t^2 players come together for signing. The players will run a joint random secret sharing protocol to generate the ephemeral key k. Also, another random number a is generated running the jrss protocol. Since the dealer makes the matrix A public in the key generation phase, the players use the same matrix A in the JRSS protocols used for generating a and k.

The size of the coalition $\mathcal{S}$ is t^2 because this coalition will be used to recover a secret $v = ak$ from the shares of the (t, n) shared secrets a and k, and this requires a coalition of at least t^2 players as described in Section 2.1. For other threshold operations a coalition of $\mathcal{S}' \subset \mathcal{S}$ of size t will be sufficient and will be used. In the following $A_{\mathcal{S}'}$ denotes the matrix consisting of rows of members of $\mathcal{S}'$ in the matrix A. Similarly, $(A \diamond A)_{\mathcal{S}}$ denotes the matrix consisting of rows of members of $\mathcal{S}$ in the matrix $(A \diamond A)$.

The steps of the partial signature computation are given below:

- Each player runs JRSS protocol twice for generating shares k and a. At the end, player i holds share k_i for joint random secret k and holds share a_i for joint random secret a for all $i \in \mathcal{S}$.
- Each player $i \in \mathcal{S}$ computes $v_i = a_i k_i$. The v_i values are the shares of the secret $v = ak$. Then v is constructed by the coalition $\mathcal{S}$ and v^{-1} is computed in $\mathbb{Z}_q^*$. Then each player in the coalition $\mathcal{S}$ computes

$$h = g^{v^{-1}} \bmod p. \tag{3.24}$$

  Note that during the reconstruction of v, the matrix $(A \diamond A)_{\mathcal{S}}$ is used.

- Each player $i \in \mathcal{S}'$ computes his partial signature of r as

$$r_i = h^{c_{i1} a_i} \bmod p, \tag{3.25}$$

  where c_{ij} is the ij-th cofactor of the matrix $A_{\mathcal{S}'}$ as described in Section 3.2.2.

3.6.4.3 Computing r

To combine the partial signatures we simply compute

$$\overline{r} = \prod_{i \in \mathcal{S}'} r_i \bmod p \tag{3.26}$$

By equation (3.2) we have

$$\overline{r} = h^{a\Delta s'} \bmod p \tag{3.27}$$

$$\overline{r} = g^{k^{-1}\Delta s'} \bmod p \tag{3.28}$$

The combiner computes $\Delta_{\mathcal{S}'}^{-1}$ in $\mathbb{Z}_q^*$. Note that here we take advantage of the fact that q is public. In the threshold RSA case, $\phi(N)$ had to be kept secret, so we couldn't compute the inverse this way. After this step r can finally be computed:

$$r = (\overline{r}^{\Delta_{s'}^{-1}} \bmod p) \bmod q. \tag{3.29}$$

After r is computed, it is made public to members of $\mathcal{S}$ not in $\mathcal{S}'$ so that it can be used during the computation of s which we describe next.

3.6.4.4 Computing s

Remember that $s = k(w + \alpha r) \bmod q$. Each member $i \in \mathcal{S}$ knows the values w, r and has shares k_i and α_i for the secrets k and α respectively.

We know that multiplication by a scalar and addition are linearity preserving operations. So, we can use the homomorphism properties of the LSSSs (described using Blakley SSS in Chapter 2) for the computation of $w + \alpha r$. But to be able to use the homomorphism property, we have to add with a share of w and not with w itself. Since matrix A is public, generating the shares of w is easy.

The steps of the computation now can be given as follows:

- Each player $i \in \mathcal{S}$ generates the column vector $v = (w, 0, \ldots, 0)$ of length t

and computes simply the dot product of row i of matrix A and vector v:

$$w_i = \sum_{j=1}^{t} A_{ij} v_j. \tag{3.30}$$

- Each player $i \in \mathcal{S}$ computes $(w_i + \alpha_i r) \bmod q$. Note that this value, is a (t, n) share of the secret value $w + \alpha r$. Each player i also has the share k_i of the secret k , so by multiplying the secrets

  $$s_i = k_i(w_i + \alpha_i r)$$

  is computed. Here, by the $(\times, \times)$-homomorphism property of LSSSs, s_i is a (t^2, n) share of the secret $s = k(w + \alpha r) \bmod q$.

- In the final step, the value s is recovered by the coalition $\mathcal{S}$ from the secrets s_i. This concludes the computation since the DSS signature (r, s) is found.

Chapter 4

Secret Sharing in General Access Structures

In this chapter we will be interested in more general access structures than threshold ones, multipartite access structures in particular. To investigate these access structures we will not use linear secret sharing schemes as Shamir or Blakley SSS. We will use Mignotte and Asmuth-Bloom secret sharing schemes which are based on Chinese remainder theorem (CRT) and are very similar to each other. An interesting question to answer in CRT-based secret sharing schemes is whether one can find a (generalized) Mignotte or Asmuth-Bloom sequence for an arbitrary access structure.

Galibus and Matveev [19] extended Mignotte's work [40] on threshold secret sharing on integers to work with polynomials. They proved that any access structure can be realized with their extended modular approach. Their method works with polynomials and constructs a generalized Mignotte sequence, but with slight modifications it can also be used to generate generalized Asmuth-Bloom sequences for integers. We implemented the modified Galibus and Matveev algorithm to work with integers. However, the method is not suitable for practical use as we will show.

We propose another more practical method of generating Asmuth-Bloom sequences for general (multipartite) access structures. We use the words general and multipartite interchangeably because every access structure is multipartite [24]. The proposed method increases the number of shares and hence decreases the information rate but the information rate is significantly better compared to the modified method of Galibus and Matveev for integers.

The organization of this chapter is as follows: First, we will define multipartite access structures which is our main focus in this chapter. Then we will describe Mignotte and Asmuth-Bloom secret sharing schemes and their generalizations. After that, the method of Galibus and Matveed and its modified version on integers will be given. After discussing the practical implications of these methods based on experiments we will present the new splitting based secret sharing scheme. We will also discuss how function sharing can be done with the new splitting-based secret sharing scheme. As an example, we will show how RSA signature computation can be done using the proposed secret sharing scheme.

4.1 Multipartite Access Structures

An interesting type of access structures is the multipartite access structure. Let $\mathcal{P} = 1, 2, \ldots, n$ be the set of players that are distributed into different disjoint classes $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, where $k \geq 2$. Every class $\mathcal{P}_i$ has n_i players, so the total number of players is $n = \sum_{i=1}^{k} n_i$. We say that an access structure is multipartite when the players in every class play the same role. The formal definition is as follows :

Definition 3. Let $\mathcal{P}$ be a set of players partitioned into $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$, i.e.

$$\mathcal{P} = \bigcup_{j=1}^{k} \mathcal{P}_j \tag{4.1}$$

$$\mathcal{P}_i \cap \mathcal{P}_j = \emptyset, \forall i \neq j. \tag{4.2}$$

Let $\sigma(S)$ be a renaming function on a player set $S \subset \mathcal{P}$ such that each renaming $i \mapsto j$ satisfies $i \in \mathcal{P}_p, j \in \mathcal{P}_p$ for $1 \leq p \leq k$. For an access structure Γ let $\sigma(\Gamma)$

be defined as:

$$\sigma(\Gamma) = \bigcup_{S \in \Gamma} \sigma(S). \tag{4.3}$$

An access structure Γ defined on the set of players $\mathcal{P}$ is multipartite with respect to partition $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k$ if $\sigma(\Gamma) = \Gamma$. In this case, we say that Γ is $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_k)$-multipartite or that Γ is $k-$multipartite.

Of course, any access structure defined on a set of n players is trivially an n-multipartite access structure, because we can always take $\mathcal{P}_1 = \{1\}, \ldots, \mathcal{P}_n = \{n\}$. But usually we want to consider the minimum possible number of classes. This optimal expression of an access structure as a multipartite one is obtained by taking $\mathcal{P}_1, \ldots, \mathcal{P}_k$ as the equivalence classes of the relation $\sim$, defined by

$$i \sim j \Leftrightarrow \tau_{ij}(\Gamma) = \Gamma,$$

where $\tau_{ij}(\Gamma)$ is the resulting access structure after transposition of players i and j in Γ. It is easy to prove that Γ is $(\mathcal{P}_1, \ldots, \mathcal{P}_k)$-multipartite, using the fact that any permutation can be expressed as a composition of transpositions.

Multipartite access structures can be represented by sets of vectors. Each vector in the access structure denotes a different group of authorized participants and the size of each vector is the number of partitions. For example, let $n = 12$ and $s = [|\mathcal{P}_1|\ |\mathcal{P}_2|\ |\mathcal{P}_3|] = [4\ 4\ 4]$, where n is the total number of players and s is a vector showing the sizes of each partition. Let $\Gamma = \{(2,3,4),(3,3,3)\}$. For this access structure the adversary structure $\overline{\Gamma}$ is found as $\overline{\Gamma} = \{(1,5,5),(5,2,5),(5,5,2),(2,3,3)\}$. In this case, a group of participants is authorized (i.e. belongs to the access structure Γ) if there are at least 2 players from partition X_1, at least 3 players from partition X_2, and at least 4 players from partition X_3; or at least 3 players from partition X_1, at least 3 players from partition X_2, and at least 3 players from partition X_3. Groups of players not satisfying the given condition are not authorized (i.e. belongs to the adversary structure $\overline{\Gamma}$).

A matrix can also be used to describe the multipartite access structures. The number of columns will be equal to the number of partitions of the player set.

Each row of the matrix denotes a group of authorized players. Let's denote this matrix with M_Γ. For the above example, obviously we have

$$M_\Gamma = \begin{pmatrix} 2 & 3 & 4 \\ 3 & 3 & 3 \end{pmatrix}.$$

4.2 Secret Sharing Schemes based on Chinese Remainder Theorem

4.2.1 Mignotte Secret Sharing

Mignotte's threshold secret sharing scheme [40], uses special sequences of integers, referred to as Mignotte sequences.

Definition 4. Let t and n be integers such that $2 \leq t \leq n$. A (t, n)-Mignotte sequence is a sequence of pairwise coprime positive integers $m_1 < m_2 < \ldots < m_n$ such that $\prod_{i=0}^{t-2} m_{n-i} < \prod_{i=1}^{t} m_i$.

The above relation is equivalent with

$$\max_{1 \leq i_1 < \ldots < i_{t-1} \leq n} \{m_{i_1} \ldots m_{i_{t-1}}\} < \min_{1 \leq i_1 < \ldots < i_t \leq n} \{m_{i_1} \ldots m_{i_t}\}.$$

Given a publicly known (t, n)-Mignotte sequence, the scheme works as follows:

- The secret d is chosen as a random integer such that $\beta < d < \alpha$, where $\alpha = \prod_{i=1}^{t} m_i$ and $\beta = \prod_{i=0}^{t-2} m_{n-i}$
- The shares d_i are chosen as $d_i = d \bmod m_i$, for all $1 \leq i \leq n$;
- Given t distinct shares $d_{i_1}, \ldots, d_{i_t}$, the secret d is recovered using the standard Chinese remainder theorem, as the unique solution modulo $m_{i_1} \ldots m_{i_t}$ of the system

$$\begin{aligned} x &\equiv d_{i_1} \bmod m_{i_1} \\ &\vdots \\ x &\equiv d_{i_t} \bmod m_{i_t}. \end{aligned}$$

Indeed, the secret d is an integer solution of the above system by the choice of the shares. Moreover, d lies in $\mathbb{Z}_{m_{i_1}\dots m_{i_t}}$ because $d < \alpha$. On the other hand, having only $t-1$ distinct shares $d_{i_1}, \dots, d_{i_{t-1}}$, we obtain only that

$$d \equiv x_0 \bmod m_{i_1} \dots m_{i_{t-1}},$$

where x_0 is the unique solution modulo $m_{i_1} \dots m_{i_{t-1}}$ of the resulting system. Therefore, in order to assure a reasonable level of security, (t, n)-Mignotte sequences with a large factor $\frac{\alpha-\beta}{\beta}$ must be chosen.

Iftene extended Mignotte's threshold scheme by introducing the generalized Mignotte sequences whose elements are not necessarily pairwise coprime [26].

Definition 5. Let t and n be integers such that $2 \leq t \leq n$. The sequence of integers $m_1, m_2, \dots, m_n$ is an (t, n) generalized Mignotte sequence if it satisfies the following condition:
$\max_{1\leq i_1<\dots<i_{t-1}\leq n}([m_{i_1}m_{i_2}\dots m_{i_{t-1}}]) < \min_{1\leq i_1<\dots<i_t\leq n}([m_{i_1}m_{i_2}\dots m_{i_t}])$, where $[m_{i_1}m_{i_2}\dots m_{i_t}]$ denotes the lowest common multiple (lcm) of $m_{i_1}, m_{i_2}, \dots, m_{i_t}$.

Obviously, every (t, n) Mignotte sequence is a (t, n)-generalized Mignotte sequence.

Generalized Mignotte scheme works similar to Mignotte's scheme by setting $\alpha = \min_{1\leq i_1<\dots<i_t\leq n}([m_{i_1}m_{i_2}\dots m_{i_t}])$ and $\beta = \max_{1\leq i_1<\dots<i_{t-1}\leq n}([m_{i_1}m_{i_2}\dots m_{i_{t-1}}])$. For secret recovery, the general variant of the Chinese remainder theorem must be used.

4.2.2 Asmuth-Bloom Secret Sharing Scheme

The Asmuth-Bloom secret sharing scheme [1] is similar to Mignotte's secret sharing scheme and also uses a special sequence of coprime integers $m_0 < m_1 < \dots < m_n$ such that

$$m_0 \cdot \prod_{i=0}^{t-2} m_{n-i} < \prod_{i=1}^{t} m_i.$$

Usually m_0 is kept as a secret and $m_1, \ldots, m_n$ are publicly known. The secret sharing scheme works as follows:

- The secret d is chosen as a random number from $\mathbb{Z}_{m_0}$.
- y is computed as $y = d + A \cdot m_0$, where A is an arbitrary integer such that $y \in \mathbb{Z}_{m_1 \ldots m_t}$
- The shares d_i are computed as $d_i = y \bmod m_i$ for all $1 \leq i \leq n$;
- Given t distinct shares $d_{i_1}, \ldots, d_{i_t}$, the secret d can be obtained as $d = x_0 \bmod m_0$, where x_0 is obtained using the standard variant of the Chinese remainder theorem, as the unique solution modulo $m_{i_1} \ldots m_{i_t}$ of the system

$$\begin{aligned} x &\equiv d_{i_1} \bmod m_{i_1} \\ &\vdots \\ x &\equiv d_{i_t} \bmod m_{i_t}. \end{aligned}$$

In the original Asmuth-Bloom SSS, the authors proposed an iterative process to solve the above system of equations. We describe a non-interactive and direct solution method given in [15]. This method is more suitable for function sharing in the sense that it does not require interaction between parties and has an additive structure which is convenient for exponentiations; and is used in function sharing schemes given in [34]. Suppose S is a coalition of t users gathered to construct the secret d.

1. Let $M_{S\setminus i}$ denote $\prod_{j \in S, j \neq i} m_j$ and $M'_{S,i}$ be the multiplicative inverse of $M_{S\setminus i}$ in $\mathbb{Z}_{m_i}$, i.e.
   $M_{S\setminus i} M'_{S,i} \equiv 1 \pmod{m_i}$.
   Then, every user i computes
   $u_i = y_i M_{S\setminus i} M'_{S,i} \bmod M_S$.
2. y is computed as
   $y = \sum_{i \in S} u_i \bmod M_S$.
3. The secret d is computed as
   $d = y \bmod m_0$.

The same procedure can be used in Mignotte SSS, except for the last step which is not needed.

Asmuth-Bloom sequences can also be generalized in a similar manner to Mignotte sequences [27]. The numbers in the sequence are not necessarily prime and satisfy the following condition:

$$m_0 \cdot \max_{1\leq i_1<\ldots<i_{t-1}\leq n}([m_{i_1}m_{i_2}\ldots m_{i_{t-1}}]) < \min_{1\leq i_1<\ldots<i_t\leq n}([m_{i_1}m_{i_2}\ldots m_{i_t}]).$$

We conclude this section by explaining 2 crucial properties of $(\lceil n/2\rceil, n)$ Asmuth-Bloom sequences. We will use these properties later in this chapter when we are presenting a new splitting-based secret sharing scheme.

Lemma 1. An $(\lceil n/2\rceil, n)$ Asmuth-Bloom sequence is also an (m, n) Asmuth-Bloom sequence for all m such that $1 < m < \lceil n/2\rceil$.

Proof. Let $t = \lceil n/2\rceil$. Consider the inequality:

$$\prod_{j=1}^{t} m_j > m_0 \prod_{j=1}^{t-1} m_{n-j+1} \tag{4.4}$$

Now for $k < t$ we have to show that

$$\prod_{j=1}^{k} m_j > m_0 \prod_{j=1}^{k-1} m_{n-j+1} \tag{4.5}$$

Comparing inequalities (4.4) and (4.5), we see that same number of numbers are removed from the products from each side of the inequality (4.4) to obtain inequality (4.5). But the numbers removed from the left side are strictly less than the numbers removed from the right side. So, inequality (4.5) holds. □

Lemma 2. An $(\lceil n/2\rceil, n)$ Asmuth-Bloom sequence is also a (k, n) Asmuth-Bloom sequence for all k such that $\lceil n/2\rceil < k \leq n-1$.

Proof. The proof is similar to the proof of the previous lemma. Now for $m > t$ consider the inequality,

$$\prod_{j=1}^{m} m_j > m_0 \prod_{j=1}^{m-1} m_{n-j+1}. \tag{4.6}$$

Here, the same number of numbers are added to the product in each side of inequality (4.4) to obtain inequality (4.6). But, since $m > t$, the numbers inserted to the left side of the inequality (4.4) are strictly greater than the numbers inserted to the right side. So, inequality (4.6) holds as well. □

4.3 Method of Galibus and Matveev

The method of Galibus and Matveev produces a generalized Mignotte sequence for an arbitrary access structure for the ring of polynomials.

In the following algorithm, n denotes the total number of players, $p_i(x)$ denotes the polynomial of user i for all $1 \leq i \leq n$. Γ denotes the access structure.

Algorithm 1 Method of Galibus and Matveev

1: **for all** maximal unauthorized subset $A \notin \Gamma$ **do**
2: {Define polynomial $pr_i(x)$ for all i for some $r(x)$}
3: **for all** $i \notin A$ **do**
4: $pr_i(x) = p_i(x) \cdot r(x)$
5: **end for**
6: **for all** $i \in A$ **do**
7: $pr_i(x) = p_i(x)$ for all $i \in A$
8: **end for**
9: Find irreducible monic $r(x)$ such that
10: $deg[p_i(x), i \in A] < deg[pr_i(x), i \in C]$ for all authorized subset C
11: **for all** $i \notin A$ **do**
12: Set $p_i(x) = p_i(x) \cdot r(x)$.
13: **end for**
14: **end for**

Initially each polynomial is set to 1. Then, at each iteration of the algorithm, a maximal unauthorized subset $A \notin \Gamma$ (i.e. there exists no unauthorized subset $B \notin \Gamma$ such that $A \subset B$) is processed. The polynomials (i.e. moduli) of all participants not belonging to A are multiplied by some irreducible monic $r(x) \in GF(q)[x]$, where $GF(q)[x]$ denotes the polynomial ring and $GF(q)$ is the Galois field of prime order q. $r(x)$ is chosen such that after all multiplications the

following condition is fulfilled:

$$deg[p_i(x), i \in A] < deg[p_i(x), i \in C],$$

for any authorized subset C. This condition holds true, as Galibus and Matveev show, after some other maximal unauthorized subset is chosen and the same operation is performed for the irreducible monic $r_2(x) \neq r(x)$ at a subsequent iteration of the algorithm. Thus, after repeating the operation for all maximal unauthorized subsets the realization of the Mignotte sequence for the given access structure Γ is obtained.

Our modifications to the method of Galibus and Matveev for integers are as follows: We want our modified algorithm to produce generalized Asmuth-Bloom sequences instead of generalized Mignotte sequences, so p_0 is chosen as a random prime number of specified bit length in the beginning. Then at each step of the algorithm the following condition is checked:

$$p_0 \cdot [p_i, i \in A]) < \min_C([p_i, i \in C]), \tag{4.7}$$

for the current unauthorized subset A and for all authorized subsets. Here C is an authorized subset ($C \in \Gamma$), and the lowest common multiple of p_is in C are minimum compared to other authorized subsets. If the condition is not satisfied a prime number p_c is chosen such that $(p_0 \cdot [p_i, i \in A]) < (p_i \cdot p_c, i \in C])$ for any authorized subset C. After finding such p_c, all moduli not belonging to participants from A are multiplied with p_c. Thus, as in the original algorithm of Galibus and Matveev, after this operation is repeated for every maximal unauthorized subset and the desired sequence (a generalized Asmuth-Bloom sequence in this case) is generated. The modified version of the algorithm for integers is given in Algorithm 2.

This algorithm allows us to generate generalized Asmuth-Bloom sequences for arbitrary access structures. Nevertheless, our experiments indicate that the algorithm only has theoretical importance. Our experiments with the algorithm show that the generated numbers are very big and impractical to use even for very small access structures.

Algorithm 2 Modified algorithm for integers

```
Generate prime p_0 of specified bit length
for all maximal unauthorized subset A ∉ Γ do
    Find M = min_C([p_i, i ∈ C]) where C ∈ Γ
    if (p_0 · [p_i, i ∈ A]) < M then
        Continue
    else
        Find prime p_c s.t. p_0 · [p_i, i ∈ A] < [p_i · p_c, i ∈ C] for all C ∈ Γ
        for all i ∉ A do
            Set p_i = p_i · p_c.
        end for
    end if
end for
```

Table 4.1 shows the average and maximum bit lengths of the generated generalized Asmuth-Bloom sequences for 4 different access structures and for p_0 of length $32, 64, 128, 256$ and 512 bits. The top row of the table shows the different access structures. In each of the access structures, there are 2 partitions. The vector s shows the sizes of the partitions. Each row of the matrix A shows a different group of authorized participants. As seen from the table average and maximum bit lengths increase linearly with bit length of p_0. The values with an * next to them are obtained without waiting for the algorithm to stop and the shown maximum and average values are the values at the time the algorithm is stopped. So, their actual values will be higher. Note that, the algorithm used to generate generalized Asmuth-Bloom sequences is a deterministic algorithm and obtained average and maximum values are obtained by running the algorithm just once, not from multiple runs.

Clearly the obtained maximum and average bit lengths indicate that the algorithm is not of practical value. We can say that it only has theoretical importance. The results obtained when the number of partitions is set to 3 is even worse as the required number of multiplications increase significantly.

Bits	A = {(2 3)} s = (4 4)	A = {(2 3),(3 2)} s = (4 4)	A = {(2 3)} s = (5 5)	A = {(2 3),(3 2)} s = (5 5)
32	Max : 652 Avg : 327	Max : 7162 Avg : 5612	Max : 1489 Avg : 726	Max : 53631 Avg : 42558
64	Max : 1324 Avg : 663	Max : 14554 Avg : 11404	Max : 3025 Avg : 1475	Max : 91729 Avg : 73031
128	Max : 2668 Avg : 1336	Max : 29338 Avg : 22988	Max : 6097 Avg : 2973	Max : 219711 Avg : 174350
256	Max : 5356 Avg: 2679	Max : 58906 Avg: 46156	Max : 12241 Avg: 5968	Max : 160455* Avg: 103670*
512	Max : 10732 Avg: 5367	Max : 56722 Avg: 92492	Max : 24529 Avg: 11958	Max : 214456* Avg: 202365*

Table 4.1: Maximum and average bit lengths of generalized Asmuth-Bloom sequences generated by the modified Galibus and Matveev algorithm

4.4 The New Method Based on Splitting

For generating Mignotte or Asmuth-Bloom sequences for arbitrary access structures another approach is possible as we will show next. We describe the method using the multipartite access structures similar to Galibus and Matveev's method. Unlike the previously described method of Galibus and Matveev, and its modified version working on integers, the new method generates more than one sequence.

The idea of the new method is quite simple: For sharing a secret d, for each row r of the access structure matrix M_Γ, we generate $d_{\mathcal{P}_1}, d_{\mathcal{P}_2}, \ldots, d_{\mathcal{P}_k}$ such that

$$d_{\mathcal{P}_1} + d_{\mathcal{P}_2} + \ldots + d_{\mathcal{P}_k} = d \bmod m_0$$

and share $d_{\mathcal{P}_1}$ for participants in partition 1, share $d_{\mathcal{P}_2}$ in partition 2 and so on. So, if there are m rows in the access structure matrix M_Γ, each player ends up with m shares. Obviously, this idea is not specific to any secret sharing scheme. However, here we focus only on using this method with Asmuth-Bloom secret sharing scheme. Note that it suffices to generate one Asmuth-Bloom sequence for each partition. Moreover, if there are partitions with the same size, the same Asmuth-Bloom sequence can be reused. In the extreme case where all partitions have the same size, just one Asmuth-Bloom sequence is generated and used in all partitions. After briefly describing the method, we next formally give the

algorithm for Asmuth-Bloom secret sharing scheme.

In the following assume there are k partitions of the participants and n_i denotes the number of participants in partition i. The access structure Γ is a multipartite access structure and the access structure matrix M_Γ consists of m rows and k columns, which is equal to the number of partitions.

For each partition we generate an Asmuth-Bloom sequence with common m_0. More specifically, for partition j we generate an $(\lceil n_j/2 \rceil, n_j)$ Asmuth-Bloom sequence. Normally, one would be required to generate an Asmuth-Bloom sequence for each different pair (a_{ij}, n_j), where a_{ij} denotes the element of M_Γ at row i and column j. However, as we showed in Lemmas 1 and 2, an $(\lceil n/2 \rceil, n)$ Asmuth-Bloom sequence is also an (m, n) Asmuth-Bloom sequence for all $m < \lceil n/2 \rceil$ and also a (k, n) Asmuth-Bloom sequence for all $k > \lceil n/2 \rceil$. So, the generated $(\lceil n_j/2 \rceil, n_j)$ sequence can be used for each different value of a_{ij} for all i.

Algorithm 3 Splitting-Based Secret Sharing Scheme for General Access Structures based on Asmuth-Bloom SSS

for all i such that $1 \leq i \leq k$ **do**
  Set $t = \lceil n_i/2 \rceil$
  Generate $m_0 < m_{i_1} < m_{i_2} < \ldots < m_{i_{n_i}}$
  such that $\prod_{j=1}^{t} m_{ij} > m_0 \prod_{j=1}^{t-1} m_{i_{n_i-j+1}}$
end for
for all row i of matrix M_Γ **do**
  generate $d_{\mathcal{P}_1}, d_{\mathcal{P}_2}, \ldots, d_{\mathcal{P}_k}$ such that $d_{\mathcal{P}_1} + d_{\mathcal{P}_2} + \ldots + d_{\mathcal{P}_k} \bmod m_0 = d$
  for all i such that $1 \leq i \leq k$ **do**
    Share $d_{\mathcal{P}_i}$ in partition i
  end for
end for

Algorithm 4 Sharing of $d_{\mathcal{P}_i}$ in partition i

Let $M_i = \prod_{j=1}^{n_i} m_{ij}$
Choose A_i as a random integer such that $0 \leq y_i \leq M_i$
where $y_i = d_{\mathcal{P}_i} + A_i m_0$
for all j such that $1 \leq j \leq n_i$ **do**
  $y_{ij} = y_i \bmod m_{ij}$ {Generate share of player j of partition i}
end for

Note that for each vector in the access structure we have to regenerate the

sum $d = \sum_{j=1}^{k} \mathcal{P}_j \bmod m_0$. Otherwise, i.e. if we use the same sum more than once, unauthorized groups can find the secret by combining their shares given for different vectors in the access structure. As an example consider a secret sharing scheme with 10 players. Let the access structure Γ be defined as $\Gamma = \{(5,3),(3,5)\}$. Obviously subsets of $(4,4)$ are unauthorized, but a group of players having 4 players from each partition can find S_2 from the shares of given for vector $(5,3)$ and can find S_1 from the shares given for vector $(3,5)$. Then combining these would yield the secret. Obviously, this situation has to be avoided.

The main drawback of the proposed secret sharing scheme is the fact that it reduces the information rate by giving each partition more than one share. Since, we cannot find one Asmuth-Bloom sequence that suits our needs this is a price we have to pay. In [20] it is shown that not all access structures can be realized using pairwise coprime moduli. They call access structures that can be realized using pairwise coprime moduli as elementary access structures and give the characterization of these access structures with the help of the monomial cut concept. On the other hand, the modification of the same authors' algorithm to the ring of integers which produces generalized Asmuth-Bloom sequences produce very big numbers prohibiting the algorithms' practical value. So, even though the secret sharing scheme proposed here decreases the information rate, it compares favorably to its alternatives.

The splitting-based secret sharing scheme suits naturally to be used with function sharing schemes. In the next subsection we show how RSA signatures can be computed when the secret is shared using the proposed splitting-based secret sharing scheme.

4.4.1 Threshold RSA signature scheme with the proposed secret sharing scheme

In [34], Kaya et al. investigated how threshold cryptography can be conducted with the Asmuth-Bloom secret sharing scheme and presented function sharing schemes for RSA, ElGamal and Paillier cryptosystems. Here, we will follow their

approach for realizing RSA signature computation using the proposed splitting-based secret sharing scheme as the underlying secret sharing scheme. When an authorized group of participants come together, participants from each partition can compute their partial signatures as in [34]. Then each partial signature coming from each partition is multiplied to generate the incomplete signature. The given method for RSA signature computation in [34] first calculates an incorrect signature and then this incorrect signature is corrected by using a procedure which utilizes the public keys of the system. Here, the required housekeeping is a little more complicated. The computed partial signatures of each partition will be incorrect. It is not possible to correct these partial signatures at this step since there is no information to be used for correction (i.e. no public keys or partial public keys for each partition). However, we can use the correction procedure after the partial signatures are combined. In the following we give the steps of RSA signature computation when the RSA private d is shared using the proposed splitting-based secret sharing method.

- *Setup:* In the RSA setup phase, choose the RSA primes $p = 2p^{'} + 1$ and $q = 2q^{'} + 1$ where $p^{'}$ and $q^{'}$ are also large random primes. $N = pq$ is computed and the public key e and private key d are chosen from $\mathbb{Z}^{*}_{\phi(N)}$ where $ed \equiv 1 \pmod{\phi(N)}$. Use the above secret sharing scheme for sharing d with $m_0 = \phi(N) = 4p^{'}q^{'}$. Note that each participant will receive multiple shares, the number of which is equal to the number of rows of the access structure matrix.

- *Signing:* Let w be the hashed message to be signed and suppose the range of the hash function is $\mathbb{Z}^{*}_{N}$. Let there be p partitions. Assume a coalition S consisting of t_1 players from partition 1 , ..., t_k players from partition k wants to obtain the signature $s = w^d \bmod N$. Let S_p denote the subset of S having players only from partition p. For this coalition to be authorized there has to be at least one row r in the access structure matrix such that $r \geq (t_1, \ldots, t_k)$. Without loss of generality assume assume $r = (t_1, \ldots, t_k)$, we can always have this by not considering shares of players coming from partitions already having enough number of players.

- *Generating partial results*:
  Each user $i \in S_j$ computes
  $u_i = y_i M'_{S_j,i} M_{S_j \setminus i} \bmod M_{S_j}$,
  $s_i = w^{u_i} \bmod N$.
- *Combining partial results:*
  The incomplete signature $\overline{s}$ is obtained by combining the s_i values in each partition $\mathcal{P}_j$ and then multiplying the results.
  $\overline{s} = \prod_{j=1}^{k} \prod_{i \in \mathcal{P}_j} s_i \bmod N$.
- *Correction:*
  Let $\kappa_j = w^{-M_{S_j}}$ be the corrector of partition j. The incomplete signature can be corrected by trying

$$(\overline{s}\kappa_1^{j_1}\kappa_2^{j_2}\ldots\kappa_p^{j_p})^e = \overline{s}^e(\kappa_1^e)^{j_1}(\kappa_2^e)^{j_2}\ldots(\kappa_p^e)^{j_p} \stackrel{?}{\equiv} w \pmod{N} \tag{4.8}$$

  for

$$\begin{aligned} 0 \leq \quad & j_1 \quad < t_1, \\ 0 \leq \quad & j_2 \quad < t_2, \\ & \vdots \\ 0 \leq \quad & j_p \quad < t_p. \end{aligned}$$

  Then the signature is computed by

$$s = \overline{s}\kappa_1^{\delta_1}\kappa_2^{\delta_2}\ldots\kappa_p^{\delta_p}, \tag{4.9}$$

  where $\delta_1, \ldots, \delta_p$ denote the values of $j_1, \ldots, j_p$ that satisfy (4.8). For details of why this correction procedure is needed and why it works, the reader may consult [34].

- *Verification:* Verification is the same as the standard RSA signature verification.

Chapter 5

Function Sharing Implementations

In this chapter we describe our implementations of several secret sharing and function sharing schemes. These implementations are carried out in order to verify the correctness of the proposed schemes. Also, observing the efficiency in terms of computation time was our aim.

We implemented the RSA signature computation based on linear secret sharing schemes given in Chapter 3. The basic version of the proposed function sharing scheme (without robustness feature) appeared in [5]. The robust version as given in this thesis along with application to other public key cryptosystems appeared in [4]. We implemented only the basic version, the robust version is not implemented yet. As a result of the implementation, we observed that the computation of the cofactors is the most time consuming step of the solution procedure. Since we only need the first row of the adjugate matrix (transpose of the cofactor matrix), we didn't compute all the cofactors in our implementation.

In addition to the function sharing schemes given in Chapter 3, the splitting-based secret sharing scheme described in Chapter 4 is implemented. Also, the modified version of Galibus and Matveev algorithm is implemented to get the average and maximum bit lengths of the generated generalized Asmuth-Bloom

sequences. The results of our experiments with this algorithm appeared in Chapter 4. As the bit lengths given in Table 4.1 indicate, this algorithm is not practical.

Additionally, several other function sharing schemes that are recently proposed are implemented. The threshold RSA signature computation, threshold Paillier decryption and threshold ElGamal decryption functions using Asmuth-Bloom secret sharing scheme was given in [34]. These function sharing schemes given in [34] are robust. However, we didn't implement the robustness features of these schemes. Another function sharing scheme given in [35] shows how threshold Digital Signature Standard (DSS) signatures can be computed with Asmuth-Bloom secret sharing scheme as the underlying secret sharing scheme. This signature scheme is also implemented.

Asmuth-Bloom based function sharing schemes are implemented using Java programming language. The threshold RSA signature scheme using linear secret sharing schemes given in Chapter 3 is implemented in Matlab. The splitting-based secret sharing scheme and modified Galibus and Matveev algorithms are also implemented in Matlab. In Matlab implementations, Matlab—Java interface is utilized to be able to use BigInteger class from java.math package.

The developed function sharing schemes share the same modules for secret sharing. Several command line parameters are defined. These parameters are: $-n$ for the total number of participants, $-t$ for the threshold value and $-S$ for the coalition. Upon program start the mode of operation is selected such as RSA signature with linear SSS, Paillier decryption with Asmuth-Bloom SSS etc.

Chapter 6

Conclusion and Future Work

In this thesis, we investigated some aspects of the secret sharing and function sharing problems. We focused on Blakley SSS which is a linear SSS. In secret sharing and function sharing literature, many extensions to and solutions based on Shamir SSS can be found. After investigating homomorphic properties of Blakley SSS, we used these properties to extend Blakley secret sharing in various ways. We examined homomorphic properties of Blakley SSS and used these properties as building blocks in the extensions to the Blakley SSS. We showed that how verifiability, proactivity features can be achieved with Blakley SSS. Another extension was showing how secret sharing without a dealer can be done with Blakley SSS.

We also showed how some function sharing schemes can be realized when a linear SSS is used as the underlying secret sharing scheme. We presented a robust RSA threshold signature scheme based on a linear SSS. The proposed signature scheme generalizes Shoup's threshold RSA signature based on Shamir secret sharing, and is as efficient and practical as Shoup's scheme.

Besides RSA, we showed that this approach can be extended to other public key cryptosystems where the private key is used in the exponent. We demonstrated how Paillier decryption function can be shared by this approach. We also

showed threshold DSS signature computation using the same approach. Threshold DSS signature computations made use of the mentioned homomorphic properties of linear secret sharing schemes. ElGamal and Naccache-Stern knapsack cryptosystems are some other cryptosystems that can benefit from the proposed solution.

Another secret sharing scheme of our interest was Asmuth-Bloom secret sharing. We investigated ways of finding Asmuth-Bloom sequences for arbitrary access structures. We modified a method proposed in the literature for this purpose and implemented it to see it in practice. The numbers produced by this algorithm turned out to be impractical to use. Because of the impracticality of this algorithm, at the cost of reducing the information rate, we proposed a new method for enabling Asmuth-Bloom secret sharing for general access structures. The proposed secret sharing scheme compares favorably to the mentioned modified algorithm in terms of information rate. Function sharing schemes can be implemented with the proposed secret sharing scheme, and we showed an example threshold RSA signature computation using the proposed secret sharing scheme.

For future work, obtaining more compact Asmuth-Bloom sequences and hence improving the information rate can be considered. Also, ways of improving the efficiency of the proposed function sharing scheme can be investigated. This can be achieved, for example by finding a way of applying the corrector functions locally at each partition instead of globally (for the entire coalition $\mathcal{S}$) as proposed. Another issue to consider can be proposing other function sharing schemes based on the proposed secret sharing scheme.

Finally, we implemented the proposed secret and function sharing schemes in addition to some other recently proposed function sharing schemes. The implementations are not done to be used in real world secret and/or function sharing schemes, but to show the correctness and measure the efficiency of the proposed methods.

Bibliography

[1] C. Asmuth and J. Bloom. A modular approach to key safeguarding. *IEEE Transactions on Information Theory*, 29(2):208–210, 1983.

[2] J. C. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *CRYPTO '88: Proceedings of the 8th Annual International Cryptology Conference on Advances in Cryptology*, volume 403 of *LNCS*, pages 27–35. Springer-Verlag, 1990.

[3] G. Blakley. Safeguarding cryptographic keys. In *Proc. of AFIPS National Computer Conference*, 1979.

[4] I. N. Bozkurt, K. Kaya, and A. A. Selçuk. Practical threshold signatures with linear secret sharing schemes. In *AFRICACRYPT '09: Proceedings of the 2nd International Conference on Cryptology in Africa*, volume 5580 of *LNCS*, pages 167–178, Berlin, Heidelberg, 2009. Springer-Verlag.

[5] I. N. Bozkurt, K. Kaya, A. A. Selçuk, and A. M. Güloğlu. Threshold cryptography based on Blakley secret sharing. In *3rd Information Security and Cryptology Conference*, 2008.

[6] E. F. Brickell. Some ideal secret sharing schemes. In *EUROCRYPT '89: Proceedings of the workshop on the theory and application of cryptographic techniques on Advances in cryptology*, volume 434 of *LNCS*, pages 468–475, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[7] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Proc. of CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer-Verlag, 1992.

[8] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *SFCS '85: Proceedings of the 26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, Washington, DC, USA, 1985. IEEE Computer Society.

[9] J. Cohen Benaloh. Secret sharing homomorphisms: keeping shares of a secret secret. In *Proceedings on Advances in cryptology—CRYPTO '86*, volume 263 of *LNCS*, pages 251–260, London, UK, 1987. Springer-Verlag.

[10] R. Cramer, I. Damgard, and U. Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *EUROCRYPT'2000*, volume 1807 of *LNCS*, pages 316–334. Springer-Verlag, 2000.

[11] Y. Desmedt. Some recent research aspects of threshold cryptography. In *Proc. of ISW '97, 1st International Information Security Workshop*, volume 1196 of *LNCS*, pages 158–173. Springer-Verlag, 1997.

[12] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In *Proc. of CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer-Verlag, 1990.

[13] Y. Desmedt and Y. Frankel. Shared generation of authenticators and signatures. In *Proc. of CRYPTO'91*, volume 576 of *LNCS*, pages 457–469. Springer-Verlag, 1992.

[14] Y. Desmedt and Y. Frankel. Homomorphic zero-knowledge threshold schemes over any finite abelian group. *SIAM Journal on Discrete Mathematics*, 7(4):667–679, 1994.

[15] C. Ding, D. Pei, and A. Salomaa. *Chinese Remainder Theorem: Applications in Computing, Coding, Cryptography*. World Scientific, 1996.

[16] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[17] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS'87: Proc. of the IEEE Symposium on Foundations of Computer Science*, pages 427–437. IEEE Press, 1987.

[18] P. A. Fouque, G. Poupard, and J. Stern. Sharing decryption in the context of voting or lotteries. In *Proc. of FC 2000, 4th International Conference on Financial Cryptography*, volume 1962 of *LNCS*, pages 90–104. Springer-Verlag, 2001.

[19] T. Galibus and G. Matveev. Generalized Mignotte's sequences over polynomial rings. *Electronic Notes on Theoretical Computer Science*, 186:43–48, 2007.

[20] T. Galibus, G. Matveev, and N. Shenets. Some structural and security properties of the modular secret sharing. In *Proc. of SYNASC'2008*, 2008.

[21] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust and efficient sharing of RSA functions. In *Advances in Cryptology CRYPTO '96*, volume 1109 of *LNCS*, pages 157–172, 1996.

[22] R. Gennaro, S.Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In *Eurocrypt'96: Advances in Cryptology*, volume LNCS 1070, pages 157–172, Berlin/New York. Springer-Verlag.

[23] R. Gennaro, S.Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. *Information and Computation*, 164(1):54–84, 2001.

[24] J. Herranz and G. Saez. New results on multipartite access structures. *IEE Proceedings of Information Security*, 153(4):153–162, 2006.

[25] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive secret sharing. In *Proc. of CRYPTO'95*, volume 963 of *LNCS*, pages 339–352. Springer-Verlag, 1995.

[26] S. Iftene. A generalization of Mignotte's secret sharing scheme. In *Proceedings of the 6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, September 2004*, pages 196–201. Mirton Publishing House, 2004.

[27] S. Iftene. *Secret Sharing Schemes with Applications in Security Protocols*. PhD thesis, University Alexandru Ioan Cuza of Iaşi, Faculty of Computer Science, 2007.

[28] I. Ingemarsson and G. J. Simmons. A protocol to set up shared secret schemes without the assistance of a mutually trusted party. In *EUROCRYPT'91: Advances in Cryptology*, volume 473 of *LNCS*, pages 266–282. Springer-Verlag, 1990.

[29] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings of the IEEE Global Telecommunications Conference, Globecom '87*, pages 99–102. IEEE Press, 1987.

[30] W. Jackson, K. Martin, and C. O'Keefe. Mutually trusted authority free secret sharing schemes. *Journal of Cryptology*, 10:261–289, 1997.

[31] M. Karchmer and A. Wigderson. On span programs. In *Proc. of 8-th Annual Structure in Complexity Theory Conference*, pages 102–111. IEEE Computer Society Press, 1993.

[32] E. D. Karnin, J. W. Greene, and M. E. Hellman. On secret sharing systems. *IEEE Transactions on Information Theory*, 29:35–41, 1983.

[33] K. Kaya. *Threshold Cryptography With Chinese Remainder Theorem*. PhD thesis, Bilkent University, Department of Computer Engineering, 2009.

[34] K. Kaya and A. A. Selçuk. Threshold cryptography based on Asmuth-Bloom secret sharing. *Information Sciences*, 177(19):4148–4160, 2007.

[35] K. Kaya and A. A. Selçuk. Sharing DSS by the Chinese Remainder Theorem. Cryptology ePrint Archive, Report 2008/483, 2008.

[36] N. Koblitz. *A Course in Number Theory and Cryptology*. Springer-Verlag, 1987.

[37] A. Lysyanskaya and C. Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In *Proc. of ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer-Verlag, 2001.

[38] R. J. McEliece and D. V. Sarwate. On sharing secrets and Reed-Solomon codes. *Communications of the ACM*, 24(9):583–584, 1981.

[39] C. Meadows. Some threshold schemes without central key distributors. *Congressus Numerantium*, 46:187–199, 1985.

[40] M. Mignotte. How to share a secret? In *Proc. of the Workshop on Cryptography*, volume 149 of *LNCS*, pages 371–375. Springer-Verlag, 1983.

[41] D. Naccache and J. Stern. A new public key cryptosystem. In *EUROCRYPT'97: Advances in Cryptology*, volume 1233 of *LNCS*, pages 27–36. Springer-Verlag, 1997.

[42] V. Nikov, S. Nikova, and B. Preneel. On multiplicative linear secret sharing schemes. In *Proc. of INDOCRYPT'03*, volume 2904 of *LNCS*, pages 135–147, 2003.

[43] C. Padro and G. Saez. Secret sharing schemes with bipartite access structure. *IEEE Transactions on Information Theory*, 46(7):2596–2604, 2000.

[44] P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In *Proc. of EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer-Verlag, 1999.

[45] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO'91: Proc. of the 11th Annual International Cryptology Conference on Advances in Cryptology*, volume 576 of *LNCS*, pages 129–140, London, UK, 1992. Springer-Verlag.

[46] A. D. Santis, Y. Desmedt, Y. Frankel, and M. Yung. How to share a function securely? In *Proc. of STOC'94*, pages 522–533. ACM Press, 1994.

[47] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *CRYPTO '99: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, volume 1666 of *LNCS*, pages 148–164, London, UK, 1999. Springer-Verlag.

[48] A. Shamir. How to share a secret? *Communications of the ACM*, 22(11):612–613, 1979.

[49] V. Shoup. Practical threshold signatures. In *Proc. of EUROCRYPT'2000*, volume 1807 of *LNCS*, pages 207–220. Springer-Verlag, 2000.

[50] M. Tompa and H. Woll. How to share a secret with cheaters. *Journal of Cryptology*, 1(2):133–138, 1988.

[51] C. F. van Loan. The ubiquitous Kronecker product. *Journal of Computational and Applied Mathematics*, 123(1-2):85–100, 2000.

Appendix A

Basic Notation

Secret Sharing

n	total number of participants
t	threshold
k	the number of partitions
n_j	the number of participants in partition j
d	the shared secret
d_j	share of user j for secret d
$d_{\mathcal{P}_j}$	the (sub)secret shared in partition j
$\mathcal{P}$	the set of all participants
$\mathcal{P}_j$	the set of all participants from partition j
$\mathcal{S}$	the coalition of users
$\mathcal{S}_j$	the coalition of users from partition j
$1, 2, \ldots, n$	the (labels of) participants
Γ	the authorized access structure
$\overline{\Gamma}$	the unauthorized access structure
M_Γ	access structure matrix in multipartite access structure

Number Theory

$\phi(m)$	the value of Euler's totient function in m
$\lambda(N)$	Carmichael function of N
$GF(q)$	Galois field of prime order q
$[a_1, \ldots, a_n]$	the least common multiple of the integers $a_1, \ldots, a_n$
$\gcd(a_1, \ldots, a_n)$	the greatest common divisor of the integers $a_1, \ldots, a_n$
$\mathbb{Z}_m$	the set $\{0, 1, \ldots, m-1\}$, for some $m \geq 1$
$\mathbb{Z}_m^*$	the set $\{a \in \mathbb{Z}_m \mid \gcd(a, m) = 1\}$
$a \mid b$	a divides b
$a^{-1} (\mathrm{mod} m)$	multiplicative inverse of a modulo m, for some $a \in \mathbb{Z}_m^*$
$\log_\alpha \beta$	the discrete logarithm of β to the base α

Appendix B

Acronyms

SSS	secret sharing scheme
FSS	function sharing scheme
LSSS	linear secret sharing scheme
VSS	verifiable secret sharing
JRSS	joint random secret sharing
CRT	Chinese remainder theorem
DSS	digital signature standard
PKC	public key cryptosystem