Secret Sharing and Secure Multi-party Computation

Michael Mortensen

1. July 2007

Department of Informatics
University of Bergen
PB. 7800
N-5020 BERGEN

Preface

This thesis explores the different secret sharing schemes from the 1970s until today. We present the schemes and provide descriptions and examples on how they work. More time is devoted to exploring monotone span programs, a generalisation of secret sharing schemes, as these are the building blocks of all modern research on secret sharing.

We also investigate an application of secret sharing known as unconditionally secure multi-party computation.

Acknowledgements

First and foremost I would like to thank my thesis supervisor Matthew Geoffrey Parker, not only for introducing me to the interesting concept of secret sharing, but also for always taking the time for a talk, no matter how busy he was.

I would also like to thank my friends at department for many good times and lots of encouragement, but especially Haakon Nilsen who also took the time to spell check my thesis.

List of Notation

$\mathcal{F}$	-	A field
$\mathcal{P}$	-	The set of players or participants
P_i	-	Player i from the set $\mathcal{P}$
Γ	-	Access structure
Δ	-	Adversary structure
$\mathbb{R}$	-	Vector a random values
r_i	-	a random value in $\mathbb{R}$
λ	-	A recombination vector
$\mathcal{M}$	-	Monotone Span Program
M	-	A matrix
$M^{(1)}$	-	The first vector column of M
$M^{(i,\ldots,n)}$	-	A matrix consisting of columns i through n of M
$M_{(i)}$	-	The i-th row of M
G^v	-	Local complementation on vertex v in the graph G

List of Acronyms

LSSS	-	Linear secret sharing scheme
MPC	-	Multi-party computation
LC	-	Local Complementation
MSP	-	Monotone Span Program

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

We are all familiar with the scenario where the president and his highest ranking general both have a key, so that the participation of both of them is required to unlock the trigger for the nuclear missile. The digital analogy of this would be that they both have a secret piece of information, and only the combination of these two pieces will be accepted as a working key by the launch computer. In cryptography, we call this secret sharing.

Besides the exciting application of warfare management, there are many other applications of the cryptographic primitive called secret sharing. Modern cryptographic secret sharing, attributed to Shamir [32] and Blakley [6], was first designed for key safeguarding, but has since been applied far beyond its original intent. Today, secret sharing is a cornerstone in secure multi-party computation [4] [11] [23] [10] [16], electronic voting [25], metering schemes [28], and distributed key distribution [12] [29].

Today, computers and working environments are becoming more and more distributed. This greatly increases the need for protocols which secure the distribution of tasks while protecting the privacy of users and the reliability of results. This problem is known as secure multi-party computation [34]. One of the paradigms in secure multi-party computation uses secret sharing, which offers unconditionally secure protocols solving the multi-party computation problem.

1.1 Objective of Thesis

The objective of this thesis is to provide the reader with an overview of the evolution of secret sharing as well as introducing the necessary preliminaries and terminology, providing a deeper insight to the mechanisms of modern secret sharing. In addition, we will describe a practical application of secret sharing demonstrating the versatility of the primitive.

1.2 Thesis Outline

Chapter 2 introduces the classic concept of secret sharing, as well as the modern schemes created in the 1970s and 1980s. We review many of the different schemes and see how they relate to one another.

Chapter 3 builds on the constructions of Chapter 2 and introduces a more generalised notion of secret sharing, one that is the de facto standard today. We study this generalisation in detail and define several operations that manipulate the secret sharing scheme.

Chapter 4 presents an actual application of secret sharing. We look at a form of secure multi-party computation that builds on the secret sharing primitive.

Chapter 2

Simple Secret Sharing and Threshold Schemes

2.1 Introduction

Secret sharing is nothing new; it is a problem that has been encountered and solved many times. The solutions vary in theory, application and elegance. The simplest form of a secret sharing scheme is one that requires all n participants to be present in order to unlock the secret while keeping it hidden for any smaller group. A few of these will be presented briefly in section 2.2. More complex schemes also exist, schemes that require a threshold number of people to cooperate in order to reconstruct the secret and even more flexible schemes that allow predefined groups of people to "unlock" the secret, using a structure known as an *access structure*, which will be discussed in chapter 3.

2.2 Simple Secret Sharing

There exists a multitude of secret sharing schemes which require all the participants to cooperate in order to reconstruct the secret. We briefly mention one in Section 2.2.1.

2.2.1 Additive Secret Sharing

Given a secret $s \in \mathcal{F}$, the dealer D select $n-1$ random integers $\mathbb{R} = \{r_1, r_2, r_{n-1}\}$ uniformly from $\mathcal{F}$. D then computes

$$s_n = s - \sum_{i=i}^{n-1} r_i \; mod \; \mathcal{F} \tag{2.1}$$

D sends each player P_i $1 \leq i \leq n-1$ the share $s_i = r_i$, and the share s_n is sent to P_n.

The reconstruction of secret $s \in \mathcal{F}$ is trivial; simply add all of the shares together:

$$s = \sum_{i=1}^{n} s_i \; mod \; \mathcal{F}$$

The above additive secret sharing scheme requires all participants to contribute their shares in order to reconstruct the secret. If one or more of the participants are missing, no information about the original secret can be recovered; such a scheme is known as a *perfect* secret sharing scheme.

Theorem A *perfect secret sharing scheme* is perfect in an information theoretic sense when the required $\mathcal{P}$ participants can reconstruct the secret $s \in \mathcal{F}$, but any smaller set cannot discover anything about the secret.

Proof. Given a secret $s \in \mathcal{F}$ and a random uniform distribution of the shares or *shadows* of the secret among $\mathcal{P}$ participants, all participants are needed to reconstruct the secret. Consider the situation where $|\mathcal{P}| - 1$ participants attempt to compute the secret s:

$$s' = \sum_{i=1}^{|\mathcal{P}|-1} s_i \tag{2.2}$$

By adding their values together, they can compute $s' = s + s_{|\mathcal{P}|}$. Since the random value $s_{|\mathcal{P}|}$ is unknown, they have no information on what the secret s may be. $\square$

2.3 Threshold Secret Sharing Schemes

In 1979, both Shamir and Blakley presented simple, albeit powerful secret sharing schemes that allowed a t-threshold of n people, where $t \leq n$, to reconstruct the secret. Both solved an impractical real world problem often found in combinatorics textbooks [22]:

> Eleven scientists are working on a secret project. They wish to lock up the documents in a cabinet so that the cabinet can be opened if and only if six or more of the scientists are present. What is the smallest number of keys to the locks each scientist must carry?

In the real world the number of locks on the cabinet would be $\binom{11}{5} = 462$ while the number of keys each scientist would have to carry would be $\binom{10}{5} = 252$. Luckily, mathematics offers a much cleaner and more practical solution. From geometry, we know that given two arbitrary distinct points on the circumference of a circle, we do not have enough information to reconstruct the entire circle. However, given three distinct points, we can reconstruct the entire circle. If we think of this circle as the secret, we can see that we have just constructed a simple 3-threshold secret sharing scheme. While this circle obviously has severe limitations, there exist other structures which can have an arbitrary number of points and hence an arbitrarily sized threshold. One such scheme was constructed by Shamir in [32]. His solution used curves and reconstructed the secret by interpolation when a threshold of t people supplied their part. In Section 2.3.1, we will demonstrate this scheme using polynomial interpolation. In Section 2.3.2, we briefly present another scheme, invented by Blakley [6], that reconstructs the secret by the intersection of hyperplanes. Another secret sharing scheme, by Asmuth and Bloom [1], that uses congruence classes to solve the secret sharing problem, is presented in Section 2.3.3.

Definition A *(t,n)-threshold secret sharing scheme* is a secret sharing scheme that can divide a secret $s \in \mathcal{F}$ into shares $\{s_1, s_2, \ldots, s_n\} \in \mathcal{F}$ so that $t \leq n$ and:

1. Given any set of t or more shares s_i, s can be reconstructed.
2. Any set of fewer than t shares gives no information about s.

2.3.1 Shamir's Secret Sharing Scheme

Given n participants $\mathcal{P} = \{P_1, \ldots, P_n\}$, we can use polynomial interpolation to construct a (t, n)-threshold secret sharing scheme that will require a subset $A \subseteq \mathcal{P}, |A| \geq t$ in order to successfuly reconstruct the secret.

Creating the Shares

In order to create the shares, the dealer D first selects a secret $s \in \mathcal{F}$ to share. We then construct a random polynomial $f(x)$ with a degree of $deg(f) = t-1$.

$$f(x) = s + r_1 x + r_2 x^2 + ... + r_{t-1} x^{t-1} \ mod \ \mathcal{F} \tag{2.3}$$

subject to the following conditions:

- The field $\mathcal{F} > n$ and $\mathcal{F}$ is $GF(q)$ for some prime power q.
- The secret $s \in \mathcal{F}$.
- The threshold $t \leq n$.
- The coefficients $\{r_1, \ldots, r_{t-1}\}$ are chosen independently and randomly from the interval $[0, \mathcal{F})$.

Each share s_i of the secret can then be created by an evaluation of the function f. In other words:

$$s_1 = f(1), s_2 = f(2), \ldots, s_n = f(n).$$

Example *(Polynomial construction)* Let $\mathcal{F} = 17$, $s = 4$, $t = 3$, and $r_{\{1..t-1\}} = \{3, 6\}$. Our polynomial is then, as stated in equation (2.3),

$$f(x) = 4 + 3x + 6x^2 \ mod \ 17$$

and some of our secret shares are:

$$\begin{aligned} s_1 &= f(1) = 4 + 3 \cdot 1 + 6 \cdot 1^2 \ mod \ 17 = 13 \\ s_2 &= f(2) = 4 + 3 \cdot 2 + 6 \cdot 2^2 \ mod \ 17 = 0 \\ s_3 &= f(3) = 16 \\ s_7 &= f(7) = 13 \end{aligned}$$

Reconstructing the Secret

We can reconstruct the secret using polynomial interpolation. A minimum of t participants, one more than the degree of the polynomial, will have to contribute to the reconstruction of the polynomial. The information needed from each participant is a tuple consisting of his value for x and the output of the polynomial function on x. In other words, each participant has a tuple $(x, q(x) = s_x)$. As no two participants share the same value for x, the tuples are in Lagrange form, and the interpolation polynomial in the Lagrange form is defined as:

$$L(z) = \sum_{i=1}^{t} q(x_i) \cdot \prod_{j=1}^{t,j \neq i} \frac{z - x_j}{x_i - x_j} \; mod \; \mathcal{F} \tag{2.4}$$

Since we are only interested in the first coefficient, s, we can simplify equation (2.4) by setting $z = 0$:

$$\begin{aligned} L(0) &= \sum_{i=1}^{t} q(x_i) \cdot \prod_{j=1}^{t,j \neq i} \frac{0 - x_j}{x_i - x_j} \; mod \; \mathcal{F} \\ &= \sum_{i=1}^{t} q(x_i) \cdot \prod_{j=1}^{t,j \neq i} \frac{x_j}{x_j - x_i} \; mod \; \mathcal{F} \\ &= \sum_{i=1}^{t} q(x_i) \cdot \prod_{j=1}^{t,j \neq i} x_j \cdot (x_j - x_i)^{-1} \; mod \; \mathcal{F} \end{aligned} \tag{2.5}$$

In fact, we can generalise this a bit more; by writing the equation as

$$\begin{aligned} L(0) &= \sum_{i=1}^{t} q(x_i) \cdot \lambda_i \; mod \; \mathcal{F} \\ \lambda_i &= \prod_{j=1}^{t,j \neq i} \frac{0 - x_j}{x_i - x_j} \end{aligned} \tag{2.6}$$

we see that λ_i is independent of the shares and only depends on the number of shares used in the reconstruction of the polynomial. This means that we can pre-calculate the values of λ_i and use them later when we recombine the secret.

Definition The vector $\lambda = \{\lambda_1, \ldots, \lambda_n\}$ such that $s = \sum_{i=1}^{t} s_i \lambda_i$ is known as the *recombination vector.*

Example *Reconstruction of the secret using polynomial interpolation* In the previous example we constructed a polynomial mod 17 and generated the shares $s_1 = 13, s_2 = 0, s_3 = 16, s_7 = 13$. Using equation (2.5), and three of the shares created (for example s_1, s_2, and s_7) our secret is

$$\begin{aligned}
L(0) &= \sum_{i=1}^{t} q(x_i) \cdot \prod_{j=1}^{t,j\neq i} x_j \cdot (x_j - x_i)^{-1} \; mod\ 17 \\
&= 13 \cdot \prod_{j=1}^{t,j\neq i} x_j \cdot (x_j - x_i)^{-1} + 0 \cdot \prod_{j=1}^{t,j\neq i} x_j \cdot (x_j - x_i)^{-1} + \\
&\quad 13 \cdot \prod_{j=1}^{t,j\neq i} x_j \cdot (x_j - x_i)^{-1} \; mod\ 17 \\
&= 13 \cdot \left(2 \cdot (2-1)^{-1} \cdot 7 \cdot (7-1)^{-1}\right) + \\
&\quad 13 \cdot \left(1 \cdot (1-7)^{-1} \cdot 2 \cdot (2-7)^{-1}\right) \; mod\ 17 \\
&= 13 \cdot (2 \cdot 1 \cdot 7 \cdot 3) + 13 \cdot (1 \cdot 14 \cdot 2 \cdot 10) \; mod\ 17 \\
&= 4
\end{aligned}$$

The relationship between this method and reconstruction using the Chinese remainder theorem is illustrated in Appendix 2.3.3.

2.3.2 Blakley's Scheme

Given an n-dimensional non-parallel hyperplanes they will intersect at a specific point. Some coordinate of this point will be the secret. This is the idea behind Blakley's threshold secret sharing theme [6]. Figure 2.3.2 illustrates this in three dimensions.

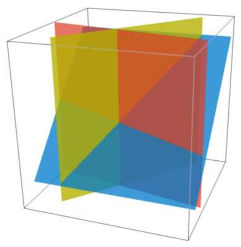


Figure 2.1: Illustration of Blakey's scheme in three dimensions. The secret is the position where all three planes intersect.

2.3.3 The Asmuth & Bloom Secret Sharing Scheme

The idea behind this scheme, first proposed by Asmuth and Bloom [1], is that the keys, also called shadows, are congruence classes of a number associated with the original key.

Creating the Shares

To share the secret s, $s > 0$ among n people with a t-threshold, choose a number $p > s$ and a set of numbers $m_1 < m_2 < m_3 < \ldots < m_n$ such that $gcd(m_i, m_j) = 1$ for $i \neq j$, $gcd(m_i, p) = 1 \forall i$, and $\prod_{i=1}^{t} m_i > p \prod_{i=1}^{t-1} m_{n-i+1}$. Calculate the value $\mathcal{M} = \prod_{i=1}^{t} m_i$ and select an arbitrary integer $\mathcal{A}$ such that $0 \leq y < \mathcal{M}$ where $y = x + \mathcal{A} \cdot p$. The keys are then $S_i = y \ mod \ m_i$.

Example *Key construction* Let our secret be $s = 2$ and select a set of numbers that meet the constraints mentioned above. For this example we will use $p = 3$, $t = 3$, $n = 4$, and $m_1 = 5, m_2 = 7, m_3 = 9, m_4 = 11$. Our number $\mathcal{M} = 5 \cdot 7 \cdot 9 = 315$. Let our random integer $\mathcal{A}$ be 50. Thus, $y = x + \mathcal{A} \cdot p = 2 + 50 * 3 = 152$ which clearly meets the condition $0 \leq y < \mathcal{M}$. Our keys are then

$$\begin{aligned}
s_1 &= 152 \ mod \ 5 = 2 \\
s_2 &= 152 \ mod \ 7 = 5 \\
s_3 &= 152 \ mod \ 9 = 8 \\
s_4 &= 152 \ mod \ 11 = 9
\end{aligned}$$

Reconstructing the Secret

Reconstructing the secret is fairly simple in this scheme. First, find the original y-value by applying the Chinese remainder theorem on the set of congruences. In other words, solve the congruence system:

$$\begin{aligned}
y &\equiv s_1 \ mod \ m_1 \\
y &\equiv s_2 \ mod \ m_2 \\
y &\equiv s_3 \ mod \ m_3 \\
&\vdots \\
y &\equiv s_t \ mod \ m_t
\end{aligned}$$

After the y-value has been reconstructed, the original secret is

$$s = y \ mod\ p \tag{2.7}$$

where p is a public variable that was chosen during the creation of the keys.

Example Given t keys or shadows from the shadows $s_1 \dots s_n$ created in Example 2.3.3, we can rediscover the secret s by first finding the value y by applying the Chinese remainder theorem on $t = 3$ of our congruences:

$$\begin{aligned} y &\equiv 5\ mod\ 7 \\ y &\equiv 8\ mod\ 9 \\ y &\equiv 9\ mod\ 11 \end{aligned}$$

Here $y = 152$, so by equation (2.7) we find that our secret $s = 152\ mod\ 3 = 2$.

2.3.4 Mignotte's Threshold Secret Sharing Scheme

Another scheme that uses the Chinese remainder theorem to solve the problem of secret sharing is the Mignotte's threshold secret sharing scheme [24]. It is quite similar to the scheme mentioned in section 2.3.3, but differs in the requirements and restrictions on the input data and choice of coprime moduli.

Creating the Shares

To share a secret among $n \geq 2$ people with a threshold $t \leq n$, create n coprime integers such that $m_1 < m_2 < \dots < m_{n-1} < m_n$, $m_{n-t+2} \cdot m_{n-t+3} \cdots m_n < m_1 \cdot m_2 \cdots m_t$ and choose a secret s which lies within the interval $[m_{n-t+2} \cdot m_{n-t+3} \cdots m_n, m_1 \cdot m_2 \cdots m_t]$. Each share is then $s_i = s\ mod\ m_i$.

Reconstructing the Secret

Given a set of t shares the secret can be recovered by using the Chinese remainder theorem on the given set of congruence classes

$$\begin{aligned} s &\equiv S_{i_1}\ mod\ m_{i_1} \\ &\vdots \\ s &\equiv S_{i_t}\ mod\ m_{i_t} \end{aligned}$$

2.4 Linear Secret Sharing

An interesting aspect of all the above secret sharing schemes is that they use ideas from linear algebra to solve the problem of secret sharing. Actually, if we were to use the Asmuth&Bloom scheme on polynomials instead of integers, we would have generalised the Asmuth&Bloom scheme to Shamir's scheme. In fact, with small modifications, all the above schemes can be generalised as was shown in [21].

As they are nearly equivalent, many features, such as key updating algorithms, are easily applied across the different schemes. Other traits, such as information security, are equally as good from one scheme to the next.

From here on in this chapter we will focus on different traits and security aspects of linear secret sharing schemes (LSSSs). We will primarily use Shamir's scheme in our descriptions, but the methods used apply equally as well to the LSSSs.

Updating the Keys

In some scenarios, such as a company where employees may come and go and where board members are exchanged annually, it may be necessary to secretly update the keys held by the remaining active participants. This can easily be accomplished. First, generate $t-1$ random α-values. Then, create a new polynomial, with zero as the first coefficient,

$$P(x) = 0 + \sum_{i=1}^{t-1} \alpha_i \cdot x^i \; mod \; \mathcal{F} \tag{2.8}$$

and calculate the shares $s'_1 \dots s'_n$ with this polynomial. Once this is done, distribute each share s'_i to the person with the corresponding s_i share and have him calculate his new share $s_i^{new} = s'_i + s_i$. Once the new share is created, s_i and s'_i should be destroyed.

Example Suppose we have the secret $s = 5$ and the Shamir secret sharing polynomial $f(x) = 5 + 3x + 2x^2 \; mod \; 7$. For the four participants $\mathcal{P} =$

$\{P_1, P_2, P_3, P_4\}$ the dealer would create the following shares:

$$\begin{aligned} s_1 &= f(1) = 3 \\ s_2 &= f(2) = 5 \\ s_3 &= f(3) = 4 \\ s_4 &= f(4) = 0 \end{aligned}$$

Now suppose that the dealer does not want participant P_3 to hold a valid share any longer, to exclude P_3 he creates a new polynomial $p(x) = 0 + 6 \cdot x^1 + 1 \cdot x^2 \ mod\ 7$ and generates the shares:

$$\begin{aligned} s'_1 &= p(1) = 0 \\ s'_2 &= p(2) = 2 \\ s'_4 &= p(4) = 5 \end{aligned}$$

and distributes them to their corresponding participants. Each player that receives such a share computes his new share in the scheme:

$$\begin{aligned} s_1^{new} &= s_1 + s'_1 = 3 + 0 = 3 \\ s_2^{new} &= s_2 + s'_2 = 5 + 2 = 0 \\ s_4^{new} &= s_4 + s'_4 = 0 + 5 = 5 \end{aligned}$$

Using the new shares the participants can reconstruct the correct secret. Conversely, any collusion of two new shares and the share s_3 does not reconstruct the correct secret.

Verifying the Shares

In [15], P. Feldman introduced a verifiable secret sharing (VSS) scheme based upon the Shamir-scheme. Using *homomorphic* encryption (as defined in Appendix A.4.3), the Feldman scheme allows the participants to verify whether or not the shares they have received are consistent. It should be noted, however, that while the Feldman scheme binds a player to a value, the secret is now only *computationally secure*, i.e., retrieving the secret without the correct number of shares is computationally intractable.

One homomorphic encryption method that allows us to utilize this scheme is the use of discrete logarithms. For example, if a dealer D wanted to commit to a value s, i.e., allow the players to validate that they have a valid share of s, he would first generate the shares with the polynomial $f(x) = s + \sum_{i=1}^{t-1} a_i \cdot x^i \ mod\ \mathcal{F}$ and would then select a suitable integer q such that $r = \mathcal{F}q + 1$ is prime and g is a generator of $\mathbb{Z}_r^*$. He would then compute the values $c_0 = g^s, c_1 = g^{a_0}, \ldots, c_{t-1} = g^{a_{t}-1}$ and broadcast these values to the shareholders. The shareholders could then verify their shares by computing

$$g^{s_i} = c_0 \prod_{j=1}^{t-1} c_j^{i^j} \tag{2.9}$$

Another method of verifiable secret sharing where the secret remains secure in the information theoretic sense, but where the consistency of the shares is only computationally secure, is a method by Torben Pedersen [30]. The Pedersen VSS scheme allows a player to non-interactively check whether the share he has received is consistent. It works as follows:

Let q be a Sophie Germain prime, i.e., $p = 2q + 1$ is also prime. Let G be a subgroup of $\mathbb{Z}_p^*$ of order q and g a generator of G. Let g and h be elements of G chosen by a trusted party before use of the scheme or by some other cryptographic primitive (like coin flipping) so that $log_g h$ remains unknown to the participants.

Define the operation commit to be:

$$Commit(s,t) = g^s h^t \ mod\ p \tag{2.10}$$

where $s \in \mathbb{Z}_q$ is the value D wishes to commit to and $t \in \mathbb{Z}_q$ is a random value chosen by D. The commit operation is computationally binding, that is, the player committing to s cannot open a commitment to $s' \neq s$ unless he can solve $log_g h$.

In order to create a verifiable secret sharing scheme using the Pedersen commitment, create two polynomials $f(x) = \alpha_0 + \alpha_1 x + \cdots + \alpha_{t-1}x^{t-1}$ and $f(x) = \beta_0 + \beta_1 x + \cdots + \beta_{t-1}x^{t-1}$ by using Shamir's secret sharing scheme. Let $\alpha_0 = s$ and $\beta_0 = t$. For each participant P_i, create the shares $s_i = f(i)$ and $s'_i = f'(i)$. Also create the commitments $c_i = Commit(\alpha_i, \beta_i)$ where $0 \leq i \leq t-1$. Let the commitments $c_0, \ldots, c_{t-1}$ be publically available. Each participant P_i can now check the validity of his shares (s_i, s'_i) by the equation:

$$c_i = s_i^g {s'}_i^h = \prod_{j=0}^{t-1} c_j^{i^j} \tag{2.11}$$

2.4.1 Security

As it was with the simple secret sharing schemes, the linear threshold secret sharing schemes are also *perfectly* secure.

We have shown that any subset of participants consisting of at least t members can reconstruct the secret. Let us assume that an adversary has obtained $t-1$ shares. For each possible value in the interval $[0, \mathcal{F})$, he can now construct one unique polynomial f' with degree $t-1$ such that $f'(0) = s'$. Even though one of these values will contain the correct secret, each of the values are equally likely, hence by knowing $t-1$ of the shares the adversary still has learned nothing about the secret.

2.4.2 Limitations

Threshold schemes, albeit powerful, have some impractical limitations. In a threshold scheme, we presume that every participant is equal. However, to paraphrase George Orwell, some participants are more equal than others. That is, in certain situations we might want to trust some participants more than we trust others. For example, in a network of computers, where each computer represents a participant, we might want to require a higher threshold of computers that are more likely to be corrupted, like those connected to the outside world, and a lower threshold for the more trusted computers. In other words, we would like to define differently sized subsets of the participants needed to reconstruct the secret. The structure consisting of all these sets is called an *access structure*, and will be covered in Chapter 3.

Chapter 3

Access Structures & Monotone Span Programs

3.1 Introduction

The threshold schemes presented in the previous chapter only allowed *any* subset of a size t or greater to reconstruct the secret. This approach has obvious disadvantages where we want a more fine-grained configurable scheme. Ideally, we would like a method where we could define a set of potentially differently sized authorised subsets. This flexibility is exactly what *access structures* provides us with. Introduced in [19], an access structure, denoted by Γ, consists of a set of authorised subsets where each authorised subset has the ability to reconstruct the secret.

Definition A perfect secret sharing scheme realising the access structure Γ is a method of sharing a secret $\mathcal{S}$ among a set of $\mathscr{k}$ participants (denoted by $\mathcal{P}$), in such a way that the following two properties are satisfied [33]:

1. If an authorised subset of participants $B \subseteq \mathcal{P}$ pool their shares, then they can determine the value of $\mathcal{S}$
2. If an unauthorised subset of participants $B \subseteq \mathcal{P}$ pool their shares, then they can determine nothing about the value of $\mathscr{s}$.

Definition The *unauthorised* or *adversary structure*, denoted by Δ, consists of all the sets that are not in Γ. The tuple (Γ, Δ) is an access structure if $\Gamma \cap \Delta = \emptyset$. An access structure is said to be *complete* if $\Gamma \cup \Delta = \mathcal{P}$.

Definition An access structure Γ is said to be *monotone increasing* if it satisfies the following property:

- If $B \subseteq \mathcal{P}$ is an authorised subset of Γ and $B \subseteq C$ then C is also an authorised subset of Γ.

Likewise, the unauthorised set Δ is *monotone decreasing*, in other words, if a set A is in Δ then any set $B \subset A$ is also in Δ.

Definition All subsets of Γ that cannot be split into smaller authorised subsets are known as minimal sets. The collection of these sets form the basis of the access structure. This set of minimal subsets is denoted by Γ_0.

Example If we have five participants $(\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_5)$ and an access structure Γ with the authorised sets $\{\mathcal{P}_1, \mathcal{P}_2\}$, $\{\mathcal{P}_1, \mathcal{P}_3, \mathcal{P}_5\}$, $\{\mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4\}$, $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$ thus,

$$\begin{aligned} \Gamma_0 &= \{\mathcal{P}_1, \mathcal{P}_2\} \cup \{\mathcal{P}_1, \mathcal{P}_3, \mathcal{P}_5\} \cup \{\mathcal{P}_2, \mathcal{P}_3, \mathcal{P}_4\} \\ \Gamma &= \Gamma_0 \bigcup \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\} \end{aligned}$$

As the set $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\} \supset \{\mathcal{P}_1, \mathcal{P}_2\}$ it is not part of the basis Γ_0

3.2 Monotone Circuits

3.2.1 Circuit Construction

An elegant method of realising an access structure with a perfect secret sharing scheme was provided by J. Benaloh and J. Leichter in [5].

The idea is to first build a *monotone circuit* that describes that access structure, then to create the necessary secret sharing schemes described by the circuit. The secret sharing schemes used are the ones mentioned in the previous chapter, but now each participant in an authorised subset will hold a piece needed to reconstruct the secret for that subset. The shares of each subset are independent of the shares in another subset.

To form the monotone circuit, we can first express the access structure as collection of disjunctions on conjunctions.

Example Given the access structure mentioned in Example 3.1, our circuit will have to realise the following disjunctions of conjunctions.

$$(\mathcal{P}_1 \wedge \mathcal{P}_2) \vee (\mathcal{P}_1 \wedge \mathcal{P}_3 \wedge \mathcal{P}_5) \vee (\mathcal{P}_2 \wedge \mathcal{P}_3 \wedge \mathcal{P}_4)$$

A monotone circuit realising an access structure Γ will have $\mathcal{P}$ inputs $(x_1, \ldots, x_{\mathcal{P}})$, one for each participant, and one output $s \in \mathcal{F}$ corresponding to the resulting secret. The circuit is called monotone as changing any of the false inputs to true will never result in the output changing from true to false.

Using a monotone circuit C, a dealer can realise Γ by creating a (n, n)-secret-sharing-scheme for each conjunction. For example, given the circuit in Figure 3.1, realising the access structure seen in 3.1, we would, for every conjunction $\wedge_i$, create a (n, n)-secret sharing scheme $(s_i \Rightarrow (s_{i1}, \ldots, s_{in}))$ where $n = in(\wedge_i)$ and in is a function that returns the number of wires going into $\wedge_i$. Each participant P_j connected to $\wedge_i$ would be assigned a share.

The resulting secret sharing scheme realises the access structure Γ by utilising a perfect secret sharing scheme for each authorised subset $A \in \Gamma$. It is not completely efficient, however, as if a participant belongs to several subsets, as is the case in the above example, he will receive several shares.

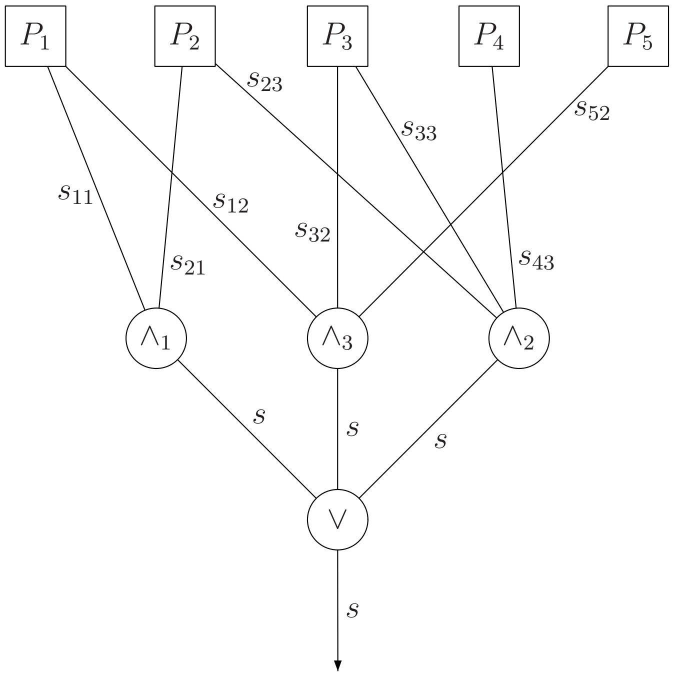


Figure 3.1: A monotone circuit. The final output is the secret $\mathcal{S}$.

Using access structures, a player may receive several shares. Sometimes this is necessary [5], but sometimes it may be foolish, as it is possible to realise a threshold access structure with a different secret sharing scheme for every subset of a size equal to the threshold instead of just creating a threshold scheme which does the same thing efficiently. This leads to a definition of an ideal secret sharing scheme.

Definition [7] An *ideal secret sharing scheme* is a scheme where each participant holds a share of the secret which is the same size as the secret.

Roughly speaking, this means that if our secret $s \in \mathcal{F}$ is represented by $b = log_2|\mathcal{F}|$ bits, then the share held by P_i can also be represented by $b = log_2|\mathcal{F}|$ bits. If the scheme is not ideal, there will be a player holding at least two shares, thus the number of bits needed to represent his shares will exceed the number of bits needed to represent the secret.

The threshold schemes mentioned in Chapter 2 are all ideal.

3.3 Monotone Span Programs

3.3.1 Preliminaries and Notation

A vector of size x in the field K is denoted by K^x. An $m \times d$ matrix M defines a linear map from $\mathcal{F}^d$ to $\mathcal{F}^m$. The kernel or nullspace of M, denoted by $ker(M)$ is the set of vectors $v \in V$ such that $Mv = \mathbf{0}$. The image of M, denoted by $im(M)$ is the set vectors $v \in \mathcal{F}^m$ such that $Mu = v$ for some $u \in \mathcal{F}^d$. $\langle a, b\rangle$ denotes the standard inner product.

The operator $\otimes$ denotes the *Kronecker product* (explained in Appendix A.4.1).

Given an adversary structure Δ, we say that the structure is $\mathbb{Q}^2$ if for any two sets of the structure their union is not the entire set. Similarly, a $\mathbb{Q}^3$ structure has a similar restriction, but is limited to three sets.

Example A adversary structure $\Delta = \{\{P_1, P_2, P_3\}, \{P_3, P_5\}, \{P_3, P_4\}\}$ is $\mathbb{Q}^2$, but not $\mathbb{Q}^3$, while the structure $\Delta = \{\{P_1, P_2\}, \{P_3\}, \{P_4\}, \{P_5\}\}$ is $\mathbb{Q}^3$.

3.3.2 Monotone Span Programs

Monotone Span Programs, or MSPs, were first introduced by Karchmer and Widgerson in [20]. Monotone Span Programs, a linear algebraic model of computation, have a one to one mapping to linear secret sharing schemes [20], [2].

MSPs define a monotone function, as we shall see later in this section.

Definition *A monotone function* f is a function from $\{0,1\}^n$ to $\{0,1\}$ where

$$A \subseteq B \Rightarrow f(A) \leq f(B) \tag{3.1}$$

An access structure Γ defines a monotone function as follows:

$$f_\Gamma(A) = \begin{cases} 0, & if\ A \notin \Gamma, \\ 1, & if\ A \in \Gamma \end{cases}$$

Conversely, a monotone function also defines an access structure Γ_f:

$$\Gamma_f = \{A \subseteq \mathcal{P} : f(A) = 1\}$$

Definition A Monotone Span Program (MSP) $\mathcal{M}$ is a quadruple $(\mathcal{F}, M, \epsilon, \psi)$ where $\mathcal{F}$ is a field, M is a $d \times e$ matrix (with d rows and $e \leq d$ columns over $\mathcal{F}$. ψ is a surjective labelling function that assigns one or more rows of the matrix M to a player P. ϵ is a chosen target vector of necessary size. The size of the MSP $\mathcal{M}$ is d, the number of rows of M.

Let M_A denote a submatrix of M containing only the rows of M which belong to the members of $A \subseteq \mathcal{P}$, given by the labelling function ψ. An MSP is said to compute the access structure Γ when $\epsilon \in im(M_A^T)$ iff $A \subseteq \Gamma$. In other words, the MSP $\mathcal{M}$ realises the monotone function f_Γ by

$$f(A) = 1 \Leftrightarrow \epsilon \in im(M_A^T), A \in \Gamma.$$

We say that A is *accepted* by $\mathcal{M}$ if A is in Γ; otherwise, A is *rejected* by $\mathcal{M}$.

For convenience, we will use the target vector $\epsilon = (1, 0, \ldots, 0)^T$. This is without loss of generality, since a change of the basis will also change the target vector. That is, given an invertible matrix B with the proper dimensions:

$$\epsilon \in im(M_A^T) \Leftrightarrow B\epsilon \in im((BM^T)_A) \tag{3.2}$$

.

If $A \in \Gamma$, hence also accepted by $\mathcal{M}$, then $\mathcal{M}$ contains a recombination vector λ such that $M_A^T\lambda = \epsilon$. Using the recombination vector, the secret can then be reconstructed by the following relations:

$$\langle \lambda, M_A(s, \mathbf{x}) \rangle = \langle M_A^T \lambda, (s, \mathbf{x})^T \rangle = \langle \epsilon, (s, \mathbf{x})^T \rangle = s \tag{3.3}$$

where s is the secret and $\mathbf{x}$ is the dealer's random input. The example below illustrates this:

Example Using the Vandermonde-matrix defined in example A.1.2, Shamir's secret sharing scheme reconstruction as an MSP can be computed as follows: If $A \subseteq P$ and $|A| \geq k$ (k is the threshold of the scheme, as mentioned in A.1.2), compute $\lambda \in \mathcal{F}^{|A|}$ such that $M_A^T \lambda = \epsilon$.

So, given our matrix M (mod 17) (illustrated in table 3.1):

Player			
1	1^0	1^1	1^2
2	2^0	2^1	2^2
3	3^0	3^1	3^2
4	7^0	7^1	7^2

Table 3.1: A Vandermonde matrix for a threshold sharing scheme.

and $A \subset \mathcal{P} = \{1, 2, 4\}$, the matrix defined by M_A is:

$$M_A = \begin{pmatrix} 1^0 & 1^1 & 1^2 \\ 2^0 & 2^1 & 2^2 \\ 7^0 & 7^1 & 7^2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 7 & 15 \end{pmatrix} \mathit{mod\ 17}$$

And our recombination vector λ from $M_A^T \lambda = \epsilon$ is

$$\lambda = \begin{pmatrix} 8 & 2 & 8 \\ 7 & 5 & 5 \\ 3 & 10 & 4 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \equiv \begin{pmatrix} 8 \\ 2 \\ 8 \end{pmatrix} \mathit{mod\ 17}$$

Given the vector containing the shares held by the participants we can use the relationship given in Equation 3.3 to reconstruct the secret $s = [8, 2, 8]^T \cdot [13, 0, 13]^T = 4$.

Theorem [20] Let a secret s be shared by an MSP $\mathcal{M}$ and a set of participants $A \subset \mathcal{P}$. If $f(A) = 1$ the players can correctly determine s. If $f(A) = 0$ the players learn no information about s.

Proof. As defined by the monotone span program, $f(A) = 1$ iff $\epsilon \in im(M_A^T)$ which means there has to exist a recombination vector λ such that $M_A^T \lambda = \epsilon$. Given such a vector we can use equation 3.3 to reconstruct the secret. However, if $f(A) = 0$ then $\epsilon \notin im(M_A^T)$ and there must exist a vector $\mathbf{v}$ so that $M_A^T \mathbf{v} = \mathbf{0}$ and $\langle \epsilon, \mathbf{v} \rangle \neq 0$, ie. $\mathbf{v}_1 \neq 0$. This implies that given $M_A^T \mathbf{w} = s'$ then $M_A^T(\mathbf{w} + j\mathbf{v}) = s', j \in \mathcal{F}$. This means that for the players in A, every possible share is equiprobable [20]. $\square$

3.3.3 Realising General Access Structure with Monotone Span Programs

In the previous section we showed how to realise a threshold access structure using an MSP. In this section, we will briefly show how to realise a general access structure with an MSP using an example.

Example Assume we have four players, $P = \{1, 2, 3, 4\}$, and an access structure $\Gamma = \{\{1, 2\}, \{2, 3\}, \{3, 4\}\}$. We want to share one secret bit $\{0, 1\} \in S$ such that only the sets in Γ can recover the secret. Using a simple secret sharing scheme, we divide the secret amongst each set in Γ.

Participant	Share
1	a
2	s+a, b
3	s+b
4	b

Table 3.2: Secret sharing pieces

As we can see in table 3.2, by partitioning in the manner shown, we have a simple secret sharing scheme that realises the access structure Γ. We can describe the secret share belonging to each of the participants as a matrix:

$$\mathbf{P_1} = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}, \mathbf{P_2} = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \mathbf{P_3} = \begin{pmatrix} 1 & 0 & 1 \end{pmatrix}, \mathbf{P_4} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}$$

Now we can combine the matrices of the participants to form the matrix M. We also create the surjective labelling function ψ that assigns one or more of the rows to the rightful participant:

$$\mathbf{M} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{array}{l} \Rightarrow P_1 \\ \Rightarrow P_2 \\ \Rightarrow P_2 \\ \Rightarrow P_3 \\ \Rightarrow P_4 \end{array}$$

Allowing this matrix to be the matrix M in our MSP $\mathcal{M}$, the dealer can now split secrets, and the authorised subsets can reconstruct them. In the case the target vector ϵ is something other than $1(\mathbf{0})$, we apply equation 3.2 to obtain the desired target vector.

Using the MSP $\mathcal{M}$, we can now share a secret.

1. The dealer generates a random column vector $\mathbf{x} \in \mathcal{F}$ subject to the condition $x_1 = s$ (illustrated in Figure A.3.1).
2. The dealer then computes each share $s_i = M_i\mathbf{x}$ and transmits s_i secretly to the player P_i.

$$\begin{pmatrix} \text{Rows belonging to } P_1 \\ \hline \text{Rows belonging to } P_2 \\ \hline \vdots \\ \hline \text{Rows belonging to } P_n \end{pmatrix} \begin{pmatrix} s \\ \hline r_1 \\ \vdots \\ r_{n-1} \end{pmatrix} = \begin{pmatrix} s_{p1} \\ s_{p2} \\ \vdots \\ s_n \end{pmatrix}$$

Figure 3.2: Creation of shares

Reconstruction of the secret is done in the same manner in which we reconstructed the secret in the example in Subsection 3.3.2.

3.3.4 Multiplicative MSPs

Given two *d-vectors*, $\mathbf{x}$ and $\mathbf{y}$, let $\mathbf{x}\diamond\mathbf{y}$ denote the vector containing all entries of $\mathbf{x}$ and $\mathbf{y}$ in the form of $x_i \cdot y_j$ where $\psi(i) = \psi(j)$* [11]. Thus, if participant P_i has d_i rows in $\mathbf{x}$ and $\mathbf{y}$, then P_i will have d_i^2 rows in $\mathbf{x} \diamond \mathbf{y}$.

Example Given two vectors with ψ labelling:

$$\mathbf{x} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_3 \end{matrix}, \mathbf{x} = \begin{pmatrix} a' \\ b' \\ c' \\ d' \end{pmatrix} \begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_3 \end{matrix}$$

*ψ is a surjective labelling function of an MSP $\mathcal{M}$ as defined in Section 3.3.2.

then the vector $\mathbf{x} \diamond \mathbf{y}$ is:

$$\mathbf{x} \diamond \mathbf{y} = \begin{array}{cc} \begin{pmatrix} a \cdot a' \\ b \cdot b' \\ c \cdot c' \\ c \cdot d' \\ d \cdot c' \\ d \cdot d' \end{pmatrix} & \begin{array}{c} P_1 \\ P_2 \\ P_3 \\ P_3 \\ P_3 \\ P_3 \end{array} \end{array}$$

Definition An MSP is said to be *with multiplication* (as defined by [11]) if there exists a recombination vector $\mathbf{r}$ such that for any two random vectors $\mathbf{x}, \mathbf{x}'$ where $x_0 = s$ and $x'_0 = s'$ it holds that:

$$s \cdot s' = \langle \mathbf{r}, M(s, \mathbf{x}) \diamond M(s', \mathbf{x}') \rangle \tag{3.4}$$

An MSP $\mathcal{M}$ is said to be *strongly multiplicative* if for any subset $A \in \Delta$ of dishonest players, $\mathcal{M}_{\bar{A}}$ is multiplicative.

Constructing a Multiplicative MSP

In [11], it was shown that we can construct a multiplicative MSP $\mathcal{M}$ from any MSP with a $\mathbb{Q}^2$ ($\mathbb{Q}^3$) adversary structure, of a size at most twice the size of the original MSP.

Lemma [11] Given an MSP $\mathcal{M}_0 = (\mathcal{F}, M_0, \psi, \epsilon)$ computing the monotone function f, we can construct a new multiplicative MSP $\mathcal{M}$ computing the same function.

Example Given an MSP $\mathcal{M}_0 = (\mathcal{F}, M_0, \psi, \epsilon)$ construct its dual MSP $\mathcal{M}_0^{\perp} = (\mathcal{F}, M_1 = M_0^{\perp}, \psi, \epsilon)$. A simple and elegant way of constructing the dual MSP $\mathcal{M}_0^{\perp}$ can be found in [14]. Using M_0 and M_1, construct the matrix M consisting of M_0 in its upper left corner and M_1 in its lower right corner.

$$M = \begin{pmatrix} M_0 & \overbrace{0}^{k} & \emptyset \\ \emptyset & M_1^{(1)} & M_1^{(2 \ldots n)} \end{pmatrix}$$

Let $\mathbf{k}$ be the column in M that passes through the first column of M_1. Add this column to the first column of M and delete $\mathbf{k}$. Using the labelling functions ψ from MSP $\mathcal{M}_0$ and $\mathcal{M}_0^{\perp}$, we assign each row of M to some player P_i. The new MSP $\mathcal{M} = (\mathcal{F}, M, \psi, \epsilon)$ now computes the function $f = f \vee f^{\perp}$, is multiplicative, and twice the size of the original MSP $\mathcal{M}_0$.

3.3.5 Operations on MSPs

Addition

Lemma [26] Given the access structures Γ_1 and Γ_2 realised with MSP $\mathcal{M}_1$ and $\mathcal{M}_2$, then the MSP

$$M = \begin{pmatrix} M_1^{(1)} & M_1^{(2,\ldots,n)} & 0 \\ M_2^{(1)} & 0 & M_2^{(2,\ldots,n)} \end{pmatrix} \begin{matrix} \psi_1 \\ \psi_2 \end{matrix} \tag{3.5}$$

realises $\Gamma_1 + \Gamma_2$. For an example of this operation, see Appendix A.2.1.

This method can also be used to realise any general access structure.

Example For each authorised set $A \in \Gamma$, construct a (n, n)-threshold MSP as described in Subsection 3.3.2. The addition of each of these MSPs results in an MSP computing Γ. The MSP is rather inefficient, though, having a size of $\sum_{i=1}^{|\Gamma|} size(\mathcal{M}_i)$.

Multiplication

Lemma [26] Let $\mathcal{M}_1$ and $\mathcal{M}_2$ be two MSPs that realise the access structures Γ_1 and Γ_2. Then the MSP $\mathcal{M}$, with

$$M = \begin{pmatrix} M_1^{(1)} & -M_1^{(1)} & M_1^{(2,\ldots,n)} & 0 \\ 0 & M_2^{(1)} & 0 & M_2^{(2,\ldots,n)} \end{pmatrix} \begin{matrix} \psi_1 \\ \psi_2 \end{matrix} \tag{3.6}$$

realises that access structure $\Gamma_1 \times \Gamma_2$.

Insertion

It is often useful to start with an access structure with small and simple schemes and, from them, build a larger one by inserting new participants. Using a construction by *Nikov et al.* in [26], we can construct a new access structure from two other structures where we replace a player from the first access structure with the sets of the second. In other words, given two access structures Γ_1 and Γ_2 over $\mathcal{P}_1$ and $\mathcal{P}_2$ and their corresponding MSPs $\mathcal{M}_1$ and $\mathcal{M}_2$, we will construct a new MSP $\mathcal{M}$ where a player $P_i \in \Gamma_1$ is substituted with the sets from Γ_2.

Given $\mathcal{M}_1 = \{\mathcal{F}, M_1, \psi_1, \epsilon\}$ and $\mathcal{M}_2 = \{\mathcal{F}, M_2, \psi_2, \epsilon\}$ let M_{1,p_i} be the matrix M_1 containing only the rows belonging to P_i and $M_{1,\bar{p_i}}$ be the matrix M_1 excluding all the rows belonging to P_i. Let $q = size(M_{1,p_i})$ be the number of rows belonging to P_i and the vector $\mathbf{u^j} = (0,\ldots,0,1,0,\ldots,0)^T \in \mathcal{F}^q$ be a column vector with 1 in its j-th position. Let the matrix $\widetilde{M} = \mathbf{u^j} \otimes M_2^{(2,\ldots,n)}$ and $\widehat{M} = M_{1,p_i} \otimes M_2^{(1)}$. The matrix

$$M = \begin{pmatrix} \widehat{M} & \widetilde{M} \\ M_{1,\bar{p_i}} & 0 \end{pmatrix} \tag{3.7}$$

realises the access structure $\Gamma_1(P_i \rightarrow \Gamma_2)$.

Reduction of an authorised set

We introduce the concept of a reduced access structure, which given an access structure $\Gamma = \{A_1, A_2, \ldots, A_n\}$ and a player $P_i \in \mathcal{P}$ results in an access structure $\Gamma_{-P_i} = \{A_1 \backslash P_i, \ldots, A_n \backslash P_i\}$. We present the original structure as an MSP and show the necessary operations needed to create an MSP representing the access structure Γ_{-P_i}.

Theorem Given an MSP $\mathcal{M}$ realising an access structure Γ we can create a new MSP $\mathcal{M}_{-P_i}$ realising the access structure $\Gamma_{-P_i} = \{A_1 \backslash P_i, \ldots, A_n \backslash P_i\}$.

Given an MSP $\mathcal{M} = (\mathcal{F}, M, \psi, \epsilon)$, we construct a new MSP $\mathcal{M}_{-P_i}$ by taking the rows belonging to P_i and giving them to each participant that share an authorised set with P_i. For each player P_j we then remove each row belonging to P_j which is linearly dependent on other rows belonging to P_j iff the corresponding entry in the recombination vector is 0.

Proof. Assume that $P_i \in A_j$ and $A_j \in \Gamma$, the reduction of A_j on P_i will result in a set $A_j \backslash P \in \Gamma$. If all the participants in the set A_j receive P_i's shares it is trivial to see that the set A can still reconstruct the secret s. The sharing of P_i's shares will also not result in a group of participants gaining the unintended ability to reconstruct the secret. This is because if a set of players $P_{k,\ldots,n}$, after receiving the shares of P_i, can successfully reconstruct the secret, then they could have, in collusion with P_i, reconstructed the same secret before P_i's shares were shared. $\square$

Reducing an MSP *may* destroy the (strong) multiplication property of an MSP. Likewise, a reduced MSP from a $\mathbb{Q}^2$ ($\mathbb{Q}^3$) MSP may not be $\mathbb{Q}^2$ ($\mathbb{Q}^3$) any longer, rendering it useless for secure MPC.

Combination of MSPs

By generalising an example from [31] we will in this section show how to combine an MSP, which results are shared with several MSPs into a single MSP.

As the matrix of an MSP corresponds to a linear map from $\mathcal{F}^d$ to $\mathcal{F}^m$, given MSPs $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$ over $\mathcal{F}$ for the boolean functions $f_0, f_1, \ldots, f_n$, where f_0 takes n variables, then there must exist an MSP $\mathcal{M}$ computing the function $f_0(f_1, \ldots, f_n)$.

To construct such a MSP, given the MSPs $\mathcal{M}_0, \mathcal{M}_1, \ldots, \mathcal{M}_n$, create a new matrix M (with row labelling) as follows:

$$M = \left(\begin{array}{c|ccc} M_{0(1)} \otimes M_1^{(1)} & M_1^{(2,\ldots,n)} & \cdots & \emptyset \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline M_{0(n)} \otimes M_n^{(1)} & \emptyset & \cdots & M_n^{(2,\ldots,n)} \end{array}\right) \begin{array}{c} \psi_1 \\ \vdots \\ \psi_n \end{array} \tag{3.8}$$

That is, for each matrix M_i $(1 \leq i \leq n)$, create a tensor-matrix from the row vector i of M_0 and the first column vector of M_i, $\widetilde{M_i} = M_{0(i)} \otimes M_i^{(1)}$. Let $c_i = \mathit{columns}(M_i) - 1$. Append $\widetilde{M_i}$ with $\sum_{j=1}^{i-1} c_j$ $\mathbf{0}$ columns followed by the the columns $M_i^{2,\ldots,n}$ and $\sum_{j=i+1}^{n} c_j$ $\mathbf{0}$ columns.

The MSP $\mathcal{M} = (\mathcal{F}, M, \psi, \epsilon)$ is now equivalent to sharing a secret with $\mathcal{M}_1$ and resharing each resulting share s_i with the MSP $\mathcal{M}_i$. In other words, the MSP $\mathcal{M}$ now computes the function $f_0(f_1, \ldots, f_n)$.

An example of this procedure using four threshold MSPs is provided in Appendix A.2.2.

3.3.6 Known Results and Open Problems on MSPs

- It has been shown that we can create a multiplicative MSP from any MSP with a $\mathbb{Q}^2$ adversary structure. However, it is still not known whether the same result holds for *strongly multiplicative MSPs* with $\mathbb{Q}^3$ adversary structures.
- Another method of constructing multiplicative MSPs was presented by *Nikov et al.* in [27]. They define the concept of a *resulting MSP* that is the result of local multiplication of shares distributed by two different MSPs. To compute a *multiplicative resulting MSP*, an MSP is created

for Γ_1 and its dual access structure Γ_2. This was shown to be a more efficient method of constructing a multiplicative MSP. But, similar to the results of *Cramer et al.* [11], it is still not known which initial access structures yield a *strongly multiplicative MSP*.

- As we have illustrated, MSPs are an elegant construction to realise a monotone function. A lot of ongoing research is focused on proving lower and upper bounds of an MSP. As each row of the matrix in an MSP corresponds to a share in a LSSS, a lower bound on an MSP implies a lower bound on the number of shares in an LSSS. This is of great importance in the practial applications of linear secret sharing and secure multi-party computation. The best lower bound for a monotone span program computing a boolean function with n-inputs variables over any field is $n^{\Omega(logn)}$ [17]).
- Serge Fehr ([13]) introduced the concept of an *extended span program*. Unlike the monotone span program, which is defined over a field, the extended span program is a generalisation allowing it to be defined over rings. These programs share the same problems as MSPs.

3.4 Dynamic Graphs and Access Structures

Definition Given a graph $G = (V, E)$, the local complementation (LC) on a vertex $v \in V$ is the graph G^v where the subgraph of the neighbourhood of v is complemented. The complementation of G^v is $G = (G^v)^v$.

As stated earlier, Shamir schemes are ideal schemes. An interesting observation is that the local complementation of any vertex in a graph representing the Shamir scheme (a complete graph) yields a complete bi-partite graph (see Figure 3.3), which can also be represented with an ideal scheme using the Brickell Vector Space Construction [8].

In fact, it turns out that local complementation of a vertex v belonging to an independent set of any complete bi-partite graph can be realised with an ideal scheme provided that each member of the independent set I where $v \in I$ holds the same share of the secret. This result does not seem to apply to local complementation on a vertex in complete multi-partite graphs, even though these graphs can also be realised with an ideal scheme [8].

When investigating complete multi-partite graphs we did however stumble upon, as far as we know, a seemingly unknown attribute of LC on complete

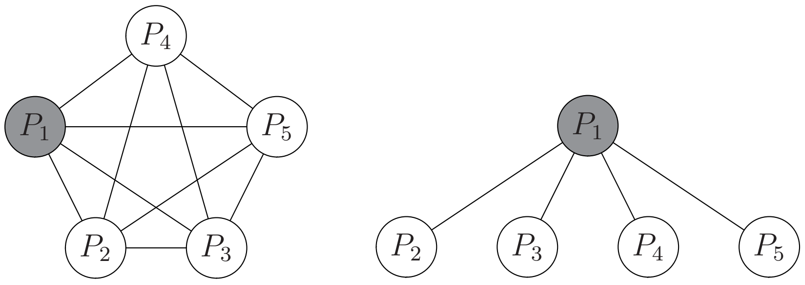


Figure 3.3: Local Complementation on the vertex labelled P_1

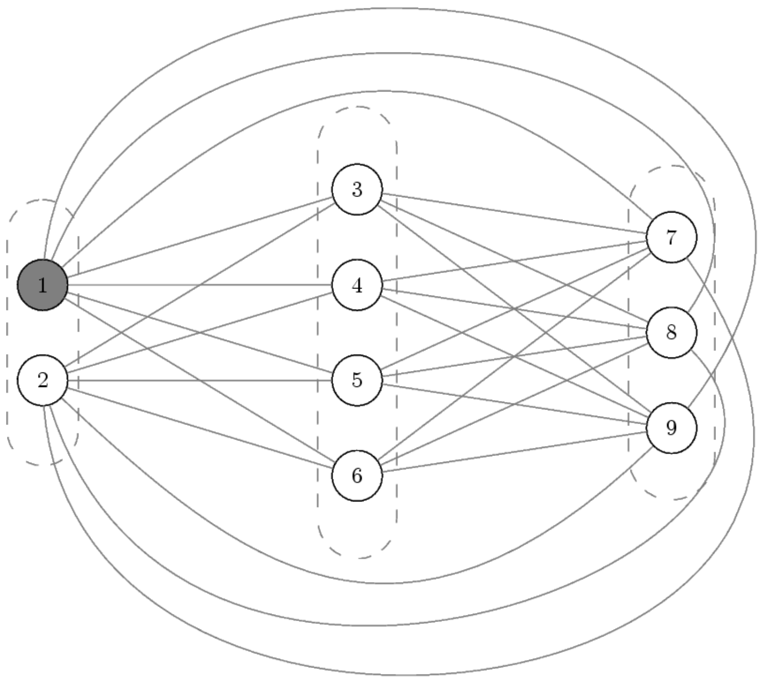
multi-partite graphs. Running LC on a vertex v in a complete multi-partite graph yields a different graph where all independent sets I_i where $v \notin I_i$ have been inverted with respect to itself and all other independent sets I_j where $v \notin I_j$, see figure 3.4 for an illustration of this. It seems however, in addition to LC on the original vertex v, LC on any other vertex u where $u, v \in I_a$ results in the same graph as LC on v.

Theorem Given a complete multi-partite graph, LC on any vertex from an independent set corresponds exactly to LC on any other vertex in the same independent set, that is, $G^u = G^v \ \forall \ u, v \in I$.

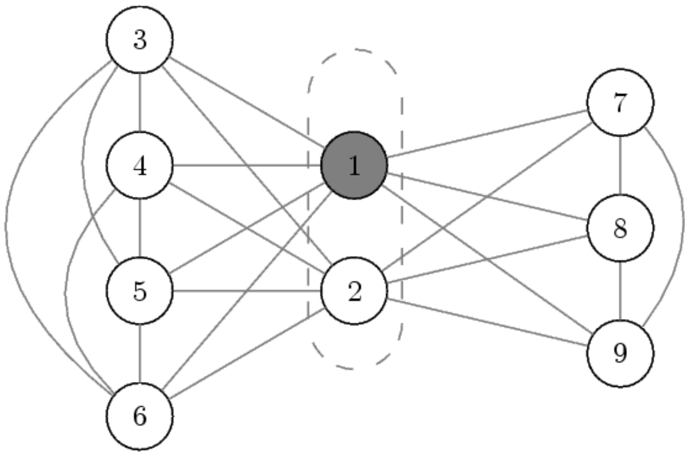
Proof. Let $Adj(G)$ be an adjacency matrix (as defined in Appendix A.4.2) that represents the graph G. LC on a vertex v corresponds to adding (modulo 2) the row $i \in Adj(G)$, belonging to v, to all other rows j where $i_j \neq 0$, ignoring additions along the matrix diagonal.

Clearly, all rows within the same independent set are identical, hence LC on one of them is equivalent to LC on a different one, which means that $G^u = G^v \ \forall \ u, v \in I_j$. □

Corollary As $G^u = G^v \ \forall \ u, v \in I$, provided that G is complete multi-partite, then $(G^u)^v = G \ \forall \ u, v \in I$.



(a) Complete Multi-partite Graph, independent sets are within the dotted ovals



(b) LC on Vertex 1. Independent set is within the dotted oval.

Figure 3.4: Complete Multi-partite Graph and Local Complementation

Chapter 4

Secure Multi-party Computation

4.1 Introduction

First introduced by Yao [34], the term *secure multi-party computation* usually refers to the situation where a group of people wishes to compute the value of a public function F on set of data $S = \{s_1, s_2, \ldots, s_n\}$, but with the restriction that none of the participants can learn more about the global result than what they could learn from their own input and the public information. This was nicely exemplified by Yao in a problem that is known as the *Millionaire problem*:

> Two millionaires wish to know who is richer; however, they do not want to find out inadvertently any additional information about each others wealth. How can they carry out such a conversation?

Generally, we divide secure multi-party computation into two groups, *computationally secure MPC* and *information theoretically secure multi-party computation*. The former is based on unproven cryptographic primitives that are assumed computationally infeasible, while the latter is *unconditionally secure* even if the adversary has unlimited computing power. In the latter, we also make the somewhat impractical, but not impossible, assumption that there exists an unconditionally secure channel between each of the participants.

As you may recall, linear secret sharing offers an unconditionally secure method of sharing information among a group of players. This important attribute is the reason why LSSSes are a cornerstone in secure multi-party

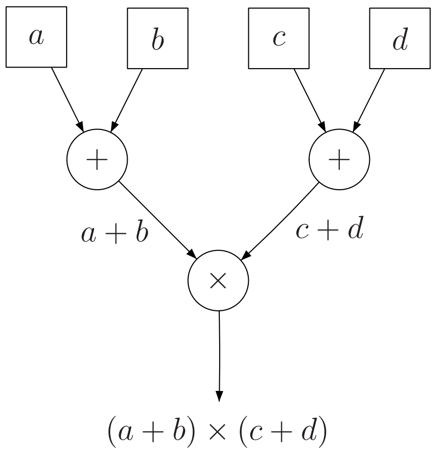


Figure 4.1: Illustration of an arithmetic circuit.

computation.

4.2 Preliminaries

Corrupt players in multi-party computation are generally divided into the following groups:

- A *passive* adversary is a group of participants in an adversary structure where the players follow the protocol correctly, but collaborate in information gathering and sharing with the goal of violating the privacy of other players.
- An *active* adversary is a set of players that, in coordination with each other, intentionally violate the protocol with the intent to disrupt the computation, in order to produce incorrect results and/or violate the privacy of the other players.

Both types of adversary can be *static* or *adaptive*. A *static adversary* is an adversary who chooses a set of players to corrupt *before* the execution of the protocol. An *adaptive adversary* can choose which players to corrupt during the execution of the protocol.

An arithmetic circuit C simply describes the input gates, intermediate calculations and resulting output. An example is illustrated in figure 4.2.

It is also useful to discuss MPC in terms of the *adversary structure* instead of the access structure. Therefore, when we give a threshold t of an adversary structure, the secret sharing polynomial used will be of degree t and we will require $t + 1$ shares in order to correctly reconstruct the secret.

4.3 Commitment Scheme

Definition A *commitment scheme*, for an adversary structure Δ, is a scheme that allows a player P_i to commit to a value a while keeping the value hidden in the presence of an Δ-adversary and also binding P_i to the value in such a way that when he in a later stage decides to reveal the value, only the value a will be accepted among the other players.

For computationally secure MPC, we can use a VSS scheme like the ones mentioned in Section 2.4, but for unconditionally or perfectly secure MPC information, we will use a commitment scheme devised by *Cramer et al.* in [11].

Commitment scheme

COMMIT(s) *Commitment* allows a player to commit to a value s.

CTP(s,j) *Commitment transfer protocol* allows a player P_i to transfer a commitment of s to player P_j.

CSP(s) *Commitment sharing protocol* allows a player P_i to convert a commitment to s into a set of commitments on the shares of s $(s_1, s_2, \ldots, s_n)$. In other words, each player P_j will be committed to his share s_j.

CMP *Commitment multiplication protocol* allows a player P_i who is committed to a, b and $c = ab$ to prove to the other players that c is indeed equal to $a \cdot b$.

OPEN(s) Open reveals a commitment, i.e., the value is revealed to all participants. Only the correct value, the one D commited to, is accepted by the honest players.

The commitment scheme is also homomorphic, that is given two commitments C_a and C_b each player can compute non-interactively the commitment C_{a+b} and C_{ab}.

Commitment In order for a dealer D to commit to a value s, he could simply share s among the players. This would work if we could guarentee that D was honest, but if D was corrupt, he could send inconsistent shares to the different players. To avoid this, we must force D to distribute consistent shares:

1. D chooses a symmetric $e \times e$ matrix R at random and sets $R_{1,1}$ to s. For each row $\mathbf{v_i}$ belonging to P_i in M, D sends the vector $\mathbf{u_i} = R\mathbf{v_i^T}$ to P_i. The first element of $\mathbf{u_i}$ is s_i, P_i's share in s.
2. P_i sends to each P_j the value $s_{ij} = \langle \mathbf{v_j}, \mathbf{u_i} \rangle$. P_j compares the received value with $\langle \mathbf{v_i}, \mathbf{u_j} \rangle$ and broadcasts[*] the message *fail(i,j)* if they are not equal [†].
3. If a fail(i,j) is received, D broadcasts the value s_{ij}. If any of the players fail to agree that the value s_{ij} is correct, they broadcast an *accusation* that D is corrupt.
4. Say P_j accuses D of corruption. D can disprove the accusation by broadcasting the information sent to P_j in step 1.
5. Each player checks the values broadcasted by D to see if they are consistent with the values they have received. If they are not he sends an *accusation* that D is corrupt. By the $\mathbb{Q}^2$ ($\mathbb{Q}^3$) property of the adversary structure, the protocol will only be rejected if at least one of the honest players sends an *accusation*. Likewise, the protocol will be accepted if all the accusing players are in Δ.

An example of a commitment check is given in appendix A.3.2.

Commitment Transfer Protocol A commitment transfer protocol allows a player P_i, who has a commitment to s to transfer the commitment to a player P_j. If P_j and P_i are honest, the protocol leaks no information to the adversary. P_j learns the value s in the process.

1. P_i securely sends P_j all the information he used to create a commitment C to s. This includes s.
2. P_j creates a new commitment C' to s using the information received in step 1 and checks whether or not $C' - C = 0$.

If any of the above steps fail, P_i or P_j must be corrupt. To disprove his corruption, P_i can open s.

Commitment Sharing Protocol A commitment sharing protocol is a protocol that allows a player P_i committed to a value s to secret share

[*] *Fitzi et al.* [16] shows how to create a perfectly secure broadcast channel for any $\mathbb{Q}^3$ adversary structures.

[†] As $\langle \mathbf{v_j}, \mathbf{u_i} \rangle = \langle \mathbf{v_j} R, \mathbf{v_i}^T \rangle = \langle \mathbf{v_i}, R\mathbf{v_j}^T \rangle = \langle \mathbf{v_i}, \mathbf{u_j} \rangle$ P_i and P_j can check the consistency of their shares.

$s \Rightarrow s_1, \ldots, s_n$ so that each player P_j is committed to the share s_j.

To accomplish this, P_i, already committed to s, generates a random vector $\mathbb{R}$ of size $e-1$ and commits to each value in $\mathbb{R}$. Using CTP, P_i transfers the commitment of s_j to P_j and hence P_j also learns s_j. Since s_j is a result of linear operation on the previously committed values, P_j can check that s_j is indeed a share of s.

Commitment Multiplication Protocol A commitment multiplication protocol allows a player committed to the values a, b and $c = ab$ to convince the other players that $ab = c$. If the scheme is strongly multiplicative, the CMP will be perfectly secure. Otherwise it will have a negligible error probability ε.

CMP with negligible error:

1. P_i has already committed to the values a, b and $c = ab$. We'll denote those commitments C_a, C_b and C_c. In order to convince the other players that c is indeed equal to $a \cdot b$, P_i chooses a random β and creates the commitment C_β and another commitment for the value βb, we will call it $C_{\beta b}$.
2. The other players generate a random challenge $r \in \{0, 1\}$ using the appropriate protocols.
3. P_i opens the commitment $rC_a + C_\beta$, which reveals the value $r_1 = ra + \beta$, and he also opens a commitment to $r_1 C_b - C_{\beta b} - rC_c$, which should reveal 0 as

$$\begin{aligned}
r_1 C_b - C_{\beta b} - rC_c &= \\
(ra + \beta)C_b - C_{\beta b} - rC_c &= \\
ra \cdot C_b + \beta \cdot C_b - C_{\beta b} - rC_c &= \\
rC_{ab} + C_{\beta b} - C_{\beta b} - rC_c &= \qquad \text{(By homomorphism)} \\
rC_c - rC_c + C_{\beta b} - C_{\beta b} &= \\
&= 0
\end{aligned}$$

If P_i is honest, then all the opened values are either random or 0. If P_i can answer two different random challenges correctly, then $ab = c$ with an error probability of $\frac{1}{2}$. This probability can be reduced by iterating the process until the desired probability ε is reached [9].

CMP with zero error:

1. P_i has committed to three values a, b and c.

2. Using CSP, P_i creates and distributes shares of a, b and c. Each player P_j receives the shares a_j, b_j and $c_j = a_j \cdot b_j$ and is committed to them.

3. Since P_i is committed, we know that the shares of a, b and c are consistent and $f_a(0) = a$, $f_b(0) = b$ and $f_c(0) = c$ where $deg(f_a) = t$, $deg(f_b) = t$, and $deg(f_c) = 2t$. Each player checks whether or not his shares compute $c_i = a_i \cdot b_i$ and broadcasts an *accusation* if this fails.

In order to have a CMP with zero error the secret sharing schemes must be strongly multiplicative. That is, $t < n/3$. That means that there are at least $n-t$ honest players in the scheme. And since $n-t > 2t$, where $2t$ is also the maximum degree of f_c, the honest players can always correctly reconstruct the polynomial if $c = ab$. If $c \neq ab$, at least one honest player, in addition to the corrupt players, would have to accuse P_i.

This protocol can also be generalised to work for any secret sharing scheme with a $\mathbb{Q}^3$ adversary structure, provided that the scheme is realised with a *strongly multiplicative* MSP [11].

Open reveals a commitment. To open a commitment on s, the dealer D broadcasts s and all the shares of $s = \{s_1, \ldots, s_n\}$. If the number of players that agree to the broadcasted values of s and s_i is greater than the set of players from the adversary structure Δ, then the opening of s is accepted.

4.4 Unconditionally Secure Multi-party Computation

Using the linear secret sharing schemes we introduced the previous chapters, we can construct a secure multi-party computation scheme. First, we define the necessary arithmetic operations for computation in a passive adversary case. Computations in an active adversary setting will be addressed later in this section.

Threshold schemes used to securely compute a function f with a passive static or adaptive adversary can only compute a function securely if $t < n/2$. For a static or adaptive active adversary where a broadcast channel does not exist, the bound is $t < n/3$. Upon closer inspection, we see that a (strongly) multiplicative threshold function is $\mathbb{Q}^2$ ($\mathbb{Q}^3$). This leads us to a more general protocol for multi-party computation:

Lemma [11]

- We can compute any function with a passive adversary structure provided that our secret sharing scheme is resilient to a $\mathbb{Q}^2$ adversary structure.
- To compute a function f in the presence of an active adversary our adversary structure must be $\mathbb{Q}^3$.

4.4.1 Secure MPC with a Passive Adversary

Given two instances of Shamir's secret sharing scheme, the participants can compute the **addition** of the secret simply by adding the shares of one instance to their corresponding shares in the other instance.

$$f_1 + f_2 = (s_{1,1} + s_{2,1}) + (s_{1,2} + s_{2,2}) + \ldots + (s_{1,n} + s_{2,n}) = s_1 + s_2 \tag{4.1}$$

Multiplication of a constant c can be computed by having each participant P_i compute $c_i = s_1 \cdot c$. The resulting shares $c_1, \ldots, c_n$ determine $s \cdot c$.

Multiplication is a bit more complicated as the multiplication of two polynomials would result in a new polynomial with degree of at most $deg(f_1) + deg(f_2) = 2t$ and the coefficients of the new polynomial would not be randomly distributed. To solve this problem, we perform a sanity operation, a reshare, after every multiplication that reduces the degree of $f_1 \cdot f_2$ and adds uniformly random values to all coefficients in $f_1 \cdot f_2$, except for first coefficients of each polynomial, ie, the secret. We will illustrate how to perform multiplication of two polynomials generated using Shamir's secret sharing scheme with an example.

Example Given two values a and b, we can securely compute the value $c = ab$ with a passive adversary by executing the following steps:

1. Share $a \Rightarrow a_1, a_2, \ldots, a_n$ and $b \Rightarrow b_1, b_2, \ldots, b_n$ such that P_i receives shares a_i and b_i
2. Each player P_i then computes the product of his two shares, $c_i = a_i \cdot b_i$
3. Each player P_i then shares $c_i \Rightarrow c_{i,1}, c_{i,2}, \ldots c_{i,n}$ and sends the shares to their respective players.

4. Each player P_j can now compute the value $\tilde{c_j}$ using the values received and the recombination vector.
$$\begin{pmatrix} c_{1,1} & c_{2,1} & \cdots & c_{n,1} \\ c_{1,2} & c_{2,2} & \cdots & c_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1,n} & c_{2,n} & \cdots & c_{n,n} \end{pmatrix} \begin{pmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{pmatrix} = \begin{pmatrix} \tilde{c_1} \\ \tilde{c_2} \\ \vdots \\ \tilde{c_n} \end{pmatrix}$$
5. The shares $\tilde{c_1}, \tilde{c_2}, \ldots, \tilde{c_n}$ determine $c = ab$ completing the multiplication.

A practical example of this procedure is given in Appendix A.3.1.

Using these primitives, we can now evaluate an arithmetic circuit C over a field $\mathcal{F}$ computing a function f such that when the circuit completes, each player will have a share of the resulting computation.

Theorem There exists functions that cannot be securely computed with a passive adversary if the adversary structure is not at least $\mathbb{Q}^2$ [4].

Proof. Consider for example the OR-function between two players. It is easy to see that this function can never be computed by the two participants, each providing one bit, without one of them leaking information. □

Conversely, we can compute any function f securely with a passive adversary provided that the adversary structure is at least $\mathbb{Q}^2$. To prove this, it is sufficient to show that given three values $a, b, c \in \mathcal{F}$, we can always securely compute $a + b$, $c \cdot a$, and $a \cdot b$ [4]. This was shown in the above example.

4.4.2 Secure MPC with an Active Adversary

In order to safely compute a function f on a set of values where we have a *static* or *adaptive* active adversary we require a method that allows the participants to check whether a player is executing the protocol correctly and providing valid shares. In other words, we need a stronger primitive that allows players to *commit* to a value. To achieve this, we will use the commitment scheme described in Section 4.4.2.

Using this scheme, we can now construct a *information theoretic secure MPC* protocol resilient against an active adaptive adversary $\mathbb{Q}^3$ structure.

Assume two committed input values a and b shared with CSP so that each player P_j holds a commitment to that share a_j and b_j.

Scenario	**Adversary**	**Broadcast**	**Structure**
Information theoretic	Passive	No	$\mathbb{Q}^2$
Information theoretic	Active & adaptive	Yes	$\mathbb{Q}^2$
Information theoretic	Active & adaptive	No	$\mathbb{Q}^3$
Computational secure	Active & adaptive	No	$\mathbb{Q}^2$

Table 4.1: Security of secure multi-party computation

To compute the **addition** of a and b, each player P_i adds his two shares $c_i = a_i + b_i$ and computes a commitment for $a_i + b_i$.

Multiplication of the values a and b:

1. Each player P_i multiplies his shares $\tilde{c}_i = a_i \cdot b_i$ and commits to the result. Each P_i then performs $\mathrm{CMP}(C_a, C_b, C_{\tilde{c}_i})$ where C_a, C_b, and $C_{\tilde{c}_i}$ are the commitments to a_i, b_i and $\tilde{c}_i$.
2. Each player then shares his commitment to $\tilde{c}_i$ using the CSP protocol.
3. Every player now computes the value $c_j = \sum_{i=1}^{n} \lambda_i c_{ij}$ and a commitment $C_{c_j}$for it. Players can check if the value is correct because $C_{c_j} = \sum_{i=1}^{n} \lambda_i C_{ij} = \sum_{i=1}^{n} \lambda_i C_{ji}$.

If a participant fails in any of the above steps, he is disqualified, and if the adversary structure is $\mathbb{Q}^3$, his input can be ignored, i.e., we remove corrupt players from the recombination vector. The reconstruction is still possible, because the number of honest players is sufficient enough to reconstruct the missing local multiplication. To illustrate this, recall that for a $\mathbb{Q}^3$ threshold scheme the adversary threshold is $t < n/3$. Given this requirement, the number of honest players is at least $n - t > 2t$, which means that there exist enough honest players to reconstruct the missing local multiplication, which would be a polynomial of degree $2t$ [11].

If we allow for a negligible error and assume a broadcast channel, then $n/2 > t \geq n/3$ is sufficient for secure multi-party computation [9]. [11] showed that we can construct a general secure multi-party computation scheme from *any* linear secret sharing scheme provided that the access structure allows MPC and VSS. That is, we can construct a secure multi-party computation protocol from any $\mathcal{M}$ with a $\mathbb{Q}^2$ ($\mathbb{Q}^3$) adversary structure.

4.5 Conclusions

As we have seen, unconditionally secure MPC is a very useful and applicable cryptographic problem, but rather expensive. Especially multiplication and commitments come with a rather large overhead in communication. Using a computationally secure commitment scheme reduces the overhead, but at the expense of unconditional provable security.

If we are willing to sacrifice unconditional security, we can create stronger protocols, tolerating larger adversary structures [18] [23]. Table 4.1 lists the known results for the different secure MPC models.

Chapter 5

Summary and Conclusion

5.1 Summary

In this thesis, we have seen the different secret sharing schemes developed from the 1970s and to the 1990s until they were finally generalised as monotone span programs in 1993. We have also covered the versatility of this generalisation, by showing different operations on it and seeing how this affects the access structures they realise.

In addition, we have studied how we can use monotone span programs to implement verifiable secret sharing as well as information theoretically secure multi-party computation.

5.2 Conclusion

Even though secret sharing is a field of research which has been explored significantly since it was first discovered, much still remains unexplored. The relationship between access structures and graphs looks interesting, but we did not encounter many papers discussing this during our research. [26] dedicated a subsection to access structures with forbidden sets as star-topologies and [3] proved a lower-bound for MSPs computing a 6-clique function, but beyond this, little seems to be explored.

5.3 Open Problems

- The following conjecture was given by *A. Beimel* [2] and to this day has still not been proven or disproven:

  Conjecture There exits a $\varepsilon > 0$ such that for every positive integer n there is an access structure over n participants to which their exists secret sharing scheme which distribute shares of a length exponential to the number of parties.

- It was shown by [10] how to construct a multiplicative MSP for any MSP realizing a $\mathbb{Q}^2$ adversary structure. A method of constructing a strongly multiplicative MSP for an MSP realizing a $\mathbb{Q}^3$ is still not known even though it has been proved that such an MSP does indeed exist.

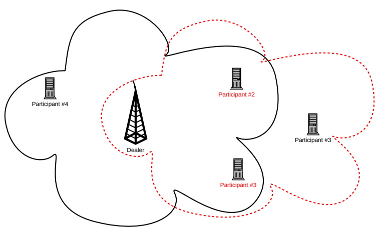
- In this thesis we showed that all bi-partite graphs have a local complementation which can be realised with an ideal scheme. An interesting question is, does there exist other graph structures which have similar properties. If so, operations such as LC may assist the creation of efficient schemes for relatively complex graphs by altering simpler graphs which are known to contain certain properties.

5.3.1 Secure MPC on Distributed Networks

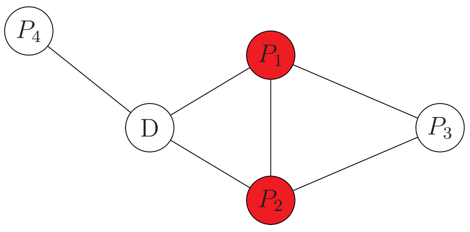
Secure multi-party computation is inherently distributed, yet MPC on a distributed or mobile network presents many difficulties that do not seem to have simple solutions. Earlier issues are still present, such as share distribution and secure reconstruction, but many are vastly more complex now that a direct link to every other player does not exist any longer.

We can surely assume that some participants will have to pass their keys via other participants to reach the dealer and vice-versa. This opens for man-in-the-middle type attacks for active adversaries and simplified eavesdropping for passive adversaries. Figure 5.1a illustrates this problem. In this scenario, all communication between participant P_4 and the dealer D must go via participants P_2 or P_3.

As illustrated with red-coloured nodes in figure 5.1b, we can surely assume that the members of the unauthorised structure Δ are able to cooperate in order to corrupt the protocol.



(a) Network distribution and coverage



(b) Network visualized as a graph

Figure 5.1: Distributed network

The problem of a passive adversary can be solved by having pre-shared keys. It can even be solved with computational security using public key cryptography, since a passive adversary will execute the protocol correctly. However, the bigger challenge is handling active adversaries. As they may retain a transmission or act as a man-in-the-middle, no communication via them can be trusted, defeating the purpose of the protocol.

Little work seems to have been done on the area of secure MPC on distributed networks.

Appendix A

Appendix

A.1 Methods to reconstruct a Shamir secret

A.1.1 The Chinese Remainder Theorem

Using the primes generated by the ideal $(x-x_j)$, where x_j is the value x from the tuple $(x, q(x))$ held by the participants, we can reconstruct our secret as follows:

$$\begin{aligned} M(x) &= (x-x_{j_1})\cdot(x-x_{j_2})\cdots(x-x_{j_k}) \qquad (A.1)\\ m_t(x) &= \frac{M(x)}{(m-x_{j_t})} \qquad (A.2)\\ q(x) &= \sum_{t=1}^{k} s_t\langle m_t^{-1}(x)\rangle_{(x-x_{j_t})}\, m_t(x)\ mod\ \mathcal{F} \qquad (A.3)\end{aligned}$$

Example To illustrate this, we use the values found in the example in subsection 2.3.1, $S_1 = 13, S_2 = 0, S_3 = 16, S_7 = 13$. Since our threshold is 3, we can use any three distinct shares to reconstruct the secret. For this example, we will use S_1, S_2 and S_7, thus $M = (x-1)\cdot(x-2)\cdot(x-7)$.

Using equations A.1 and A.2, all modulo our prime $p = 17$, we find the values

$$\begin{aligned}
m_1(x) &= \frac{M}{(x-1)} = (x-2)\cdot(x-7) \\
\langle M_1(x)\rangle_{(x-1)} &= (1-2)\cdot(1-7) = 6 \\
\left\langle M_1(x)^{-1}\right\rangle_{(x-1)} &= 6^{-1} = 3 \\
m_2(x) &= \frac{M}{(x-2)} = (x-1)\cdot(x-7) \\
\langle M_2(x)\rangle_{(x-2)} &= (2-1)\cdot(2-7) = -5 = 12 \\
\left\langle M_2(x)^{-1}\right\rangle_{(x-2)} &= 12^{-1} = 10 \\
m_3(x) &= \frac{M}{(x-7)} = (x-1)\cdot(x-2) \\
\langle M_3(x)\rangle_{(x-7)} &= (7-1)\cdot(7-2) = 30 = 13 \\
\left\langle M_3(x)^{-1}\right\rangle_{(x-7)} &= 13^{-1} = 4
\end{aligned}$$

which we then use in Equation A.3:

$$\begin{aligned}
q(x) &= 13\cdot 3\cdot(x-2)(x-7)+ \\
&\quad 0\cdot 12\cdot(x-1)(x-7)+ \\
&\quad 13\cdot 4\cdot(x-1)(x-2)\ mod\ 17 \\
&= 5\cdot(x^2-9x+14)+ \\
&\quad 1\cdot(x^2-3x+2) \\
&= 5x^2+6x+2+x^2+14x+2 \\
&= 6x^2+3x+4
\end{aligned}$$

This gives us the same polynomial that was used to construct the shares in the example in subsection 2.3.1.

A.1.2 Linearity in Ideal Threshold Secret Sharing Schemes

It is also possible to express the shares as a system of linear equations. Each of the original coefficients can be expressed as a linear equation:

$$S_x = M + \alpha_1 \cdot x_j^1 + \alpha_2 \cdot x_j^2 + \cdots + \alpha_{k-1}\cdot x_j^{k-1}\ mod\ \mathcal{F},\ 1 \le j \le k$$

This can be expressed in matrix form $\mathbf{A}x = b \; mod \; \mathcal{F}$ where

$$\mathbf{A} = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{k-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{k-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & x_k^2 & \cdots & x_k^{k-1} \end{pmatrix}, x = \begin{pmatrix} M \\ \alpha_1 \\ \vdots \\ \alpha_k \end{pmatrix}, b = \begin{pmatrix} S_1 \\ S_2 \\ \vdots \\ S_k \end{pmatrix} mod \; \mathcal{F}$$

The above matrix **A** is a *Vandermonde matrix* and therefore its determinant can easily be found using the well-known formula

$$detA = \prod_{1 \leq i < j \leq k} (x_i - x_j) \; mod \; \mathcal{F}$$

This eases the creation of an inverse matrix $\mathbf{A}^{-1}$, which can be used to find the solution to the system of linear equations.

Example Given the polynomial and the shares from the example in Subsection 2.3.1, we construct the following matrix and vectors.

$$\mathbf{A} = \begin{pmatrix} 1^0 & 1^1 & 1^2 \\ 2^0 & 2^1 & 2^2 \\ 7^0 & 7^1 & 7^2 \end{pmatrix}, x = \begin{pmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \end{pmatrix}, b = \begin{pmatrix} 13 \\ 0 \\ 13 \end{pmatrix} mod \; 17$$

A solution $\mathbf{A}^{-1}b = x$ is then:

$$\begin{pmatrix} 8 & 2 & 8 \\ 7 & 5 & 5 \\ 3 & 10 & 4 \end{pmatrix} \begin{pmatrix} 13 \\ 0 \\ 13 \end{pmatrix} \equiv \begin{pmatrix} 4 \\ 3 \\ 6 \end{pmatrix} mod \; 17$$

where the resulting x-vector contains the exact same coefficients used to construct the original polynomial.

A.2 MSP Examples

A.2.1 Addition of MSPs

Given two access structures $\Gamma_1 = \{\{P_1, P_2, P_3\}\}$ and $\Gamma_2 = \{\{P_2, P_4, P_5\}\}$, realised with the MSPs $\mathcal{M}_1 = (\mathcal{F}, M_1, \psi_1, \epsilon)$ and $\mathcal{M}_2 = (\mathcal{F}, M_2, \psi_2, \epsilon)$, where

$\mathcal{F}$ is GF(7) and

$$M_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 2 \end{pmatrix} \overbrace{\begin{matrix} P_1 \\ P_2 \\ P_3 \end{matrix}}^{\psi_1}, M_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 2 \end{pmatrix} \overbrace{\begin{matrix} P_2 \\ P_4 \\ P_5 \end{matrix}}^{\psi_2}$$

.

Using Equation 3.5, we create a new MSP $\mathcal{M} = (\mathcal{F}, M, \psi, \epsilon)$ where

$$M = \left(\begin{array}{c|cc|cc} 1 & 1 & 1 & 0 & 0 \\ 1 & 2 & 4 & 0 & 0 \\ 1 & 3 & 2 & 0 & 0 \\ \hline 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 2 & 4 \\ 1 & 0 & 0 & 3 & 2 \end{array}\right) \overbrace{\begin{matrix} P_1 \\ P_2 \\ P_3 \\ P_2 \\ P_4 \\ P_5 \end{matrix}}^{\psi}$$

.

The new MSP $\mathcal{M}$ realises that access structure $\Gamma = \{\{P_1, P_2, P_3\}, \{\{P_2, P_4, P_5\}\}$.

A.2.2 Combination of MSPs

Given the MSPs $\mathcal{M}_0, \mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3$ where $\mathcal{F}$ is $GF(7)$, $\epsilon = (1, 0, \ldots, 0)^T$ and

$$M_0 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 2 \end{pmatrix} \begin{matrix} \mathcal{M}_1 \\ \mathcal{M}_2 \\ \mathcal{M}_3 \end{matrix}, M_1 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 2 \\ 1 & 4 & 2 \end{pmatrix} \begin{matrix} \mathcal{P}_1 \\ \mathcal{P}_2 \\ \mathcal{P}_3 \\ \mathcal{P}_4 \end{matrix}$$

$$M_2 = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 4 & 2 \end{pmatrix} \begin{matrix} \mathcal{P}_1 \\ \mathcal{P}_2 \\ \mathcal{P}_4 \end{matrix}, M_3 = \begin{pmatrix} 1 & 3 & 2 \\ 1 & 4 & 2 \\ 1 & 5 & 4 \end{pmatrix} \begin{matrix} \mathcal{P}_3 \\ \mathcal{P}_4 \\ \mathcal{P}_5 \end{matrix}$$

Let the secret $s = 6$. Let the random vectors for each sharing be

$$\mathbb{R}_0 = \begin{pmatrix} 3 \\ 5 \end{pmatrix}, \mathbb{R}_1 = \begin{pmatrix} 1 \\ 4 \end{pmatrix}, \mathbb{R}_2 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \mathbb{R}_3 = \begin{pmatrix} 6 \\ 4 \end{pmatrix}$$

Let M be the resulting matrix after applying Equation 3.8:

$$M = \left(\begin{array}{ccc|cccccc}
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 2 & 4 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 3 & 2 & 0 & 0 & 0 & 0 \\
1 & 1 & 1 & 4 & 2 & 0 & 0 & 0 & 0 \\
\hline
1 & 2 & 4 & 0 & 0 & 1 & 1 & 0 & 0 \\
1 & 2 & 4 & 0 & 0 & 2 & 4 & 0 & 0 \\
1 & 2 & 4 & 0 & 0 & 4 & 2 & 0 & 0 \\
\hline
1 & 3 & 2 & 0 & 0 & 0 & 0 & 3 & 2 \\
1 & 3 & 2 & 0 & 0 & 0 & 0 & 4 & 2 \\
1 & 3 & 2 & 0 & 0 & 0 & 0 & 5 & 4
\end{array}\right)
\begin{array}{c}
P_1 \\ P_2 \\ P_3 \\ P_4 \\ \hline P_1 \\ P_2 \\ P_4 \\ \hline P_3 \\ P_4 \\ P_5
\end{array}$$

Multiplying this matrix with the vector $\mathbf{v} = (s, \mathbb{R}_0^T, \mathbb{R}_1^T, \mathbb{R}_2^T, \mathbb{R}_3^T)^T$ gives us

$$M\mathbf{v} = (5, 4, 4, 5, 0, 5, 0, 2, 1, 1)^T$$

This is exactly equal to the result we would achieve by first sharing s with MSP $\mathcal{M}_0$ and resharing each resulting share by $\mathcal{M}_i$:

$$M_0 \begin{pmatrix} 6 \\ 3 \\ 5 \end{pmatrix} = \begin{pmatrix} 0 \\ 4 \\ 4 \end{pmatrix}$$

Setting $s_1 = 0, s_2 = 4, s_3 = 4$, we get

$$M_1 \begin{pmatrix} s_1 \\ 1 \\ 4 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \\ 4 \\ 5 \end{pmatrix}, M_2 \begin{pmatrix} s_2 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 5 \\ 0 \end{pmatrix}, M_3 \begin{pmatrix} s_3 \\ 6 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

which combined to one vector is

$$M\mathbf{v} = (5, 4, 4, 5, 0, 5, 0, 2, 1, 1)^T$$

A.3 Secure Multi-party Computation Examples

A.3.1 Passive Threshold MPC

Let $\mathcal{M}$ be the MSP $\mathcal{M} = (\mathcal{F}, M, \psi, \epsilon)$ where $\mathcal{F} = 7$, ϵ is the vector $(1, 0, \ldots, 0)^T$ and

$$M = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix} \begin{array}{c} P_1 \\ P_2 \\ P_3 \\ P_4 \end{array}$$

The recombination vector of M is $\lambda = (4, 1, 4, 6)^T$.

Given input $a = 3$ and $b = 5$, we wish to securely compute $c = ab$. To share a and b, we create two random vectors $R_a = (4)^T$ and $R_b = (1)^T$ and set the first element of R_a to a and the first element of R_b to b. The computed shares are then:

$$\begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} 3 \\ 4 \end{pmatrix} = \overbrace{\begin{pmatrix} 0 \\ 4 \\ 1 \\ 5 \end{pmatrix}}^{\text{Shares of a}} \text{ and } \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} 5 \\ 1 \end{pmatrix} = \overbrace{\begin{pmatrix} 6 \\ 0 \\ 1 \\ 2 \end{pmatrix}}^{\text{Shares of b}}$$

For each player P_i, we compute $c_i = a_i b_i$. That gives us $c_1 = 0$, $c_2 = 0$, $c_3 = 1$ and $c_4 = 3$. which we reshare to reduce its degree and add randomness. To do this, each player creates a random vector:

$$R_{p1} = (5)^T, R_{p2} = (1)^T, R_{p3} = (4)^T, R_{p4} = (2)^T$$

and, like before, we add the share c_i belonging to p_i as the first element of R_{p_i} and, using M, we create the new shares:

$$M \cdot \begin{pmatrix} 0 \\ 5 \end{pmatrix} = \overbrace{\begin{pmatrix} 5 \\ 3 \\ 1 \\ 6 \end{pmatrix}}^{\text{Shares of } c_1}, M \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \overbrace{\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix}}^{\text{Shares of } c_2}$$

$$M \cdot \begin{pmatrix} 1 \\ 4 \end{pmatrix} = \overbrace{\begin{pmatrix} 5 \\ 2 \\ 6 \\ 3 \end{pmatrix}}^{\text{Shares of } c_3}, M \cdot \begin{pmatrix} 3 \\ 2 \end{pmatrix} = \overbrace{\begin{pmatrix} 5 \\ 0 \\ 2 \\ 4 \end{pmatrix}}^{\text{Shares of } c_4}$$

	p_1	p_2	p_3	p_4
p_1	5	1	5	5
p_2	3	2	2	0
p_3	1	3	6	2
p_4	6	4	3	4

Table A.1: Shares of c_{ij}

P_i then sends P_j his respective share of c_{ij}. Table A.3.1 illustrates which players have which shares after this step completes.

Each player can then compute

$$\tilde{c_j} = \sum_{i=1}^{n} \lambda_i c_{ij}$$

which gives us

$$\begin{aligned}
\tilde{c_1} &= \langle \lambda, [5,1,5,5]^T \rangle = 71 \; mod \; 7 = 1 \\
\tilde{c_2} &= \langle \lambda, [3,2,2,0]^T \rangle = 22 \; mod \; 7 = 1 \\
\tilde{c_3} &= \langle \lambda, [1,3,6,2]^T \rangle = 43 \; mod \; 7 = 1 \\
\tilde{c_4} &= \langle \lambda, [6,4,3,4]^T \rangle = 64 \; mod \; 7 = 1
\end{aligned}$$

The shares $\tilde{c_1}$ determine c

$$\langle \lambda, [1,1,1,1]^T \rangle = 15 \; mod \; 7 = 1$$

completing the computation.

A.3.2 Commitment Example

Suppose the dealer D wants to commit to a value $s = 5$. Given the MSP $\mathcal{M} = (\mathcal{F}, M, \psi, (1, 0, \ldots, 0)^T)$ where $\mathcal{F}$ is $GF(7)$ and M is a $d \times e$ matrix:

$$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 2 \end{pmatrix} \begin{array}{c} \overbrace{P_1}^{\psi} \\ P_2 \\ P_3 \end{array}$$

Create a random symmetric $e \times e$ matrix R and set the first element of R to s:

$$R = \begin{pmatrix} s & 1 & 1 \\ 1 & 2 & 5 \\ 1 & 5 & 1 \end{pmatrix}$$

Let $\mathbf{v}_i$ denote the row in M belonging P_i. For each player P_i, D sends the vector $\mathbf{u}_i = R\mathbf{v}_i$ to P_i.

$$R\overbrace{\begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}}^{\mathbf{v}_1} = \overbrace{\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}}^{\mathbf{u}_1}, R\overbrace{\begin{pmatrix} 1 \\ 2 \\ 4 \end{pmatrix}}^{\mathbf{v}_2} = \overbrace{\begin{pmatrix} 4 \\ 4 \\ 1 \end{pmatrix}}^{\mathbf{u}_2}, R\overbrace{\begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}}^{\mathbf{v}_3} = \overbrace{\begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix}}^{\mathbf{u}_3}$$

Now suppose player P_1 and P_3 wish to check the consistency of their shares. First P_1 calculates the value:

$$\langle \mathbf{v}_3, \mathbf{u}_1 \rangle = \langle (1,3,2), (0,1,0) \rangle = 3$$

and sends it to P_3. P_3 calculates the value:

$$\langle \mathbf{v}_1, \mathbf{u}_3 \rangle = \langle (1,1,1), (3,3,4) \rangle = 3$$

and sends it to P_3. They then compare the values and see if they match, confirming whether or not their shares are consistent.

A.4 Definitions

A.4.1 The Kronecker product

The **Kronecker product**, or *tensor product*, denoted $\otimes$, is an operation on two arbitrarily sized matrices that results in a third block matrix.

Definition Given an $m \times n$ matrix A and a $p \times q$ matrix B, the Kronecker product is an $mp \times nq$ matrix:

$$A \otimes B = \begin{pmatrix} A_{1,1}B & \cdots & A_{1,n}B \\ \vdots & \ddots & \vdots \\ A_{m,1}B & \cdots & A_{m,n}B \end{pmatrix} \tag{A.4}$$

A.4.2 Adjacency Matrix

Definition An adjacency matrix M, for an unweighted and undirected graph $G = (V, E)$, is a $|V| \times |V|$ symmetrical matrix where:

$$M_{u,v} = \begin{cases} 0 & \text{if } (u, v) \notin E, \\ 1 & \text{if } (u, v) \in E. \end{cases}$$

We assume that vertices of G are never connected to themselves, so $M_{i,i} = 0 \ \forall \ i$.

A.4.3 Homomorphic Encryption

Definition *Homomorphic encryption* is an encryption scheme that allows operations, such as multiplication and addition, to be performed on ciphertext values, resulting in a ciphertext that is equal to performing identical operations on the plaintext and then encrypting it.

Bibliography

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

[2] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution*. PhD thesis, Dept. of Computer Science, Technion, 1996.

[3] Amos Beimel, Anna Gal, and Michael S. Paterson. Lower bounds for monotone span programs. In *IEEE Symposium on Foundations of Computer Science*, pages 674–681, 1995.

[4] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10, New York, NY, USA, 1988. ACM Press.

[5] J. Benaloh and J. Leichter. Generalized secret sharing and monotone functions. In *CRYPTO '88: Proceedings on Advances in cryptology*, pages 27–35, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[6] G.R. Blakley. Safeguarding cryptographic keys. *AFIPS Conference Proceedings*, 48:313–317, 1979.

[7] Ernest 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*, pages 468–475, New York, NY, USA, 1990. Springer-Verlag New York, Inc.

[8] Ernest F. Brickell and Daniel M. Davenport. On the classification of ideal secret sharing schemes (extended abstract). In *CRYPTO '89: Proceedings on Advances in cryptology*, pages 278–285, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[9] R. Cramer and I. Damgard. Multiparty computation - an introduction. 2002.

[10] R. Cramer, I. Damgard, and U. Maurer. Span programs and general secure multi-party computation, 1998.

[11] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret sharing scheme. Cryptology ePrint Archive, Report 2000/037, 2000.

[12] Paolo D'Arco and Douglas R. Stinson. On unconditionally secure robust distributed key distribution centers. In *ASIACRYPT '02: Proceedings of the 8th International Conference on the Theory and Application of Cryptology and Information Security*, pages 346–363, London, UK, 2002. Springer-Verlag.

[13] Serge Fehr. Span programs over rings and how to share a secret from a module. Master's thesis, ETH Zurich, Institute for Theoretical Computer Science, 1998.

[14] Serge Fehr. Efficient construction of the dual span program. Manuscript, May 1999.

[15] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *Proc. 28th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 427–437, 1987.

[16] Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. General adversaries in unconditional multi-party computation. In *ASIACRYPT*, pages 232–246, 1999.

[17] Anna Gal. A characterization of span program size and improved lower bounds for monotone span programs. In *ACM Symposium on Theory of Computing*, pages 429–437, 1998.

[18] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[19] M. Ito, A. Saito, and T. Nishizeki. Secret sharing scheme realizing general access structure. In *Proceedings IEEE Globecom '87*, pages 99–102. IEEE, 1987.

[20] Mauricio Karchmer and Avi Wigderson. On span programs. In *Proceedings of the 8th Structures in Complexity*, pages 102–111, 1993.

[21] S C Kothari. Generalized linear threshold scheme. In *Proceedings of CRYPTO 84 on Advances in cryptology*, pages 231–241, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[22] C. L. Liu. *Introduction to Combinatorial Mathematics.* McGraw-Hill, New York, 1968.

[23] Ueli Maurer. Secure multi-party computation made simple. *Discrete Appl. Math.*, 154(2):370–381, 2006.

[24] M. Mignotte. How to share a secret? In Thomas Beth, editor, *Cryptography - Proceedings of the Workshop on Cryptography, Burg Feuerstein, Germany*, pages 371–375, Berlin, 1983. Springer-Verlag. Lecture Notes in Computer Science Volume 149.

[25] C. Andrew Neff. A verifiable secret shuffle and its application to e-voting. In *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125, New York, NY, USA, 2001. ACM Press.

[26] Ventzislav Nikov and Svetla Nikova. New monotone span programs from old. Cryptology ePrint Archive, Report 2004/282, 2004.

[27] Ventzislav Nikov, Svetla Nikova, and Bart Preneel. On multiplicative linear secret sharing schemes. In *INDOCRYPT*, pages 135–147, 2003.

[28] Ventzislav Nikov, Svetla Nikova, Bart Preneel, and Joos Vandewalle. Applying general access structure to metering schemes. Cryptology ePrint Archive, Report 2002/102, 2002.

[29] Ventzislav Nikov, Svetla Nikova, Bart Preneel, and Joos Vandewalle. On distributed key distribution centers and unconditionally secure proactive verifiable secret sharing schemes based on general access structure. In *INDOCRYPT '02: Proceedings of the Third International Conference on Cryptology*, pages 422–436, London, UK, 2002. Springer-Verlag.

[30] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO '91: Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, pages 129–140, London, UK, 1992. Springer-Verlag.

[31] S. Radomirovi. Investigations into span programs with multiplication. Master's thesis, Institute for Theoretical Computer Science, ETH Zurich, 1998.

[32] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[33] Douglas R. Stinson. *Cryptography: Theory and Practice (Discrete Mathematics and Its Applications)*. CRC Press, March 1995.

[34] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd IEEE Symposium on Foundations of Computer Science (FOCS '82)*, pages 160–164, 1982.