Dissertation

on

Multi Party Computation

Thesis Submitted in the partial fulfillment of the requirements for the degree of

Master of Technology

in

Computer Science and Engineering

Submitted by

ACHYUT GHOSH

17CS4307

Under the guidance of

Dr. Jaydeep Howlader

Department of Computer Science and Engineering
National Institute of Technology Durgapur
M. G. Avenue, A-Zone, Durgapur - 713209
West Bengal, India

May 2019

Certificate Of Approval

The forgoing thesis is hereby approved as a creditable study of Technological subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite with degree for which it has been submitted. It is to be understood that by this approval, the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approve the thesis only for the purpose for which it has been submitted.

Board of Thesis Examiner

-
-
-
-

Declaration

I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which has been accepted for the award of any other degrees or diplomas of the university or other institutes of higher learning, except which due acknowledgment has been maid in this text.

.....

Achyut Ghosh
National Intitute of Technology,Durgapur

Certificate

This is to certify that the thesis entitled **"Multi Party Computation"** *submitted by Mr Achyut Ghosh to the National Institute of Technology,Durgapur, India towards partial fulfilment of the requirments for the award of Degree of Master of Technology in Computer Science and Engineering with specialization in Information Technology is a bonafide record of the work carried out by him under my supervision and guidance.*

.....
(Signature of Project Guide)
Asst.Prof. Jaydeep Howlader
Department of Computer Science and Engineering

.....
(Signature of HoD)
Prof. Tandra Pal
Department of Computer Science and Engineering

Acknowledgement

I hereby wish to express my sincere gratitude and respect to Assistant Prof. Jaydeep Howlader, Dept. of Computer Science and Engineering, NIT, Durgapur, under whom I had proud privilege to work. His valuable guidance and encouragement have really led me to the path of completion of this project. Any amount of thanks would not be enough for the valuable guidance of my supervisor. It was thus that working under his expertise and receiving a part of his tremendous knowledge became rewarding experience.

I would also like to thank all the faculty members of CSE dept. for their devoted help. I also cordially thank all laboratory assistants for their co-operation in the Computer laboratory.

Finally, I would like to pen down my gratitude towards my family members for their continuous support and encouragement.It would have not been possible to complete my work with out their support.

.....

Achyut Ghosh
Department of Computer Science and Engineering
National Intitute of Technology,Durgapur

Abstract

Today, the world is full of secrets. Sometimes, we may like to know something about them without revealing the secrets themselves.For example, whether I have more money than my friend or whether two satellites would collide without publishing their moving trajectories. Secure Multi Party Computation allowes us to jointly compute some functions while keeping the privacy of our inputs.
Multi Party Computation is based on Secret Sharing , which is a thechnique for protecting sensetive data such as Cryptographic keys. It is used to distribute a secret value to a number of parts-shares-that have to be combinrd together to access the original value. The first secret sharing schemes were proposed by Shamir and Blakley. This work describes t-out-of-n threshold secret sharing scheme and its properties.
In secure multi-party shuffling, multiple parties, each holding an input, want to agree on a random permutation of their inputs while keeping the permutation secret. This problem is important as a primitive in many privacy-preserving applications such as anonymous communication, location-based services, and electronic voting. We have used multi party shuffling to implement a mix network.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In todays world, we have a huge amount of information. That data could be used to figure out trends which could, for example, allow us to make wiser business decisions. If we would live in a world without secrets and where everyone trusts each other, then we could simply publish all the information and analyse it. In the real world, however, there are many things, that people consider private, such as their medical or financial details. Companies have business secrets, which they do not want to reveal either. Therefore, it would be great, if there would be a way to analyse data without compromising anyones privacy. The latter is exactly what *Secure Multi Party Computation* (SMPC) allows us to do.

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.

In the general case we have N parties $P_1, P_2, ..., P_N$ with private inputs $x_1, x_2, .. ., x_N$ respectively. The goal is to compute a function $f(x_1, x_2, ..., x_N)$ and learn nothing more than what they would have if a separate trusted party had collected their inputs, computed function f for them, and the return the result to all parties.

There are two important requirements on any protocol for secure computation, namely *privacy* and *correctness* . The privacy requirement suggests that nothing but what is absolutely essential should be learned. More specifically, all parties should learn nothing more but the computation output. The correctness requirement states that every party should receive the correct computation output, that is an adversary should not be able to cause the result of the computation to be different than the outcome of the function the parties agreed to compute. 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 [18] and Blakley [3], was first designed for key safeguarding, but has since been applied far beyond its original intent. Today, secret sharing is an important feature in secure multi-party computation [12], electronic voting [14] and distributed key distribution [15].
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 [19]. 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 History and Literature

Secure multi-party computation was introduced in 1982 by Andrew Yao[19]. As a first general solution, Goldreich, Micali, and Wigderson [10] presented a passively secure protocol that allows n players to securely compute any given function even if a passive adversary corrupts any $t < n$ players, and an actively secure protocol that tolerates an active adversary corrupting any $t < n/2$ of the players. The security of the protocols is cryptographic, that is the adversary is assumed to be polynomially bounded. Chaum, Damgrd, and van de Graaf[6] improved the bound for the active model in the sense that the input of one player can even be information-theoretically hidden. Galil, Haber, and Yung [9] considered efficiency and several corruption types in the cryptographic model. Ben-Or,

Goldwasser and Wigderson[2] proved that in the secure-channels model without broadcast, perfect security for n players can be achieved even if the adversary can corrupt any set of less than $n/2$ players (passive case) or, alternatively, any set of less than $n/3$ players (active case). These bounds are tight. The same results were obtained independently by Chaum, Crpeau and Damgrd [5] in an unconditional model with exponentially small error probability. The bound for the active model was improved by Rabin and Ben-Or [17] and independently by Beaver [1] by assuming a broadcast channel and tolerating a negligible error probability. They proposed protocols that provide unconditional security against an active adversary that may corrupt any $t < n/2$ of the players. Combining the advantages of unconditional security (against an adversary that corrupts a certain fraction of the players) and cryptographic security (against an adversary with limited computing power), Chaum [4] presented a protocol which provides unconditional security against an adversary that is limited by the number of players he can corrupt, and provides cryptographic security against an adversary who may corrupt any number of players.

Secret-sharing is one of the central ingredients of almost any MPC protocol. First secret-sharing schemes have been reported independently by Blakley [3] and Shamir [18].

Chapter 2

Cryptographic Tools and Mathematical Foundation

In this chapter, the existing cryptographic tools, that are used in our protocol, are introduced. Some of the basic mathematical properties are also discussed.

2.1 Group

A **Group** $(\mathbb{G}, \oplus)$ is a non-empty set $\mathbb{G}$ which is *closed* under a binary operation $\oplus$ (that is, for any $x, y \in \mathbb{G}, x \oplus y \in \mathbb{G}$) and also satisfies the following properties:

1. **Identity** There is an element e in $\mathbb{G}$, such that for every $x \in \mathbb{G}, e \oplus x = x \oplus e = x$.

2. **Inverse** For every x in $\mathbb{G}$ there is an element $y \in \mathbb{G}$ such that $x \oplus y = y \oplus x = e$, then y is called the inverse element of x, where again e is the identity.

3. **Associativity** The following identity holds for every $x, y, z \in \mathbb{G}$:

$$x \oplus (y \oplus z) = (x \oplus y) \oplus z$$

- A group $\mathbb{G}$ is called *abelian (or commutative)* group if $a \oplus b = b \oplus a$, for all $a, b \in \mathbb{G}$.

- The order of a group $\mathbb{G}$ is the cardinality of $\mathbb{G}$.

2.2 Ring

A **Ring** $(\mathbb{R}, \oplus, \otimes)$ is a non-empty set $\mathbb{R}$ which is *closed* under two operations $\oplus$ and $\oplus$ and satisfying the following properties:

1. $(\mathbb{R}, \oplus)$ forms an abelian group.
2. $\otimes$ operator is associative in $\mathbb{R}$. For every $a, b, c \in \mathbb{R}$,

$$a \otimes (b \otimes c) = (a \otimes b) \otimes c$$

3. $\otimes$ is distributive over $\oplus$. For every $a, b, c \in \mathbb{R}$ the following identities hold:

$$a \otimes (b \oplus c) = (a \otimes b) \oplus (a \otimes c)$$

   and

$$(b \oplus c) \otimes a = b \otimes a \oplus c \otimes a$$

4. There exists a multiplicative identity e_1 (e_1 is not equals to the additive identity e_0).

2.3 Field

A **Field** $(\mathbb{F}, \oplus, \otimes)$ is a commutative ring where all elements $a \in \mathbb{F}$ have multiplicative inverse expect e_0(additive identity). Field has the following properties:

1. $\mathbb{F}$ is an abelian group under $\oplus$.
2. The set $\mathbb{F}$ without the additive identity e_0 is an abelian group under $\otimes$.

2.4 Finite Fields

Finite fields are used in Shamir secret sharing and in MPC, thus a brief description is given. A finite field, also called a *Galois field*(GF), is a field with a finite number of elements. The order of a finite field is always a prime or a power of a prime.

$GF(p)$ also denoted as $\mathbb{F}_p$, where p is a prime number, is simply the field of integers modulo p, is called **Prime Field**. This means one can perform operations (addition, subtraction, multiplication) as usually done on integers, followed by reduction modulo p. For instance, in $GF(7)$, $5 + 4 = 9$ is reduced to 2 modulo 7.

The reason for working modulo p is to achieve *perfect secrecy,* that is, given fewer than threshold t shares, the secret can be any element in the field and thus those shares do not supply any additional information about the secret. Simply working over all the reals makes it impossible to state the same qualities, because the shares cannot be uniformly distributed over the reals.

2.5 Mathematical foundations of secret sharing

2.5.1 Polynomial evaluations

We start by describing basis properties of polynomials. Let us consider a field $\mathbb{F}$ and denote the set of all polynomials over $\mathbb{F}$ by $\mathbb{F}$ [x]. Let $f(x) = f_0+f_1x+...+f_kx^k$ be a polynomial in $\mathbb{F}$[x]. We also fix a vector $a = [a_0, ..., a_n] \in \mathbb{F}^n$ so that all values a_i are different and nonzero. Then we define the polynomial evaluation mapping $\mathbf{eval} : \mathbb{F}[x] \to \mathbb{F}^n$ as follows. We evaluate the polynomial $f(x)$ on the vector a and present the result as a vector.
$\mathbf{eval}(f) := [f(a_0), ..., f(a_n)]^T$.

In the following theorem operations between vectors in $\mathbb{F}^n$ are defined elementwise. That is, if $u, v \in \mathbb{F}^n$ and $\oplus$ is a binary operator, then:
$u \oplus v := [(u_1 \oplus v_1), ..., (u_n \oplus v_n)]^T$

Theorem . *For any two polynomials f and $g \in \mathbb{F}[x]$ and a scalar value $r \in \mathbb{F}$ the following conditions hold:*

1. *Additivity* : $\mathbf{eval}(f + g) = \mathbf{eval}(f) + \mathbf{eval}(g)$.
2. *Multiplicativity* : $\mathbf{eval}(f \cdot g) = \mathbf{eval}(f) \cdot \mathbf{eval}(g)$.
3. *Multiplicativity w.r.t. scalar values:* $\mathbf{eval}(r \cdot f) = r \cdot \mathbf{eval}(f)$

*The mapping **eval** is a linear transformation.*

Proof. The conditions hold because of the duality with the respective polynomial operations:

1. Additivity: $(f + g)(a) = f(a) + g(a)$
2. Multiplicativity: $(f \cdot g)(a) = f(a) \cdot g(a)$
3. Multiplicativity w.r.t. scalar values: $(r \cdot f)(a) = r \cdot f(a)$

The conclusion that the mapping is a linear transformation directly follows from the above conditions. Thus we have shown that **eval** is a linear mapping between the evaluation positions of the polynomial and the result vector.

Theorem (Lagrange interpolation theorem). *Let $\mathbb{F}$ be a field and $a_0, \ldots, a_n, y_0, \ldots, y_n \in \mathbb{F}$ so that all values a_i are distinct. Then there exists only one polynomial f over $\mathbb{F}$ so that deg $f \leq n$ and $f(a_i) = y_i, (i = 0, \ldots, n)$.*

The Lagrange interpolation polynomial can be computed as the sum

$$f(x) = y_0 b_0(x) + \cdots + y_n b_n(x)$$

where the base polynomials $b_i(x)$ are defined as

$$b_i(x) = \prod_{j=0, i \neq j}^{n} \left(\frac{x - a_j}{a_i - a_j}\right)$$

2.6 Basic idea of Mix Network

Mix networks are routing protocols that create hard-to-trace communications by using a chain of proxy servers known as *mixes* which take in messages from multiple senders, shuffle them, and send them back out in random order to the next destination (possibly another mix node). This breaks the link between the source of the request and the destination, making it harder for eavesdroppers to trace

end-to-end communications. Furthermore, mixes only know the node that it immediately received the message from, and the immediate destination to send the shuffled messages to, making the network resistant to malicious mix nodes.

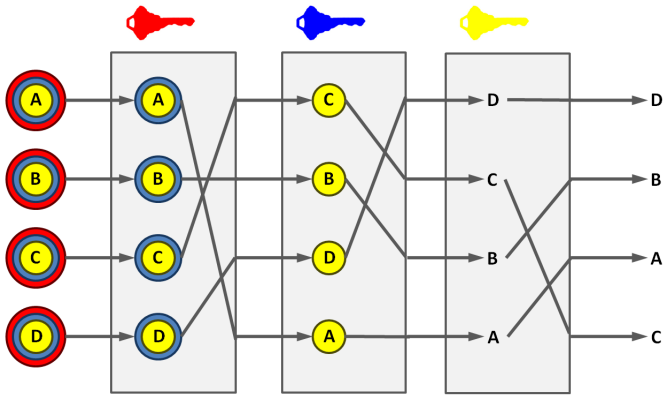

Figure 2.1: Simple decryption mix net. Messages are encrypted under a sequence of public keys. Each mix node removes a layer of encryption using its own private key. The node shuffles the message order, and transmits the result to the next node.

Each message is encrypted to each proxy using public key cryptography; the resulting encryption is layered like a Russian doll (except that each "doll" is of the same size) with the message as the innermost layer. Each proxy server strips off its own layer of encryption to reveal where to send the message next. If all but one of the proxy servers are compromised by the tracer, untraceability can still be achieved against some weaker adversaries. The concept of mix networks was first described by David Chaum in 1981 [7].

Chapter 3

Secure Multiparty Computation

This chapter addresses secure multiparty computation (MPC) which is the problem of how mutually distrusting parties can compute a function without revealing their individual input values to each other. More formally the problem of multiparty computation is defined as follows [8]:

Definition *Secure multiparty computation (MPC) can be defined as the problem where n participants* $P_1, \ldots, P_n$ *agree on a function* f *and wish to compute and reveal to each participant* $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n)$*, where each input* x_i *is a secret input provided by participant* P_i *. Player* P_i *will only learn* y_i *, nothing more.*

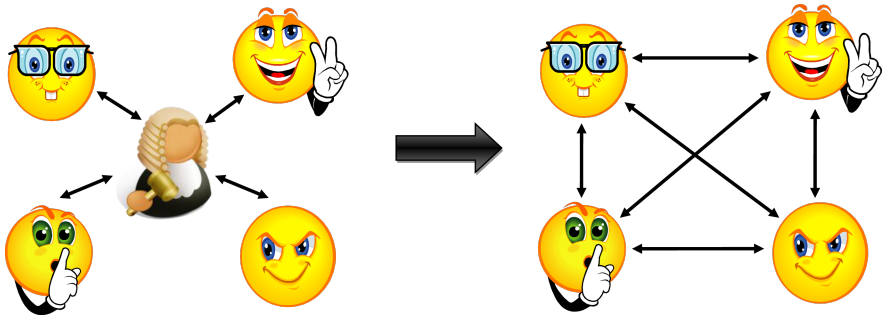

Figure 3.1: A trusted party scenario to the left and the idea of MPC (simulation of a trusted party) to the right.

Security in this context means guaranteeing the correctness of the output as well as the privacy of the participants inputs, even if some participants try to cheat.

3.1 Adversaries

Participants in a MPC protocol do not necessarily behave as intended. Corrupt players may occur and they are divided into the following two main groups [16]:

- A *passive adversary* is where players perform the protocol correctly, but collaborate in information gathering and sharing with the goal of getting access to sensitive information of other players. Such players are sometimes called *semihonest*.
- An *active adversary* is a group of players deviating from the protocol in order to disrupt the computation. The goal is to produce incorrect results and/or violate the privacy of other players. Such players are sometimes called *byzantine*.

Both types can be *static* or *adaptive*. A *static adversary* means that the set of corrupted players is chosen before the protocol starts. An *adaptive adversary* on the other hand, can choose which players to corrupt during the execution of the protocol.

3.2 Network Assumptions

Traditionally secure multiparty computations were done with the assumption of a *synchronous* network, with a known maximum drift rate. In this setting it is easy to divide a protocol into logical units called rounds. In each round, each player may send a message to each other and the messages are delivered before the next round begins.

Modern networks, like the Internet, are though mostly *asynchronous*. Data is transmitted without the use of an external clock signal and problems like congestion, delay, lost packets and other transmission errors might occur.

In addition the players communicate with each other over channels. Many MPC protocols assume the existence of *pairwise* channels among the players. For a MPC protocol to be information-theoretically secure, it is assumed that there exists an unconditionally secure channel between each of the participants.

3.3 Security

This section describes some classical results on security and threshold adversaries. Generally, the security can be divided into two groups [11]:

1. *Information-theoretically secure MPC* is unconditionally secure, i.e., it is not possible to cheat or violate the security even if an adversary has unrestricted computing power. It can be further classified into the following levels:

   (a) *Perfect security:* The result of a real execution of the protocol with a real adversary must be exactly the same as the result of an ideal execution with a trusted party and an ideal adversary.

   (b) *Statistical security:* The result of the real protocol execution need only be statistically close to the result of an ideal execution.

2. *Cryptographically secure MPC* is based on unproven cryptographic primitives that are assumed to be computationally infeasible.

Chapter 4

Secret Sharing Schemes

4.1 Introduction

In this chapter theory related to secret sharing is presented. Secret sharing is closely related to MPC and a few such schemes are thus described. Shamir secret sharing will be the most important as it is used in MPC.

Imagine setting up a launch program for a nuclear missile. You do not want to give a single person this responsibility. In order to launch the missile it is desirable that two people will need to enter a password and turn their keys at the same time. This example is comparable to the concept of secret sharing. More formally:

Definition 1 *A secret sharing scheme refers to a method which distributes shares of a secret among a group of participants in such a way that the secret can only be reconstructed when the shares are combined together.*

4.2 Additive Secret Sharing

There exist several secret sharing schemes which differ in theory and application. The most trivial ones require all n participants in order to reconstruct the secret. Additive secret sharing is an example of such a scheme and is described next.

Assume we wish to split the secret s among n people, then $n1$ random numbers $r_1, \ldots, r_{n1}$ need to be selected. It is impossible to choosea random integer in a way that all integers are equally likely (the sum of the infinitely many equal probabilities, one for each integer, cannot be 1). Therefore an integer p larger than all possible messages that might occur is selected. Then s and r are regarded as numbers mod p . The share for player P_n is computed:

$$s_n = s - \sum_{i=1}^{n-1} r_i \mod p$$

The rest of the players $P_i, 1 \leq i \leq n-1$, get the shares $s_i = r_i$. In order to reconstruct the secret, the sum of all shares needs to be computed:

$$s = \sum_{i=1}^{n} s_i \mod p$$

The above technique is absolutely secure if done properly. Each piece by itself is totally worthless, but together the players can reconstruct the secret. A scheme characterized by this property is also the definition of a *perfect secret sharing scheme.*

4.2.1 Creating the Shares

Below is a simple example to better understand how additive secret sharing really works. Assume we have a $secret = 13$ which should be secret shared among $n = 3$ participants. We define an integer $p = 20$ to work over, together with $n-1$ random numbers $r_1 = 3$ and $r_2 = 8$. The following calculations are then possible:

$$s_1 = r_1 \mod p = 3 \mod 20 = 3$$

$$s_2 = r_2 \mod p = 8 \mod 20 = 8$$

$$s_3 = s \textstyle\sum_{i=1}^{n-1} r_i \mod p = 13(3+8) \mod 20 = 2$$

4.2.2 Reconstructing the Secret

Reconstructing the secret is easy, just add all shares together and we see that the sum 13 is equal to the original secret.

$$s = \sum_{i=1}^{n} s_i \mod p = 3 + 8 + 2 \mod p = 13$$

One problem with this protocol is that with a missing share, no one can reproduce the secret. Schemes that solve this issue are called threshold schemes and are described next.

4.3 Shamir Secret Sharing Scheme

In difference to additive secret sharing, threshold schemes allow a subset of the players to reconstruct the secret. More formally a (t, n)-threshold scheme is defined as follows:

Definition 2 *A (t,n)-threshold scheme is a method of sharing a secret s among a set of n participants such that any subset consisting of t participants can reconstruct the secret s, but no subset of smaller size can reconstruct s.*

1. **Correctness:** s is uniquely determined by any t shares from $\{s_1, \ldots, s_n\}$ and there exists an algorithm that efficiently computes s from these t shares.
2. **Privacy:** having access to any $t - 1$ shares from $\{s_1, \ldots, s_n\}$ gives no information about the value of s, i.e., the probability distribution of $t - 1$ shares is independent of s.

In 1979, both Adi Shamir [18] and George Blakley [3] independently presented such threshold schemes. Shamirs method is using polynomial interpolation, while Blackley works on the intersection of hyperplanes. Only Shamirs secret sharing will be described here, as this is the scheme implemented in MPC.

4.3.1 Overview

In order to illustrate Shamir secret sharing, a $(2, n)$ scheme is designed. Assume a secret is to be shared among n participants. In Figure 4.1 the point $(0, s)$ on the y-axis, which corresponds to

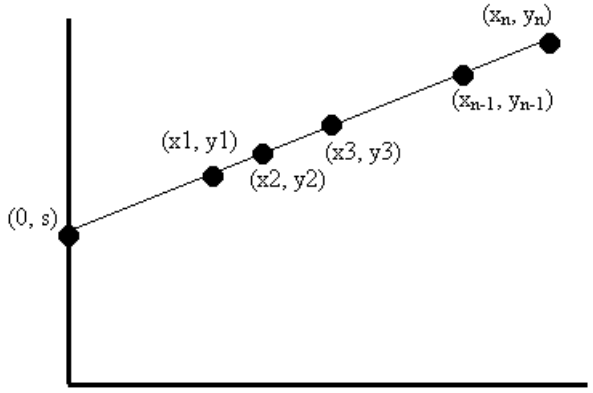

Figure 4.1: The linear curve corresponding to a (2,n)-threshold scheme

the secret, is selected. Drawing a random line through this point gives n points on that line. Each point $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ represents a share.

Since two points determine a line, it is clear that two participants can discover the secret by drawing a line between their points, and from there check where the line intersects the y axis. If holding only one share, infinite possibilities would exist for a line through this point. This means that with one point, no information about the secret is revealed.

Extending the idea to a $(3, n)$ scheme, a curve determined by three points is needed. This corresponds to the quadratic function $y = a_0 + a_1 \cdot x + a_2 \cdot x^2$. Again the secret is

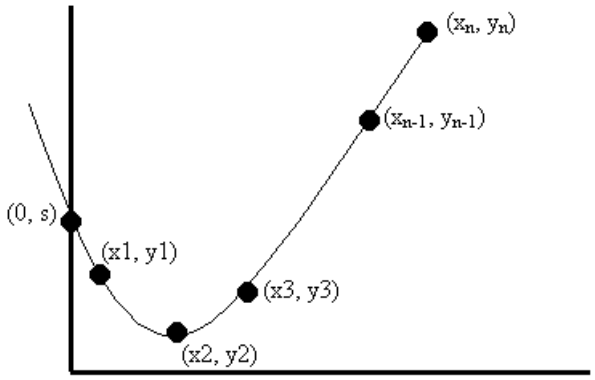

Figure 4.2: The quadratic curve corresponding to a (3,n)-threshold scheme

located on the y-axis before drawing a random curve corresponding to a quadratic function that goes through the point. Finally n points are selected as seen in Figure 4.2 which all represent a share.

4.3.2 Creating the Shares

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

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

$$f(x) = s + r_1 x + r_2 x^2 + \cdots + r_{t-1} x^{t-1} \mod p$$

subject to the following conditions:

- $p > n$ and $\mathbb{F}_p$ is a prime field for some prime number p.
- The secret $s \in \mathbb{F}_p$.
- The threshold $t \leq n$.
- The coefficients $\{r_1, \ldots, r_{t-1}\}$ are chosen independently and randomly from the interval $[0, p)$.

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 $p = 17, s = 4, t = 3$, and $r_{1..t1} = \{3, 6\}$. Our polynomial is then, as stated in the equation for $f(x)$,

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

and some of our secret shares are:

$$s_1 = f(1) = 4 + 3 \cdot 1 + 6 \cdot 1^2 \mod 17 = 13$$

$$s_2 = f(2) = 4 + 3 \cdot 2 + 6 \cdot 2^2 \mod 17 = 0$$

$$s_3 = f(3) = 16$$

$$s_7 = f(7) = 13$$

4.3.3 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:

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

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

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

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

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

we see that b_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 b_i and use them later when we recombine the secret.

Definition The vector $b = \{b_1, \ldots, b_n\}$ such that $s = \sum_{i=1}^{t} s_i b_i$ 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 the equation of $L(0)$, and three of the

shares created (for example s_1, s_2, and s_7) our secret is

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

so we have reconstructed the secret, $s = 4$.

The algorithms for sharing the secret *Share* and reconstructing the secret *Recons* are discussed here:

Algorithm 1: Generate Shamir's Share $[s] \leftarrow Share(\mathbb{F}_p, t, n, s)$

Data: Number of parties n, threshold t, secret s and a prime field $\mathbb{F}_p$
Result: Share $[s]$

```
foreach i ∈ {1, 2, ..., n} do
    [s]_i ← s + Σ_{l-1}^{t} a_l i^l
    send [s]_i to party P_i
end
return [s]
```

Algorithm 2: Reconstruct Shamir's Share $s \leftarrow Recons(A, [s])$

```
for each party P_i ∈ A send [s]_i to all parties
all parties compute s ← Σ_{P_i∈A} [s]_i Π_{Pj∈A, j≠i} (-j)/(i-j)
return s
```

Chapter 5

Secure computation with shares

A secret sharing scheme is *homomorphic* if it is possible to compute new valid shares from existing shares.

Definition *Let s and t be two values and $[s] = [s_1, \ldots, s_n]$ and $[t] = [t_1, \ldots, t_n]$ be their shares. A secret sharing scheme is $(\oplus, \otimes)$-homomorphic if shares $[(s_1 \otimes t), \ldots, (s_n \otimes t)]$ uniquely determine the value $s \oplus t$.*

5.1 Basic Operations

We will now show what can be done with the shares once they have been distributed. We will investigate the possibility of using the *homomorphic* property of the secret sharing scheme to perform operations with the shares. In the following assume that a $t - out - of - n$ threshold scheme is used. Assume that we have n parties $P_1, \ldots, P_n$ and they have their secret shares from a prime field $\mathbb{F}_p$.

5.1.1 Addition of two shares

Assume that we have shared values $[u] = [u_1, \ldots, u_n]$ and $[v] = [v_1, \ldots, v_n]$. Because the evaluation mapping is a linear transformation, we can add the shares

of $[u]$ and $[v]$ to create a shared value $[w]$ so that $u + v = w$.

Algorithm 3: Protocol for adding two Shamir shares $[w] \leftarrow Add([u], [v])$

Data: shares u_t and v_t
Result: share $[w]$ that represents the sum of $[u]$ and $[v]$
1 $w_t = u_t + v_t$

Each party t has to run the protocol given in Algorithm 3 to add two shared values.

5.1.2 Multiplication with a scalar

Assume that we have a shared value $[u] = [u_1, \ldots, u_n]$ and a public value c. Again, thanks to the linear transformation property of the evaluation mapping we can multiply the shares u_i with c so that the resulting shares represent the value $[w] = c[u]$.

Algorithm 4: Protocol for multiplying Shamir shares by a scalar value $[w] \leftarrow CMul(c, [u])$

Data: shares u_t and a public value c
Result: share w_t that represents the value of $c[u]$
1 $w_t = cu_t$

Algorithm 4 shows the protocol for multiplication a share value by a scalar.

5.1.3 Multiplication of two shares

Multiplication of secrets requires an interactive protocol, which is shown as Algorithm 5. The idea is from the multiplication protocol proposed in [2]. Given two secrets $[a], [b]$, each party P_i computes a share $[a]_i[b]_i$. As a secret $[a]$ is represented by a uniformly random t-degree polynomial f in Shamirs secret sharing scheme, the result polynomial of multiplication will have a degree of $2t$, and not uniformly random, thus $[a]_i[b]_i$ is not the correct share of $[ab]$.
To ensure the result polynomial has a degree of t, and be uniform random, the parties need to interactively compute a new random t-degree polynomial h, where $h(0) = ab$. The restriction $2t+1 \leq n$ must hold, otherwise it would be impossible

to reconstruct the result from the $2t$-degree polynomial.

Algorithm 5: Multiplication of secrets $[c] \leftarrow Mul([a], [b])$

Data: $[a], [b]$, where $a, b \in \mathbb{F}_p$
Result: $[c]$, where $c = a \cdot b, c \in \mathbb{F}_p$
1 **foreach** $i \in \{1, 2, \ldots, 2t+1\}$ ***in parallel*** **do**
2 | P_i computes $z_i \leftarrow [a]_i \cdot [b]_i$
3 | P_i shares z_i into $[z_i] \leftarrow Share(z_i)$
4 **end**
5 **foreach** $j \in \{1, 2, \ldots, n\}$ ***in parallel*** **do**
6 | P_j computes $[c]_j \leftarrow \sum_{i=1}^{2t+1}([z_i]_j \prod_{l=1, l\neq i}^{2t+1} \frac{-l}{i-l})$
7 **end**

Complexity Analysis. Algorithm 5 for the multiplication of secrets requires one round and one invocation. Note that only the first $2t+1$ parties are involved in the computation of z_i , this can save some communication and computation cost. To further optimize communication and computation load, it is possible to select any arbitrary subset of $2t + 1$ parties.

5.2 Interactive Generation of Random Secret

Many MPC protocols require the ability to generate shared random secrets. There are two different techniques for the generation of random secrets.

5.2.1 Shared Random Element

Algorithm 6 is used to jointly compute a shared secret $[r]$ for some unknown $r \in \mathbb{F}_p$. The idea is to compute $r = \sum_{i=1}^{n} x_i$, where $x_i \in \mathbb{F}_p$ is chosen uniformly random by P_i. If atleast one party P_i is honest, then at least one x_i is uniformly random, thereby r is uniformly random.

Algorithm 6: Shared random element $[r] \leftarrow Rand(\mathbb{F}_p)$

Data: $\mathbb{F}_p$
Result: [r], where $r \in \mathbb{F}_p$
foreach $i \in \{1, 2, \ldots, n\}$ ***in parallel*** **do**
  P_i picks a random $x_i \in \mathbb{F}_p$
  P_i shares x_i into $[x_i] \leftarrow Share(x_i)$
end
$[r] \leftarrow \sum_{i=1}^{n} [x_i]$

Complexity Analysis. Protocol *Rand* requires one round and one invocation.

5.2.2 Shared Random Bit

Algorithm 7 generates a shared random bit $[b]$, where $b \in \{0, 1\} \subset \mathbb{F}_p$. This protocol requires a prime modulus p with the property that $p \equiv 3(mod\ 4)$. Algorithm *Rand* is used to generate a uniformly random value $[r]$ which is then squared and opened. If the product is zero, the protocol fails and retries, otherwise the parties compute the $v = u^{(p+1)/4} = r^{(p+1)/2} = r^{(p-1)/2} \cdot r$. Note that $r^{(p-1)/2}$ is the Legendre Symbol $\left(\frac{r}{p}\right) = \pm 1$ for $r \neq 0$, then $b = 2^{-1} \cdot (v^{-1} \cdot r + 1) = 2^{-1} \cdot (\pm 1 \cdot r^{-1} \cdot r + 1) = 2^{-1} \cdot (\pm 1 + 1)$. Thus $b = 1$ if $\left(\frac{r}{p}\right) = 1$, and $b = 0$ if $\left(\frac{r}{p}\right) = -1$.

Algorithm 7: Shared random bit $[b] \leftarrow Rand_2(\mathbb{F}_p)$

Data: $\mathbb{F}_p$, where $p \equiv 3(mod\ 4)$
Result: $[b]$, where $b \in \{0, 1\} \subset \mathbb{F}_p$
$[r] \leftarrow Rand(\mathbb{F}_p)$
$[u] \leftarrow Mul([r], [r])$
$u \leftarrow Recons([u])$
if $u = 0$ **then**
  abort and retry
else
  $v \leftarrow u^{(\frac{p+1}{4})}$
  $[b] \leftarrow 2^{-1} \cdot (v^{-1} \cdot [r] + 1)$
end

The value of Legendre symbol $\left(\frac{r}{p}\right)$ depends on whether r is quadratic residue or not, and the probability that $\left(\frac{r}{p}\right) = 1$ is $1/2$ for $r \in \mathbb{F}_p$, thus the bit b is uniformly random. The protocol fails with probability $1/p$, which is negligible

for a large p that $p > 2^k$, where k is a security parameter.

Complexity Analysis. Protocol $Rand_2$ requires three rounds and three invocations.

5.2.3 Shared Random Invertible Element

Algorithm 8 generates a non-zero shared random field element $[r]$, and optionally its inverse $[r^{-1}]$. The parties use the protocol $Rand$ to generate two shared random secrets $[x]$ and $[y]$, and then reveal their product $[x \cdot y]$. If the product is 0, the protocol fails and retries, while if not, a non-zero shared random field element $[r]$ and its inverse $[r^{-1}]$ can be obtained in a trivial way.
We use $Rand^*$ to denote the protocol that only generates a nonzero shared random field element $[r]$ without its inverse $[r^{-1}]$. Note that the protocol fails with probability $2/p$, when x or y is zero.

Algorithm 8: Shared random invertible element $[r], [r^{-1}] \leftarrow RandInv(\mathbb{F}_p)$

Data: $\mathbb{F}_p$
Result: $[r], [r^{-1}]$, where $r \in \mathbb{F}_p$
$[x] \leftarrow Rand(\mathbb{F}_p)$
$[y] \leftarrow Rand(\mathbb{F}_p)$
$[c] \leftarrow Mul([x], [y])$
$c \leftarrow Recons([c])$
if $c = 0$ **then**
  abort and retry
else
  $[r] \leftarrow [x]$
  $[r^{-1}] \leftarrow c^{-1} \cdot [y]$
end

Complexity Analysis. The random secrets $[x]$ and $[y]$ can be generated in parallel, thus the overall complexity is three rounds and four invocations.

Chapter 6

Mixing using Multi Party Shuffling

In secure multi-party shuffling (MPS) problem, a group of parties each holding an input value want to randomly permute their inputs while ensuring no party can map any of the outputs to any of the input holders better than can be done with a uniform random guess. A MPS protocol is a useful primitive for achieving privacy and robustness in many applications such as anonymous communication [7], location-based services, electronic voting [14], secure auctions , and general multi-party computation [12].

6.1 Previous work

Shuffling in the multi-party setting has already been studied, primarily in the context of mix-nets. As first defined by Chaum [7] , a mix-net consists of a chain of servers (called mixes) that randomly reorder a sequence of messages in a way that the correlation with the corresponding input messages remains hidden. To ensure honest behavior in the malicious setting, a verifiable shuffling technique is often used, where each mix is asked to prove correctness of its shuffles without leaking how the shuffle was performed.

Previously some multi party shuffling technique was designed based on sorting [13]. They have used two finite fields $\mathbb{F}_p$ and $\mathbb{F}_q$ for two primes p and q, where

$q < p$. Each party holds a secret value $x_i \in \mathbb{F}_p$ and a uniform and independent random value $r_i \in \mathbb{F}_q$. So every party has a pair of values $(r_i, x_i) \in \mathbb{F}_p \times \mathbb{F}_q$, which is sorted according to the first elements of the pairs. For sufficiently large prime q , the algorithm randomly shuffles the sequence of inputs $\prod(x_1, \ldots, x_n)$(where $\prod$ is a random shuffling) with high probability.

6.2 Multi Party Shuffling based on random seeds

6.2.1 Preliminaries

We will perform all operations under prime field $\mathbb{F}_p$. The operations on shares discussed in the earlier chapter will be required in this protocol. Such as, *Add*, Mul and $Rand_2$.

6.2.2 Our protocol for shuffling

Suppose there are n parties $P = (P_1, \ldots, P_n)$ and they have their *secrets* $s_i \in \mathbb{F}_p$ for a big prime number p. The secrets are shared among n parties using shamir's secret sharing protocol. Now, the parties want to $shuffle$ their shares such that no adversary can detect which share belongs to whom.

In $Shuff$ function, we have used a divide and conquer approach to call the *BinSwap* function. To shuffle the shares we need to generate $(n-1)$ number of random bits using $Rand_2$ protocol, where n is the number of parties. $Shuff$ function will be called in a recursive manner.

Algorithm 9: Shuffling of shares $(s_1, \ldots, s_n) \leftarrow Shuff(< s_1, \ldots, s_n >, P)$

Data: $\mathbb{F}_p$, where $p \equiv 3(mod\ 4)$, Shares $(s_1, \ldots, s_n) \in \mathbb{F}_p$
Result: Shuffled Shares

```
for i ≤ n − 1 do
    b[i] ← Rand_2
end
mid=(start+end)/2
if end = start + 1 then
    BinSwap(start, end, Binindex)
end
else if end ≥ start + 1 then
    for i <= mid − start do
        BinSwap(start + i, mid + 1 + i, Binindex)
    end
end
if start < mid then
    Shuff(start, mid, 2 ∗ Binindex + 1)
end
if mid + 1 < end then
    Shuff(mid + 1, end, 2 ∗ Binindex + 2)
end
```

In *BinSwap* function , we have used two shares and a random bit. This will be called upon two conditions as shown in the $Shuff()$ function. Based on the value of the random bit *swap* will occur.

$$[\bar{x}] = [b][x] + (1 - [b])[y]$$

$$[\bar{y}] = [b][y] + (1 - [b])[x]$$

Suppose *BinSwap* function is called with two shares $[x]$ and $[y]$. $[\bar{x}]$ and $[\bar{y}]$ are the updated shares after swapping. If the value of the random bit $[b]$ is 1 then $[\bar{x}]$ and $[\bar{y}]$ will retain $[x]$ and $[y]$ respictively. Otherwise, If the value of the random bit $[b]$ is 0, then swapping will be done.

Algorithm 10: Swapping of two shares $\leftarrow$ $BinSwap(indexA, indexB, Binindex)$

Data: Index of two parties and indes of random bit: $indexA$, $indexB$, $Binindex$

1 $P[indexA] =$
$\mathbf{Add}(\mathbf{Mul}(P[indexA], b[Binindex]), \mathbf{Mul}(P[indexB], (1 - b[Binindex])))$

2 $P[indexB] =$
$\mathbf{Add}(\mathbf{Mul}(P[indexB], b[Binindex]), \mathbf{Mul}(P[indexA], (1 - b[Binindex])))$

6.3 Mixing using shuffling

6.3.1 Architecture

We have introduced a new idea of mixing by using our shuffling protocol. In the existing setup of mix network the nodes are used to operate one after another which is the bottleneck of the system. If one node breaks down or performs some malicious operations, the total system gets corrupted. We have used shuffling to construct a new kind of mix network, which is synchronous *i.e.* each node works collaboratively.

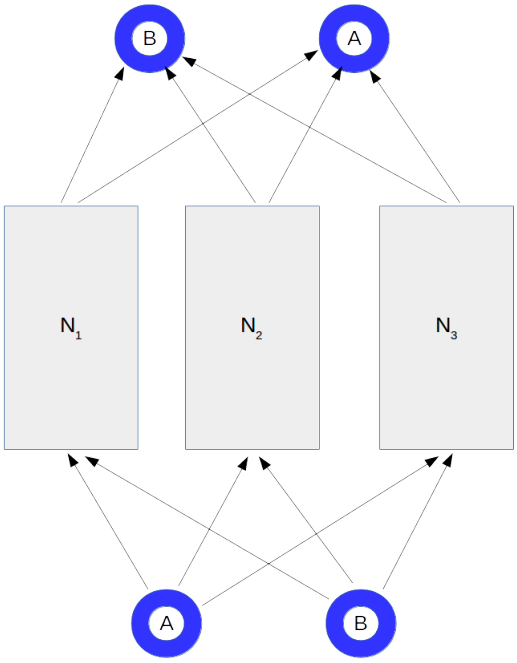

Figure 6.1: Architecture of Mixing using Shuffling

Suppose there are two parties with their secrets A and B from a prime field and there are three nodes in the mix net. The parties share their secrets to the nodes and get the shuffled output as shown in figure.

6.3.2 Clusturing and setup

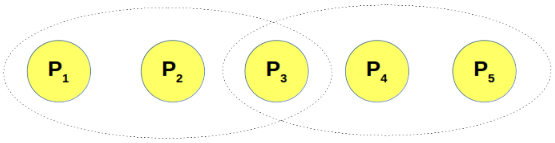

Figure 6.2: Creation of Clustures

Let there are n parties $P_1, \ldots, P_n$ with their secrets s_i (for $i = 1, 2, \ldots, n$) from a prime field $\mathbb{F}_p$. We have combined z (where $z = \lfloor n/2 \rfloor + 1$) number of parties to create a clusture, which will act like a node. We denote this clustures as $\mathcal{N}$. Let we have total k number of clustures, $\mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_k)$.

6.3.3 Algorithm

The input to the mixnet will be batch of tuples ,$Mix(B =< s_1, \ldots, s_n >, \mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_k))$. The algorithm is given below:

Algorithm 11: Mixing of shares $B_k \leftarrow Mix(B =< s_1, \ldots, s_n >, \mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_k))$

Data: Shares of n parties, $B =< s_1, \ldots, s_n >$ and the clustures, $\mathcal{N} = (\mathcal{N}_1, \ldots, \mathcal{N}_k))$
Result: Shuffled Shares B_k

```
B_0 = B
for i = 1 to k do
    B_i ← Shuff(B_{i-1}, N_i)
end
return B_k
```

So we get the shuffled shares B_k from B by using the Mix algorithm.

6.3.4 Result

Here we present the simulation of the mixing protocol on a stand-alone system[1]. We compute the time for single shuffle with varying the number of users. During our computation we consider:

1. All users (i.e. n) are participating in the mixing.
2. Number of parties are always odd.
3. Number of parties in a clusture is also odd.

No. of parties n	Clusture size z	Time for one shuffle t
5	3	0.4 seconds
9	5	0.2 seconds
13	7	0.7 seconds
17	9	2.0 seconds
21	11	4.5 seconds
25	13	9.0 seconds
29	15	18.0 seconds
33	17	33.0 seconds
37	19	53.0 seconds
41	21	1 minutes 22 seconds
45	23	2 minutes 09 seconds
49	25	2 minutes 58 seconds
53	27	4 minutes 18 seconds
57	29	5 minutes 40 seconds
61	31	8 minutes 19 seconds

Table 6.1: The simulation result of mixing. First column shows the number of parties n participated in mixing. The second column shows the clusture size z. The third column depicts the time for one shuffle operation

[1]Intel(R)Core(TM)i7CPU 2.70-2.70GHz with 8GB RAM

Chapter 7

Conclusion and Future Work

We have introduced a new idea of mixing by using multi party shuffling. This setup of mix network is more robust as it is synchronous. In our protocol every nodes participate in mixing simultaneously and work collaboratively.

Several open problems remain. First, we have to improve our architecture so that we can operate with more number of parties. Second, we have implemented the protocol in a stand-alone system. We need to implement the protocol in distributed platform for real-time operation.

References

[1] D. Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, 1991. 3

[2] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 1–10. ACM, 1988. 3, 21

[3] G. R. Blakley et al. Safeguarding cryptographic keys. In *Proceedings of the national computer conference*, volume 48, 1979. 2, 3, 14

[4] D. Chaum. The spymasters double-agent problem. In *Conference on the Theory and Application of Cryptology*, pages 591–602. Springer, 1989. 3

[5] D. Chaum, C. Crépeau, and I. Damgard. Multiparty unconditionally secure protocols. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 11–19. ACM, 1988. 3

[6] D. Chaum, I. B. Damgård, and J. Van de Graaf. Multiparty computations ensuring privacy of each partys input and correctness of the result. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 87–119. Springer, 1987. 2

[7] D. L. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. 8, 25

[8] I. Damgård. Theory and practice of multiparty computation. In *International Conference on Security and Cryptography for Networks*, pages 360–364. Springer, 2006. 9

[9] Z. Galil, S. Haber, and M. Yung. Cryptographic computation: Secure fault-tolerant protocols and the public-key model. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 135–155. Springer, 1987. 2

[10] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987. 2

[11] E. Kushilevitz, Y. Lindell, and T. Rabin. Information-theoretically secure protocols and security under composition. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 109–118, New York, NY, USA, 2006. ACM. 11

[12] U. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006. 2, 25

[13] M. Movahedi, J. Saia, and M. Zamani. Secure multi-party shuffling. In *International Colloquium on Structural Information and Communication Complexity*, pages 459–473. Springer, 2015. 25

[14] C. A. Neff. A verifiable secret shuffle and its application to e-voting. In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pages 116–125. ACM, 2001. 2, 25

[15] V. Nikov, S. Nikova, B. Preneel, and J. Vandewalle. On distributed key distribution centers and unconditionally secure proactive verifiable secret sharing schemes based on general access structure. In *International Conference on Cryptology in India*, pages 422–435. Springer, 2002. 2

[16] R. Oppliger. Contemporary cryptography, artech house. *Inc., Norwood, MA*, 2005. 10

[17] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 73–85. ACM, 1989. 3

[18] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979. 2, 3, 14

[19] A. C.-C. Yao. Protocols for secure computations. In *FOCS*, volume 82, pages 160–164, 1982. 2