Efficiency Optimizations on Yao's Garbled Circuits and Their Practical Applications

A thesis submitted to the
Graduate School of Natural and Applied Sciences

by

Osman BiÇER

in partial fulfillment for the
degree of Master of Science

in

CyberSecurity Engineering

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in CyberSecurity Engineering.

APPROVED BY:

Prof. Dr. Ensar Gül .....
(Thesis Advisor)

Dr. Mehmet Sabir Kiraz .....
(Thesis Co-advisor)

Assoc. Prof. Amir Azaron .....

Asst. Prof. Erdinç Öztürk .....

This is to confirm that this thesis complies with all the standards set by the Graduate School of Natural and Applied Sciences of Istanbul Şehir University.

DATE OF APPROVAL: 24 Jan 2017

SEAL/SIGNATURE:

Declaration of Authorship

I, Osman Biçer, declare that this thesis titled, 'Efficiency Optimizations on Yao's Garbled Circuits and Their Practical Applications' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: 

Date: 24 Jan 2017

"The scientist is not a person who gives the right answers, he is one who asks the right questions."

Claude Lévi-Strauss

Efficiency Optimizations on Yao's Garbled Circuits and Their Practical Applications

Osman BiÇER

Abstract

The advance of cloud computing and big data technologies brings out major changes in the ways that people make use of information systems. While those technologies extremely ease our lives, they impose the danger of compromising privacy and security of data due to performing the computation on an untrusted remote server. Moreover, there are also many other real-world scenarios requiring two or more (possibly distrustful) parties to securely compute a function without leaking their respective inputs to each other. In this respect, various secure computation mechanisms have been proposed in order to protect users' data privacy. Yao's garbled circuit protocol is one of the most powerful solutions for this problem. In this thesis, we first describe the Yao's protocol in detail, and include the complete list of optimizations over the Yao's protocol. We also compare their advantages in terms of communication and computation complexities, and analyse their compatibility with each other. We also look into generic Yao implementations (including garbled RAM) to demonstrate the use of this powerful tool in practice. We compare those generic implementations in terms of their use of garbled circuit optimizations. We also cover the specific real-world applications for further illustration. Moreover, in some scenarios, the functionality itself may also need to be kept private which leads to an ideal solution of secure computation problem. In this direction, we finally cover the problem of Private Function Evaluation, in particular for the 2-party case where garbled circuits have an important role. We finally analyse the generic mechanism of Mohassel *et al.* and contribute to it by proposing a new technique for the computation of the number of possible circuit mappings.

Keywords: Secure Multi-Party Computation, Secure Two-Party Computation, Private Function Evaluation, Yao's Garbled Circuits, Garbled RAM

Yao'nun Karıştırılmış Devresi Protokolü Optimizasyonlarının Verimliliği ve Pratik Uygulamaları

Osman Biçer

Öz

Bulut bilişim ve büyük veri teknolojilerinin ilerlemesi insanların bilişim sistemlerini kullanma yollarında büyük değişimler getirmiştir. Bu teknolojiler hayatımızı büyük ölçüde kolaylaştırırken, aynı zamanda hesaplamaların uzak bir sunucuda yapılması nedeniyle bilgilerin mahremiyetini ve güvenliğini tehlikeye atmaktadırlar. Birbirine yeterince güvenemeyen iki veya daha fazla tarafın bir fonksiyonu güvenli olarak hesaplamasını gerektiren gerçek hayatta karşılaşılabilecek birçok durum vardır. Bu sebeple, kullanıcıların veri mahremiyetini koruyan çeşitli güvenli hesaplama yöntemleri önerilmiştir. Yao'nun karıştırılmış devresi protokolü bu güvenli hesaplama problemine karşı önerilmiş en güçlü çözümlerden biridir. Bu tezde, öncelikle Yao protokolünü ve bu protokolün optimizasyonu için önerilmiş gelişmelerin tam listesini anlatmaktayız. Aynı zamanda, bu gelişmeleri iletişim ve hesaplama zorluğu olarak kıyaslıyoruz ve birbirleriyle uyumluluklarını analiz ediyoruz. Bu güçlü protokolün pratikteki kullanımını göstermek amacıyla çeşitli genel Yao uygulamalarını (karıştırılmış RAM dahil) inceliyoruz. Bu uygulamaları kullandıkları karışık devre optimizasyonlarına göre kıyaslıyoruz. Özel olarak bazı gerçek-hayat uygulamalarıyla Yao protokolünü daha da örneklendiriyoruz. Hesaplanacak fonksiyonun da gizli bir bilgi olması durumunda, onun da gizlenmesinin tam bi mahremiyet için gerekli olduğu unutulmamalıdır. Bu doğrultuda geliştirilmiş olan gizli fonksiyon hesaplama yöntemlerini, özellikle karışık devrelerin önemli bir rolünün olduğu iki taraflı durum için tezimizde anlatıyoruz. Son olarak Mohassel ve Sadeghian'ın geliştirmiş olduğu mekanizmayı ele alıyoruz ve olası devre haritalarının sayısını hesaplamak için kullanılacak yeni bir teknik önererek buna katkıda bulunuyoruz.

Anahtar Sözcükler: Güvenli Çok-Taraflı Hesaplama, Güvenli İki-Tarafli Hesaplama, Gizli Fonksiyonel Hesaplama, Yao'nun Karıştırılmış Devreleri, Karıştırılmış RAM

To my mom and dad for their patience and love

Acknowledgments

First of all, I would like to thank to my advisor Prof. Dr. Ensar Gül for his contributions and support. I would also thank to my co-advisor Dr. Mehmet Sabır Kiraz to whom I cannot express my gratitude for introducing the area of cryptographic protocols to me, for encouraging me studying Yao's garbled circuits, for his contributions during the preparation of my thesis, his perfect guidance, suggestions and feedbacks. Without his input, this thesis could not have been completed as good as it is right now. He is quite enthusiastic in teaching and guiding which makes him the best mentor that a master student wish to have during this hard process.

I would also thank to Muhammed Ali Bingöl for interesting discussions, insightful suggestions, and successful observations on my thesis. His ideas and feedbacks have helped me a lot to improve the thesis significantly. I am also grateful to Dr. Osmanbey Uzunkol and Dr. İsa Sertkaya for reviewing my thesis, making comments and giving quite helpful feedbacks.

I would also thank to Mike Rosulek although we have not met personally. His talk in Simons Institute, University of California, Berkeley, namely A Brief History of Practical Garbled Circuit Optimizations gave me the basic understanding of the Yao's protocol and the starting point for my research of garbled circuit optimizations.

The last but not the least, I cannot express my gratitude for my mom and dad for supporting me in my all decisions, including going after the area of secure computation. They have always helped me, guided me and been the perfect parents whom any child would ever hope for.

Contents

List of Figures

List of Tables

Abbreviations

§	described in [chapter/section/subsection]
MPC	Secure **M**ulti-**P**arty **C**omputation
SFE	**S**ecure **F**unction **E**valuation
GMW	**G**oldreich-**M**icali-**W**idgerson
2PC	Secure **2**-**P**arty **C**omputation
RAM	**R**andom **A**ccess **M**emory
PRF	**P**seudo-**R**andom **F**unction
MAC	**M**essage **A**uthentication **C**ode
DKC	**D**ual **K**ey **C**ipher
OT	**O**blivious **T**ransfer
HE	**H**omomorphic **E**ncryption
CPU	**C**entral **P**rocessing **U**nit
ct	**c**iphert**t**ext
edt	total **e**ncryption and/or **d**ecryption **t**ime
P&P	**P**oint & **P**ermute
GRR3	**G**arbled **R**ow **R**eduction **3** ciphertexts
GRR2	**G**arbled **R**ow **R**eduction **2** ciphertexts
lsb	**l**east **s**ignificant **b**it
cpg	**c**ycles **p**er **g**ate
bpg	**b**ytes **p**er **g**ate
PFE	**P**rivate **F**unction **E**valuation
IBE	**I**dentity **B**ased **E**ncryption
ORAM	**O**blivious **R**andom **A**ccess **M**emory
erf	**e**rror **f**unction
CTH	**C**ircuit **T**opology **H**iding

PGE	**P**rivate **G**ate **E**valuation
ow	**o**utgoing **w**ire
iw	**i**ncoming **w**ire
EP	**E**xtended **P**ermutation
OMAP	**O**blivious **M**apping
OEP	**O**blivious Evaluation of **E**xtended **P**ermutation
SN	**S**witching **N**etwork
PN	**P**ermutation **N**etwork
OSN	**O**blivious Evaluation of **S**witching **N**etworks

Chapter 1

Introduction

Two rich people want to determine which one of them is richer so that he would pay the bill for the dinner. However, none of them is willing to permit the other learn more information about his personal wealth than what the mere knowledge of who is richer does. They start discussing how they could achieve this just by talking to each other. They are quite sure that both will always tell the truth since they are honourable businessmen who cannot take the risk of being caught while lying. On the other hand, both suspect that the other may try to deduce information about his wealth from the conversation. After some time of discussion, they come to the conclusion that it is impossible to decide who is richer under these conditions since they do not know much about *secure computation* techniques.

This famous problem is known as "millionaires' problem" proposed by Andrew Yao [6]. He has also proposed a cryptographic solution for this problem, and generalized it to the secure computation of any function [6, 7]. His later work has showed that any function that can be computed by a polynomial-size circuit can be computed securely [7]. The problem has further widened and solved for the case of more than two parties [6, 8]. Yao's research is followed by many others' in constituting an active subfield of cryptography known as *secure multi-party computation* (MPC) or *secure function evaluation* (SFE), which aims solving the problem of two or more parties computing a function jointly without revealing their secret inputs to each other.

There are many real-life examples where MPC techniques can be applied, including financial systems [9], cooperation of intelligence agencies, companies and governments

[10, 11], electronic elections [12], electronic auctions [13, 14], secure biometric identification [15–17], secure e-mail filtering [18], *etc.* In fact, there is no bound for the areas where MPC may be used, and it can be adopted in any case some parties are required to compute a function on their private data.

Various methods have been proposed for MPC, including generic methods and function specific methods. Although function specific methods usually run more efficiently, they are limited in use due to the fact that each of them works for only one function. It is quite inefficient to design a method and to prove its security for each different function unless the function will be used many times. An example of frequently used functions is the Hamming distance calculation which is used in many scenarios, including biometric checks [15] *etc.* Hence, designing a specific protocol for it while proving its security makes sense [16, 17]. However, general research approach is towards the generic methods which can be applied to arbitrary functions.

Generic methods have been developed for usage in an unlimited set of functions. Usually one method is better than the other for different computational settings. For instance, *homomorphic encryption* will be a very good fit for arithmetic circuits if an efficient fully homomorphic encryption scheme become available in the future [2]. However, currently the proposed fully homomorphic encryption schemes are inefficient for practical secure computation.

The most efficient methods for secure computation of functions represented as boolean circuits include *GMW protocol* [8] and *Yao's garbled circuit protocol* (Yao's protocol). The former usually gives better results in the presence of at least three parties, while the latter is usually better for two-party case.

Yao's protocol remains one of the most important paradigms for MPC, especially in the case of *secure two-party computation* (2PC) [5]. In particular, it is valuable for its constant round complexity. Since the time it was proposed by Andrew Yao in [7], it has become one of the major fields in modern cryptographic research. It is constantly being optimized in terms of communication complexity and computation complexity.

While the research for optimizing Yao's protocol scheme continues, various practical applications using Yao's protocol have also been developed. These applications demonstrate that it is a promising cryptographic primitive for a wide range of applications, including

privacy preserving data mining, efficient secure two-party computation, private function evaluation *etc.*

In this thesis, we first describe the Yao's protocol in detail, and include the complete list of optimizations over the Yao's protocol. We also compare their advantages in terms of communication and computation complexities, and analyse their compatibility with each other. We also look into generic Yao implementations (including garbled RAM) to demonstrate the use of this powerful tool in practice. We compare those generic implementations in terms of their use of garbled circuit optimizations. We also cover the specific real-world applications for further illustration. Moreover, in some scenarios, the functionality itself may also need to be kept private which leads to an ideal solution of secure computation problem. In this direction, we finally cover the problem of Private Function Evaluation, in particular for the 2-party case where garbled circuits have an important role. We finally analyse the generic mechanism of Mohassel *et al.* and contribute to it by proposing a new technique for the computation of the number of possible circuit mappings.

1.1 Overview of the Thesis

Research goal:

Our goal in this thesis is to compare the advantages of currently known Yao's protocol optimizations in terms of communication and computation complexities, to analyse their compatibility with each other, to demonstrate their role with a view towards its practical and real-world applications and in private function evaluation. We intend to describe the current state of the art for Yao's protocol, since it is hard to find many comprehensive works about it. We believe that this work will be quite useful to cryptography community as a study material as well.

Organization of the thesis:

Chapter 1: Introduction

Chapter 1 is dedicated to introduction and overview of the thesis.

Chapter 2: Preliminaries

Chapter 2 is dedicated to generic MPC methods, and to cryptographic basis. We also included a section for circuit concepts which is assumed to be helpful for the people with potentially different backgrounds.

Chapter 3: Yao's Garbled Circuit Protocol

Chapter 3 includes general description and formal definition of Yao's protocol, as well as the generic Yao's protocol template together with its security properties.

Chapter 4: Garbled Circuit Optimizations

Chapter 4 presents known garbled circuit optimizations in a chronological order (*i.e.*, P&P (§4.2), GRR3 (§4.3), free `XOR` (§4.4), GRR2 (§4.5), fle`XOR` (§4.6), half gates (§4.7)). We analyze these optimizations in terms of their relations and contradictions as well as their compatibility with each other. One of our aims is to give a clear overview, therefore, we did not get involved with proofs and other related complex formulas.

Chapter 5: Practical Implementations of Yao's Protocol

Chapter 5 composes of generic Yao's protocol applications and some real-world examples, including pipelining method, garbled RAM, MPC for satellite collusion probability, and privacy preserving data mining.

Chapter 6: Private Function Evaluation

Chapter 6 is dedicated to private function evaluation. We intend to describe Mohassel *et al.*'s generic PFE scheme, which is the most efficient to date, and its application to Yao's protocol. We contribute to it by proposing a new technique for the computation of the number of possible circuit mappings.

Chapter 7: Conclusion and Discussions

Chapter 7 concludes with general discussions of garbled circuit optimization techniques, Yao's protocol applications and private function evaluation.

Chapter 2

Preliminaries

In this chapter, we will present the basic concepts of secure computation techniques. First, we will show the required properties for a secure computation scheme. We will continue with general adversary models in cryptographic protocols. This will be followed by circuit concepts useful for MPC techniques which, we suppose, will be quite helpful for people new to the area. Then, we will present general cryptographic primitives. We will also give the summary of oblivious transfer protocol, homomorphic encryption, and GMW protocol.

2.1 Requirements of Secure Multi-Party Computation

To formally claim and prove the security of an MPC protocol, some general security properties are required [10]. The most central of these properties are described in [10] by Lindell *et al.* as follows:

1. *Correctness:* The output that is delivered to each party (*i.e.* each participant of the MPC protocol) is guaranteed to be correct.

2. *Privacy*: None of the participants is allowed to learn anything more about other participants' inputs than what he can learn from the output itself.

3. *Independence of inputs:* The protocol may not allow any of the parties to choose his input based on other parties' inputs. This property is different from privacy since choosing an input dependent on another party's unknown input is possible .

4. *Guaranteed output delivery:* In the end of the protocol, honest parties should receive their outputs no matter how hard corrupt parties try to prevent it.

5. *Fairness:* A party whether he is corrupt or not can receive his output if all of the parties receive their outputs. For detailed information about how to achieve efficient fair MPC, we refer the reader to [19, 20].

Lindell *et al.* stress that this list does not define security, but rather compose of the requirements that any secure protocol must conform [10].

2.2 Adversary Models

Security of cryptographic protocols are formalized and proved against adversaries with different capabilities [2].

2.2.1 Semi-Honest Adversaries

The *semi-honest* (also known as passive, or honest-but-curious) threat model is the standard adversary model for MPC. Here parties typically follow the protocol as they are supposed to but may try to deduce information about another party's input from the protocol transcript [21]. If a protocol is secure against semi-honest adversaries, it does not allow them to learn any extra information from the protocol.

2.2.2 Covert Adversaries

Covert adversaries constitute the type of adversaries that are allowed to deviate from the protocol with a restriction that they must evade being caught while they are doing so [2]. It can be safely assumed that in many political, social and business scenarios, the gain from cheating is overweighted by the results of being caught. If those deviations are detected with a certain frequency (*e.g.*, 1 out of 10 times), such a protocol can be considered secure enough. If a protocol is secure against covert adversaries, it allows catching those adversaries with a certain probability if they deviate from the protocol.

2.2.3 Malicious Adversaries

The strongest type of adversaries is the *malicious adversaries* (also known as active adversaries), which may deviate from the protocol arbitrarily so that they can extract the other parties private inputs or alter the computation outcome [2]. If a protocol is secure against malicious adversaries, a corrupt party will be caught whenever he deviates from the protocol.

Throughout this thesis, we focus on the security against semi-honest adversaries due to the following reasons [21]:

1. There are many real-world situations where modelling the parties as semi-honest adversaries is appropriate:
   (a) where parties are legitimately trusted but there is a legal need for preventing them from divulging information, or for protection against break-ins in the future.
   (b) where the software used for MPC can hardly be changed by participants without being detected, either due to software attestation use or the fact that internal controls are in place (*e.g.*, when parties are government agencies, or large corporations).
2. Securing protocols against semi-honest adversaries is an important step toward construction of secure protocols against stronger adversaries. There are generic ways of altering them to achieve security against covert or malicious adversaries [20, 22].

2.3 Corruption Models

Apart from the above adversary models, there also exist static and adaptive corruption models.

Static corruption model: This model implies that if a party is honest in the beginning, he always remains honest; whereas if a party is corrupted in the beginning, he always remains corrupted [10].

Adaptive corruption model: Instead of including a fixed number of corrupted parties, adaptive corruption model suggests that the number of corrupted parties may increase during the computation. However, if a party gets corrupted, it remains that way from then on [10]. Therefore, there may never be a decrease in the number of corrupted parties.

2.4 Circuit Concepts

For a generic MPC protocol to take place, first a function must be written as a combination of common building blocks, *i.e.*, they must be represented as *circuits*. The number of types of building blocks is limited. Therefore, by showing how to compute each building block, a generic MPC scheme permits calculation of unlimited functions. Standard circuit representations generally used in MPC protocols are boolean circuits and arithmetic circuits [2].

2.4.1 Boolean circuits

In engineering and computer science, functions are classically represented as *Boolean circuits* [2]. A boolean circuit basically composes of *logic gates* and *wires* connecting them [23]. Figure 2.1 shows an example boolean circuit whose wires are a, b, c, d, e, f, h, k, and o, and gates are $g1$, $g2$, $g3$, $g4$, and $g5$.

a, b, and c are the *input*s of the circuit in Figure 2.1, d, e, f, h, and k are the *intermediate wires*, and o is the *output* wire. A boolean circuit may have more than one output as well. A wire is exactly 1 bit that may have one of the two truth values, *i.e.*, either `TRUE` (also denoted as 1 or `High`) or `FALSE` (also denoted as 0 or `Low`). When 2 wires cross each other, they are connected if there is a big dot in the connection point, otherwise they are not connected. For example a and b cross each other but not connected (the same applies to d and e in Figure 2.1).

A logic or boolean gate generally takes 1 or 2 wires as input (although there is no certain limitation) and outputs exactly 1 wire. Formally a d-input gate G_d is a boolean function mapping $d > 0$ bits input to 1-bit output, *i.e.* [2]:

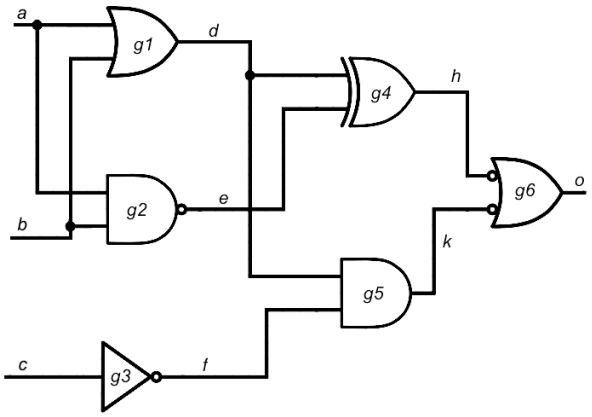

FIGURE 2.1: An example boolean circuit.

$$G_d : (in_1, \ldots, in_d) \in \{0,1\}^d \rightarrow (out) \in \{0,1\} \tag{2.1}$$

For the gates of the circuit in Figure 2.1, the left sides are used for the input, the right side are used for the output. For example $g1$ in Figure 2.1 takes a and b as inputs and outputs d. However, gates may be rotated in a different circuit. In this case, one needs to look at the two asymmetric sides of a gate. Generally, the larger assymetric side of the gate is the side of inputs and the narrower assymetric side is for the output. *A wire can only be an output of exactly 1 gate*, although it can be input to multiple gates [23].

In Figure 2.1, $g1$ is an `OR` gate ($d \leftarrow a \vee b$), $g4$ is an `XOR` gate ($h \leftarrow d \oplus e$), and $g5$ is an `AND` gate ($k \leftarrow d \wedge f$). If there is a bubble on the wire, its truth value is inverted after the bubble. For example, $g2$ would have been an `AND` gate without the bubble on its output. But the bubble means the output is inverted. Actually, there is a special name for the type of $g2$, it is a `NAND` gate ($e \leftarrow (a \wedge b)'$). $g3$ would have been a `buffer` gate without the bubble on its output. A `buffer` gate outputs the input as it is. However, with the bubble $g2$ is a `NOT` gate ($f = c'$). $g6$ would have been an `OR` gate without the bubbles on its inputs. Now, it takes the inputs inverted, and `OR`s them afterwards ($o \leftarrow h' \vee k'$). Actually $g6$ is another representation of a `NAND` gate due to the logic identity $h' \vee k' = (h \wedge k)'$. There also exist `NOR` gates represented as an `OR` gate with a bubble on its output.

The truth table of a gate shows the relation between its possible inputs and its possible outputs. The truth table of a gate has 2^k rows where k is the number of its input wires.

TABLE 2.1: Truth table of an `AND` gate ($g5$ in Figure 2.1).

d	f	$k = d \wedge f$
0	0	0
0	1	0
1	0	0
1	1	1

The truth table of the `AND` gate $g5$ in Figure 2.1 can be seen in Table 2.1.

In fact, there are basically 2^4 different 2-input gates in total. However, some of them are trivial (*i.e.*, the ones whose output depends only one of the inputs and the ones whose output depends none of the inputs). Those gates can be replaced by more efficient representations, *e.g.*, wires, `NOT` gates, *etc.* The remaining non-trivial gates fall into the category of either even gates or odd gates [24].

Definition 2.1. Even gates are the 2-input gates whose truth table has 2 `FALSE` outputs and 2 `TRUE` outputs.

Definition 2.2. Odd gates are the 2-input gates whose truth table has either 3 `FALSE` outputs and 1 `TRUE` output or 1 `FALSE` output and 3 `TRUE` outputs.

There are only 2 non-trivial even gates which are `XOR` and `XNOR`, and 8 non-trivial odd gates, including `OR`, `AND`, `NOR`, `NAND`, *etc* [24].

The *size* of a boolean circuit means the number of its gates [25]. The *depth* of a boolean circuit means the number of gates in the longest path that must be taken from any input to any output [25]. The *topology* of a boolean circuit means the connections between its gates [25]. A boolean circuit can uniquely be defined by its topology and its gates.

The *topological order* of a boolean circuit is that when its gates are indexed as $G_1, \ldots, G_n$, i^{th}, a gate G_i does not get the output of a succeeding gate $G_{j>i}$ as its input [2]. Intuitively, in order to compute a gate, all of its input wires must be known, which can be ensured by computing the gates in topological order. By computing the gates one-by-one in topological order the whole boolean circuit can be computed. The topological order is not necessarily unique for a given boolean circuit [2].

A group of gate types $(G_1, \ldots, G_n)$ is *Turing-complete*, if and only if any probabilistic polynomial time algorithm can be represented by a combination of those gates [26].

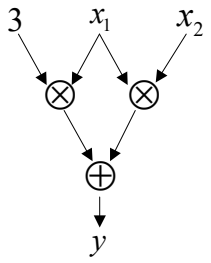

FIGURE 2.2: An example arithmetic circuit [2].

Examples are (`AND`,`XOR`) and (`NAND`). Building a `NAND` gate from a group of gates is an easy way to see whether that group of gates is Turing-complete or not.

A decrease in the number of gates in a circuit also means a decrease in overall cost of an MPC protocol in terms of computation complexity, and communication complexity. There are various techniques for circuit optimizations. Some circuit optimization techniques intend to reduce the number of odd gates at the cost of increasing the even gates. They could also be useful in some MPC techniques [27, 28].

2.4.2 Arithmetic circuits

A more compact representation for functions is *arithmetic circuits* [2]. Unlike boolean circuits where wires are chosen from $\mathbb{Z}_2$, here wires have values chosen from $\mathbb{Z}_{m\geq 2}$. The gates operations are either modular addition + or modular multiplication $\times$. Figure 2.2 shows an example arithmetic circuit.

One can express any boolean circuit as an arithmetic circuit over $\mathbb{Z}_2$. However, if $\mathbb{Z}_m$ has a modulus m which is sufficiently large, then the resulting arithmetic circuit representation of a function will probably have much lower size than its boolean circuit representation, since a single operation will be enough for each integer addition or multiplication [2].

Computations on both positive and negative integers x can be simulated by arithmetic circuits, since one can map them into elements of $\mathbb{Z}_m : \mathbb{Z} \to \mathbb{Z}_m, x \to x \ \texttt{mod} \ m$ [2].

2.5 Cryptographic Basis

As the cryptographic basis for this thesis, we present symmetric and public key encryptions, cryptographic hash functions, pseudo-random functions and message authentication codes. We will include only a brief summaries of them due to the fact that the details of them are not necessary for understanding protocols and that vast majority of our readers will probably have an acquaintance with them. However, at the end of this section we present dual-key ciphers in more detail because of their intensive use in Yao's protocol and supposed unfamiliarity of some readers with them.

2.5.1 Symmetric Encryption

A *symmetric encryption* scheme uses the same cryptographic key k for both encryption of plaintext and decryption of ciphertext [29]. A well-known example is `AES` encryption [29]. The notation $c \leftarrow E_k(m)$ means that a plaintext message m is encrypted with a key k resulting in a ciphertext c.

Decryption is generally denoted as either $m \leftarrow D_k(c)$ or the inverse of E, namely $m \leftarrow E_k^{-1}(c)$.

2.5.2 Public Key Encryption

A *public key encryption* scheme uses different keys for encryption and decryption. *Public keys* which are known publicly as their name implies are used for encryption, while private keys which are known only to their owners are used for decryption [30]. Any user can encrypt a message with the public key of the receiver, but the resulting ciphertext can be decrypted only with the receiver's private key. The notation $c \leftarrow E_{pk_i}(m)$ means that a plaintext message m encrypted with a public key pk_i of i^{th} person results in a ciphertext c.

Decryption with the secret key sk_i of the i^{th} person is denoted as either $m \leftarrow D_{sk_i}(c)$ or $m \leftarrow E_{pk_i}^{-1}(c)$. The well-known public key cryptosystems are `ElGamal` [31] and `RSA` [30].

2.5.3 Cryptographic Hash Function

A *cryptographic hash function* $H(m)$ maps an arbitrary size message m to a fixed size ℓ-bit string $c \leftarrow H(m)$ [32]. Throughout this thesis when we say *hash function*, we refer to a cryptographic hash function.

Hash functions are ideally modelled in the *random oracle model* [33]. A random oracle is a theoretical black-box responding to every unique query with a true random number picked from its output domain. It records its responses to unique queries so that it can respond a query the same way every time it is repeated. A well-known hash function scheme is `SHA256` [34].

2.5.4 Pseudo-Random Function

A *pseudo-random function* (`PRF`) is a function that can be used for pseudo-random generation, *i.e.*, it can be modelled as random oracle. It is denoted as $\texttt{PRF}(x)$ on an input x. Its representation can be extended as $\texttt{PRF}_k(x)$ to include the use of a private key k [2].

An instantiation of `PRF` can be achieved with a block cipher, *e.g.*, `AES`, or a hash function, *e.g.*, `SHA256`. In case a `PRF` with the same key k is repeatedly used, the `AES` instantiation would be more efficient since its key schedule needs to be run just once [2].

2.5.5 Message Authentication Code (`MAC`)

A *message authentication code* (`MAC`) is a fixed-sized data that is used for authentication of a message. It is denoted as $\texttt{MAC}_k(m)$ on an input message m that needs to be authenticated and a private key k [2].

The `MAC` value provides protection for both data integrity and authenticity of a message since it allows the detection of any changes in the message content by the verifiers possessing the private key k.

2.5.6 Dual-Key Cipher

A *dual-key cipher* (`DKC`) is a cryptographic notion proposed by Bellare *et al.* in [3]. A `DKC` formally represents a two-key lockbox where both keys are required for openning

the box. A DKC is a function E associating a security parameter $k \in N$ where N is the set of positive integers and keys $A, B \in \{0,1\}^k$ with a k-bit pseudo-random number $E_{A,B} : \{0,1\}^k \to \{0,1\}^k$. Let $D_{A,B} : \{0,1\}^k \to \{0,1\}^k$ denote the inverse of this function [3].

Decryption of DKC may also be denoted by the inverse function notation $E^{-1}_{A,B} : \{0,1\}^k \to \{0,1\}^k$ instead of $D_{A,B} : \{0,1\}^k \to \{0,1\}^k$.

Throughout this thesis an encryption with two keys mean a DKC unless it is stated otherwise.

So far, a variety of DKC schemes have been proposed. Among them, an earlier one is Equation (2.2) proposed by Naor *et al.* in [14]. For every encryption, PRF is called twice. PRF may be implemented as a keyed hash.

$$E_{A,B}(C) \to \texttt{PRF}(A, \texttt{gateID}) \oplus \texttt{PRF}(B, \texttt{gateID}) \oplus C \tag{2.2}$$

Lindell *et al.* proposed a more efficient DKC scheme Equation (2.3) in [35]. It requires one hash per encryption, which reduces the computational cost significantly.

$$E_{A,B}(C) \to H(A||B||\texttt{gateID}) \oplus C \tag{2.3}$$

Kreuter *et al.* proposed the DKC scheme Equation (2.4) in [36]. An AES256 encryption is used instead of a hash function. Kreuter *et al.* shows that this improvement reduces the computational cost around 25%.

$$E_{A,B}(C) \to \texttt{AES256}(A||B||\texttt{gateID}) \oplus C \tag{2.4}$$

Bellare *et al.* proposed the *state-of-the-art* DKC scheme Equation (2.5)[1] in [37] which eliminates the need for key precessing in each AES encryption by using a constant key k_c for all of them.

$$E_{A,B}(C) \to \texttt{AES128}_{k_c}(K) \oplus K \oplus C \tag{2.5}$$

[1] $K = 2A \oplus 4B \oplus gateID$

2.6 Secret Sharing

Secret sharing refers to the methods where a secret value is distributed amongst a group of parties, each having a share from the secret [38]. To reconstruct the secret, parties need to combine a sufficient number of shares together; since individual share of a party is useless on its own. There have been various secret sharing schemes proposed so far. Here we will introduce only some of them which will be helpful throughout this thesis.

2.6.1 `XOR` Sharing

`XOR` *sharing* (also known as *boolean sharing*) is a secret sharing type where for an ℓ-bit value x shared by m parties, the share of a party i is an ℓ-bit value x_i, and when the shares of all m parties `XOR`ed bitwise together the result is x, *i.e.*, $x = x_1 \oplus \ldots \oplus x_m$ [39]. There is no number limit for parties in `XOR` sharing. However, if any of the parties keeps his share, the rest of the parties cannot even get close to learning the shared value.

2.6.2 Arithmetic Sharing

Arithmetic sharing is similar to `XOR` *sharing* in that there is no number limit for parties and that if any of the parties keeps his share, the rest of the parties cannot even get close to learning the shared value [39]. It is a secret sharing type where for an ℓ-bit value x shared by m parties, the share of a party i is an ℓ-bit value x_i, and when the shares of all m parties added together in a modulus n which conforms $2 \leq n \leq 2^\ell$ the result is x, *i.e.*, $x = x_1 + \ldots + x_m \ \texttt{mod}\ n$.

2.6.3 Yao Sharing

Yao sharing is a secret sharing type where 1 bit is shared by 2 parties [39]. In order to share a bit b, the first party P_1 picks 2 random ℓ-bit strings B_0 and B_1. The second party P_2, without knowing b, keeps only B_b. P_1 does not know which of the 2 strings kept by P_2, and P_2 does not know the other string picked by P_1. Only together, they can evaluate b. Although keeping costly strings for a bit does not look very efficient at first, Yao sharing has certain advantages for 2PC which will be obvious when we describe Yao's protocol in §3.

2.6.4 Shamir's Secret Sharing

Shamir's secret sharing is an effective secret sharing scheme proposed by Adi Shamir [38, 40] where a group of n users share a secret data D. The scheme permits any predefined $(k+1) \leq n$ or more users to reconstruct the secret. However, no information about D can be recovered by k or less users. This scheme can also be referred to as $(k+1, n)$-threshold secret sharing scheme, where $(k+1)$ is the threshold and n is the number of users sharing the secret.

All users have a different point in two-dimensional plane, $(x_1, y_1), \ldots, (x_n, y_n)$. All of the points must be chosen such that they are on a k-degree polynomial. Therefore, any $k+1$ of these shares suffices for Lagrange's interpolation. The secret value is the evaluation of the polynomial on axis $x = 0$.

2.7 Oblivious Transfer

An *1-out-of-m oblivious transfer* (1-out-of-m OT) protocol is a two-party asymmetric[2] protocol where one of the parties is the sender, and the other one is the receiver [4]. The sender has the set of values $\{x_1, \ldots, x_m\}$ and the receiver has an index i. At the end of the protocol, the receiver should only learn one of the sender's inputs, which is x_i; whereas the sender should not learn anything about the index i. An efficient 1-out-of-m OT technique can be found in [41].

The high computational complexity of OT is a major source of inefficiency. In order to reduce this cost, some optimizations (*e.g.*, extended OT [42]) have been proposed.

There also exist OT protocols for settings with more parties, known as *multi-party oblivious transfer*. A multi-party OT is a protocol where one of the parties holds the values $x_1, \ldots, x_m$, but multiple parties secret share the choice index i. At the end of the protocol, the parties learn shares of x_i instead of learning it as a whole. The party holding the initial values is called the sender, whereas the other ones are called the receivers.

[2] An asymmetric protocol means that parties play different roles during the protocol.

2.8 Homomorphic Encryption

Homomorphic Encryption (HE) schemes are used for secure evaluation of arithmetic circuits since they permit computation of multiplication and addition on ciphertexts [2]. An *additively* HE scheme allows only unlimited addition on encrypted data; whereas a *multiplicative* HE scheme allows only unlimited multiplication on it. An encryption scheme having both multiplicatively and additively HE property is called *fully homomorphic encryption* (FHE).

There was a wide-spread belief that FHE does not exist until recently. Gentry has been the inventor of the first FHE scheme [43]. Unfortunately, huge sizes and computational costs of current FHE schemes make them too inefficient to be used in practical applications no matter how much effort has been given for improving their performances. The problem is that a FHE scheme must allow algebraic operations while providing strong security assumptions, which makes the costs grow substantially.

2.9 Goldreich-Micali-Wigderson (GMW) Protocol for MPC

One of the commonly used MPC schemes is *Goldreich-Micali-Wigderson* (GMW) protocol that uses `XOR` sharing (§2.6), and is proposed in [8]. It proposes MPC of boolean circuits with gates `AND` and `XOR` against semi-honest adversaries (§2.2.1).

`XOR` gates can be computed locally and are communication free [4]. To illustrate, to compute $c = a \oplus b$, each party i only needs to use its shares $c_i = a_i \oplus b_i$ in order to receive his output share c_i. However, to compute an `AND` gate, parties are required to communicate for 1-out-of-4 OT (§2.7). In the case of 2 parties, to compute their output shares of $a \wedge b$, P_1 constructs the evaluation table for both input shares of P_2 and they engage in a 1-out-of-4 OT (§2.7) where P_2's inputs are used as the choice index. To extend the protocol for m parties, $\binom{m}{2}$ runs of the OT protocol is required. One can also see it as one run of a multi-party 1-out-of-4 OT protocol where the choice indices are a and b [4].

Chapter 3

Yao's Garbled Circuit Protocol

Even though Yao's protocol has more than two-party applications, its use will be held limited to 2PC. It is an asymmetric protocol, which means that parties play different roles while the protocol is running. One of the parties has the role of the *garbler*, whereas the other one becomes the *evaluator*. The protocol is intended to be secure in the semi-honest model (§2.2.1). It runs on boolean functions, so first a function must be converted to a boolean circuit. Figures 3.1, 3.2 and 3.3 have been taken from Mike Rosulek's presentation in Simons Institute, University of California, Berkeley, namely *A Brief History of Practical Garbled Circuit Optimizations*.

A gentle introduction. Yao's garbled circuit protocol is briefly as follows (later we propose it in a more formal model):

Assume Alice and Bob are trying to compute a function f whose boolean circuit is given in Figure 3.1. Throughout this thesis, Alice will be the garbler, Bob will be the evaluator. Alice's input is x including bits a and c, and Bob's input is y including bits b and d.

`Garbling:`

1. Alice picks random and computationally indistinguishable masking values for possible truth values `FALSE` and `TRUE` of each wire.
2. She encrypts the output masking values of each gate using their corresponding input masking values as the `DKC` key (§2.5.6). This way she gets four ciphertexts for each gate in the circuit as in Figure 3.2.

`Input Transfer:`

3. She sends all ciphertexts for each gate, as well as her masked input values for a and c to Bob. He takes his own masked input values from Alice using 1-out-of-2 OT (§2.7).

`Evaluating:`

4. Bob decrypts the related ciphertext (we will come to this later) gate-by-gate in *topological order*, reaching the output masking values of the circuit. Topological order means from the inputs to the output. The rule is that if the output of a gate g_1 is input to another gate g_2, g_1 must be evaluated before g_2. In this case the gate order might be chosen as 1, 2, 3, 4, 5.

`Output Reveal:`

5. Bob tells Alice the output masking values, and Alice sends the output of the function $f(x, y)$ to Bob.

The flow of communication between the garbler and the evaluator is summed up in Figure 3.3.

3.1 Formal Definiton of Yao's Protocol

The Yao's protocol scheme proposed by Bellare *et al.* in [3] brought a significant jump by defining the procedures involved in a secure Yao's protocol. A conventional circuit

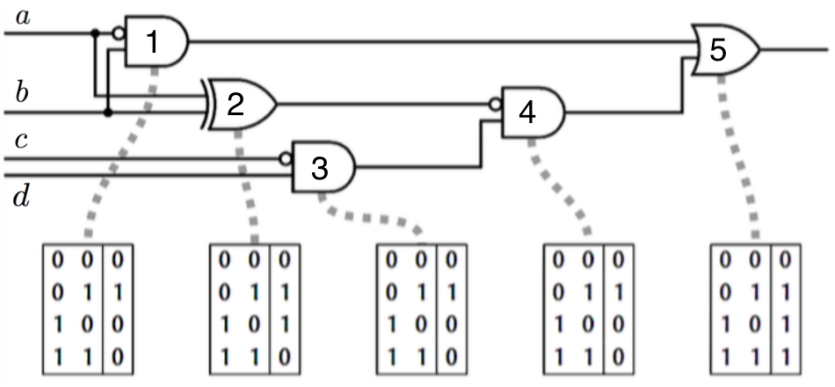

FIGURE 3.1: A boolean circuit of a function f with the truth table of the gates included.

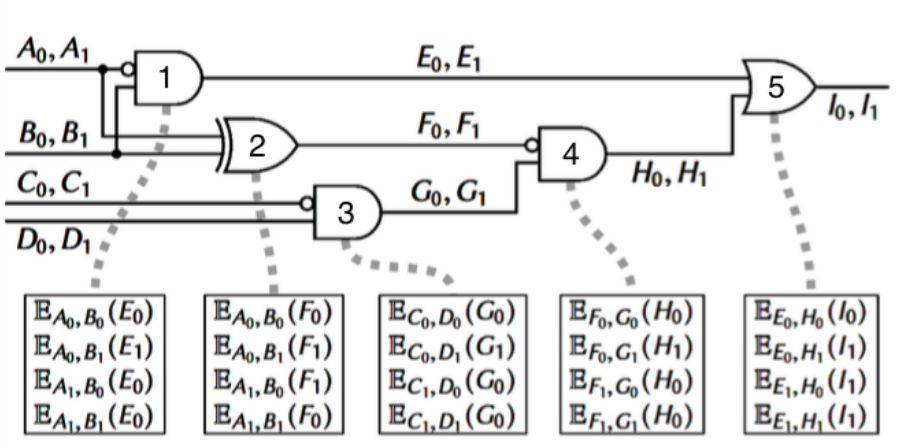

FIGURE 3.2: Garbling the circuit in Figure 3.1

can be defined as $f = (n, m, q, A, B, G)$ where the numbers of its inputs, its outputs, and its gates are $n \geq 2$, $m \geq 1$, and $q \geq 1$, respectively. The number of its wires is denoted as $r = n + q$. The sets of the circuit `Inputs`, `Wires`, `OutputWires` and `Gates` are defined as $\texttt{Inputs} = \{1, \ldots, n\}$, $\texttt{Wires} = \{1, \ldots, n+q\}$, $\texttt{OutputWires} = \{n+q-m+1, \ldots, n+q\}$, and $\texttt{Gates} = \{n+1, \ldots, n+q\}$. Then the function identifying each gate's first incoming wire is $A : \texttt{Gates} \rightarrow \texttt{Wires}\backslash\texttt{OutputWires}$. The function identifying each gate's second incoming wire is $B : \texttt{Gates} \rightarrow \texttt{Wires}\backslash\texttt{OutputWires}$. The function determining the functionality of each gate is $G : \texttt{Gates} \times \{0,1\}^2 \rightarrow \{0,1\}$. The requirement is that $A(g) < B(g) < g$ for all $g \in \texttt{Gates}$.

Bellare *et al.* defines the generic garbling scheme consisting of `Gb`, `En`, `Ev`, and `De` algorithms which are described as folows (see also Figure 3.4 and Algorithm 1) [3]:

1. **Garble** (`Gb`): `Gb` procedure takes 1^k and a boolean circuit f as input, and outputs

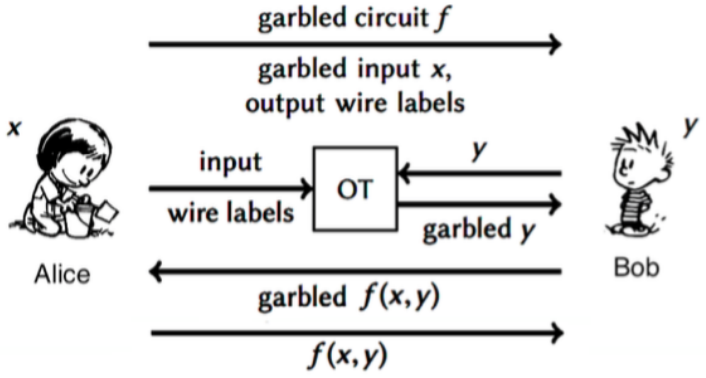

FIGURE 3.3: Communication flow in the semi-honest Yao's protocol.

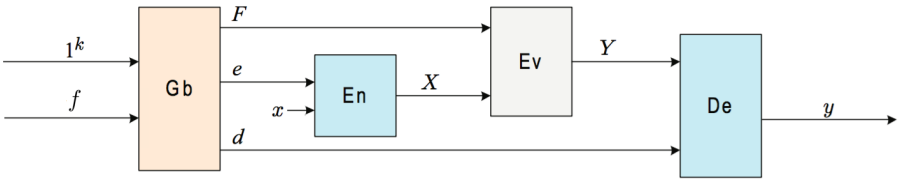

FIGURE 3.4: The flow of procedures in Yao's protocol in [3].

(F, e, d), where F is a garbled circuit, e is the encoding information, and d is the decoding information. The **for**-loop on Line 3 of Algorithm 1 assigns masking values for every wire in the circuit for both `TRUE` and `FALSE`. It also assures that the last bits of the assigned masking values for a wire, which we call label bits, differ from each other. The **for**-loop on Line 6 of Algorithm 1 encrypts the possible output masked with their corresponding input masking values for each gate. It also orders the ciphertexts with respect to the label bits (`lsb`) of input masking values so that the order does not leak information (we will call this technique point and permute).

2. **Encode** (`En`): `En` procedure takes (e, x) as input, where e is as we mentioned above and x is a suitable input for f, and outputs a garbled input X. In this scheme, encoding is directly assigning the pre-known masking values for the inputs.

3. **Evaluate** (`Ev`): `Ev` procedure takes (F, X) as input, and outputs a garbled output Y. The **for**-loop on Line 22 of Algorithm 1 decrypts only one ciphertext related to a gate with its input masking values and with respect to their label bits.

4. **Decode** (`De`): `De` procedure takes (d, Y) as input, and outputs a plain output y. In this scheme, decoding is directly assigning the pre-known outputs for the masking values obtained by the `Ev` procedure.

Correctness property is that Equation (3.1) holds for all possible input x where $(F, e, d) \leftarrow \texttt{Gb}(1^k, f)$.

$$\texttt{De}(d, \texttt{Ev}(F, \texttt{En}(e, x))) = f(x) \tag{3.1}$$

Algorithm 1 Garbled circuit scheme [3].

procedure $\textsc{Gb}(1^k, f)$ ▷ Garbling phase
  $(n, m, q, A', B', G) \leftarrow f$
  for $i \in \{1, \ldots, n+q\}$ **do**
    $t \leftarrow \{0,1\}$, $X_i^0 \leftarrow \{0,1\}^{k-1}t$, $X_i^1 \leftarrow \{0,1\}^{k-1}\bar{t}$
  end for
  for $(g, i, j) \in \{n+1, \ldots, n+q\} \times \{0,1\} \times \{0,1\}$ **do**
    $a \leftarrow A'(g)$, $b \leftarrow B'(g)$, $A \leftarrow X_a^i$, $\mathtt{a} \leftarrow \mathtt{lsb}(A)$, $B \leftarrow X_b^i$, $\mathtt{b} \leftarrow \mathtt{lsb}(B)$
    $T \leftarrow g \parallel \mathtt{a} \parallel \mathrm{b}$, $P[g, \mathtt{a}, \mathtt{b}] \leftarrow E_{A,B}(X_g^{G_g(i,j)})$
  end for
  $F \leftarrow (n, m, q, A', B', P)$
  $e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$
  $d \leftarrow (X_{n+q-m+1}^0, X_{n+q-m+1}^1, \ldots, X_{n+q}^0, X_{n+q}^1)$
  return (F, e, d)
end procedure

procedure $\textsc{En}(e, x)$ ▷ Encoding phase
  $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$
  $x_1 \ldots x_n \leftarrow x$, $X \leftarrow (X_1^{x_1}, \ldots, X_n^{x_n})$
  return X
end procedure

procedure $\textsc{Ev}(F, X)$ ▷ Evaluating phase
  $(n, m, q, A', B', P) \leftarrow F$, $(X_1, \ldots, X_n) \leftarrow X$
  for $g \leftarrow n+1$ **to** $n+q$ **do**
    $a \leftarrow A'(g)$, $b \leftarrow B'(g)$, $A \leftarrow X_a^i$, $\mathtt{a} \leftarrow \mathtt{lsb}(A)$, $B \leftarrow X_b^i$, $\mathrm{b} \leftarrow \mathtt{lsb}(B)$
    $T \leftarrow g \parallel \mathtt{a} \parallel \mathtt{b}$, $X_g \leftarrow D_{A,B}(P[g, \mathtt{a}, \mathtt{b}])$
  end for
  return $(X_{n+q-m+1}, \ldots, X_{n+q})$
end procedure

procedure $\textsc{De}(d, Y)$ ▷ Decoding phase
  $(Y_1, \ldots, Y_m) \leftarrow Y$, $(Y_1^0, Y_1^1, \ldots, Y_m^0, Y_m^1) \leftarrow d$
  for $i \in \{1, \ldots, m\}$ **do**
    if $Y_i = Y_i^0$ **then** $y_i \leftarrow 0$
    else if $Y_i = Y_i^1$ **then** $y_i \leftarrow 1$
    else return $\perp$
    end if
  end for
  return $y \leftarrow y_1 \ldots y_m$
end procedure

3.2 Security Properties of Yao's Protocol

We need some parameters in order to appreciate the security of a garbling scheme. The security parameters defined by Bellare *et al.* are *privacy*, *obliviousness*, and *authenticity* [3].

3.2.1 Privacy

Privacy is achieved by a garbling scheme if no more information about the input x must be revealed by the collection (F, X, d) than that is revealed by $f(x)$ [3, 5]. Let (f, x) be chosen by the adversary. Then either the circuit is garbled to $(F, e, d) \leftarrow \textsc{Gb}(1^k, f)$, the input is encoded as $X \leftarrow \textsc{En}(e, x)$, the adversary getting (F, X, d); or the

simulator S devises a *fake* $(\bar{F}, \bar{X}, \bar{d})$ depending solely on the security parameter k, the side information[1] $\phi = \Phi(f)$, and the output $y = \texttt{Ev}(f, x)$. The $(\bar{F}, \bar{X}, \bar{d})$ produced by the simulator must be indistinguishable from the ones coming from the actual garbling scheme.

3.2.2 Obliviousness

Obliviousness is achieved by a garbling scheme if (F, X) reveals nothing more than the side information[1] $\Phi(f)$ about f or x [3, 5]. To compare obliviousness with privacy (§3.2.1), where the output is learned by the evaluator, here, he does not learn that since d is kept hidden. The output can be revealed by a private scheme even without d, while x can be revealed by an oblivious scheme once d is exposed. Let (f, x) be chosen by the adversary. Either the circuit is garbled to $(F, e, d) \leftarrow \texttt{Gb}(1^k, f)$, the input is encoded as $X \leftarrow \texttt{En}(e, x)$, and the adversary getting (F, X); or the simulator S to devises a *fake* $(\bar{F}, \bar{X})$ depending solely on k, and $\phi = \Phi(f)$. The $(\bar{F}, \bar{X})$ produced by the simulator must be indistinguishable from from the ones coming from the actual garbling scheme.

3.2.3 Authenticity

Authenticity is achieved by a garbling scheme if from (F, X), an adversary cannot construct a garbled output $\bar{Y}$ which is not authentic, *i.e.* $\texttt{De}(d, \bar{Y}) \neq \perp$ only if $\bar{Y} = \texttt{Ev}(F, X)$, except for negligible probability [3, 5].

[1] Side-information means any information about the circuit which the protocol does not intend to hide, like its size or its topology. $\Phi(f)$ is the side-information function which maps f to ϕ.

Chapter 4

Garbled Circuit Optimizations

Since we have introduced the generic garbled circuit framework, it is time to present the optimizations on it in detail. We start with describing the parameters of a garbled circuit scheme that can be optimized and their relevant importance. We then continue with optimization techniques, along with comparing them with each other and presenting the relations between them. At the end, we have included a useful table to show the compatibility of various garbling techniques. Figures 4.1, 4.2, 4.3, 4.4, 4.5 and 4.6 have been taken from Mike Rosulek's presentation in Simons Institute, University of California, Berkeley, namely *A Brief History of Practical Garbled Circuit Optimizations*.

Mainly, there are three parameters related to Yao's protocol that can be optimized: the *size* of the garbled circuit which limits the communication complexity cost, the *computation time* required both for encryption and decryption, and the *security* of the protocol [44]. The size of the garbled circuit is important because it usually needs to be transmitted to the evaluator over a limited channel. Clearly, the computation time required is also an important parameter for both parties.

4.1 General Focus

4.1.1 The Size Parameter

The size of the garbled circuit is usually the primary parameter due to the limits of the communication channel. The most effort in the garbled circuit research has been

dedicated to make it smaller. Reducing it even in the expense of worse computation times or weaker hardness assumptions is often preferable [44].

A reduction in the size of a garbled circuit generally comes from a decrease in the number of ciphertexts needed per gate. Circuits can grow to contain billions of gates, meaning each garbled circuit can be gigabytes in size. Our primary goal in this chapter is to cover garbled gate size optimization techniques.

4.1.2 The Computation Time Parameter

Computation time is related to time consumptions of `Gb` and `Ev` procedures. Naturally the research aims to make them shorter. The computation time may be even more important when the `CPU` resource of a party is restricted, such as a mobile device. The improvements in `DKC` schemes (§2.5.6) schemes proposed are also for this parameter. The gate garbling techniques may also improved for this parameter as well [44].

4.1.3 Security Parameter

A garbling scheme must conform the security properties (§3.2) although in some cases authenticity parameter may be omitted. If the hardness assumptions of the building blocks of a scheme (*e.g.*, `DKC` scheme (§2.5.6), gate garbling technique) is stronger, the protocol will also be more secure [44].

The rest of this chapter is especially dedicated to the techniques related to the optimizations in the size parameter. However, the techniques will also be compared for the other parameters whenever it is necessary. After the description of each technique, there will be a size and computation time scoreboard for comparing that technique with the previous ones (see Tables 4.1, 4.2, 4.3, 4.4, 4.5, 4.6). The time for encryptions and decryptions for both `DKC` schemes and symmetric schemes assumed to be the same and denoted as `edt` (for encryption/decryption time). ct stands for ciphertexts.

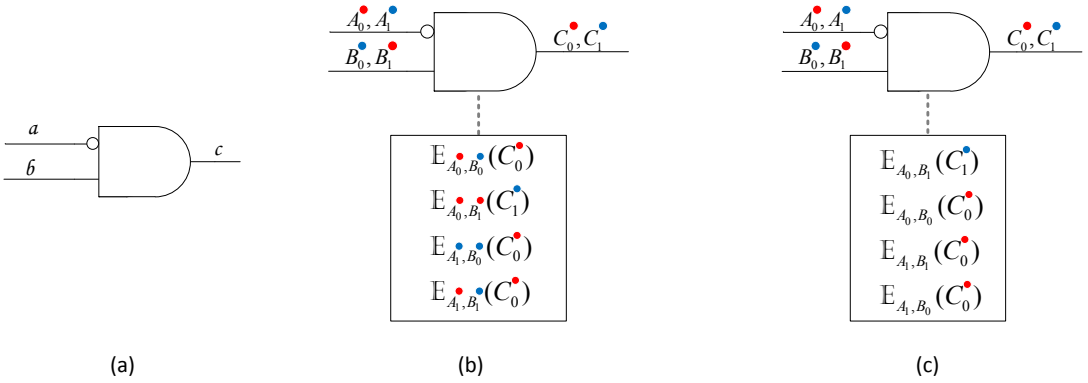

FIGURE 4.1: (a) The gate to be evaluated. (b) Label assignment. (c) Rearrangement of ciphertexts canonically with respect to input labels.

4.2 Point and Permute

The evaluator needs to know which one of the ciphertexts for a gate must be decrypted during the evaluation process. However, he cannot be allowed to deduce the truth value of any of inputs or outputs. The oldest and yet secure method achieving is *point and permute* (P&P), and suggested by Beaver *et al.* in [45].

`Garbling:`

1. Alice and Bob want to compute the output of the gate in Figure 4.1 (a) where a and b is the input c is the output.

2. Alice chooses masking values of wires such that each masking value has one of the two possible labels (the one for a is either A_0 or A_1, the one for b is either B_0 or B_1, and the one for c is either C_0 or C_1), and for a given wire both masking values have different labels (see Figure 4.1 (b)). The label needs to be something that can be directly detectable from the masking value (*e.g.*, its last bit). For example, if the masking value on the wire a corresponding to the truth value `FALSE` (A_0) has 0 on the last bit, then the masking value for the truth value `TRUE` (A_1) must have 1 on the last bit. The truth value cannot be detected from the label of the masking value. Alice encrypts the possible output masking values of the gate with the corresponding input masking values ($E_{A_0,B_0}(C_0)$, $E_{A_0,B_1}(C_1)$, $E_{A_1,B_0}(C_0)$, and $E_{A_1,B_1}(C_0)$).

3. Alice rearrange the ciphertexts with respect to the input labels, as in Figure 4.1 (c). During the evaluation, Bob will know which ciphertext he must decrypt from

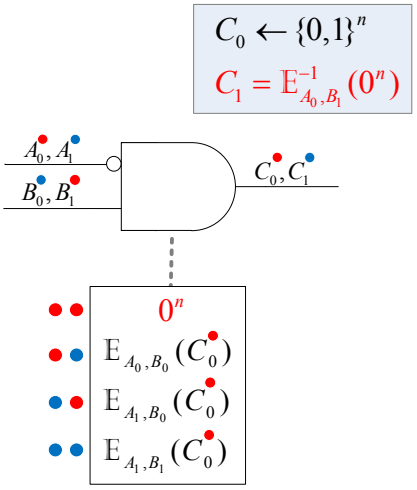

FIGURE 4.2: Garbled row reduction 3 ciphertexts.

the labels of the inputs. This way, ciphertexts are ordered unrelated to the truth values of wires and any information leakage is prevented.

The number of ciphertexts per gate that needs to be transmitted is 4 in this method. 4 encryption and 1 decryption are the computational cost for each gate (see Table 4.1).

4.3 Garbled Row Reduction 3 Ciphertexts

Instead of choosing the masking values of the output of a gate randomly as in P&P (§4.2), in [14] Naor *et al.* suggested a smarter way, called *garbled row reduction 3 ciphertexts* (GRR3).

`Garbling:`

1. Alice and Bob want to compute the output of the gate in Figure 4.2.

TABLE 4.1: Optimization Scoreboard (P&P)

Method	**Odd / Even Gate Size**	**Enc. Time per Odd / Even Gate**	**Dec. Time per Odd / Even Gate**
P&P	4 ct / 4 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`

ct: ciphertexts; `edt`: total encryption and/or decryption time

2. Alice choose the masking value of the first output in label order such that all bits of the resulting ciphertext is 0 (*i.e.*, by decrypting all 0, $C_1 \leftarrow E^{-1}_{A_0,B_1}(0^n)$). The masking value reached will still be pseudo-random.

3. Since there is no need to send the first ciphertext, sending 3 ciphertexts per gate suffices.

Although GRR3 results in smaller-sized garbled circuits than the ones resulted from P&P (§4.2), it has little affect on the computation cost since the gain coming from one less encryptions goes to the decryption of the first ciphertext (see Table 4.2).

4.4 Free `XOR`

One of the greatest jumps in the garbled circuit technology has been the free `XOR` technique, which is proposed by Kolesnikov and Schneider in [46]. It basically eliminates the need for any ciphertext transmission and any calculation for `XOR` gates. The function can be compiled such that the number other gates are minimized. Usually they are just `AND` gates, since (`XOR`, `AND`) is Turing complete.[1]

`Garbling:`

1. Alice and Bob want to compute the output of the `XOR` gate in Figure 4.3 (a).

2. The masking value for `TRUE` in a wire a can be written as the one for `FALSE` in that wire A `XOR`ed with some offset $\triangle_A$, which is a random value having the same number of bits as A and B, as in Figure 4.3 (b). The masking value for `FALSE` becomes A, and the masking value for `TRUE` becomes $A \oplus \triangle_A$. Alice also writes the masking values of b and c the same way.

[1]The number of `AND` gates in the Boolean functions is called *multiplicative complexity*. Reducing it at the expense of increasing `XOR`s is already an active research topic [27].

Table 4.2: Optimization Scoreboard (GRR3)

Method	**Odd / Even Gate Size**	**Enc. Time per Odd / Even Gate**	**Dec. Time per Odd / Even Gate**
P&P (§4.2)	4 ct / 4 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
GRR3	3 ct / 3 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`

ct: ciphertexts; `edt`: total encryption and/or decryption time

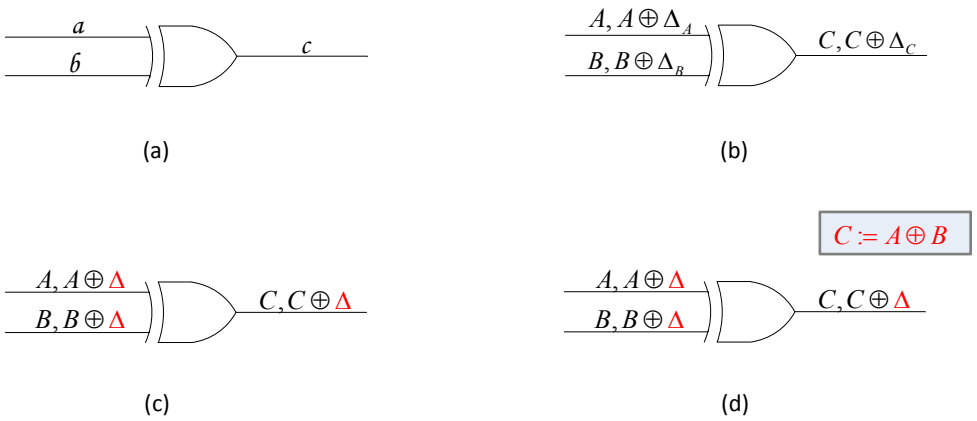

FIGURE 4.3: (a) `XOR` gate with masked values on its wires. (b) `XOR` gate whose masked values interpreted with offsets. (c) `XOR` with the same offset in the masked values on wires. (d) `XOR` gate arranged for free `XOR` technique

3. Alice sets the offsets of all wires be the same secret value $\triangle$ as in Figure 4.3 (c). Even if there are more than one gate in a circuit, all wires must be given the same offset so that the free `XOR` method can be applied. Offset must be kept as a secret by the garbler.

4. Alice choose the masking value for `FALSE` in the output, `XOR` of those for `FALSE` in the inputs as in Figure 4.3 (d). This makes transmitting any ciphertext for an `XOR` gate unnecessary.

`Evaluating:`

5. Bob just `XOR`s the masking value of the inputs to calculate the masking value of the output.

`AND` gates can be encrypted as in GRR3 (§4.3), and 3 cipher texts needs to be transmitted (see Figure 4.4). Labels still exist, and ciphertexts must be ordered accordingly. The offset must be chosen such that for a given wire both masking values have different labels (*e.g.*, its `lsb` must be 1 if the label is the last bit). Since the same offset is used in both inputs and the payload, there is a need for a circularity assumption for the encryption scheme used [47].

Free `XOR` technique, makes `XOR`s completely free for transmission and computation in both the garbler's side and the evaluator's side. This has a huge impact, not just for freeing `XOR`s but also permitting the minimization of the other gates at the expense of increasing `XOR`s (see Table 4.3).

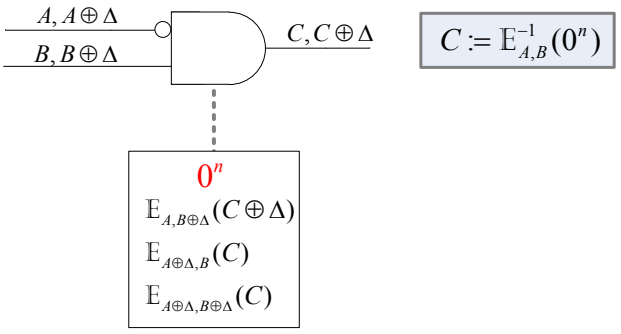

FIGURE 4.4: Encryptions of a gate other than `XOR` in the free `XOR` technique.

4.5 Garbled Row Reduction 2 Ciphertexts

Pinkas *et al.* proposed a method called *garbled row reduction 2 ciphertexts* (GRR2) in order to reduce the number of transferred ciphertexts in [24]. GRR2 is based on Shamir's secret sharing (§2.6). It is especially good for reducing the size in case of abundant `AND` gates [24].

`Garbling:`

1. Alice and Bob want to compute the output of the odd gate in Figure 4.5 (a).
2. Alice calculates K_1, K_2, K_3, and K_4 by decrypting all 0 for all possible input combinations (*e.g.*, $K_1 \leftarrow E^{-1}_{A_0,B_0}(0^n)$, $K_2 \leftarrow E^{-1}_{A_0,B_1}(0^n)$, $K_3 \leftarrow E^{-1}_{A_1,B_0}(0^n)$, $K_4 \leftarrow E^{-1}_{A_1,B_1}(0^n)$).
3. Using the rows which give the same output (in this case the rows 1, 3, 4) Alice plots a 2^{nd} degree polynomial $P(x)$ (*e.g.*, the red parabolas in Figure 4.5 (b)).
4. Alice also plots another 2^{nd} degree polynomial $Q(x)$ from the excluded row (here the row 2), $P(5)$, and $P(6)$ (*e.g.*, the blue parabolas in Figure 4.5 (b)).

TABLE 4.3: Optimization Scoreboard (Free `XOR`)

Method	**Odd / Even Gate Size**	**Enc. Time per Odd / Even Gate**	**Dec. Time per Odd / Even Gate**
P&P (§4.2)	4 ct / 4 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
GRR3 (§4.3)	3 ct / 3 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
Free `XOR`	3 ct / free	4 `edt` / free	1 `edt` / free

ct: ciphertexts; `edt`: total encryption and/or decryption time

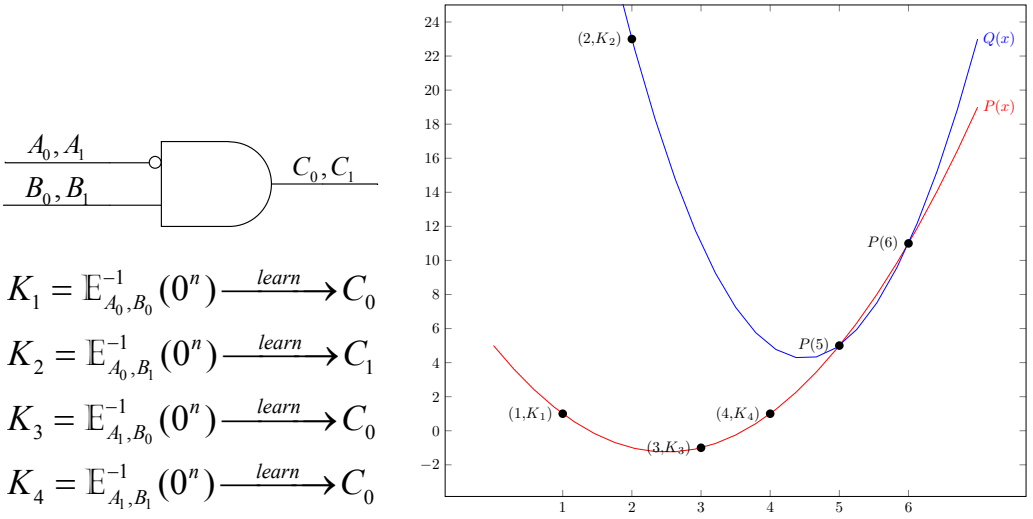

FIGURE 4.5: (a) The odd gate to be garbled. (b) Plots of two polynomials obtained from K_1, K_2, K_3, and K_4.

`Evaluating:`

5. Alice sends only the intersection points $P(5)$ and $P(6)$. Bob will get another point by decrypting all 0 with the masking values that he gets in the input. He will be able to reaching only one of the polynomials, not knowing which one. The output masking value will be the evaluation of this polynomial at $x = 0$ (*e.g.*, $C_0 = P(0)$ and $C_1 = Q(0)$).

The position in this scheme leaks information. Moreover, since the wire masking values are not chosen but calculated pseudo-random values, it is impossible to directly use the P&P (§4.2) technique. Instead, Pinkas *et al.* proposed adding a one bit *external value* c_i for each wire. External values, like labes, are different for the `TRUE` and `FALSE` truth values unrelated to the truth value. Just like labels, external values are used for ordering. To calculate the external value of the output of a gate, 4 additional M_r bits are sent. The evaluator, then, just needs to `XOR` the first bits of both input masking values and the related M_r bit to find out the output external value. Since he does not know the masking values for other truth values of the input wires, he cannot find out the external values for the other output.

`Garbling:`

Table 4.4: Optimization Scoreboard (GRR2)

Method	**Odd / Even Gate Size**	**Enc. Time per Odd / Even Gate**	**Dec. Time per Odd / Even Gate**
P&P (§4.2)	4 ct / 4 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
GRR3 (§4.3)	3 ct / 3 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
Free `XOR` (§4.4)	3 ct / free	4 `edt` / free	1 `edt` / free
GRR2	2 ct / 2 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`

ct: ciphertexts; `edt`: total encryption and/or decryption time

1. For an even gate, Alice similarly calculates K_1, K_2, K_3, and K_4 as in the odd gate case, in order of the external values.

2. Somewhat differently from the previous procedure, she plots the two 1^{st} degree polynomials each passing through the two points which correspond to the same output value. For instance, if both K_1 and K_3 are for the rows corresponding to `TRUE`, she plots $P(x)$ passing through $(1, K_1)$ and $(3, K_3)$ and $Q(x)$ passing through $(2, K_2)$ and $(4, K_4)$. She sends $P(5)$ and $Q(5)$, along with the 4 additional M_r bits. She makes sure that ordering $P(5)$ and $Q(5)$ is according to the external value of the output of the gate just like using them the same as label bits, so that the evaluator know which one to use.

`Evaluating:`

3. The evaluator decrypts all 0 with the masking values of the inputs. With two points in hand he plots the 1^{st} degree polinomial evaluate it at $x = 0$ and reaches the output masking value.

Referring to Shamir's secret sharing (§2.6), two t-length values and 4 M_r bits $(2t + 4)$ are needed to be sent per gate. For the sake of simplicity, we can take it as 2 ciphertexts per gate (see Table 4.4).

Although GRR2 is good for reducing the sizes of odd gates, it has a major drawback: incompatibility with free `XOR` (§4.4). This is because the output masking values of the gates garbled with the GRR2 technique are pseudo-random numbers which cannot be set to the same offset.

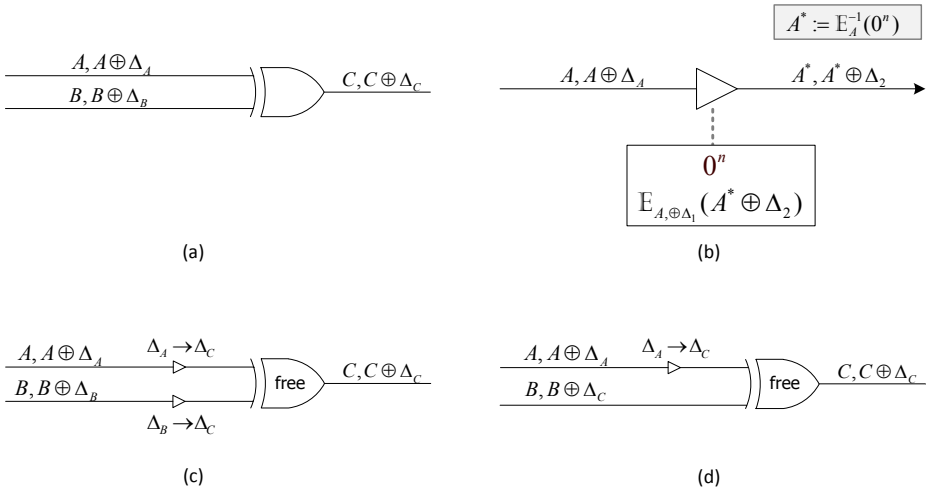

FIGURE 4.6: (a) An `XOR` gate with different offsets in its inputs and output. (b) A buffer gate to carry the offset of a wire. (c) An `XOR` gate offsets of whose inputs are carried to the offset of its output by two imaginary buffer gates. (d) An `XOR` gate the offset of whose an input is carried to the offset of its output by an imaginary buffer gate.

4.6 Fle`XOR`

The incompatibility of free `XOR` (§4.4) and GRR2 (§4.5) causes an inconvenient situation where both may be better for different circuits depending on the proportion of `XOR` and `AND` gates. To solve this issue, Kolesnikov *et al.* proposed the fle`XOR` technique in [44]. Fle`XOR` may reduce the number of ciphertexts for an `XOR` gate even if it has different offsets on its wires. With this technique, `XOR` gates requires 1 or less ciphertext most of the time. It may cost 2 ciphertexts, only if the output masked value of the XOR gate has different offset from its inputs. Actually, most of the time, the output masked value may be chosen such that it has the same offset at least one of the inputs

`Garbling:`

1. Alice and Bob want to compute the output of the `XOR` gate in Figure 4.6 (a).

2. The idea is that if it was possible to carry the input wires to the same offset level with the output wire, which is $\triangle_C$, the `XOR` gate would be free. Figure 4.6 (b) depicts an imaginary buffer gate which can be used to carry the offset of a wire. Alice encrypts the output masked values with their corresponding inputs as $E_A(A^*)$ and $E_{A\oplus\triangle_1}(A^*\oplus\triangle_2)$. She order them by P&P (§4.2), and since A^* can be any random value, she can let the first one in order all 0. Therefore, sending just one cipher text for a buffer gate suffices.

TABLE 4.5: Optimization Scoreboard (Fle`XOR`)

Method	**Odd / Even Gate Size**	**Enc. Time per Odd / Even Gate**	**Dec. Time per Odd / Even Gate**
P&P (§4.2)	4 ct / 4 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
GRR3 (§4.3)	3 ct / 3 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
Free `XOR` (§4.4)	3 ct / free	4 `edt` / free	1 `edt` / free
GRR2 (§4.5)	2 ct / 2 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
Fle`XOR`	2 ct / {0,1,2} ct	4 `edt` / {0,2,4} `edt`	1 `edt` / {0,1,2} `edt`

ct: ciphertexts; `edt`: total encryption and/or decryption time

3. Alice needs at most two imaginary buffer gates for an `XOR` gate to carry the inputs to the same offset level as the outputs (see Figure 4.6 (c)).

4. Most of the time, one imaginary buffer per `XOR` gate will be enough since Alice can let the offset of the output the same as one of the inputs (see Figure 4.6 (d)). And if the inputs and the output have the same offset, the `XOR` gate will be free.

fle`XOR` technique can be combined with GRR2 (§4.5) in order to reduce the number of ciphertexts for `AND` gates (see Table 4.5). The combined scheme proposed by Kolesnikov *et al.* can be seen in Algorithm 2.

The notation used in Algorithm 2 is similar to the one in Algorithm 1. `XORGates`(f) denotes the set of `XOR` gates in f. $\mathtt{C}_{X_i}$ denotes the external value of the wire whose masking value is X_i. V_{ij} denotes the value used in the interpolation related to the order ij. m_{ij} denotes the one bit value used to mask the external value. X_{ai} denotes the masking value on the wire a, i being the external value. w_{ai} denotes the truth value on the wire a, i being the external value. c_{ij} denotes the bits sent for the calculation of the external value of the output of a gate, ij being the order coming from the input external values [44].

4.7 Half Gates

The *half gates* method, which is proposed by Zahur *et al.* in [5], proves that sending 2 ciphertexts can be enough for an `AND` gate while `XOR` gates are still free. The same offset is kept throughout the whole circuit wires, like the free `XOR` (§4.4). It is based upon the idea that if one of the sides knows the truth value on an input wire of an `AND` gate, it

Algorithm 2 The combined fleXOR and GRR2 scheme proposed by Kolesnikov *et al.* in [44].

procedure $\mathtt{G_B}(1^k, f)$ ▷ Garbling phase
  $(n, m, q, A', B', G) \leftarrow f$
  for $i \in \{1, \ldots, n\}$ **do** $t \leftarrow \{0,1\}$, $X_i^0 \leftarrow \{0,1\}^{k-1}t$, $X_i^1 \leftarrow \{0,1\}^{k-1}\bar{t}$
  end for
  for $g \in \{n+1, \ldots, n+q\}$ in a **safety-respecting** order **do** $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
    if $g \in \mathtt{XORGates}(f)$ **then**
      if $X_a^0 \oplus X_a^1 = X_b^0 \oplus X_b^1$ **then** $X_g^0 \leftarrow X_a^0 \oplus X_b^0$, $X_g^1 \leftarrow X_a^0 \oplus X_b^1$, $P[g] \leftarrow \perp$
      else
        if $\mathtt{C}_{X_a^0} = 0$ **then** $X_a^{\bar{0}} \leftarrow H(X_a^0, g \parallel 00)$, $X_a^{\bar{1}} \leftarrow X_a^{\bar{0}} \oplus X_b^0 \oplus X_b^1$
          $X_g^0 \leftarrow X_a^{\bar{0}} \oplus X_b^0$, $X_g^1 \leftarrow X_a^{\bar{0}} \oplus X_b^1$, $P[g] \leftarrow H(X_a^1, g \parallel 00) \oplus X_a^{\bar{1}}$
        else $X_a^{\bar{1}} \leftarrow H(X_a^1, g \parallel 00)$, $X_a^{\bar{0}} \leftarrow X_a^{\bar{1}} \oplus X_b^0 \oplus X_b^1$
          $X_g^0 \leftarrow X_a^{\bar{0}} \oplus X_b^0$, $X_g^1 \leftarrow X_a^{\bar{0}} \oplus X_b^1$, $P[g] \leftarrow H(X_a^0, g \parallel 00) \oplus X_a^{\bar{0}}$
        end if
      end if
      $\mathtt{C}_{X_g^0} \leftarrow \mathtt{C}_{X_a^0} \oplus \mathtt{C}_{X_b^0}$, $\mathrm{C}_{X_g^1} \leftarrow \mathtt{C}_{X_g^0}^{-}$
    else
      $\mathrm{C}_{X_g^0} \leftarrow \{0,1\}$ $\mathtt{C}_{X_g^1} \leftarrow \mathtt{C}_{X_g^0}^{-}$
      for $(i,j) \in \{0,1\}^2$ **do** $V_{ij} \parallel m_{ij} \leftarrow H(X_{ai}, X_{bj}, g \parallel i \parallel j)$, $c_{ij} \leftarrow \mathtt{C}_{X_g^{w_{ai} \wedge w_{bj}}} \oplus m_{ij}$
      end for
      $Q \leftarrow interp\{(2i + j, V_{ab}) \mid w_{ai} \wedge w_{bj} = 0\}$
      $R \leftarrow interp\{(2i + j, V_{ab} \mid w_{ai} \wedge w_{bj} = 1), (4, Q(4)), (5, Q(5))\}$
      $X_g^0 \leftarrow Q(-1)$, $X_g^1 \leftarrow R(-1)$, $P[g] \leftarrow (Q(4), Q(5), c_{00}, c_{01}, c_{10}, c_{11})$
    end if
  end for
  $F \leftarrow (n, m, q, A', B', P)$, $e \leftarrow (X_1^0, X_1^1, \ldots, X_n^0, X_n^1)$, $d \leftarrow (X_{n+q-m+1}^0, X_{n+q-m+1}^1, \ldots, X_{n+q}^0, X_{n+q}^1)$
  return (F, e, d)
end procedure

procedure $\mathtt{E_N}(e, x)$ ▷ Encoding phase
  $(X_1^0, X_1^1, \ldots, X_n^0, X_n^1) \leftarrow e$, $x_1 \ldots x_n \leftarrow x$, $X \leftarrow (X_1^{x_1}, \ldots, X_n^{x_n})$, **return** X
end procedure

procedure $\mathtt{E_V}(F, X)$ ▷ Evaluating phase
  $(n, m, q, A', B', P) \leftarrow F$, $(X_1, \ldots, X_n) \leftarrow X$
  for $g \leftarrow n+1$ **to** $n+q$ **do** $a \leftarrow A'(g)$, $b \leftarrow B'(g)$
    if $g \in \mathtt{XORGates}(f)$ **then**
      if $P[g] \leftarrow \perp$ **then** $X_g \leftarrow X_a \oplus X_b$
      else
        if $\mathtt{C}_{X_a} = 0$ **then** $X_a^{-} \leftarrow H(X_a, g \parallel 00)$
        else $X_a^{-} \leftarrow P[g] \oplus H(X_a, g \parallel 00)$
        end if
      end if
      $\mathtt{C}_{X_g} \leftarrow \mathtt{C}_{X_a} \oplus \mathtt{C}_{X_b}$
    else
      $V* \parallel m* \leftarrow H(X_a, X_b, g \parallel C_{X_a} \parallel C_{X_b})$
      $R* \leftarrow interp\{(2C_{X_a} + C_{X_b}, V*), (4, Q(4)), (5, Q(5))\}$
      $X_g \leftarrow R*(-1)$, $\mathtt{C}_{X_g} \leftarrow c_{\mathtt{C}_{X_a}\mathtt{C}_{X_b}} \oplus m*$
    end if
  end for
  return $(X_{n+q-m+1}, \ldots, X_{n+q})$
end procedure

procedure $\mathtt{D_E}(d, Y)$ ▷ Decoding phase
  $(Y_1, \ldots, Y_m) \leftarrow Y$, $(Y_1^0, Y_1^1, \ldots, Y_m^0, Y_m^1) \leftarrow d$
  for $i \in \{1, \ldots, m\}$ **do**
    if $Y_i = Y_i^0$ **then** $y_i \leftarrow 0$
    else if $Y_i = Y_i^1$ **then** $y_i \leftarrow 1$
    else return $\perp$
    end if
  end for
  return $y \leftarrow y_1 \ldots y_m$
end procedure

is enough to send just one ciphertext. The method divides the `AND` gate into two `AND` gates where one of the parties knows the truth value on an input wire. The name of the method comes from this division.

A, B, C, C_1, and C_2 are the masking values for the wires a, b, c (output of the `AND` gate), c_1 (output of the garbler half gate), and c_2 (output of the evaluator half gate), respectively. $\triangle$ denotes the common offset as in free `XOR` (§4.4).

`Garbling:`

1. Alice and Bob want to compute the output of an `AND` gate whose inputs are a and b.

2. An `AND` gate can be written as an `XOR` of two `AND` gates as in Equation (4.1) where r is a randomly chosen bit only known to Alice. Alice chooses it to be the label bit of the B, which is the masked value for `FALSE` on the wire b. r is still unknown to Bob.

$$a \wedge b = (a \wedge r) \oplus [a \wedge (b \oplus r)] \tag{4.1}$$

`Garbler Half Gate:`

3. $a \wedge r$ is the garbler half gate, whereas $a \wedge (b \oplus r)$ is the evaluator half gate. For the output of the garbler half gate $c_1 \leftarrow a \wedge r$, Alice needs to send $E_B(C_1)$ and $E_{B \oplus \triangle}(C_1 \oplus r\triangle)$. Since she knows the value of r, there is just 2 input combinations. She orders the ciphertexts with respect to the label bit of b. Row reduction (§4.3) is also possible by letting the 1^{st} ciphertext in all 0. She calculates the 2^{nd} ciphertext from the value she reaches by decrypting the first one. Thus, sending just 1 ciphertext is enough for the garbler half gate.

4. During the evaluation of the garbler half gate, Bob decrypts the related cipher text depending on the label bit of the masking value on the wire b. Since the order is by labels he can not learn the truth value of b.

`Evaluator Half Gate:`

5. For the evaluator half gate, Alice needs to let Bob learn $q = b \oplus r$ without learning b or r. Actually, it is whatever Bob gets as the label bit of the masked value on wire b. This was the main reason why r was chosen as the label bit of B in the beginning.

6. To garble the evaluator half gate $c_2 \leftarrow a \wedge q$, there are two ways Alice may go depending on the value of r. If r is `FALSE`, Alice sends two ciphertexts $E_B(C_2)$ and $E_{B \oplus \triangle}(C_2 \oplus A)$ in this order strictly. Otherwise, Alice sends two ciphertexts $E_{B \oplus \triangle}(C_2)$ and $E_B(C_2 \oplus A)$ in this order strictly. Moreover, the 1^{st} ciphertext can be let all 0 and the 2^{nd} one can be calculated from it. Therefore, sending only one ciphertext for the evaluator half gate also suffices.

7. If Bob gets `FALSE` as q, he decrypts the first ciphertext using the masking value on the wire b, arriving at the masking value of the output of the evaluator half gate. Otherwise, he decrypts the second ciphertext using the value on the wire b, and `XOR`s the result with the masking value on the wire a, arriving at the masking value of the output of the half gate.

The evaluator does not learn the truth values of a, b, r, or c_2 (if $q = 1$, of course, otherwise he learns c_2). In the end, the results of the half gates must be `XOR`ed, in order to obtain the final output of the `AND` gate.

With the half gates technique, an `AND` gate costs 2 cipher texts and `XOR`s are free, which makes the half gates technique the optimum from size point of view among the methods developed so far (see Table 4.6). Zahur *et al.* have also proven that decreasing the size of an `AND` gate further is impossible.

Table 4.6: Optimization Scoreboard (Half Gates)

Method	**Odd / Even Gate Size**	**Enc. Time per Odd / Even Gate**	**Dec. Time per Odd / Even Gate**
P&P (§4.2)	4 ct / 4 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
GRR3 (§4.3)	3 ct / 3 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
Free `XOR` (§4.4)	3 ct / free	4 `edt` / free	1 `edt` / free
GRR2 (§4.5)	2 ct / 2 ct	4 `edt` / 4 `edt`	1 `edt` / 1 `edt`
Fle`XOR` (§4.6)	2 ct / {0,1,2} ct	4 `edt` / {0,2,4} `edt`	1 `edt` / {0,1,2} `edt`
Half Gates	2 ct / free	4 `edt` / free	2 `edt` / free

ct: ciphertexts; `edt`: total encryption and/or decryption time

The Complete Scheme. For a boolean circuit f, a numeric index is assigned to each wire in the circuit. The sets of input wires, output wires, output wires of `XOR` gates in $faredenotedas\texttt{Inputs}(f)$, $\texttt{Outputs}(f)$, and $\texttt{XORGates}(f)$, respectively. These functions can also be applied to garbled version F of f as $\texttt{Inputs}(F)$, $\texttt{Outputs}(F)$, and $\texttt{XORGates}(F)$. v_i denotes the one bit truth value on the i^{th} wire in a circuit. If the output wire of a gate has index i, that gate is named as i^{th} gate. The wire masking values for `FALSE` and `TRUE` on the i^{th} wire is denoted as $W_i^0, W_i^1 \in \{0,1\}^k$, respectively. The security parameter of the scheme is denoted as k. For each wire masking value W, the label bit is its least significant bit $\texttt{lsb}W$. For the i^{th} wire, define $p_i = \texttt{lsb}W_i^0$. Being named as the permute bit of the wire, that value is a secret kept by the generator. Intuitively, if label bit a masking value on a wire is s_i, that masking value is $W_i^{s_i \oplus p_i}$, and corresponds to the truth value $s_i \oplus p_i$. W_i implies that the evaluator does not know v_i. The free `XOR` offset is denoted as $R \in \{0,1\}^k$. We have $\texttt{lsb}R = 1$ so that $\texttt{lsb}W_i^0 \neq \texttt{lsb}W_i^1$, and the complementary masking values on wires have different label bits. Sometimes $\wedge$ is omitted and two symbols is juxtaposed to imply `AND` ($ab = a \wedge b$). $H : \{0,1\}^k \times \mathbb{Z} \to \{0,1\}^k$ denotes a hash-function that is usable in garbled circuits.

$$(v_\mathrm{a}, v_\mathrm{b}) \to (a_\mathrm{a} \oplus v_\mathrm{a}) \wedge (a_\mathrm{b} \oplus v_\mathrm{b}) \oplus a_\mathrm{c} \tag{4.2}$$

The technique can be further generalized such that it can be applied any odd gate (`OR`, `NOR`, `NAND`, etc.), since all of them can be written as in Equation (4.2) where a_a, a_b, a_c are constants. For example, an `AND` gate results from setting all to `FALSE`, an `OR` gate results from setting all to `TRUE`. The construction of half gate is shown step-by-step in Table 4.7. Note that the a values does not affect what the evaluator does.

TABLE 4.7: The construction of half gates for computing Equation (4.2) [5].

Generator half gate: p_b **known to generator**	**Evaluator half gate:** $v_b \oplus p_b$ **known to evaluator**
Computes:	Computes:
$f_G(v_a, p_b) \leftarrow (v_a \oplus a_a)(p_b \oplus a_b) \oplus a_c$	$f_E(v_a, v_b \oplus p_b) \leftarrow (v_a \oplus a_a)(v_b \oplus p_b)$
Before GRR and Permutation:	Before GRR:
$H(W_a^0) \oplus f_G(0, p_b)R \oplus W_{Gc}^0$	$H(W_b^{p_b}) \oplus W_{Ec}^0$
$H(W_a^1) \oplus f_G(1, p_b)R \oplus W_{Gc}^0$	$H(W_b^{p_b \oplus 1}) \oplus W_{Ec}^0 \oplus W_a^{a_a}$
After GRR and permutation:	After GRR (permutation not needed):
$T_{Gc} \leftarrow H(W_a^0) \oplus H(W_a^1) \oplus (p_b \oplus a_b)R$	$T_{Ec} \leftarrow H(W_b^0) \oplus H(W_b^1) \oplus W_a^{a_a}$
$W_{Gc}^0 \leftarrow H(W_a^{p_a}) \oplus f_G(p_a, p_b)R$	$W_{Ec}^0 \leftarrow H(W_b^{p_b})$
Generator sends T_{Gc}	Generator sends T_{Ec}

The complete garbling procedure for an entire circuit proposed by Zahur *et al.* is shown in Algorithm 3 [5]. All gates are assumed to be either an `AND` or an `XOR` gate. Since `De` never returns $\perp$, this scheme does not satisfy the authenticity criterion. In order to make it authentic, Zahur *et al.* propose the following changes:

- The **for**-loop on Line 13 of Algorithm 3 must be changed as:

---

for $i \in$ `Outputs`(f) **do**
  $j \leftarrow$ `NextIndex`()
  $d_i \leftarrow (H(W_i^0, j), H(W_i^1, j))$
end for

- The **for**-loop on Line 54 of Algorithm 3 must be changed as:

---

for $d_i \in d$ **do** $j \leftarrow$ `NextIndex`(), parse $(h_0, h_1) \leftarrow d_i$
  if $H(Y_i, j) = h_0$ **then** $y_i \leftarrow 0$
  else if $H(Y_i, j) = h_1$ **then** $y_i \leftarrow 1$
  else return $\perp$
  end if
end for

Algorithm 3 The complete half gates garbling scheme proposed by Zahur *et al.* in [5].

procedure $\textsc{Gb}(1^k, f)$ ▷ `Garbling phase`
  $R \twoheadleftarrow \{0,1\}^{k-1}1$
  for $i \in \texttt{Inputs}(f)$ **do**
    $W_i^0 \twoheadleftarrow \{0,1\}^k$, $W_i^1 \leftarrow W_i^0 \oplus R$, $e_i \leftarrow W_i^0$
  end for
  for $i \notin \texttt{Inputs}(f)$ **do** {*in topo. order*}
    $\{a, b\} \leftarrow \texttt{GateInputs}(f, i)$
    if $i \in \texttt{XORGates}(f)$ **then** $W_i^0 \leftarrow W_a^0 \oplus W_b^0$
    else $(W_i^0, T_{Gi}, T_{Ei}) \leftarrow \textsc{GbAnd}(W_a^0, W_b^0)$, $F_i \leftarrow T_{Gi}, T_{Ei}$
    end if
    $W_i^1 \leftarrow W_i^0 \oplus R$
  end for
  for $i \in \texttt{Outputs}(f)$ **do**
    $d_i \leftarrow \texttt{lsb}(W_i^0)$
  end for
  return (F, e, d)
end procedure

private procedure $\textsc{GbAnd}(W_a^0, W_b^0)$ ▷ `Garbling AND gates`
  $p_a \leftarrow \texttt{lsb}(W_a^0)$, $p_b \leftarrow \texttt{lsb}(W_b^0)$
  $j \leftarrow \texttt{NextIndex}()$, $j' \leftarrow \texttt{NextIndex}()$
  {*First half gate*}
  $T_G \leftarrow H(W_a^0, j) \oplus H(W_a^1, j) \oplus p_b R$
  $W_G^0 \leftarrow H(W_a^0, j) \oplus p_a T_G$
  {*Second half gate*}
  $T_E \leftarrow H(W_b^0, j') \oplus H(W_b^1, j') \oplus W_a^0$
  $W_E^0 \leftarrow H(W_b^0, j') \oplus p_b(T_E \oplus W_a^0)$
  {*Combine two halves*}
  $W_0 \leftarrow W_G^0 \oplus W_E^0$
  return (W^0, T_G, T_E)
end private procedure

procedure $\textsc{En}(e, x)$ ▷ `Encoding phase`
  for $e_i \in e$ **do** $X_i \leftarrow e_i \oplus x_i R$
  end for
  return X
end procedure

procedure $\textsc{Ev}(F, X)$ ▷ `Evaluating phase`
  for $i \in \texttt{Inputs}(F)$ **do**
    $W_i \leftarrow X_i$
  end for
  for $i \notin \texttt{Inputs}(F)$ **do** {*in topo. order*}
    $\{a, b\} \leftarrow \text{GateInputs}(F, i)$
    if $i \in \texttt{XORGates}(F)$ **then** $W_i \leftarrow W_a \oplus W_b$
    else $s_a \leftarrow \texttt{lsb}(W_a)$, $s_b \leftarrow \texttt{lsb}(W_b)$, $j \leftarrow \texttt{NextIndex}()$, $j' \leftarrow \texttt{NextIndex}()$
      $T_{Gi}, T_{Ei} \leftarrow F_i$, $W_{Gi} \leftarrow H(W_a, j) \oplus s_a T_{Gi}$, $W_{Ei} \leftarrow H(W_b, j') \oplus s_b(T_{Ei} \oplus W_a)$
      $W_i \leftarrow W_{Gi} \oplus W_{Ei}$
    end if
    $W_i^1 \leftarrow W_i^0 \oplus R$
  end for
  for $i \in \texttt{Outputs}(F)$ **do** $Y_i \leftarrow W_i$
  end for
  return Y
end procedure

procedure $\textsc{De}(d, Y)$ ▷ `Decoding phase`
  for $d_i \in d$ **do** $y_i \leftarrow d_i \oplus \text{lsb}Y_i$
  end for
  return y
end procedure

4.8 Our Compatibility Analysis of Garbled Circuit Optimizations

We conclude this chapter with a useful table which reflects the compatibility of garbled circuit optimizations with each other (see Table 4.8). ✓ and X stand for compatible and non-compatible, respectively. For the use of external value (Ext. Val.), see Section 4.5.

TABLE 4.8: Compatibility of Garbled Circuit Optimization Techniques.

	P&P	**GRR3**	**Free XOR**	**GRR2**	**FleXOR**	**Half Gates**
P&P		✓	✓	✓(Ext. Val.)	✓(Ext. Val.)	✓
GRR3	✓		✓	X	✓	✓
Free XOR	✓	✓		X	X	✓
GRR2	✓(Ext. Val.)	X	X		✓	X
FleXOR	✓(Ext. Val.)	✓	X	✓		X
Half Gates	✓	✓	✓	X	X	

Ext. Val.: External Value (§4.5)

Chapter 5

Practical Implementations of Yao's Protocol

Various implementations have been developed so far based on Yao's protocol. Many of them utilize Yao's protocol for MPC applications, although some targets *Private Function Evaluation* (PFE). A comprehensive catalogue of them would have been far from the reach of just a master's thesis work. So, we will explain only some of them which are supposed to be helpful for people to see Yao's protocol in practise. They also reflect the importance of Yao's protocol and the areas it can be applied in the future. First, we will start with introducing some of the generic MPC solutions that use Yao's protocol. We compare those generic implementations in terms of their use of garbled circuit optimizations. At the end, we will present some real-world applications.

5.1 Generic Usage of Yao's Protocol in Practice

5.1.1 Pipelined Implementation (`FastGC`)

The memory required to store the entire garbled circuit is generally a limitation. Huang *et al.* proposed pipelining optimization in their framework in [21] to reduce the required memory. The garbled circuit generation and evaluation procedures can be done simultaneously, eliminating the need for keeping the entire garbled circuit in memory and the need for preparation of the entire garbled circuit before its transmission to the evaluator,

which results in a decrease in total Yao's protocol time. `FastGC` framework automates pipelined implementation, so that the only need remaining is the construction of the desired circuit [21].

At the beginning of the computation the circuit structure is instantiated by both the garbler and the evaluator. While the protocol is being executed, the generator garbles each gate in topological[1] order, and transmits it over the network as soon as it is produced. When a garbled gate is received by the evaluator, it is associated with the corresponding gate of the circuit and evaluated. A gate is eliminated as soon as it has been evaluated, so that the memory use would be minimal. This technique is called *pipelined implementation*. Note that it also reduces total Yao's protocol time of at the expense that both parties needs to be online at the same time.

5.1.2 Garbled RAM

The notion of *garbled RAM* was introduced by Lu and Ostrovsky in [48]. Gentry *et al.* have later improved it using *identity-based encryption* (IBE)[2] in [50] for provable security. It differs from Yao's garbled circuits in that it permits direct garbling of a RAM program, *without converting it into a boolean circuit*. A RAM program whose run-time is T can be converted into a Turing Machine whose run-time is $O(T^3)$ resulting in a boolean circuit of size $O(T^3 \log T)$, whereas the size and computation time of a garbled RAM program is only proportional to its running time on a RAM [50]. The inefficiency is even more prominent in the setting of *big data* [50]. In this case, efficient programs, such as binary search, run in sub-linear time with the size of the data, however their boolean circuit representations run in linear time with the size of the data.

Just like garbled circuits, garbled RAM includes a garbler who garbles the program, and sends it to the evaluator. Evaluator evaluates the garbled program using the garbled inputs and, unlike the case of Yao's protocol, outputs the actual output of the RAM program. Like the garbled circuits, garbled RAM targets security againist semi-honest adversaries ((§2.2.1). Gentry *et al.*'s scheme of garbled RAM is explained in detail below [50].

[1] Safety-respecting if the garbling method is fle`XOR`.

[2] Identity based encryption (IBE) is a form of public key encryption (§2.5.2) where a user's public key is his identity. In generic public key cryptosystems, private keys are chosen randomly and public keys are produced from them. However, in IBE the private keys are generated from users' public keys [49].

The notation $P^D(x)$ denotes a RAM program P which accesses a memory containing data D and takes an input x. Imagining D as a *huge database* controlled by the evaluator and P as a *database query* that has read or write access to the database and whose parameter is a value x (like P searches x in D) would help for understanding the notions.

A garbled RAM scheme can be used to garble P, D, x into $\bar{P}$, $\bar{D}$, $\bar{x}$, such that $\bar{P}$, $\bar{D}$, $\bar{x}$ reveals only $P^D(x)$. Furthermore, the sizes of $\bar{P}$, $\bar{D}$, $\bar{x}$ are only proportional to their corresponding plain texts. Similar to Yao's the garbled circuits, garbling x consists of providing a subset of masking values.

A RAM program P can be represented as a colleciton of `CPU`-Step Circuits which execute a single `CPU` step. Equation (5.1) shows the execution of `CPU` step j. The input to the circuit $C^P_{\texttt{CPU}}$ is the current `CPU` $\texttt{state}_j$ and a bit $b_j^{\texttt{read}}$ which resides in the memory location assigned in the previous cycle. Its outputs are an updated $\texttt{state}_{j+1}$, the next reading location $i_(j+1)^{\texttt{read}}$, a location $i_{j+1}^{\texttt{write}}$ for writing to (maybe $\perp$), a bit $b_{j+1}^{\texttt{write}}$ to write into that location. The start of the computation $P^D(x)$ is in the initial state $\texttt{state}_1 = x$ and $b_0^{\texttt{read}} = 0$, and it proceeds step-by-step. In each step j, first $b_j^{\texttt{read}}$ is set to $D[i_j^{\texttt{read}}]$, and if $i_j^{\texttt{write}} \neq \perp$, $D[i_j^{\texttt{write}}]$ is set to $b_j^{\texttt{write}}$. The output of the last `CPU` step is the output of the computation $y = P^D(x)$ as `state`. P has *read-only* memory access if it never overwrites any values in memory D (*i.e.*, $i_j^{\texttt{write}}$ is always $\perp$).

$$C^P_{\texttt{CPU}}(\texttt{state}_j, b_j^{\texttt{read}}, i_j^{\texttt{write}}, b_j^{\texttt{write}}) = (\texttt{state}_{j+1}, i_{j+1}^{\texttt{read}}, i_{j+1}^{\texttt{write}}, b_{j+1}^{\texttt{write}}) \tag{5.1}$$

Gentry *et al.* propose their scheme with security against unprotected memory access (UMA) in which the initial contents of the memory D and the complete memory access pattern of `MemAccess` (including the contents) may be learned by the intruder, [50]. They also propose that encrypting the memory contents and applying oblivious RAM is enough for transforming any garbled RAM scheme with UMA security into one providing full security.

Read-only Solution. The garbled memory is made of $\bar{D}[i]$'s, each containing an IBE[2] secret key $sk_{(i,b)}$ for the public key (i, b) where i is the location and b is the data bit $D[i]$. Another future of $\bar{D}[i]$ is that it can remain and be used by the future programs. The garbled input $\bar{x}_j$ to the `CPU` step j is the masking value for the $\texttt{state}_j$, and $\bar{x}_0$

is the masking input $\bar{x}$. The `CPU` step in Equation (5.1) simply becomes the one in Equation (5.2).

$$C^P_{\texttt{CPU}}(\texttt{state}_j, b_j^{\texttt{read}}) = (\texttt{state}_{j+1}, i_{j+1}^{\texttt{read}}) \tag{5.2}$$

(5.3) shows the garbled circuit $\bar{C}^P_{\texttt{CPU,j}}$ of the step j. The problem with garbling the `CPU` step j is that the location of $b_j^{\texttt{read}}$ is not pre-known since it is the output of the previous cycle. Let $\bar{b}_{0j}^{\texttt{read}}$ denote the masking value of $b_j^{\texttt{read}}$ for `FALSE` and $\bar{b}_{1j}^{\texttt{read}}$ denote the masking value of $b_j^{\texttt{read}}$ for `TRUE`. Each garbled step j outputs a translation mapping $\texttt{translate}_{j+1} = (ct_{0(j+1)}, ct_{1(j+1)})$ where $ct_{b(j+1)} = E((i_{j+1}^{\texttt{read}}, b), r_{bj}, \bar{b}_{b(j+1)}^{\texttt{read}})$ calculated[3] by using IBE so that the evaluator can only learn the masking value of $D[i_{j+1}^{\texttt{read}}]$ using the key $\bar{D}[i_{j+1}^{\texttt{read}}]$. $\bar{b}_{0(j+1)}^{\texttt{read}}$, $\bar{b}_{1(j+1)}^{\texttt{read}}$, r_{0j} and r_{1j} are hardcoded in the step circuit j and cannot be learned directly by the evaluator due to the garbling process. $i_{j+1}^{\texttt{read}}$ is not private since the target is UMA security, and so it does not require a masking value.

$$(\bar{x}_{j+1}, i_{j+1}^{\texttt{read}}, \texttt{translate}_{j+1}) \leftarrow \bar{C}^P_{\texttt{CPU,j}}(\bar{x}_j, \bar{b}_{bj}^{\texttt{read}}) \tag{5.3}$$

Each garbled cycle j starts with the decryption of ct_{bj} (the evaluator may know which one to decrypt due to UMA security) to get $\bar{b}_{bj}^{\texttt{read}}$, except for the first cycle where $\bar{b}_{bj}^{\texttt{read}} = \perp$. The last cycle directly outputs $y = P^D(x)$.

Writing to the Memory. Similar to the read-only case, the garbled memory is made of $\bar{D}[i]$'s, each containing an *timed IBE* secret key $sk_{(u,i,b)}$ for the public key (u, i, b) where u is the cycle that i is written last time. The full step given in Equation (5.1) needs to be evaluated. Equation (5.4) shows the garbled circuit $\bar{C}^P_{\texttt{CPU,j}}$ of the step j. Unlike the read-only case, each step j writes $sk_{(j,i,b)}$ to the garbled memory address $i_j^{\texttt{write}}$ (if they are not $\perp$), and outputs $sk_{(j+1,i,b)}$ and $i_{j+1}^{\texttt{write}}$ for writing in the next cycle. Each garbled step j outputs a translation mapping $\texttt{translate}_{j+1} = (ct_{0(j+1)}, ct_{1(j+1)})$ where $ct_{b(j+1)} = E((u_{j+1}, i_{j+1}^{\texttt{read}}, b), r_{bj}, \bar{b}_{b(j+1)}^{\texttt{read}})$ calculated by using timed IBE. Here, the assumption is that there exists a polynomial size circuit `WriteTime` such that $u_{j+1} = \texttt{WriteTime}(j, \bar{x}_j, i_{j+1}^{\texttt{read}})$, and step j can call it. Just like the read-only case, the evaluator can only learn the masking value of $D[i_{j+1}^{\texttt{read}}]$ using the key $\bar{D}[i_{j+1}^{\texttt{read}}]$. $\bar{b}_{0(j+1)}^{\texttt{read}}$, $\bar{b}_{1(j+1)}^{\texttt{read}}$,

[3] r_{bj} is the randomization value to provide semantic security.

r_{0j} and r_{1j} are hardcoded in the step circuit j and cannot be learned directly by the evaluator due to the garbling process. $i_{j+1}^{\texttt{read}}$ and $i_{j+1}^{\texttt{write}}$ are not private since the target is UMA security.

$$(\bar{x}_{j+1}, i_{j+1}^{\texttt{read}}, \texttt{translate}_{j+1}, i_{j+1}^{\texttt{write}}, sk_{(j+1,i,b)}) \leftarrow \bar{C}^{P}_{\texttt{CPU,j}}(\bar{x}_j, \bar{b}_{bj}^{\texttt{read}}, i_j^{\texttt{write}}, sk_{(j,i,b)}) \quad (5.4)$$

Full Security. Gentry *et al.* propose that any garbled RAM scheme that only provides UMA security and only supports program executions with `WriteTime` calls can be transformed into a fully secure garbled RAM scheme for arbitrary programs [50]. This transformation uses *oblivious RAM* (ORAM)[4] to first compile the original program P into a new program P^* that stores/accesses its memory using ORAM. This ensures that the memory contents and access pattern of the compiled program do not reveal anything about those of the original program. Some ORAM schemes already ensure that the compiled program provides `WriteTime` calls.

5.1.3 `JustGarble`

In [37], Bellare *et al.* proposed `JustGarble` framework, which targets optimized garbling of any circuit. It is entirely open-source and can be freely downloaded from `http://cseweb.ucsd.edu/groups/justgarble`. It implements Ga (P&P (§4.2)), GaX (Free `XOR` (§4.4) without GRR3 (§4.3)), and GaXR (Free `XOR` with GRR3), using constant key 128-bit `AES` as the `DKC` (§2.5.6) as in Equation (2.5). It works both ways: garble a boolean circuit, and evaluate a garbled circuit.

`JustGarble` uses a circuit representation called Simple Circuit Description (SCD). It is based on the circuit formulation from [3]. An SCD file consists of values n, m, q, and arrays A, B, and G. If G is not present the file is a topological circuit representation. In `JustGarble`, there are modules for building circuits, garbling boolean circuits, and evaluating garbled circuits. The Build module is useful for constructing circuits, permitting working at the individual gate level or higher. SCD files are written with

[4]Oblivious RAM (ORAM), first proposed by Goldreich and Ostrovsky *et al.* [51], permit a user to hide its access pattern to a remote storage. Although the physical storage locations accessed can be observed by an adversary, it is ensured by ORAM that anything about the real access pattern may not be learned [52].

constructed circuits. The Garble module is utilized for realizing the `Gb` algorithm of the three garbling schemes given. Garble takes a circuit $f = (n, m, q, A, B, G)$ described in an SCD as input and outputs the garbled tables P that compose of the related garbled circuit $F = (n, m, q, A, B, P)$. The inputs to the Evaluate module are a topological circuit$\bar{f} = (n, m, q, A, B)$, the garbled tables P needed for evaluating, and a garbled input X. The garbled output Y is produced. JustGarble also composes of procedures to realize `De`, mapping the garbled output Y to the plain output y [37].

The `JustGarble` implementation of GaXR (Free `XOR` (§4.4) with GRR3 (§4.3)) for 36.5K gate optimized `AES` boolean circuit whose 82% are `XOR` gates has resulted in 5.40 bytes per gate (bpg) as the size, 35.0 cycles per gate (cpg) as the evaluation time, and 63.3 cpg as the garbling time. The JustGarble implementation of GaX for the same circuit, however, has yielded 23.2 cpg as the evaluation time, 55.6 cpg as the garbling time, and 11.5 as the size [37]. (With a 3.201 GHz processor, evaluating the garbled circuit is 7.25 nsec/gate and garbling it is 17.4 nsec/gate.)

5.1.4 `ABY`

`ABY` is a framework for 2PC, proposed by Demmler *et al.* in [39]. Most of the time, a mixture of MPC primitives (GMW protocol (§2.9), Yao, HE (§2.8)...) may yield more efficient implementations than what would have been if just one of them is used. Based on this idea, `ABY` uses **A**rithmetic sharing, **B**oolean sharing, and **Y**ao sharing (§2.6). The framework aims security in the semi-honest model (§2.2.1). `ABY` works like a virtual machine, and high-level languages can be compiled to it. Variables may be either in Cleartext (i.e. one of the parties knows its value, e.g. inputs and outputs) or secret shared among the two parties. `ABY` also allows efficient conversion between the different types of sharings. The user of the framework may decide which sharings to be used depending on the application.

5.1.5 `Obliv-C`

`Obliv-C` is built by Zahur and Evans as an extension of C programming language with secure computation infrastructure [53]. It supports various C features like `pointers`, `typedef`, `struct`, *etc.*, and provides new data types and constructions so that programs

would run on private inputs. It is especially designed for scalable MPC protocols, and to enhance research on new MPC techniques by easing implementation such that just writing a new library is enough instead of building a new compiler for each technique. The source code for `Obliv-C` can be found at `https://oblivc.org`.

5.1.6 `ObliVM`

Liu *et al.* proposed `ObliVM` as a programming framework for MPC [54]. It offers a domain-specific language (`ObliVM-lang`) useful for compilation of programs into suitable representations required for MPC protocols. It also provides high-level programming constructions for MPC infrastructure which can be adapted by non-specialist programmers on security as well. The source code for `ObliVM` can be found at `http://oblivm.com`.

5.1.7 `Frigate`

`Frigate` is designed by Mood *et al.* as a compiler and a circuit interpreter for MPC [55]. It can implement any function that can be written as a boolean circuit and run any MPC primitive that operates on boolean circuits. `Frigate` permits the use of C-like language with constructs and operators specifically designed for representing Boolean circuit efficiently. To improve the efficiency, the compiler is designed to favor `XOR` gates, utilizing structures like Boyar *et al.*'s full adder with four `XOR` and one `AND` [28]. `Frigate` is also *significantly fast* in terms of compilation, interpretation and execution times. The source code for `Frigate` can be found at `https://bitbucket.org/bmood/frigaterelease`.

5.1.8 Comparison Based on Garbling Optimizations Used

Now, we compare the generic frameworks for Yao's protocol based on their use of garbled circuit optimizations (see Table 5.1). ✓ and ✗ stand for compatible and non-compatible, respectively.

TABLE 5.1: Comparison of Generic Frameworks Techniques Based on Their Use of Garbled Circuit Optimizations.

	P&P	**GRR3**	**Free XOR**	**GRR2**	**FleXOR**	**Half Gates**
`JustGarble` **(2013) [37]**	✓	✓	✓	X	X	X
`ABY` **(2015) [39]**	✓	✓	✓	X	X	X
`Obliv-C` **(2015) [53]**	✓	✓	✓	✓	✓	✓
`ObliVM` **(2015) [54]**	✓	✓	✓	X	X	X
`Frigate` **(2016) [55]**	✓	✓	✓	✓	✓	✓

`Obliv-C` and `Frigate` make the use of any garbled circuit optimization possible since they permit the alterations of garbling schemes although those garbling circuit techniques are not built-in. On the other hand, `JustGarble`, `ABY`, and `ObliVM` do not allow changing the built-in garbling constructions, therefore, is limited for the use of state-of-the-art garbled circuit optimization techniques. All of the frameworks allow compilations optimized for reducing the number of odd gates. We can deduce that `Frigate` is the optimum for working with garbled circuits since it offer maximum optimization options while being the most efficient one.

5.2 Real-World Applications

We give two real-world examples indicating the importance of Yao's protocol in practise.

5.2.1 Secure Computation of Satellite Collusion Probabilities

Satellite operators are very eager to protecting their satellites since they are extremely costly. One of the issues that operators are interested in is preventing collisions with other satellites. However, the operators also want to keep the trajectories of their satellites private, which makes coordination between different operators difficult. Hemenway *et al.* proposed an 2PC framework that combines GMW protocol (§2.9) and Yao's protocol for high-precision computation of satellite collusion probabilities in [11]. The framework does not target just the semi-honest model (§2.2.1) since in the case of satellite operators, it does not provide sufficient security. Instead, first, they prove the security of the

protocol in semi-honest (§2.2.1) setting. Then, they strengthen their construction by using standard arithmetic `MAC`s against malicious adversaries (§2.2.3).

For the sake of simplicity, the model of each satellite is a spherical object on a linear path in any short time window. Each satellite may deviate from its position, $\mathbf{p}$, and the distribution of these deviations are assumed to be covariance matrix[2] $\mathbf{C}$. The private input of a satellite a includes four parts: its position $\mathbf{p}_a$ in $\mathbb{R}^3$, its velocity $\mathbf{v}_a$ in $\mathbb{R}^3$, the covariance matrix $\mathbf{C}_a$ in $\mathbb{R}^{3\times3}$, and its radius R_a in $\mathbb{R}$. The algorithm which needs to be calculated securely for satellites a and b is the conjunction analysis calculation, which returns the collision probability p (see Algorithm 4).

Algorithm 4 The conjunction analysis calculation proposed in [11].

1: **Inputs:** $\{\mathbf{v}_i, \mathbf{C}_i, \mathbf{p}_i, R_i\}_{i\in a,b}$

2: $\mathbf{v}_r \leftarrow \mathbf{v}_b - \mathbf{v}_a$, $\mathbf{i} \leftarrow \dfrac{\mathbf{v}_r}{|\mathbf{v}_r|}$, $\mathbf{j} \leftarrow \dfrac{\mathbf{v}_b \times \mathbf{v}_a}{|\mathbf{v}_b \times \mathbf{v}_a|}$, $\mathbf{k} \leftarrow \mathbf{i} \times \mathbf{j}$, $\mathbf{Q} \leftarrow \begin{bmatrix}\mathbf{j} & \mathbf{k}\end{bmatrix}$, $\mathbf{C} \leftarrow \mathbf{Q}^T(\mathbf{C}_a + \mathbf{C}_b)\mathbf{Q}$

3: $(\mathbf{u}, \mathbf{v}) \leftarrow \texttt{Eigenvectors}(\mathbf{C})$, $(\sigma_x^2, \sigma_y^2) \leftarrow \texttt{Eigenvalues}(\mathbf{C})$, $\sigma_x \leftarrow \sqrt{\sigma_x^2}$, $\sigma_y \leftarrow \sqrt{\sigma_y^2}$

4: $\mathbf{u} \leftarrow \dfrac{\mathbf{u}}{|\mathbf{u}|}$, $\mathbf{v} \leftarrow \dfrac{\mathbf{v}}{|\mathbf{v}|}$, $\mathbf{U} \leftarrow \begin{bmatrix}\mathbf{u} & \mathbf{v}\end{bmatrix}$, $\begin{bmatrix}x_m \\ y_m\end{bmatrix} \leftarrow \mathbf{U}^T\mathbf{Q}^T(\mathbf{p}_b - \mathbf{p}_a)$

5: $p \leftarrow \dfrac{1}{2\pi\sigma_x\sigma_y}\displaystyle\int_{-R}^{R}\int_{-\sqrt{R^2-x^2}}^{\sqrt{R^2-x^2}} f(x,y)dydx$ where $f(x,y) = \texttt{exp}\left[\dfrac{-1}{2}\left[\left(\dfrac{x-x_m}{\sigma_x}\right)^2 + \left(\dfrac{y-y_m}{\sigma_y}\right)^2\right]\right]$

6: **return** p

Hemenway *et al.* propose GMW protocol (§2.9) for computing integer addition and multiplications [11]. To compute comparison and shift operations, they are represented as Boolean circuits and then evaluated using Yao's protocol. For compatibility with GMW protocol the garbled circuit must take secret inputs of both parties and the output of the gate must be computed as an arithmetic secret sharing (§2.6) among both sides.

- A shift operation is computed as follows: x_0 and x_1 are Alice and Bob's arithmetic shares, respectively. $(x_0 + x_1)$ needs to be shifted by an amount N which is known publicly. This can be accomplished by using Yao's protocol to compute Algorithm 5, where Alice is the garbler, and Bob is the evaluator. Bob uses OT (§2.7) to get the masking values for his inputs.

- A shift comparison is computed as follows: x_0 and x_1 are Alice and Bob's arithmetic shares, respectively. They would like to detect whether $(x_0 + x_1)$ is positive or not. This can be done by using Yao's protocol to compute Algorithm 6, where Alice is the garbler, and Bob is the evaluator. Bob uses OT (§2.7) to get the masking values for his inputs.

Algorithm 5 The shift operation computation proposed in [11].

1: **Hardwired:** $M = 2m$ and c, which are a modulus and a shift constant, respectively.
2: **Inputs:** x_0 and x_1, held by Alice and Bob, respectively. In addition a random R is provided by Alice.
3: $x \leftarrow x_0 + x_1 \ (\texttt{mod}\ M)$ using standard m-bit addition circuit.
4: $y \leftarrow x >> c$ by dropping c rightmost wires.
5: **Return:** $z_1 \leftarrow y + R \ (\texttt{mod}\ M)$ to Bob. She sets $z_0 = -R \ (\texttt{mod}\ M)$ for herself.

Algorithm 6 The comparison operation computation proposed in [11].

1: **Hardwired:** $M = 2m$, which is a modulus.
2: **Inputs:** x_0 and x_1, held by Alice and Bob, respectively. In addition a random R is provided by Alice.
3: $x \leftarrow x_0 + x_1 \ (\texttt{mod}\ M)$ using standard m-bit addition circuit.
4: $b \leftarrow sgn(x)$
5: **Return:** $z_1 \leftarrow b + R \ (\texttt{mod}\ M)$ to Bob. She sets $z_0 = -R \ (\texttt{mod}\ M)$ for herself.

Now, we return the computation of Algorithm 4. In the rest of this section, we provide the methods proposed by Hemenway *et al.* for the implementation of functions in Algorithm 4 [11].

Circuit Representation for Division: Integer division is implemented by repeated subtractions.

Circuit Representation for $\texttt{exp}(\cdot)$: The function $\texttt{exp}(\cdot)$ must be implemented by representing it as a degree-24 Taylor series. Then the Taylor coefficients can be hard-coded constants in the circuit [11].

Circuit Representation for $\sqrt{\cdot}$: Iterative Babylonian Algorithm can be used to approximate a square root. The Babylonian Algorithm computes Equation (5.5) on an input S, and an initial estimate x_0 [11].

$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{S}{x_n}\right) \tag{5.5}$$

The double integral on Line 5 of Algorithm 4 can be written as in Equation (5.6) where $g(x)$ is a sum of $\texttt{erf}$s. Simpson's Rule approximation to this integral (*i.e.*, using arcs of parabola) is suggested by Alfano in [56].

$$p = \frac{3}{\sqrt{8\pi}\sigma_x}\int_{-R}^{R} g(x)dx \tag{5.6}$$

Circuit Representation for $\texttt{erf}(\cdot)$: approximate $1 - \texttt{erf}(x)$ using the degree 96 rational function in Equation (5.7) where $a_1 = .3275911$, $a_2 = .254829592$, $a_3 = .0092705272$, $a_4 = .0001520143$, $a_5 = .0002765672$, and $a_6 = .0000430638$.

$$1 - \texttt{erf}(x) \approx \frac{1}{(1 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4 + a_5 x^5 + a_6 x^6)^{16}} \tag{5.7}$$

Hemenway *et al.* demonstrate that their framework is highly efficient. The collision probability calculation scheme proposed requires numerical estimation of a complicated integral. The work of Hemenway *et al.* proves that evaluating very complex functions is now possible by using MPC technology [11].

5.2.2 Privacy-Preserving Data Mining

Privacy-preserving data mining deals with the problem of how to run data mining algorithms on private data [10]. Mainly, privacy-preserving data mining is applied to two classic settings [10]:

1. Instead of a single party having the whole data set, two or more parties hold different parts of it. Running a data mining algorithm on the union of the parties' databases is aimed while each party's input is being kept private [10].

2. Some part of statistical data that needs to be released may be confidential. Hence, it can be first altered so that

   (a) no one's privacy is compromised by it,

   (b) Data mining algorithms can be run on the modified data set to obtain meaningful results [10].

Although both privacy problems are important, we will only deal with the first one where MPC techniques suit better. An example of the first type problem occurs in the field of medical research [10]. A group of hospitals would like to mine their patient data jointly for the medical research purposes but they also need to keep their patients' personal data private. Another example would be a cooperation scenario of intelligence agencies. These agencies cannot grant each other free access to their confidential databases because of the high security standards they must obey [10].

The relationship of privacy preserving techniques and MPC is so wide that we cannot cover it here comprehensively. Instead here we will examine and explain common notions of classification problem and ID3 algorithm and their relationships with MPC.

Classification problem. The input of a *classification problem* is a database structured such that each of its rows is a *transaction* and each of its columns is an *attribute* which may have different values (*e.g.*, each row may be a patient, and each column may be a different type of symptoms that is found in the patient) [10]. One of those attributes in the database is the main one, named as the class attribute (*e.g.*, it may represent whether the patient has lung cancer or not) [10]. We aim to use the database for prediction of the class of a new transaction by examining only its non-class attributes [10].

Another example would be credit risk analysis of a bank that wishes to identify which customers are likely to be profitable before giving them a loan [57]. Then the class attribute is defined as `Profitable-customer` (its values may be `YES` or `NO`) by the bank. The database attributes used for prediction include: `Home-Owner`, `Income`, `Years-of-Credit`, and `Other-Delinquent-Accounts`. In order to ensure proper decision making, various rules are defined by the bank. For example [57]:

If (`Other-Delinquent-Accounts` $= 0$) and (`Income` $> 30k$ or `Years-of-Credit` > 3) **then** `Profitable-customer` $=$ `YES` [`accept credit-card application`]

The collection of those rules that cover all possible transactions can be used for classification of a customer as profitable or not. The classification may include a probability of error [57].

Decision tree. Being a rooted tree, a *decision tree* has internal nodes, each corresponding to an attribute, and the edges leaving each node, corresponding to the possible values of the attribute [10]. The tree also has leaves, each containing the expected class value for a transaction that has the attribute values in the path from the root to that leaf. By using a decision tree, the class of a new transaction can be predicted by following the nodes from the root until the leaf. [10].

Figure 5.1 shows an example decision tree for identifying profitable customers as in the previous scenario. However, it reflects only a small portion of the tree. The whole tree would have many more nodes, edges, and leaves.

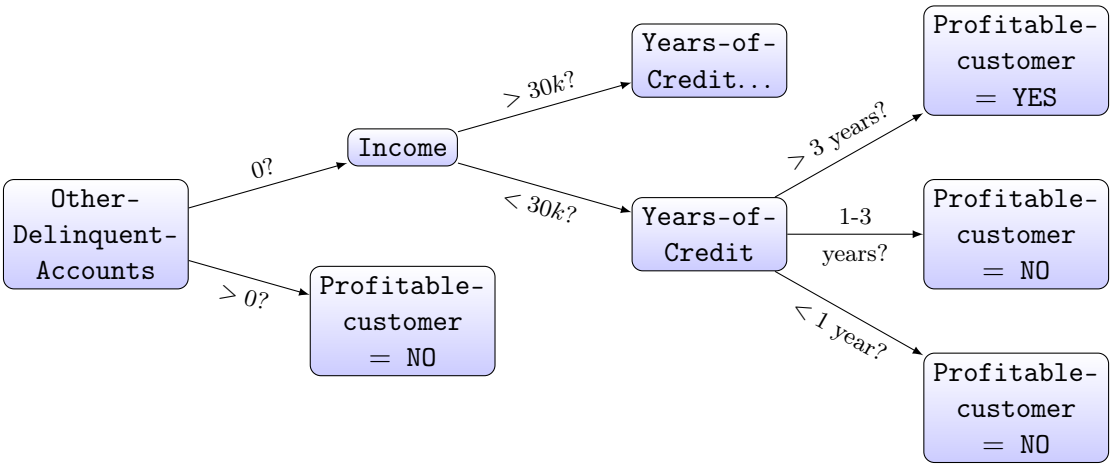

FIGURE 5.1: A decision tree for credit eligibility.

ID3 Algorithm. One of the well-known ways for designing decision trees is the use of *ID3 algorithm* [10]. The construction of the tree starts from the root node, goes top-down recursively. At each node the attribute is chosen based on its ability of classifying the transactions on its own. If an attribute is chosen for a node, the remaining transactions are partitioned by it, resulting in a smaller database which composes of the related transactions [10].

The main principle of ID3 is choosing the attribute which is best at predicting the class of the transaction. This is done by searching the attribute that decreases the information of the class to the maximum degree [10]. Namely, by choosing the attribute maximizing the *information gain*, which is the difference between the entropies of the class attribute for all transactions and for ones having the same value for a give attribute [10]. The resulting decision tree is a smaller one consistent with the database due to the greedy algorithm used in searching [10].

Privacy preserving distributed computation of ID3. We include a setting involving two parties, each having a database with different transactions to which the same set of attributes applies [10]. The parties aim at computing a decision tree of the union of their databases by using the ID3 algorithm [10]. Lindell and Pinkas describe an efficient privacy preserving protocol to solve this problem in [57].

According to Lindell and Pinkas, direct application of Yao's protocol faces some major problems, mainly the large sizes of input databases require too many OTs (§2.7), resulting in huge communication and computation costs [10]. Moreover, the boolean

circuit conversion of ID3 results in a very large circuit, because of myriad repetitions of information gain calculation which is the basic step of the algorithm [10].

Lindell and Pinkas observe that MPC of each node can be done separately [10, 57]. Starting with the root node, for each node a secure computation is invoked. Its output is revealed to both parties and the computation goes with the next node in the path. This does not compromise the protocol security since the assigned attribute to each node is also a part of the final output. Just like the non-privacy preserving implementation of ID3, both parties separately partition the rest of their transactions after an attribute is assigned to a node. This way, Lindell and Pinkas reduce the whole protocol to proper attribute assignments for node, namely the ones resulting in the highest information gains [10]. They also show how to apply Yao's protocol to proper attribute assignment [10].

Chapter 6

Private Function Evaluation

Consider the case that one invents an algorithm which can be used for efficiently diagnosing various diseases based on some information about a person's general health [4]. It is obvious that this algorithm would be precious, and healthcare institutions would volunteer to pay millions in order to use it. However, the inventor of the algorithm would prefer keeping it as a secret since he is regularly payed for it a lot of money. The problem is that medical institutions generally prefer keeping their patients' data private, preventing them from just giving it to the algorithm owner. Here the following question might be asked: How can those parties compute an algorithm which is known by only one of the parties while its input is known by only the other one? This problem is known as *private function evaluation* (PFE) [4]. The problem may also be widened to involve the case that the algorithm owner may also have his private inputs.

PFE is a special case of MPC in which n participants needs to compute a private function f using their private inputs $(x_1, \ldots, x_n))$, resulting in $f(x_1, \ldots, x_n)$. One of the parties P_1 holds a boolean circuit $\mathcal{C}_f$ of the function f, while each party P_i holds a private input x_i, and the parties aim to learn only the output of the circuit $\mathcal{C}_f(x_1, \ldots, x_n)$ while f or all other partys' inputs remain unknown to each of them except for P_1 who already knows f [1]. The difference of this scheme from the standard MPC setting is that here the function f and its boolean circuit representation $\mathcal{C}$ are not known publicly. There are many situations where such a PFE scheme would be useful, *e.g.* the ones where the function itself contains private information, or reveals security weaknesses, or the ones where service providers may prefer hiding their function or its specific implementation

as their Intellectual Property. Design of efficient special or generic PFE protocols is considered in a variety of papers in literature [1].

Most generic PFE solutions target the MPC of a *universal circuit* U_g taking the circuit $\mathcal{C}$ with a number of gates less than g and the inputs $x_1, \ldots, x_n$ of parties as input, and outputing $f(x_1, \ldots, x_n)$. The works based on this approach mainly aim to reduce the size of universal circuits, and to optimize their implementations with the help of various MPC techniques, such as Yao's protocol. However, they have a main source of inefficiency the massive sizes of known universal circuits. The complexity in their designs and implementations also increases the need for searching better alternatives.

In this section, we will explain the concepts and constructions for PFE proposed by Mohassel and Sadeghian in [1], especially for two-party case where Yao's protocol is involved.[1] The target security is in the semi-honest (§2.2.1) setting. Their work remains the most efficient PFE scheme to this date.

6.1 Mohassel and Sadeghian's Generic PFE Scheme [1]

Mohassel and Sadeghian present a generic PFE framework in [1]. In addition to the private inputs of parties which is hidden by any proper MPC scheme, hiding the topology of a boolean circuit $\mathcal{C}$ and the functionality of its gates suffices for hiding a circuit completely [1].

There are three types of information that Mohassel and Sadeghian's PFE scheme does not intend to hide about a circuit [1]:

1. The number of its inputs,
2. The number of its outputs,
3. The number of its gates.

Mohassel and Sadeghian suggest two different functionalities that make up the complete task of PFE [1]:

[1]For a clear explanation of multi-party case where GMW protocol is privately evaluated, we must refer the reader to [4].

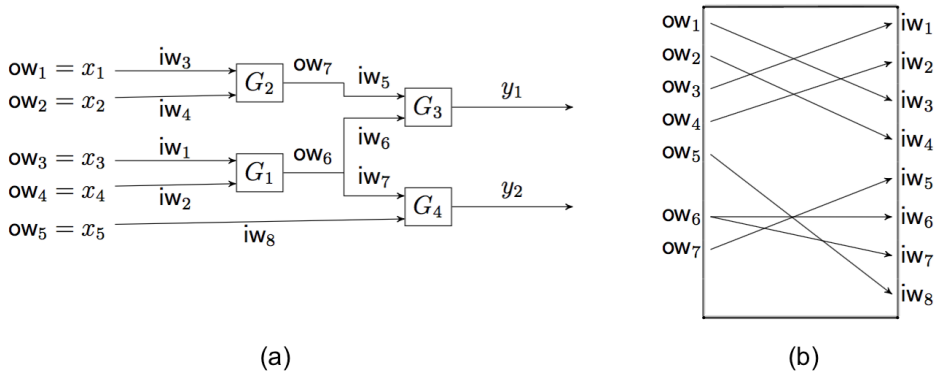

FIGURE 6.1: (a) An example circuit [4]. (b) The mapping of the circuit [4].

1. **Circuit Topology Hiding** (CTH) **Functionality.** The full description of the topology of a circuit $\mathcal{C}$ can be accomplished with the use of a mapping $\pi_{\mathcal{C}} : \mathtt{OW} \rightarrow \mathtt{IW}$. Let g, n and m denote the size, the number of inputs and the number of outputs of $\mathcal{C}$, respectively. OW (outgoing wires) is the union of the input wires of the circuit and the output wires of its non-output gates: $\{\mathtt{ow}_1 = x_1, \ldots, \mathtt{ow}_n = x_n, \mathtt{ow}_{n+1} = \mathtt{Output}(G_1), \ldots, \mathtt{ow}_{n+g-m} = \mathtt{Output}(G_{g-m})\}$. IW (incoming wires) is the set of input wires to all the gates in the circuit: $\{\mathtt{iw}_1, \ldots, \mathtt{iw}_{2g}\}$. $\pi_{\mathcal{C}}$ maps i to j (*i.e.*, $\pi_{\mathcal{C}}(i) \rightarrow j$), if and only if $\mathtt{ow}_i \in \mathtt{OW}$ and $\mathtt{iw}_j \in \mathtt{IW}$ correspond to the same wire in the circuit $\mathcal{C}$. Because an outgoing wire can correspond to more than one incoming wire, $\pi_{\mathcal{C}}$ is rarely a function. However, its inverse $\pi_{\mathcal{C}}^{-1}$ is a function since a wire can be either an output of only one gate or an input. Figure 6.1 shows an example circuit (a) and its mapping $\pi_{\mathcal{C}}$ (b). The main target of the CTH *functionality* is the oblivious application of this mapping $\pi_{\mathcal{C}} : \mathtt{OW} \rightarrow \mathtt{IW}$.

It is useful to include a computation of the number of possible mapping since it is directly related to the security of the PFE scheme. Although one may expect the number of possible mappings to be M^N due to the ability of any $\mathtt{ow}_i$ to go to any $\mathtt{iw}_j$, the exact value is smaller since an $\mathtt{ow}_i$ must go at least one $\mathtt{iw}_j$. Since π^{-1} is an onto function, computing the number of possible onto functions suffices. Applying the inclusion-exclusion principle, we get Equation (6.1) which shows the number (ρ) of possible mappings for a circuit where OW has M elements and IW has N elements [58].

$$\rho = \sum_{i=0}^{M} (-1)^i \binom{M}{i} (M-i)^N \tag{6.1}$$

2. **Private Gate Evaluation** (PGE) **Functionality.** The PGE *functionality* deals with hiding the functionality of each gate in a circuit. It can be seen as a black-box gate mechanism where only one of the parties (P_1) knows its functionality. The input of the mechanism is the shares of all parties for both inputs of the hidden gate, and it returns to the parties their shares for the output of the gate.

6.2 CTH Functionality Realization

Before describing Mohassel and Sadeghian's construction in more detail, the concept of an *extended permutation* needs to be explained [1]. A mapping $\pi : \{1 \dots M\} \to \{1 \dots N\}$ can be regarded as a permutation if it is one-to-one and onto (*i.e.* a bijection). This notion can be generalized to an extended permutation as follows: Given the positive integers M and N, a mapping $\pi : \{1 \dots M\} \to \{1 \dots N\}$ is called as an extended permutation (EP) if and only if there exists exactly one $x \in \{1 \dots M\}$ for every $y \in \{1 \dots N\}$ such that $\pi(x) = y$. x is often denoted by $\pi^{-1}(y)$. Unlike the mapping of a standard permutation, the mapping of an EP may also replicate or discard elements in the domain, allowing the domain to be larger or smaller than the range.

$n + q - m$ *oblivious mapping* (OMAP) queries and $2q$ Reveal queries are needed to be implemented in order to realize the CTH functionality (an OMAP query for each $\mathtt{ow}_i$, and a Reveal query for each $\mathtt{iw}_i$). These OMAP/Reveal queries can be combined to construct a problem known as *oblivious evaluation of the extended permutation* (OEP) to which Mohassel and Sadeghian's CTH scheme mainly address.

OEP Definition. Two-party OEP Problem 2-OEP$(\vec{\pi}, \vec{x}, \vec{t})$ is defined as follows: The first party P_1 holds an EP $\pi : \{1 \dots M\} \to \{1 \dots N\}$, and a blinding vector for outputs $\vec{t} = (t_1, \dots, t_N)$; whereas the other party P_2 holds a vector of inputs $\vec{x} = (x_1, \dots, x_M)$. Both the x_is and t_is are ℓ-bit strings. The protocol ends in P_2 learning $(x_{\pi^{-1}(1)} \oplus t_1, \dots, x_{\pi^{-1}(N)} \oplus t_N)$, while P_1 learning nothing.

Mohassel and Sadeghian construct a solution for OEP from switching networks which they observe as more efficient than the previous constructions.

Switching Networks. A *switching network* SN composes of *2-switches* which are interconnected. Its inputs are N ℓ-bit strings and a set of selection bits of each switches, while

its outputs are N ℓ-bit strings. Each switch takes two ℓ-bit strings and two selection bits as input, outputting two ℓ-bit strings. Each of the outputs may get the value of any of the input strings depending on the selection bits. This means for input values (x_0, x_1) and output values (y_0, y_1), there are four different switch output possibilities. The two selection bits s_0 and s_1 are used for determining the switch output. In particular, the switch will output $y_0 = x_{s_0}$, and $y_1 = x_{s_1}$.

The mapping $\pi : \{1 \dots N\} \to \{1 \dots N\}$ of an SN is defined as $\pi(i) = j$ if and only if after the SN is evaluated, the string on the output wire j becomes that on the input wire i. There is no need for the mapping π to be a function because the value of any input wire can be mapped to any number of output wires. However, its inverse π^{-1} must always be a function.

A *permutation network* PN is a switching network whose mapping is a permutation of its inputs. In contrast to switching networks, permutation networks compose of *1-switches*. Unlike 2-switches, they have only one selection bit s. For an input (x_0, x_1), a 1-switch outputs one of the two possible outputs: (x_0, x_1) if $s = 0$, and (x_1, x_0) if $s = 1$. 1-switch may also be called a *permutation cell*.

Waksman proposed an efficient construction for a permutation network in [59]. Mainly, his work suggests that a permutation network with $N = 2^k$ can be constructed with $N\log_2 N - N + 1$ switches, that the switch depth of the constructed PN will be $2\log_2 N - 1$, and that its computational complexity will be $O(N\log_2 N)$.

Extended Permutation from Switching Networks. Mohassel and Sadeghian propose the general method for construction of an extended permutation from switching and permutation networks [1]. However, extended permutations differ from switching networks in that the number of their inputs M and that of their outputs N need not be equal ($M \leq N$) [1]. $N - M$ additional dummy inputs are added to the real inputs of an EP $\pi : \{1 \dots M\} \to \{1 \dots N\}$ in order to simulate it as an SN.

Mohassel and Sadeghian divide a switching network into three components [1]:

1. **Dummy-value placement component.** This component takes N input strings composing of real and dummy ones. For each real input that π maps to k different outputs, the dummy-value placement component's output is the real string followed

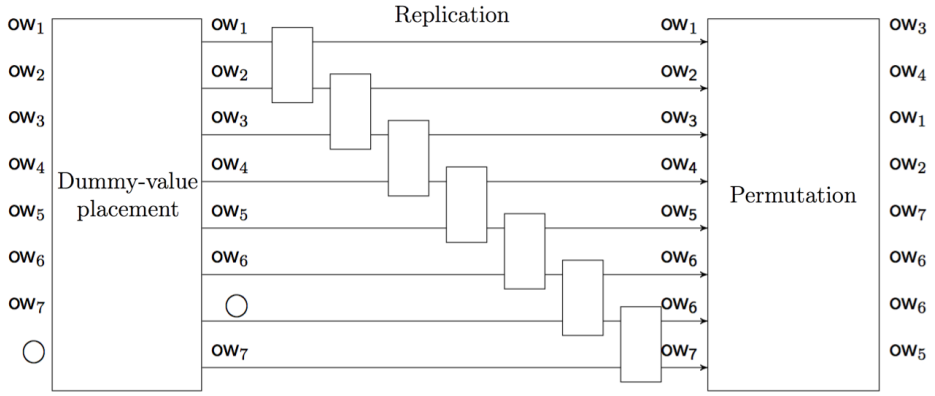

FIGURE 6.2: The switching network for EP of the circuit in Figure 6.1 [4].

by $k-1$ dummy strings. An efficient implementation of this process can be via a Waksman permutation network [59].

2. **Replication component.** This component takes the output of the dummy-value placement component as input. If a value is real, it goes unchanged. If it is a dummy value, it is replaced by the real value which precedes it. This can be computed by a series of $N-1$ 2-switches whose selection bits (s_0, s_1) are either (0,0) or (0,1). If the selection bits are (0,0), that means x_1 is dummy, and x_0 goes both of the outputs. If they are (0,1), that means both inputs are real, and both are kept on the outputs in the same order. At the end of this step, all the dummy inputs are replaced by the necessary copies of the real inputs.

3. **Permutation component.** This component takes the output wires of the replication component as input and outputs a permutation of them so that each string is placed on its final location according to the prescription of mapping π. An efficient implementation of this process can also be via a Waksman permutation network [59].

Adding up the three components, the number of switches needed for implementation of EP is $2(N\texttt{log}_2 N - N + 1) + N - 1 = 2N\texttt{log}_2 N - N + 1$. The topology of the whole switching network is the same for all N input EPs and the output depends on the selection bits.

TABLE 6.1: P_1 must learn one of these (y_0,y_1) according to his selection bits.

(s_{0u},s_{1u})	y_0	y_1
(0,0)	$x_i \oplus r_k$	$x_i \oplus r_l$
(0,1)	$x_i \oplus r_k$	$x_j \oplus r_l$
(1,0)	$x_j \oplus r_k$	$x_i \oplus r_l$
(1,1)	$x_j \oplus r_k$	$x_j \oplus r_l$

Oblivious Evaluation of Switching Networks (OSN). Now, we can return to our `OEP` problem. If the `EP` construct from switched and permutation networks can be evaluated oblivously, we have a solution. Mohassel and Sadeghian propose a method for oblivious evaluation of their building blocks, *i.e.*, 1-switches and 2-switches [1].

Recall that P_1 holds the selection bits of the switching network, and an output blinding vector $\vec{t}$ while P_2 holds the input vector $\vec{x}$. P_2 must learn the switching network's blinded output which is the `EP` of her input vector blinded with the vector $\vec{t}$; while P_1 learns $\perp$.

Secure evaluation of a single 2-switch. The express the general idea of the secure computation of whole network, Mohassel and Sadeghian describe the secure evaluation of its building block, a single 2-switch u [1]. Let the input wires of the 2-switch be w_i and w_j, and its output wires be w_k and w_l. P_2 assigns four uniformly random values r_i, r_j, r_k, r_l to the four wires of the switch. P_1 has the blinded values $x_i \oplus r_i$ and $x_j \oplus r_j$ as his shares for the two input wires. The aim is letting P_1 obtain his output shares which is the blinded values on the output wires (see Table 6.1). In fact, there are four possible output pairs ($x_i \oplus r_k$, $x_i \oplus r_l$), ($x_i \oplus r_k$, $x_j \oplus r_l$), ($x_j \oplus r_k$, $x_i \oplus r_l$), or ($x_j \oplus r_k$, $x_j \oplus r_l$) which P_1 may obtain based on the values of his selection bits s_{0u} and s_{1u}.

P_2 prepares a table with four rows: ($r_i \oplus r_k$, $r_j \oplus r_l$), ($r_i \oplus r_k$, $r_i \oplus r_l$), ($r_j \oplus r_k$, $r_i \oplus r_l$), and ($r_j \oplus r_k$, $r_j \oplus r_l$) as shown in Table 6.2. Then, P_1 and P_2 engage in a 1-out-of-4 OT (§2.7) in which P_2 inputs the four rows that he just prepared, and P_1 inputs his selection bits for the switch u. Suppose that P_1's selection bits are (0,0). This means P_1 retrieves

TABLE 6.2: P_1 gets one of these (T_0,T_1) by engaging in 1-out-of-4 OT (§2.7) with P_2.

(s_{0u},s_{1u})	T_0	T_1
(0,0)	$r_i \oplus r_k$	$r_i \oplus r_l$
(0,1)	$r_i \oplus r_k$	$r_j \oplus r_l$
(1,0)	$r_j \oplus r_k$	$r_i \oplus r_l$
(1,1)	$r_j \oplus r_k$	$r_j \oplus r_l$

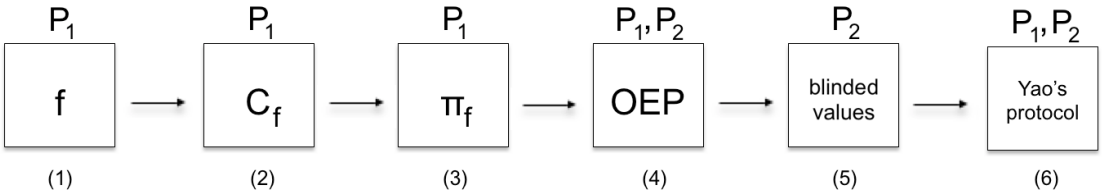

FIGURE 6.3: Basic procedures of topology hiding: (1)The function f known by P_1. (2) Circuit representation of f. (3) Circuit mapping of f. (4) `OEP` for P_2 learning blinded values. (5) The blinded values learnt by P_2. (6) Yao's protocol with the blinded values.

the first row, *i.e.*, $(r_i \oplus r_k,\ r_j \oplus r_l)$. He then `XOR`s $x_i \oplus r_i$ and $r_i \oplus r_k$, as well as $x_j \oplus r_j$ and $r_i \oplus r_l$, reaching his output shares $x_i \oplus r_k$ and $x_i \oplus r_l$.

Constant round protocol. Using the OT-based protocol proposed for 2-switches, the entire switching network can be securely computed in constant round since the protocol permits parallel OT (§2.7) runs [1]. In an *offline* stage, a set of random strings for each wire is generated, and a table for each switch is prepared by P_2. Then P_1 and P_2 run the parallel OTs (§2.7) as described above, leading to that a single row of each table is learned by P_1 according to his selection bits.

In the *online* stage, P_2 blinds his input vector with the blinding strings on the inputs of the input switches before sending them to P_1. P_1 is now able to compute the entire switching network. He just need to perform sequential `XOR`s (in topological order) to reach the blinded values on the output wires. He then applies his own blinding vector $\vec{t}$ and sends the result to P_2. P_2 removes her blinding, and obtains the output of the `OEP` [1].

Efficiency of the Mohassel and Sadeghian's `OEP`. As we mentioned before, to implement an extended permutation $\pi : 1 \ldots M \rightarrow 1 \ldots N$, $2N\texttt{log}N - N + 1$ switches are needed. In fact, 1-out-of-2 OTs (§2.7) are enough to implement `PN`s which consist of 1-switches. Moreover, 2-switches in replication component can also be implemented with 1-out-of-2 OTs (§2.7) since their outputs have 2 possibilities unlike the generic 2-switches [1]. To sum up, this protocol costs $2N\texttt{log}N - N + 1$ 1-out-of-2 OTs (§2.7). Mohassel and Sadeghian suggests the use of OT extension [60], which reduce total number of public key operations for their `OEP` to a constant value depending on the security parameter of protocol, *i.e.* $O(k)$ [1]. In this case, the number of symmetric key operations will be twice the number of OTs, which is $4N\texttt{log}N - 2N + 2$ [1].

Figure 6.3 summarizes the basic procedures of topology hiding via `OEP`. P_1 owns a function f (1). He converts f to a circuit representation C_f (2). Then he extracts the circuit mapping π_f (3). P_1 and P_2 engage in an `OEP` (4) of π_f where P_2 learns the blinded values of her input masking values (5) which she will later use in Yao's protocol (6).

Mohassel and Sadeghian show applications of their framework to arithmetic circuits, GMW protocol (§2.9), and Yao's protocol. Since the main topic of this thesis Yao's protocol and two-party cases, we will continue with its application to Yao's protocol [1].

6.3 Two-Party PFE of Yao's Protocol

Alice and Bob would like to compute a function $f(x_0, x_1)$, where x_0 is Alice's input, x_1 is Bob's input. Bob acts like P_1, and Alice acts like P_2 in Mohassel and Sadeghian's generic PFE scenario [1]. So, only Bob knows f, and the topology of the circuit $\mathcal{C}$ of f. Since (`NAND`) is Turing complete, all gates in the circuit are let to be a `NAND` gate, so that the need for `PGE` functionality can be eliminated. Alice may learn the number of gates, but she should know the circuit topology. Now, one may ask the following question: How can someone garble a circuit which she does not know? Well, cryptography can achieve many incredible things.

The protocol goes as follows [1]:

`Offline Preparation:`

1. Bob sorts the gates topologically and computes the extended permutation $\pi_{\mathcal{C}}$ corresponding to circuit $\mathcal{C}$.

2. Alice randomly generates a masking value pair (W_i^0, W_i^1) for each $\mathtt{ow}_i \in \mathtt{OW}$. This yields a total of $M = n + g - o$ pairs. Each masking value is k bits long, where k is the security parameter. The `lsb` of 2 masking values belonging to the same pair must be different so that they have different labels.

3. Alice generates a bit vector $\vec{v} = (v_1, \ldots, v_M)$ where $v_i = \mathtt{lsb}W_i^0$. She arranges each masking value pair with respect to their labels. So, they become $(W_i^{v_i}, W_i^{\bar{v_i}})$. This arrangement will be important during the garbled circuit evaluation. Moreover, she assigns those pairs to 2 vectors $\vec{p} = (p_1, \ldots, p_M)$ and $\vec{q} = (q_1, \ldots, q_M)$ where $p_i = W_i^{v_i}$ and $q_i = W_i^{\bar{v_i}}$.

4. Bob generates a random bit vector $\vec{v'} = (v'_1, \ldots, v'_N)$ where v'_j is a random bit. This yields a total of $N = 2g$ bits. He also generates random blinding pairs (t^0_j, t^1_j) for each $\texttt{iw}_j \in \texttt{IW}$ such that $\texttt{lsb}(t^0_j) = \texttt{lsb}(t^1_j)$. He assigns those pairs to 2 blinding vectors $\vec{t^0} = (t^0_1, \ldots, t^0_N)$ and $\vec{t^1} = (t^1_1, \ldots, t^1_N)$.

`Oblivious Evaluation of Switching Networks:`

5. Alice and Bob engage in an `OEP` protocol where his input is the extended permutation $\pi_{\mathcal{C}}$ and $\vec{v'}$, while her input is $\vec{v}$. As a result, Alice learns $v'' = (v''_1, \ldots, v''_N)$ where $v''_j = v_{\pi^{-1}(j)} \oplus v'_j$.

6. Alice and Bob engage in a slightly modified `OEP` protocol where his input is the extended permutation $\pi_{\mathcal{C}}$ and $\vec{t^0}$, while her input is $\vec{p}$. The output is a vector $p' = (p'_1, \ldots, p'_N)$ where $p'_j = p_{\pi^{-1}(j)} \oplus t^0_j$. The modification is that the output is not learned by Alice but fed to a new permutation network $\hat{\texttt{SN}}$.

7. Alice and Bob engage in a slightly modified `OEP` protocol where his input is the extended permutation $\pi_{\mathcal{C}}$ and $\vec{t^1}$, while her input is $\vec{q}$. The output is a vector $q' = (q'_1, \ldots, q'_N)$ where $q'_j = q_{\pi^{-1}(j)} \oplus t^1_j$. The modification is that the output is not learned by Alice but fed to $\hat{\texttt{SN}}$ as well.

8. $\hat{\texttt{SN}}$ is a switching network including N 1-swiches whose switch depth is 1. Each 1-switch u_j takes (p'_j, q'_j) as input, v'_j as the selection bit and outputs either (p'_j, q'_j) if $v'_j = 0$ or (q'_j, p'_j) if $v'_j = 1$.

9. After oblivious evaluation of $\hat{\texttt{SN}}$, Alice learns the output which is a set of N pairs whose j^{th} element is either (p'_j, q'_j) if $v'_j = 0$ or (q'_j, p'_j) if $v'_j = 1$.

`Garbling:`

10. Alice needs to arrange the blinded pairs into their original position since the truth values must be known for garbling. This can be done by using v''. If $v''_j = 0$, the pair remains unchanged, otherwise it is swapped. j^{th} element of the output will be $(\hat{W}^0_{\pi^{-1}(j)}, \hat{W}^1_{\pi^{-1}(j)})$ where $\hat{W}^b_i$ means a blinded value for W^b_i.

11. For all gates, Bob tells Alice which two of the incoming wires and which one of the outgoing wires belong to the same gate. He also tell her the outgoing wires corresponding to his and her input bits.

12. Alice garbles each gate by encrypting the masking values on the outgoing wires using the blinded values on the incoming wires as the keys. She sends Bob the garbled gates and the masking values for her inputs in `OW`. Bob gets his input masking values from her using 1-out-of-2 OT (§2.7).

`Evaluating:`

13. Using the circuit mapping $\pi_{\mathcal{C}}$, his blinding vectors $\vec{t^0} = (t_1^0, \ldots, t_N^0)$ and $\vec{t^1} = (t_1^1, \ldots, t_N^1)$, and the garbled gates told by Alice, Bob evaluates the whole garbled circuit in topological order. When an outgoing wire i is mapped to an incoming wire j, the masking value W_i on that outgoing wire is `XOR`ed with $t_j^{\mathtt{lsb}W_i}$ on the incoming wire j. These `XOR`ed (blinded) values are used as the decryption keys in the corresponding garbled gates to reveal the next masking value on the outgoing wire of the gate.

14. In the end, Bob reaches the output masking values. He tells Alice those output masking values. She decodes them and reaches $f(x_0, x_1)$. Alice tells Bob the output.

Complexity. The steps 5, 6 and 7 can be combined for only one `OEP`. Hence, this protocol requires $2N\mathtt{log}_2N - N + 1$ OTs for `OEP` and N OTs for $\hat{\mathtt{SN}}$, *i.e.* $2N\mathtt{log}_2N + 1$ OTs in total. OTs for Bob's input masking values increases the total OT requirement of complete two-party PFE protocol but they do not change its round complexity since they can be implemented in parallel with the OTs for `OEP`.

Chapter 7

Conclusion and Discussions

In this thesis, we were interested in surveying all known Yao's protocol optimizations and showing practical applications of Yao's protocol.

We have presented P&P (§4.2), GRR3 (§4.3), free `XOR` (§4.4), GRR2 (§4.5), fle`XOR` (§4.6), and half gates (§4.7) techniques in the descending order for size of garbled gates. We have compared those optimizations in terms of communication and computation complexities, and showed their compatibilities with each other.

What else can be done for optimization? Well, in science, especially in cryptography there is no end. Although Zahur *et al.* have proved that the half gates method gives the most size-optimum technique for an odd gate and yet compatible with free `XOR` [5], there are still two more optimization parameters that can be improved. There may be faster and/or more secure garbling techniques in the future. To improve on the size parameter there is a need for a revolutionary change in the traditional approach. This improvement may be a method which garbles a group of gates together instead of garbling each gate separately, resulting in a lower size.

We have also presented some generic applications as well as some real-world application examples. The generic applications include pipelining method (§5.1.1) which is useful for reducing total protocol time when the both parties of a garbling scheme is online at the same time. We also included garbling RAM (§5.1.2) which is a quite useful technique especially for applications within the realm of big data. Some generic MPC tools `JustGarble` (§5.1.3), `ABY` (§5.1.4), `Obliv-C`, `ObliVM`, and `Frigate` (§5.1.7) are also introduced briefly. We compared them in terms of the use of garbling optimization

techniques. At the end of the chapter, we have given some real-world applications, including MPC of satellite collusion probabilities (§5.2.1) and privacy preserving data mining.

We have explained private function evaluation, and Mohassel *et al.*'s PFE scheme. It is the most efficient PFE scheme known. Although their PFE scheme is limited for use right now, we know that cryptography is one of the fastest fields in computing science. It is hard to say whether it will be in use soon but someday generic PFE schemes will be in every day use for many applications, where one of the parties is also willing to hide her function since the path to developing such a technique is already open.

Bibliography

[1] P. Mohassel and S. Sadeghian. How to hide circuits in mpc an efficient framework for private function evaluation. In *Advances in Cryptology – EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 557–574, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38348-9. doi: 10.1007/978-3-642-38348-9_33. Available at `http://dx.doi.org/10.1007/978-3-642-38348-9_33`.

[2] T. Schneider. *Engineering Secure Two-Party Computation Protocols – Advances in Design, Optimization, and Applications of Efficient Secure Function Evaluation.* PhD thesis, Ruhr-University Bochum, Germany, Information Sciences, 2011. Available at `http://thomaschneider.de/papers/S11Thesis.pdf`.

[3] M. Bellare, V. Hoang, and P. Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 784–796, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1651-4. doi: 10.1145/2382196.2382279. Available at `http://doi.acm.org/10.1145/2382196.2382279`.

[4] P. Pullonen. Private function evaluation for mpc. 2015. Available at `https://courses.cs.ut.ee/MTAT.07.022/2015_spring/uploads/Main/pille-report-s15.pdf`.

[5] S. Zahur, M. Rosulek, and D. Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings,*

Part II, pages 220–250, 2015. doi: 10.1007/978-3-662-46803-6_8. Available at http://dx.doi.org/10.1007/978-3-662-46803-6_8.

[6] A. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society. doi: 10.1109/SFCS.1982.88. Available at http://dx.doi.org/10.1109/SFCS.1982.88.

[7] A. Yao. How to generate and exchange secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science*, SFCS '86, pages 162–167, Washington, DC, USA, 1986. IEEE Computer Society. ISBN 0-8186-0740-8. doi: 10.1109/SFCS.1986.25. Available at http://dx.doi.org/10.1109/SFCS.1986.25.

[8] 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*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM. ISBN 0-89791-221-7. doi: 10.1145/28395.28420. Available at http://doi.acm.org/10.1145/28395.28420.

[9] D. Bogdanov, R. Talviste, and J. Willemson. Deploying secure multi-party computation for financial data analysis. In *Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, pages 57–64, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-32946-3. doi: 10.1007/978-3-642-32946-3_5. Available at http://dx.doi.org/10.1007/978-3-642-32946-3_5.

[10] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *The Journal of Privacy and Confidentiality*, 1(1):59–98, 2009. Available at http://repository.cmu.edu/jpc/vol1/iss1/5.

[11] B. Hemenway, S. Lu, R. Ostrovsky, and W. Welser IV. High-precision secure computation of satellite collision probabilities. In *Security and Cryptography for Networks: 10th International Conference, SCN 2016, Amalfi, Italy, August 31 – September 2, 2016, Proceedings*, pages 169–187, Cham, 2016. Springer International Publishing. ISBN 978-3-319-44618-9. doi: 10.1007/978-3-319-44618-9_9. Available at http://dx.doi.org/10.1007/978-3-319-44618-9_9.

[12] R. Cramer, R. Gennaro, and B. Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Advances in Cryptology — EUROCRYPT '97: International Conference on the Theory and Application of Cryptographic Techniques Konstanz, Germany, May 11–15, 1997 Proceedings*, pages 103–118, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg. ISBN 978-3-540-69053-5. doi: 10.1007/3-540-69053-0_9. Available at `http://dx.doi.org/10.1007/3-540-69053-0_9`.

[13] P. Bogetoft, D. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. Nielsen, J. Nielsen, K. Nielsen, J. Pagter, M. Schwartzbach, and T. Toft. Financial cryptography and data security. chapter Secure Multiparty Computation Goes Live, pages 325–343. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-03548-7. doi: 10.1007/978-3-642-03549-4_20. Available at `http://dx.doi.org/10.1007/978-3-642-03549-4_20`.

[14] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM Conference on Electronic Commerce*, EC '99, pages 129–139, New York, NY, USA, 1999. ACM. ISBN 1-58113-176-3. doi: 10.1145/336992.337028. Available at `http://doi.acm.org/10.1145/336992.337028`.

[15] R. Kulkarni and A. Namboodiri. Secure hamming distance based biometric authentication. In *2013 International Conference on Biometrics (ICB)*, pages 1–6, June 2013. doi: 10.1109/ICB.2013.6613008. Available at `http://dx.doi.org/10.1109/ICB.2013.6613008`.

[16] J. Bringer, H. Chabanne, and A. Patey. Shade: Secure hamming distance computation from oblivious transfer. In *Financial Cryptography and Data Security: FC 2013 Workshops, USEC and WAHC 2013, Okinawa, Japan, April 1, 2013, Revised Selected Papers*, pages 164–176, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-41320-9. doi: 10.1007/978-3-642-41320-9_11. Available at `http://dx.doi.org/10.1007/978-3-642-41320-9_11`.

[17] M. Kiraz, Z. Genç, and S. Kardaş. Security and efficiency analysis of the hamming distance computation protocol based on oblivious transfer. *Security and Communication Networks*, 8(18):4123–4135, 2015. doi: 10.1002/sec.1329. Available at `http://dx.doi.org/10.1002/sec.1329`.

[18] J. Launchbury, D. Archer, T. DuBuisson, and E. Mertens. Application-scale secure multiparty computation. In *Programming Languages and Systems: 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 8–26, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-642-54833-8. doi: 10.1007/978-3-642-54833-8_2. Available at `http://dx.doi.org/10.1007/978-3-642-54833-8_2`.

[19] M. Kiraz and B. Schoenmakers. An efficient protocol for fair secure two-party computation. In *Topics in Cryptology – CT-RSA 2008: The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, pages 88–105, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-79263-5. doi: 10.1007/978-3-540-79263-5_6. Available at `http://dx.doi.org/10.1007/978-3-540-79263-5_6`.

[20] M. Kiraz. *Secure and Fair Two-Party Computation*. PhD thesis, Technische Universiteit Eindhoven, 2008. Available at `http://alexandria.tue.nl/extra2/200811317.pdf`.

[21] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *Proceedings of the 20th USENIX Conference on Security*, SEC'11, pages 35–35, Berkeley, CA, USA, 2011. USENIX Association. Available at `http://dl.acm.org/citation.cfm?id=2028067.2028102`.

[22] M. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of yao's garbled circuit construction. In *In Proceedings of 27th Symposium on Information Theory in the Benelux*, 2006. Available at `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.140.2627&rep=rep1&type=pdf`.

[23] A. Kumar. *Fundamentals of Digital Circuits*. Prentice-Hall Of India Pvt. Limited, 2003. ISBN 9788120317451. Available at `https://books.google.com.tr/books?id=BOVkrtiLUcEC`.

[24] B. Pinkas, T. Schneider, N. Smart, and S. Williams. Secure two-party computation is practical. In *Advances in Cryptology – ASIACRYPT 2009: 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 250–267,

Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-10366-7. doi: 10.1007/978-3-642-10366-7_15. Available at http://dx.doi.org/10.1007/978-3-642-10366-7_15.

[25] H. Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999. ISBN 3540643109.

[26] M. Sipser. *Introduction to the Theory of Computation*. International Thomson Publishing, 1st edition, 1996. ISBN 053494728X.

[27] A. Kojevnikov and A. Kulikov. Circuit complexity and multiplicative complexity of boolean functions. In *Programs, Proofs, Processes: 6th Conference on Computability in Europe, CiE 2010, Ponta Delgada, Azores, Portugal, June 30 – July 4, 2010. Proceedings*, pages 239–245, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-13962-8. doi: 10.1007/978-3-642-13962-8_27. Available at http://dx.doi.org/10.1007/978-3-642-13962-8_27.

[28] J. Boyar, R. Peralta, and D. Pochuev. On the multiplicative complexity of boolean functions over the basis $(\wedge, \oplus, 1)$. *Theoretical Computer Science*, 235(1): 43 – 57, 2000. ISSN 0304-3975. doi: http://dx.doi.org/10.1016/S0304-3975(99)00182-6. Available at http://www.sciencedirect.com/science/article/pii/S0304397599001826.

[29] J. Daemen and V. Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002. ISBN 3-540-42580-2.

[30] R. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. Available at http://doi.acm.org/10.1145/359340.359342.

[31] T. El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc. ISBN 0-387-15658-5. Available at http://dl.acm.org/citation.cfm?id=19478.19480.

[32] P. Rogaway and T. Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance,

and collision resistance. In *Fast Software Encryption: 11th International Workshop, FSE 2004, Delhi, India, February 5-7, 2004. Revised Papers*, pages 371–388, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-25937-4. doi: 10.1007/978-3-540-25937-4_24. Available at `http://dx.doi.org/10.1007/978-3-540-25937-4_24`.

[33] N. Koblitz and A. Menezes. The random oracle model: a twenty-year retrospective. *Designs, Codes and Cryptography*, 77(2):587–610, 2015. ISSN 1573-7586. doi: 10.1007/s10623-015-0094-2. Available at `http://dx.doi.org/10.1007/s10623-015-0094-2`.

[34] H. Handschuh. Sha-0, sha-1, sha-2 (secure hash algorithm). In *Encyclopedia of Cryptography and Security*, pages 1190–1193, Boston, MA, 2011. Springer US. ISBN 978-1-4419-5906-5. doi: 10.1007/978-1-4419-5906-5_615. Available at `http://dx.doi.org/10.1007/978-1-4419-5906-5_615`.

[35] Y. Lindell, B. Pinkas, and N. Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *Security and Cryptography for Networks: 6th International Conference, SCN 2008, Amalfi, Italy, September 10-12, 2008. Proceedings*, pages 2–20, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-85855-3. doi: 10.1007/978-3-540-85855-3_2. Available at `http://dx.doi.org/10.1007/978-3-540-85855-3_2`.

[36] B. Kreuter, a. shelat, and C. Shen. Billion-gate secure computation with malicious adversaries. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 285–300, Bellevue, WA, 2012. USENIX. ISBN 978-931971-95-9. Available at `http://eprint.iacr.org/2012/179`.

[37] M. Bellare, V. Hoang, S. Keelveedhi, and P. Rogaway. Efficient garbling from a fixed-key blockcipher. In *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, pages 478–492, Washington, DC, USA, 2013. IEEE Computer Society. ISBN 978-0-7695-4977-4. doi: 10.1109/SP.2013.39. Available at `http://dx.doi.org/10.1109/SP.2013.39`.

[38] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979. ISSN 0001-0782. doi: 10.1145/359168.359176. Available at `http://doi.acm.org/10.1145/359168.359176`.

[39] D Demmler, T Schneider, and M Zohner. ABY – a framework for efficient mixed-protocol secure two-party computation. In *22. Annual Network and Distributed System Security Symposium (NDSS'15)*. The Internet Society, February 8-11, 2015. doi: 10.14722/ndss.2015.23113. Available at `http://encrypto.de/code/ABY`.

[40] R. Naskar and I. Sengupta. Secret sharing and proactive renewal of shares in hierarchical groups. *CoRR*, abs/1006.1192, 2010. Available at `http://dblp.uni-trier.de/db/journals/corr/corr1006.html#abs-1006-1192`.

[41] T. Chou and C. Orlandi. The simplest protocol for oblivious transfer. In *Progress in Cryptology – LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 40–58, Cham, 2015. Springer International Publishing. ISBN 978-3-319-22174-8. doi: 10.1007/978-3-319-22174-8_3. Available at `http://dx.doi.org/10.1007/978-3-319-22174-8_3`.

[42] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*, pages 145–161, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45146-4. doi: 10.1007/978-3-540-45146-4_9. Available at `http://dx.doi.org/10.1007/978-3-540-45146-4_9`.

[43] C. Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford University, Stanford, CA, USA, 2009. AAI3382729, Available at `https://crypto.stanford.edu/craig/craig-thesis.pdf`.

[44] V. Kolesnikov, P. Mohassel, and M. Rosulek. Flexor: Flexible garbling for xor gates that beats free-xor. In *Advances in Cryptology – CRYPTO 2014: 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II*, pages 440–457, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44381-1. doi: 10.1007/978-3-662-44381-1_25. Available at `http://dx.doi.org/10.1007/978-3-662-44381-1_25`.

[45] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *Proceedings of the Twenty-second Annual ACM Symposium on Theory of Computing*, STOC '90, pages 503–513, New York, NY, USA, 1990. ACM. ISBN

0-89791-361-2. doi: 10.1145/100216.100287. Available at http://doi.acm.org/10.1145/100216.100287.

[46] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 486–498, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-70582-6. doi: 10.1007/978-3-540-70583-3_40. Available at http://dx.doi.org/10.1007/978-3-540-70583-3_40.

[47] S. Choi, J. Katz, R. Kumaresan, and H. Zhou. On the security of the "free-xor" technique. In *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 39–53. Springer, 2012. doi: 10.1007/978-3-642-28914-9_3. Available at http://dx.doi.org/10.1007/978-3-642-28914-9_3.

[48] S. Lu and R. Ostrovsky. How to garble ram programs? In *Advances in Cryptology – EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 719–734, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-38348-9. doi: 10.1007/978-3-642-38348-9_42. Available at http://dx.doi.org/10.1007/978-3-642-38348-9_42.

[49] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag. ISBN 3-540-42456-3. Available at http://dl.acm.org/citation.cfm?id=646766.704155.

[50] C. Gentry, S. Halevi, S. Lu, R. Ostrovsky, M. Raykova, and D. Wichs. Garbled ram revisited. In *Advances in Cryptology – EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 405–422, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-642-55220-5. doi: 10.1007/978-3-642-55220-5_23. Available at http://dx.doi.org/10.1007/978-3-642-55220-5_23.

[51] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996. ISSN 0004-5411. doi: 10.1145/233551.233553. Available at http://doi.acm.org/10.1145/233551.233553.

[52] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2477-9. doi: 10.1145/2508859.2516660. Available at http://doi.acm.org/10.1145/2508859.2516660.

[53] S. Zahur and D. Evans. Obliv-c: A language for extensible data-oblivious computation. *IACR Cryptology ePrint Archive*, 2015:1153, 2015. Available at http://eprint.iacr.org/2015/1153.

[54] C. Liu, X. Wang, K. Nayak, Y. Huang, and E. Shi. Oblivm: A programming framework for secure computation. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 359–376, 2015. doi: 10.1109/SP.2015.29. Available at http://dx.doi.org/10.1109/SP.2015.29.

[55] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor. Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation. In *IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016*, 2016. Available at http://dx.doi.org/10.1109/EuroSP.2016.20.

[56] S. Alfano. A numerical implementation of spherical object collision probability. *Journal of the Astronautical Sciences*, 53(1):103–109, January-March 2005. Available at http://centerforspace.com/downloads/files/pubs/JAS.V53.N01.pdf.

[57] Y. Lindell and B. Pinkas. Privacy preserving data mining. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '00, pages 36–54, London, UK, UK, 2000. Springer-Verlag. ISBN 3-540-67907-3. Available at http://dl.acm.org/citation.cfm?id=646765.704129.

[58] D. Mazur. *Combinatorics : a guided tour*. MAA textbooks. Mathematical Association of America, Washington, DC, 2010. ISBN 978-0-88385-762-5. Available at http://opac.inria.fr/record=b1133224.

[59] A. Waksman. A permutation network. *J. ACM*, 15(1):159–163, January 1968. ISSN 0004-5411. doi: 10.1145/321439.321449. Available at `http://doi.acm.org/10.1145/321439.321449`.

[60] Y. Ishai, J. Kilian, K. Nissim, and E. Petrank. Extending oblivious transfers efficiently. In *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*, pages 145–161, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-45146-4. doi: 10.1007/978-3-540-45146-4_9. Available at `http://dx.doi.org/10.1007/978-3-540-45146-4_9`.