UNIVERSITÀ DEGLI STUDI
DI MILANO

DOCTORAL SCHOOL OF COMPUTER SCIENCE

DEPARTMENT OF COMPUTER SCIENCE
"Giovani Degli Antoni"

PhD in Computer Science
Cycle XXX

Toward Lower Communication in Garbled Circuit Evaluation

PhD Thesis of:
Maryam Ehsanpour

Supervisor:
Prof. Ernesto Damiani

Co Supervisors:
Prof. Stelvio Cimato, Prof. Valentina Ciriani

PhD School Headmaster:
Prof. Paolo Boldi

Academic year 2016/17

Abstract

Secure Multi-party Computation (SMC) is a classical problem in theoretical security. In a SMC problem, two or more parties must compute correctly a function f on their respective inputs x and y, while preserving the privacy of their inputs and additional security properties.

One of the approaches proposed for addressing the SMC problem relies on the design of Garbled Circuit (GC). In *Garbled Circuits* (GCs), the function to be computed is represented as a Boolean circuit composed of binary gates. The input and output wire of each gate is masked such that the party evaluating the *Garbled Boolean Circuits* (GBC) cannot gain any information about the inputs or the intermediate results that appear during the function evaluation. The complexity of today's most efficient GC protocol depends linearly on the size of the Boolean circuit representation of the evaluated function. The total cost and run-time interaction between parties increase linearly with the number of gates and can be huge for complex GBCs. Actually, interest has grown in the efficiency of this technique and in its applications to computation outsourcing in untrusted environments.
A recent work shows that XOR gates in a Boolean circuit have no cost for the secure computation protocol. Therefore, circuits with a reduced number of non-XOR gates are more convenient and one of the possible ways to reduce the complexity of the computation is to reduce the number of non-XOR gates in the Boolean circuit.

Recalling that, the main aim of this work is to reduce the number of non-XOR gates, which directly results in a reduced number of interactions between the parties and transfer complexity at runtime, we present different approaches for reducing the communication cost of Secure Multi-party Computation (SMC) and improving the overall computation time and efficiency of the execution of SMC.

Preface

The thesis deals with a secure multi-party computation. The problem set-up is interesting in the aspect of combining multiple fundamental issues including secure distributed computing, garbling circuits, and cost-optimization. Since the garbled circuit are useful in information security, we try to optimize garbled circuit transformation and construct it which requires minimum runtime interaction between the input owners.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my thesis supervisor Prof. Ernesto Damiani, not only for introducing me to the interesting concept of garbled circuit, but also for his patience, motivation, enthusiasm, and immense knowledge. I am very grateful for his support and trust. I would also like to thank my co-supervisors, Prof. Stelvio Cimato and Prof. Valentina Ciriani for their encouragement, and their aspiring guidance and friendly advice during my project. Besides, I would like to thank my thesis referees, Prof. Anna Bernasconi, Prof. June-Koo Kevin Rhee and Prof. Faisal Shah Khan, for their insightful comments.

I would like to express my deep appreciation to my husband, Ali Ahmadzadeh, for his support in every possible way. He continuously pushed me to do my best and reinforced my strength in times when I doubted myself.

My sincere thanks also goes to Lorena Sala for helping me adjust to the new environment and made me feel comfortable.

I am especially grateful to my parents for supporting me spiritually throughout my life. I always knew that they believed in me and wanted the best for me. This thesis would not have been possible without the support of my family, my husband, Professors and my friends.

"Knowledge is the most precious of all heritage and thought is the most clear of all mirrors" Imam Ali (AS)

Contents

List of Tables

List of Figures

Short Contents

Chapter 7

Conclusions

Introduction

In a Secure Multi-party Computation (SMC) problem, two or more parties must compute correctly a function f on their respective inputs x and y, while preserving the privacy of their inputs. Besides inputs privacy, other security properties are requested for the execution of the SMC protocol, such as correctness, meaning that the output received is exactly *f(x,y)*, and independence of inputs, meaning that neither party can choose its input as a function of the other party's. In [2], different paradigms for solving secure computation problems have been proposed, some relying on Homomorphic Encryption (HE), others on Linear Secret Sharing (LSS), usually deployed when more than two parties are involved.

In the mid 1980's, Yao proposed a first approach for addressing two-party SMC [79]. Yao's construction is based on the design of a Garbled Circuit (GC), that is a normal Boolean circuit representing the function to be computed securely, but whose evaluation is performed gate by gate using a protocol to respect inputs privacy. Indeed the input and output wires of each gate are masked so that the party evaluating the GC cannot gain any information about the inputs or the intermediate results that appear during the function evaluation. For each gate, an Oblivious Transfer (OT) protocol [56] is run between the two parties so that the resulting output value can be computed without knowing the input value of the other party. Specifically, all the inputs are encrypted and during the evaluation of the output of each gate the decryption keys are exchanged. The total OT payload increases linearly with the number of gates, and can be huge for complex GCs. A possible approach to increase the efficiency of the GC technique has been proposed by Kolesnikov et al. [37] in 2008, enabling the evaluation of XOR gates essentially free communication-wise (i.e., it requires one local XOR operation, and no garbled table entries to generate or transfer). For this reason, construction of GCs requiring less non-XOR gates, achieves the goal of reducing OT payload and therefore reduces the overall communication cost. The problem then is to design the GC that represents the function f to be computed and requires the minimum interaction at runtime between the collaborating parties. In turn, since there are multiple representations of the same function, or better equivalent Boolean circuits, this means that we try to select the one that has the minimum number of non-XOR gates, since XOR gates have no cost for the execution of the secure computation protocol.

The goal of this thesis is to study the possibility of decreasing the computational and communicational cost of the SMC by reducing the number of non-XOR gates. This directly results in a decreased runtime interactions between the input owners, thus improving the overall computation time and efficiency of the execution of the SMC.

Applications for two-party secure computation have three properties: (1) the application involves inputs from two independent parties; (2) each party wants to keep its own data secret; and (3) the participants agree to reveal the output of the computation. That is, the result itself does not imply too much information about either party's private input.

Secure Multi-party Computation (SMC). Secure Multi-party Computation (SMC) protocols have been introduced to give two or more parties the capability to compute a function f of their respective inputs x and y, still keeping their inputs private, and sharing only the final result $z = f(x, y)$. In the last years, SMC was used as enabling technology for a large number of security- and privacy-critical applications (e.g., electronic auctions [48], data mining [40], remote diagnostics [14], medical diagnostics [5], or face recognition [23]).

Yao's Garbled Circuits (GCs) Protocol. Using Garbled Circuits (GCs) protocol [79], is the first approach to solve the SMC problem. Yao's GC construction introduces a protocol for the evaluation of the input function (f) represented as a Boolean Circuit composed of binary gates. The basic idea is that one party "encrypts" the circuit (using symmetric keys), the other party obliviously obtains the keys corresponding to both parties' inputs and the GC, and is able to decrypt the corresponding output. One party constructs the circuit C, and converts it into a garbled circuit and the garbled circuit is transferred to the other party. Specifically, the output of each gate in the GC is evaluated by exchanging some encrypted information between the two parties, so that none of the two parties learns any information about the inputs of the other party. So, the important note is that just a final output is shared between two parties and each party do not have any information about other party's value or any intermediate values during the protocol. For each gate, an Oblivious Transfer (OT) protocol [56] is run between the two parties so that the resulting output value can be computed without knowing the input value of the other party. The total OT payload increases linearly with the number of gates, and can be increased for complex GCs. Efficiency of GC protocol depends linearly on the size of the Boolean circuit representation of the evaluated function.

Free XOR Protocol. Kolesnikov and Schneider [37] proposed an improvement that allows XOR gates to incur zero communication with no cryptographic operations. This cryptographic protocols bring significant benefit to many SMC settings and allow to evaluate XOR gates at a substantially lower cost (i.e., in computation and communication required for creation, transfer and evaluation of the Garbled tables) than non-XOR gates such as AND gates. The *free-XOR* technique [37] allows all XOR gates to be executed by just XOR-ing the input wire labels, without the need of any encryption operations. Because of this, it is worth investing the effort to minimize the number of non-XOR gates in the construction of Garbled Circuits with the goal of reducing the communication cost. The main observation of this protocol is that it is not necessary to select all garblings independently.

Quantum Gates. One of today's most fascinating fields of research and innovation involves applying quantum phenomena to new technology. The belief is that these technologies promise to revolutionise society this century through secure communication, precision measurement, and powerful computation. Whereas in classical circuits the bit is the standard building block for storing information, in quantum circuits the information is represented by the quantum state of qu-bits. The logical properties of qubits also differ significantly from those of classical bits. Bits and their manipulation can be described using two constants (0 and 1) and the tools of boolean algebra. Qubits, on the other hand, must be discussed in terms of vectors, matrices, and other linear algebraic constructions. In particular, while a bit can have two states only, 0> or 1>, a qu-bit stores any linear combination c 0> + d 1> of 0> and 1>, where c and d are complex numbers. A quantum logic gate has the same number of inputs and outputs, and there is a bijective function between each input and output. Therefore, quantum gates are inherently reversible, meaning that any gate has a corresponding inverse operation. A quantum library contains a set of gates that compose a universal set meaning that it is possible to realize any reversible function using the gates in that library. NOT gate and two-bit Controlled-NOT (or C-NOT) gate [27] and three-bit Controlled-Controlled-NOT (or CC-NOT or Toffoli) gate [73] are some of the most common gates from NCT quantum library.

Over the last couple of decades, quantum cryptography and quantum gates have received growing attention. A large number of works in literature deals with *quantum key distribution,* i.e., the process of using quantum communication to establish a shared key between two parties (§2.2) [9].

Binary decision diagrams(BDDs). The standard representation form for logic was the sum of product (SOP) form, i.e., a disjunction (OR) of conjuctions (AND) made of literals. The advent of very large scale integration, the standard representation for logic moved from SOP to directed acyclic graphs (DAGs) [13]. A notable example of DAG where all the nodes realize the same function is binary decision diagrams (BDDs) [15]. BDDs are canonical and provide very efficient manipulation procedures. For this reason, BDDs found application in various areas, such as verification, testing, optimization, automated reasoning, etc.[44].

Multiple-Valued Logic (MVL). In the field of circuit design, Multiple Valued Logic *(MVL)* [10, 46] is a direct generalization of the standard Boolean logic, where the classical Boolean domain $B = \{0,1\}$ is replaced by $P = \{0,1,...,|P|-1\}$ with $|P|>1$. For high level design it is natural to think of multiple valued variables, rather than Boolean ones, with the aim of reducing the number of interconnections required and circuit cost to implement logic functions [75].

Problem Statement and Contributions

In this thesis we focus on improve efficiency of Garble Circuit(GC) protocol in Secure Multi-party Computation (SMC) with presenting methods for optimization of this protocol based on reducing number of non-XOR gates, which result in reducing OT payload and therefore reduce the overall communication cost. The overall research question we are interested in can be stated as follows: *"Given a Boolean function f represented by a garbled Boolean circuit (GBC) and evaluated on the private inputs held by two parties, how can we optimize this GBC transformation and how can we construct the GBC that requires minimum runtime interaction between the input owners?"*

In brief, our contributions are as follows:

- We have put forward the methods for building efficient Boolean-Circuit for SMC of functions based on Garbled Circuit (GC) protocols and decreasing the computational cost and improving the overall efficiency of the execution of the SMC.

- Using Free-XOR technique that allows XOR gates to be evaluated "for free" in GC construction, i.e., without relying on corresponding garbled gate. It can help us to reduce the number of non-XOR gates that result in reduced number of runtime interactions between the parties and the communication cost.

- Work on different design techniques (i.e., Quantum gates) and different function representation methods (i.e., BDDs, Multiple-Valued logic). Using Quantum gates can reduce the gate complexity and helps us to design quantum garbled circuit (Q-GC) equivalent to the original Boolean circuit, which can result in increasing the number of XOR gates, and decreasing the number of non-XOR gates, thus requiring less interaction. Moreover, using Binary Decision Diagrams (BDDs) method to identify non-XOR gates that could be replaced by XORs without altering the output of the circuit. This method can also result in reducing the number of operations, reducing the communication costs and circuit cost. In addition, we follow the idea of garbled circuit protocol from Boolean logic to the Multiple Valued Logic (MVL) setting to obtain more compact circuit descriptions.

In particular we have studied our methods (Quantum Garbled Circuit, BDDs, MV) for standard functionalities such as the *Millionaires' Problem*, which is the problem of determining which party has the greatest input, and for the *adder*, that is the computation of the sum of the private inputs. In all cases we try to show the improvements achieved in terms of reduced communication time.

The rest of this dissertation is organized as follows:

Secure Multi-party Computation (SMC) (§1).
Starts with a detailed introduction of two-party secure multi-party computation (SMC) with special focus on Garbled circuit (GC) protocol as a solution to solve the problem of SMC and *free-XOR* technique for improving efficiency of GC. Besides introducing compiling tools for implementing two-party secure computation and making it as a practical applications.

Introduction to Quantum Gates (§2).
Gives more detailed introduction to quantum logic concepts and simulation tool for synthesis, implementing and testing of quantum circuits.

Boolean Function Representation (§3).
Reviews concepts of Boolean Function Representation as Binary Decision Diagrams (BDDs) and gives basic information on Multiple-Valued Logic (MVL) respectively.

Secure Multi-Party Computation Exploiting Quantum Gates (§4).
Presents Garbling of Boolean function and quantum implementation of Garbled Boolean function. We show how to improve efficiency of SMC by using quantum gates instead of traditional gates in Garbled Boolean circuit construction. It also presents one of our proposed methods (Q-GBC) for improving efficiency of Garbled Boolean Circuits.

BDDs for Secure Multi-Party Computation (§5).
Provides our proposed optimization technique in GC protocol using BDDs.

MV for Secure Multi-Party Computation (§6).
proposes multiple valued GC protocol and the extension of the optimization technique for the evaluation of multiple valued gates.

Conclusions (§7).
Summarizes the contributions of our thesis and states final conclusions with directions for future work in all of our proposed techniques, Q-GBC, using BDDs and Multi-valued Logics for improving efficiency of SMC.

List of Related Publications:

"Toward Design of Garbled Circuits Using Quantum Gates", Maryam Ehsanpour, COMPSAC Conference, Building Digital Autonomy for a Sustainable World, Politecnico di Torino, 4 July - 8 July 2017, Turin, Italy.

"Exploiting Quantum Gates in Secure Computation", Maryam Ehsanpour, Stelvio Cimato, Valentina Ciriani, Ernesto Damiani, Euromicro Conference on Digital System Design, (DSD), 30 August - 1 September 2017, Vienna, Austria.

"A Multiple Valued Logic Approach for the Synthesis of Garbled Circuits", Stelvio Cimato, Valentina Ciriani, Ernesto Damiani, Maryam Ehsanpour, 25th IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC), 23 October-25 October 2017, Yas Viceroy, Yas Island, Abu Dhabi.

Chapter 1

Secure Multi-party Computation (SMC)

This chapter introduces Secure Multi-party Computation and Two-Party Secure Computation. The most efficient solution to solve the problem of SMC (Garbled Circuits Protocol) in the two-and multi-party setting is explained in §1.2, and optimization of GCs protocol has been introduced in §1.3. Finally, compiling tools for implementation of Two-Party SMC protocols are introduced in §1.4, and related work are described in §1.5.

1.1 Notations and Definitions

Due to the tremendous growth of the huge amount of computers and distributed working environments, protocols are required for protecting the privacy of users and reliability of results. This problem is known as secure multi-party computation, which is a special case of a long-studied problem in cryptography.

in Secure Multi-party Computation (SMC) the parties P_1, P_2, $\ldots$, P_k with inputs of x_1, x_2,$\ldots$, x_k want to compute some common function $f(x_1, x_2,\ldots, x_k)$ such that a party P_i can know only its own input x_i and the value of the function f. The aim of SMC protocol is to enable parties to compute a function f in a secure manner where keeping their inputs private, and sharing only the final result.

SMC problem provides a general solution to execute any computation in combinational circuits [78, 79]. It can have high communication cost as the complexity of the computation grows.

The design of efficient SMC protocols is considered for a variety of security-critical applications with sophisticated privacy and security requirements such as electronic voting, electronic auctions [48], electronic cash schemes, data mining [40], remote diagnostics [14], classification of medical data [5], or face recognition [23, 50, 60]. In addition the technology of secure multi-party computation has gain much interest recently in research community, governments and industry as a potential tool for their need.

1.1.1 Protocol Parties

Secure computation protocols have been introduced to provide two or more interacting parties with the capability of computing a function f of their respective inputs x and y, while keeping their inputs private. The protocol involves two players or participants, Alice and Bob, who want to evaluate and compute the function f together on that private data while keeping their inputs secret.

1.1.2 Security in Multi-party Computation

Security of input data is of prime importance for multi-party computation. protocol execution can be come under *"attack"* by an *adversaries.* The aim of this attack may be to learn private information or cause incorrect result of the computation. In order to prove that a protocol is secure and can endure any *adversarial attack*, a number of different definitions have been considered that ensure important security properties. The must central of these properties for multi-party computation are defined below:

- **Privacy:** Nothing should be learned more than what is necessary. Parties should learn their output and nothing else. What can be derived from the output itself, is the only information that should be learned about other parties' input. For example, the *privacy* requirement for an *election protocol* ensures that no parties should learn anything about the individual votes of other parties. Likewise, the *privacy* requirement for an *auction protocol* ensures that only the winning suggestion is revealed and clearly possible to find that all other suggestions were lower than the winner's offer.

- **Correctness:** Parties are ensured that they can receive the correct output and trusted party can not be corrupted. For example, the *correctness* requirement for an *auction protocol* ensures that the party with the highest suggestion in guaranteed to win.
- **Guaranteed Output Delivery:** Adversary should not be able to prevent honest parties from receiving their output.
- **Fairness:** Adversaries or (corrupted parties) should receive their outputs if and only if the honest parties also can receive their outputs.

Security for Multi-party Computation (MPC) can be defined an *"ideal-real-world"* consisting of both *"ideal world"* model and *"real world"* model. In the *"ideal world"* model, players give their inputs to a incorruptible trusted party that computes the function on its own and sends back the result to each party. Notice that in this ideal computation, all of the above security properties hold. In contrast, in the *"real world"* model there is no trusted party, and the parties can only exchange messages between all the parties. So, security in MPC protocol means that adversaries can perform in the *"real world"* protocol and also do in the *"ideal"* setting.

1.1.3 Secure Two-party Computation

Secure Two-party Computation allows two parties to compute correctly a function f on their respective inputs x and y, while preserving the privacy of their inputs and additional security properties and sharing only the final results. The most important of these properties are *privacy* meaning that the parties learn the output *f(x,y)* but nothing else, *correctness*, meaning that the output received is indeed *f(x,y)* and not something else, and *independence* of inputs, meaning that neither party can choose its input as a function of the other party's.

1.1.4 Comparison (Millionaires Problem)

The "Millionaires Problem" was introduced by Yao in [78] as motivation for secure computation: two millionaires want to securely compare their respective private

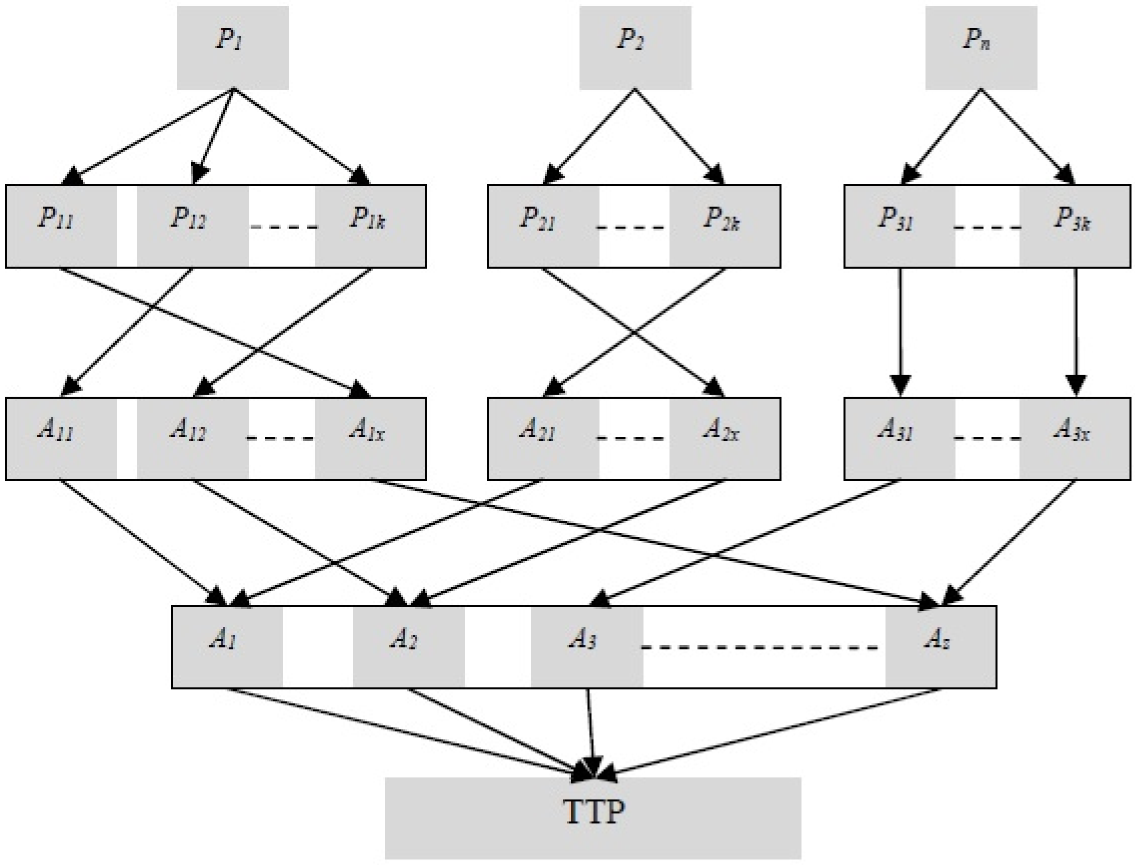


Figure 1.1: Data flow in VPP with five layer structure consisting of party layer, virtual party layer, trusted anonymizer layer, untrusted anonymizer layer and computation layer from starting to end respectively [52].

input values without revealing more information than the outcome of the comparison to the other party (e.g., about their amount of money, in which two parties are millionaires and two millionaires want to know who is richer without letting each other know about the amount of money they have). Yao's Garbled Circuit (GC) protocol (cf. §1.2) is the most efficient solution to solve the Millionaires problem.

1.1.5 Secure Multi-party Computation Using Virtual Party

Virtual Party Protocol (VPP) can be used to ensure the privacy preserving the data input by not revealing the right data. In this protocol some fake data and some virtual party are created. There are n parties P_1,P_2,P_3...,P_n. Each party P_i has data X_{i1},X_{i2},X_{i3}...,X_{im}. Each party P_i has some trusted anonymizers A_{i1},A_{i2},A_{i3}...,A_{iX}

and Z number of un-trusted anonymizers A_1,A_2,A_3...,A_Z. Each party P_i will create some fake data F_{i1},F_{i2},F_{i3}...,F_{iq}, where q is the total number of fake data entries. Each Party P_i will also create K virtual parties P_{i1},P_{i2},P_{i3}...,P_{iK}. Then the value of each data D_{i1},D_{i2},D_{i3}...,$D_{i(m+q)}$ is encrypted and distributed randomly among the virtual parties. Corresponding modifier tokens for every P_i is also created that are mixed with fake data. These modifier tokens are distributed randomly among the virtual parties that will be used in the final computation to obtain the correct result. These parties will send their data to trusted anonymizers. Trusted anonymizers distribute their data randomly among the un-trusted anonymizers and the data of un-trusted anonymizers is sent to third party. Third party will use the data and the modifier tokens to compute the result.

The whole scenario of encryption, modifier tokens, encrypted data and the method of computation can be seen in Figure 1.1.

1.1.6 Secure Multi-party Computation Using Secure Sum Protocol

In Secure Sum Computation Protocol proposed by Clifon et al. [65] all the parties want to know the sum of their individual data inputs. In this protocol parties put in a unidirectional ring and one of these parties is selected as a *protocol initiator* party. Probability of data leakage is reduced to zero by changing the positions of the parties in the ring. Randomization method is used in this protocol for computing the sum as described below:

The computation will be started by choosing a random number and add its own data input. The sum is then transmitted to the next party. The next party adds the received sum to own data and then sends this new sum to the next party. This procedure is repeated until the *protocol initiator* receives the sum of all the data and the random number. The random number is known only to the protocol initiator party, Thus it subtracts the random number from the sum and allows all the parties to know the result.

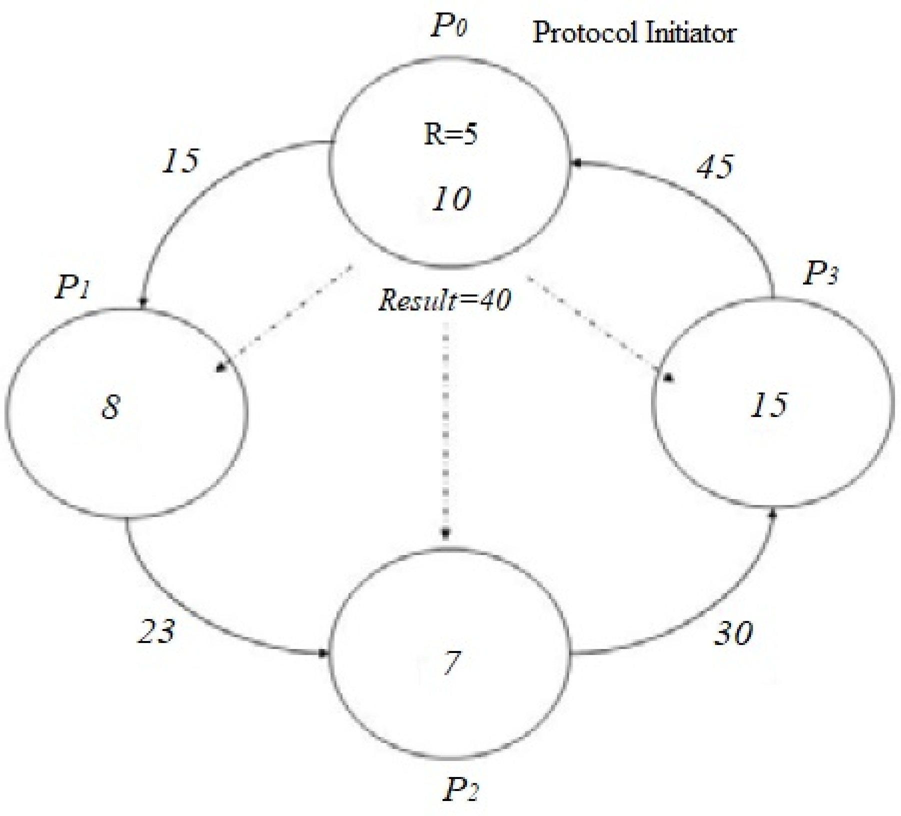


Figure 1.2: Secure Sum Protocol [65].

All scenario of this protocol are depicted in Figure 1.2 where parties where parties P_0, P_1, P_2 and P_3 with their data 10,8,7 and 15 perform secure sum computation. Initiator party P_0 selects a random number R=5 and sends to P_1 the sum of this random number and its private data (10). Also the new sum 23 can be resulted by adding the data of P_1 (8) and previous sum (15). This process is repeated until the sum 45 is received by P_0. P_0 can subtract random number from 45 and send the final sum as 40 to all the parties.

1.1.7 Secure Multi-party Computation Using Secret Sharing Schemes

Where participation of president and his highest ranking is required to unlock the trigger for the nuclear missile, we call this *secret sharing* in cryptography.

Secret sharing is used as one of the secure protocols for solving the multi-party computation problem. This protocol involves a set of n parties $\{P_1,\ldots\ P_n\}$, a dealer who has a secret, and a collection of subsets of parties called the access structure. A secret is divided into n shares, and they are privately given to n parties. An access structure is a *monotone collection* $\mathcal{A} \subseteq 2^{\{P_1,\ldots P_n\}}$ of subset of $\{P_1,\ldots\ P_n\}$. A collection $\mathcal{A}$ is monotone if $B\in\mathcal{A}$ and $B\subseteq C$ imply that C$\in\mathcal{A}$. Sets in $\mathcal{A}$ are called authorized subsets and sets not in $\mathcal{A}$ are called unauthorized subsets. Access structure is a method of sharing a secret $\mathcal{S}$ among a set of $\mathcal{K}$ participants (parties) $\mathcal{P}$ in such a way that the following two properties are satisfied:

- If an authorized subset of parties pool their shares, then they can determine the value of $\mathcal{S}$.
- If an authorized subset of parties pool their shares, then they can determine nothing about the value of $\mathcal{S}$.

1.2 Yao's Garbled Circuit Protocol

In the mid 1980's, Yao proposed a first approach for addressing two-party Secure Computation [78, 79]. Yao's garbled circuit is a way to "encrypt a computation" that reveals only the output of the computation, but reveals nothing about the inputs or any intermediate values. In this section we introduce some standard notation and concepts corresponding to Garbled Circuit protocol (§1.2.1) and then Yao's garbled circuit protocol is explained in several steps (§1.2.2).

1.2.1 Basic Concepts and Definitions

In this section we introduce common definitions, function to circuit, Boolean Circuit, Bit-String, Garbled Values and Oblivious Transfer Protocol used in this thesis.

Function to Circuit

According to Goldreich et al. in [31] it is possible to map any poly-time function f with fixed size input to a Boolean circuit C consisting of digital gates (AND,OR,NOT,..) that returns the same result.

Boolean Circuits

Boolean circuits are classical and standard representations for functions which are particularly useful for SMC. A Boolean circuit consists of n-input gates G_n and wires that performs a mapping from n input bits to one output bit, i.e.,

$$G_n : (in_1, ..., in_\mathrm{n}) \in \{0, 1\}^\mathrm{n} \to \{0, 1\}.$$

Boolean circuit is an acyclic (i.e., loop-free or feed-forward) graph used as a mathematical model of digital logic circuits, defined in terms of the logic gates.
Every gate that is used in Boolean circuit construction creates a cost. Thus, if we reduce the number of gates, we can reduce the cost involved in it. Generic gates are AND, OR, NOT, XNOR, XOR.

Bit-Strings

Bit string is a sequence of bits, which defines and represents sets of binary data that are numbered from zero to the number of bits in the string less one. The length of a bit string is also defined as the number of bits that it contains. $\{0,\ 1\}^l$ defined the space of binary strings of length l. A bit string can also contain zero or more bits.

Garbled Values

Computations in a GC are not performed on obvious values 0 or 1, but on random bit strings, called garbled values. In construction of the GC, two random bit strings (garbled values) $k^0{}_x$, $k^1{}_x$ are assigned to each wire x_i of C. Two random bit strings, which are assigned to the corresponding value 0 and 1, do not disclose to them as they are chosen randomly.

1-out-of-2 Oblivious Transfer Protocol (OT)

A foundation building block for almost all efficient protocols of secure computation is 1-out-of-2 Oblivious Transfer (OT) protocol [56, 25] that consists of two phases: the Transferring phase and the opening phase. Oblivious Transfer protocol works between two parties, a Sender (Alice) and a receiver (Bob). This protocol is called 1-out-of-2 since the receiver learns one of the 2 inputs of the sender and forget any information.

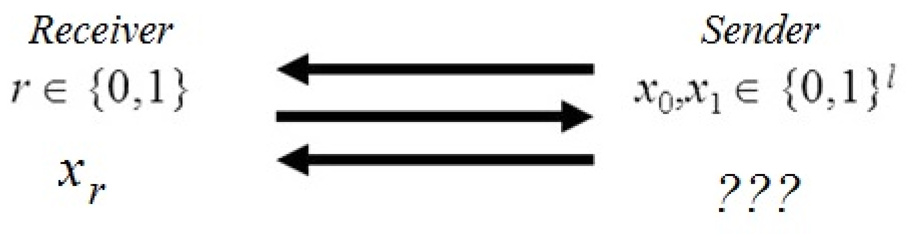


Figure 1.3: The Oblivious Transfer Protocol.

In the transferring phase, the goal of the sender (Alice) is to send a message (in general, a bit-string) to another party (Bob) in such a way that he can decide to obtain one of the 2 inputs of the sender at his choice but not both. In the opening phase, the goal of the receiver (Bob) is to open the message, but the sender never finds which message Bob received.

As you can see in Figure 1.3, a sender (Alice) has two messages (x_0,x_1). She sends a one-bit message to another party (Bob) in such a way that he can decide to obtain one of the two messages according to his choosing bit (r). The result is that the receiver learn x_r without learning anything about sender's bit while the sender learn nothing about r. Alice never finds out which message Bob received and remain totally ignorant about which of the two messages he received.

1.2.2 Description of Yao's Garbled Circuit Protocol

Yao's GC construction, often called Garbled Boolean Circuit (GBC) introduces a protocol for the evaluation of the input function (f) represented as a Boolean Circuit and is based on the encryption of the input and intermediate values, so that only the final result is shared among the parties.

Yao's Garbled Circuit construction is composed of two phases: Garbling and Evaluation, which are distinctly executed by the two parties. During the Garbling phase, Alice converts a circuit into a garbled circuit, while Bob performs the Evaluation phase taking in input the garbled circuit, executing some interactions with Alice, and finally computing the output value.

If we want to be a little more precise, a "garbling scheme" consists of the following steps:

- Alice generates a circuit representation C of function *f*.
- Alice transforms the circuit to a garbled Circuit representation by garbling every gate.
- Alice sends Garbled Circuit and her key to Bob.
- Alice and Bob perform 1-out-of-2 OTs to receive Bob's key.
- Bob evaluates the Garbled Circuit and outputs the results.

1.2.2.1 Generate Garbled Boolean Circuit

To generate the garbled Boolean circuit Alice selects and associates k-bit-string random keys to each input and output wire and for each possible value. She generates the random keys ${k^0}_x$, ${k^1}_x$ for input wire x and ${k^0}_y$, ${k^1}_y$ for input wire y and ${k^0}_z$, ${k^1}_z$ for output wire z. She puts these random keys for each wire and make *"Garbled Boolean Circuit"*. In this encrypt circuit, inputs and outputs of each gates are masked such that the parties cannot gain any information about input or intermediate result. This process is repeated for each gate composing the Boolean circuit. Encryption is important because it helps the outputs to look random and additionally prevents Bob from obtaining further information.

1.2.2.2 Generate Garbled Truth Table

Alice also encrypts the truth table of the gate using those random keys and sends the garbled truth table to Bob. She should cooperate with Bob to know about actual output. An initial table for an XOR gate and resulting garbled truth table is shown below in Figure 1.4 and Table 1.1.

1.2.2.3 Sending Alice's Garbled Values

Once that Alice has generated the garbled circuit she needs to define her own input values to send to Bob, then Alice selects the appropriate input keys for the garbled

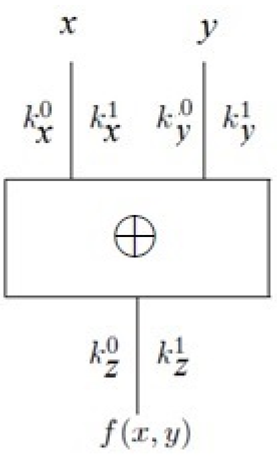
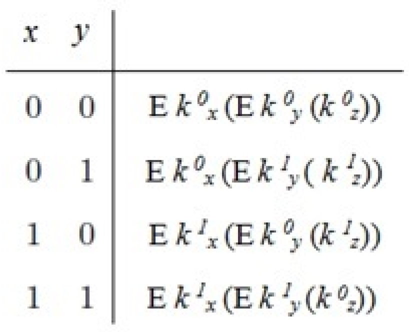


Figure 1.4: XOR gate with its corresponding wire keys.



x	y	
0	0	$E\,k^0_x(E\,k^0_y(k^0_z))$
0	1	$E\,k^0_x(E\,k^1_y(k^1_z))$
1	0	$E\,k^1_x(E\,k^0_y(k^1_z))$
1	1	$E\,k^1_x(E\,k^1_y(k^0_z))$

Table 1.1: Initial garbled circuit table for XOR gate

circuit GC. Alice sends to Bob all the keys corresponding to the values she owns for each gate. Following the previous example, if Alice's first input bit is 0, she needs to send the garbled representation $k^0{}_x$. This process is repeated for all Alice's input bits, generating Alice's garbled input values and sending them together with the garbled circuit GC to Bob.

1.2.2.4 Using OT for Bob's Input Values

Now Bob needs the keys corresponding to his input value. Alice knows the keys for each possible input value but she does not know Bob's choice of input values, therefore they engage in a 1-out-of-2 Oblivious Transfer (OT) protocol for each input value from Bob to decrypt the corresponding entries in the garbled truth table. Thus, Bob and Alice execute an OT once per each input wire. After performing OT as many times as needed, Bob ends with all the information that he needs to evaluate the circuit.

As mentioned before, the Oblivious Transfer protocol in Yao's construction, is exploited to transfer information between the two parties when evaluating the GBC. Specifically, all the inputs are encrypted and during the evaluation the output of each gate the decryption keys are exchanged, so that none of the two parties learns any information about the inputs of the other party. For each gate, an Oblivious Transfer (OT) protocol is run between the two parties so that the resulting output value can be computed without knowing the input value of the other party.

To see how "1-out-of-2" OT is used to compute the GC, let consider W_1, .., W_{u_1} be the circuit input wires corresponding to input held by Alice, and let W_{u_1+1}, .., $W_{u_1+u_2}$ be the circuit input wires corresponding to input held by Bob. Then:
(a) Alice sends to Bob the garbled values $W_1{}^{x_1}$, .., $W_{u1}{}^{x_{u1}}$.
(b)For every

$$i \in 1, .., u_2$$

Alice and Bob execute the 1-out-of-2 OT protocol, where Alice's input is ($k_0{}^{u_1+i}$, $k_1{}^{u_1+i}$), and Bob's input is yi. Bob now has the garbled tables and the garblings of all circuit's input wires. Bob evaluates the garbled circuit, and outputs *f(x, y)*.

1.2.2.5 Evaluating the Garbled Circuit

Finally, Bob will determine the corresponding output key that he can share to Alice to find out what the actual outputs were. The important note is that Bob can share just a final output with Alice and she does not have any information about Bob's value or any intermediate values during the protocol, and Bob doesn't know the input values of Alice and the intermediate values of the computation, which are randomly chosen values.

1.3 Efficient GC-based Secure Multi-Party Computation (SMC)

The total Oblivious Transfer (OT) payload and GC protocol runtime increases linearly with the number of gates, and can be huge for complex GBCs. Accordingly, the efficient GC-based Secure Multi-Party Computation (SMC) protocol is of crucial importance. We will introduce Free XORs method for improving Garbled Circuit efficiency in §1.3.1.

1.3.1 Free-XOR Protocol

An optimization of the basic GC construction has been proposed by Kolesnikov et al.[37]. In the modified protocol the evaluation of XOR gates comes "for free", in the sense that they do not require any communication for the generation and the evaluation of the associated garbled table.

The basic idea for the circuit generator is to keep a global random bit string R such that for every wire only the label W^0 (representing 0) needs to be randomly sampled while the label W^1 (representing 1) is simply set to $W^0 \oplus R$ for every binary XOR gate (with input wires subscripted with i, j and the output wire with k), the label representing 0 on the output wire is derived from xor-ing corresponding input labels, i.e.,

$$W_k{}^0 = W_i{}^0 \oplus W_j{}^0.$$

The implementation of the XOR gate, we will follow step-by-step Kolesnikov's improvement [KS08] which works as follows.

1.3.1.1.Garbling

In the first step, Alice computes the garbled Boolean circuit (GBC) as follows:

1.a Randomly choose global key offset,

$$R \in_{\mathrm{R}} \{0,1\}^{\mathrm{N}}$$

1.b For each input wire Wi of C:

(a) Randomly choose a garbled value

$$w^{0}{}_{i} = <k^{0}{}_{i}, p^{0}{}_{i}> \in \{0,1\}^{\mathrm{N}+1} \quad (keyk \in \{0,1\}^{\mathrm{N}}, p \in \{0,1\})$$

(b) Set the other garbled output value

$$w^{1}{}_{i} = <k^{1}{}_{i}, p^{1}{}_{i}> = k^{0}{}_{1} \oplus R, p^{0}{}_{i} \oplus 1$$

1.c For each gate G_i of C in topological order

(a) label G(i) with its index: label (Gi) = i

(b) If G_i is an XOR-gate Wc = XOR(Wa, Wb) with garbled input values

$$w^{0}{}_{a} = <k^{0}{}_{a}, p^{0}{}_{a}>$$

$$w^{0}{}_{b} = <k^{0}{}_{b}, p^{0}{}_{b}>$$

$$w^{1}{}_{a} = <k^{1}{}_{a}, p^{1}{}_{a}>$$

$$w^{1}{}_{b} = <k^{1}{}_{b}, p^{1}{}_{b}>$$

Set garbled output value

$$w^{0}{}_{c} = k^{0}{}_{a} \oplus k^{0}{}_{b}, p_a \oplus p_b$$

Set garbled output value $w^{1}{}_{c} = k^{0}{}_{a} \oplus k^{0}{}_{b} \oplus R, p_a \oplus p_b \oplus 1$

(c) If Gi is a 2-input gate Wc = gi(Wa, Wb) with garbled input values

$$w^{0}{}_{a} = <k^{0}{}_{a}, p^{0}{}_{a}>$$

$$w^{0}{}_{b} = <k^{0}{}_{b}, p^{0}{}_{b}>$$

$$w^{1}{}_{a} = <k^{1}{}_{a}, p^{1}{}_{a}>$$

$$w^{1}{}_{b} = <k^{1}{}_{b}, p^{1}{}_{b}>$$

Randomly choose garbled output value

$$w^0{}_c = <k^0{}_c, p^0{}_c> \in {}_{\mathrm{R}}\{0,1\}^{\mathrm{N+1}}$$

Set garbled output value

$$w^1{}_c = <k^1{}_c, p^1{}_c> = k^0{}_c \oplus R, p^0{}_c \oplus 1$$

Create Gi's garbled table. For each of 2^2 possible combinations of Gi's input values

$$v_{\mathrm{a}}, v_{\mathrm{b}} \in \{0,1\}, set$$

$$e_{v_{\mathrm{a}},v_{\mathrm{b}}} = H(k^{v_a}{}_a \parallel k^{v_b}{}_b \parallel {}_{\mathrm{i}}) \oplus \ W^{gi(va,vb)}$$

Sort entries e in the table by the input pointers, i.e. place entry $e_{\mathrm{va,vb}}$ in position

$$<h^{p\ va}{}_a\ ,\ p^{va}{}_b>$$

1.d.For each circuit-output wire Wi(the output of gate Gj) with garblings

$$w^0{}_i = <k^0{}_i\ ,\ p^0{}_i>,\ w^1{}_i = <k^1{}_i\ ,\ p^1{}_i>:$$

(a) Create garbled output table for both possible wire values

$$v \in \{0,1\}.$$

(b) Set $e_v = H(k_i^v \parallel "out" \parallel j) \oplus v$

(c) Sort entries e in the table by the input pointers, i.e.
place entry e_v in position $p^v{}_i$. There is no conflict, since

$$p^1{}_i = p^0{}_i \oplus 1.$$

In this first phase, Alice executes Kolesnikov's algorithm and uses the output of H, modeled as a Random Oracle [6], to encrypt the garbled output values in the garbled (Step 1.d(b)). Any combination of H's inputs (keys and gate indices) is used for the encryption of at most one table entry.
Alice sends the encrypted circuit to the other party, the *garbled circuit evaluator* (henceforth called *Bob*). When Bob receives the garbled circuit he only knows one garbled value per wire, and can decrypt exactly one entry of Gi's garbled table. All other entries are encrypted with at least one key that Bob does not have. Therefore, one of the two of garbled values of every wire looks random to him.

1.3.1.2. Evaluating

We now discuss Kolesnikov's GC evaluation algorithm, run by Bob. Bob receives all garbled tables, but cannot execute the randomized circuit unless it has the input values to feed to it. Alice sends her inputs to Bob through the Oblivious Transfer protocol (OT)

1.4 Compiling Tools for Implementation of Two-Party Secure Computation Protocols

While the theoretical foundations of two-party Secure Computation have been considered in [78], interest in practical SMC systems is growing and different privacy-preserving frameworks are being developed [63], [64]. So, recent implementation's tool show that SFE is ready to be used in practical applications. To make SMC usable by automatically generating protocols from high-level descriptions, several frameworks for SMC consisting of languages and corresponding tools have been developed in the last years. We review some of these tools, Fairplay (§1.4.1) and CBFS-MPC (§1.4.2) that are used in our thesis and SCAPI (§1.4.3), TASTY (§1.4.4) and ABY (§1.4.5).

1.4.1 FAIRPLAY

A number of well-established frameworks have been presented to translate the theoretical results of SMC protocol into practical applications. Fairplay[1] [43] was one of the first library published for synthesizing GC starting from the definition of the input function. It provides a method to compile given function in a high-level language into a low-level language as a circuit. Indeed, the framework includes a high-level Secure Function Definition Language (SFDL) for specifying the computed function. The definition is then compiled into a low-level description in form of a Boolean Circuit. The language used for describing the Boolean Circuit is called Secure Hardware Definition Language (SHDL). The corresponding Garbled Circuit (GC) is created from the Boolean Circuit and a garbled version of the input data is generated from the original input.

[1]http://www.cs.huji.ac.il/project/Fairplay/

1.4.1.1 Fairplay Syntax using Example

Let us consider as a simple example the SFDL description of a 2-bit Adder:

```
    program Add {
typeint = Int 2;
typeAliceInput = int;
typeBobInput = int;
typeAliceOutput = int;
typeBobOutput = int;
type Output = struct AliceOutputalice, BobOutput bob;
type Input = struct AliceInputalice, BobInput bob;

    function Output output(Input input) {
output.alice = (input.bob + input.alice);
output.bob = (input.bob + input.alice);}
}
```

The compiler, having in input the SFDL program produces the following circuit in SHDL format:

```
0 input //output$input.bob$0
1 input //output$input.bob$1
2 input //output$input.alice$0
3 input //output$input.alice$1
4 gate arity 2 table [0 0 0 1] inputs [2 0]
5 output gate arity 2 table [0 1 1 0] inputs [2 0] //output$output.alice$0
6 gate arity 2 table [0 1 1 0] inputs [3 1]
7 output gate arity 2 table [0 1 1 0] inputs [4 6] /output$output.alice$1
8 output gate arity 1 table [0 1] inputs [5] //output$output.bob$0
9 output gate arity 1 table [0 1] inputs [7] //output$output.bob$1
```

Each line specifies a wire in the generated circuit and shows input bit or a Boolean gate with its truth-table, e.g. table [0 1 1 0] shows XOR gate, table [0 0 0 1] shows AND gate, table [0 1] shows NOT gate.

1.4.2 CBFS-MPC

Circuits of Basic Functions Suitable for Multi-Party Computation (CBFS-MPC)[2]is a tool developed by the Bristol cryptography group. This tool is a set of basic combinatorial circuits, which may be useful for testing binary-circuit based SMC and two-party computation. They use *Cadence Encounter RTL*[3] compiler in conjunction with the *Faraday FSA0A_C 0.18 mm ASIC Standard Cell Library*[4] for synthesis. The resulting circuit reports the number of AND, XOR and INV gates for each circuit design in a given format, where each line contains in order the number of input wires, number of output wires, list of input wires and list of output wires, and the gate type (XOR, AND, or INV). For example, 2 1 3 4 5 XOR means that w5 = XOR (w3,w4).

1.4.3 SCAPI

Secure Computation API (SCAPI)[5] [21] is an open-source Java library (also has a c++ version) for implementing secure two-party and multiparty computation protocols. The main advantages of this framework are *flexibility*, *extendibility*, *efficiency* and *ease of use*. Its *flexibility* means that protocols implemented using SCAPI can be easily changed and replaced because of using primitives and sub-protocols in an abstract way. *Extendibility* means that the design of SCAPI ensure that any new implementations of primitives and sub-protocols that are even more efficient can be utilized in all existing protocols even if they were previously implemented. SCAPI can achieve also *efficient* property by supporting of highly efficient low-level libraries using JNI. Finally, because of focusing on keeping SCAPI easy to build and use, it has property of *"ease of use"*.

Libscapi is developed by *Bar Ilan University Cryptography Research Group* that try to promote investigate in Academy and Industry practitioners by providing e.g., high performance implementation on standard Linux and using modern techniques like Pipelining and TCP optimization and providing a common platform for bench-

[2]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/
[3]https://www.cadence.com/content/cadence/
[4]http://freelibrary.faraday-tech.com
[5]https://github.com/cryptobiu/libscapi

marking different algorithms and implementation.

1.4.4 TASTY

Tool for Automating Secure Two-partY computations (TASTY) [32] is used for describing, generating and compiling efficient secure two-party computation protocols that can generate protocols based on Homomorphic Encryption (HE) [71] and Garbled Circuit (GC) as well as combinations of both. HE can perform efficient addition and multiplication functions, whereas GC is better for non-linear functionality such as comparison function. By combining both HE and GC, it can provide optimizations for practical secure two-party computation with low latency and allows to automatically generate efficient secure protocols for many privacy-preserving applications, e.g., face recognition and remote diagnostics.

1.4.5 ABY

ABY[6] is a framework for efficient mixed-protocol secure two-party computation based on Arithmetic sharing, Boolean sharing, and Yao's garbled circuits. It can also provide highly efficient computation based on pre-computed *oblivious transfer extensions*[7] and combine protocols.

1.5 Related Work.

Several approaches have been considered in the last few years [55, 33, 41, 69, 38], for improving communication running time in two-party secure computation using garbled circuit (GC) protocol.

The proposed approach in 2009 by Benny Pinkas and Thomas Schneider [55], presents a number of optimisations which reduce the effective size of the circuit and size of the garbled table (GT) by 50 percent, which can result in improving the communication cost for transmitting the circuit between the parties. They modeled the underlying key derivations (KDFs) as correlation robust using the *Free-XOR* technique [37] (§1.3.1), which in this situation they are able to reduce the data needed

[6]https://github.com/encryptogroup/ABY
[7]https://github.com/encryptogroup/OTExtension

to be sent for the other gates by 25 percent. Indeed, using the *free*-XOR gates result in *3/4(1-p)N* amount of data needed to be sent per circuit gate, where N denotes the amount of data needed to be sent for circuit in the Yao construction and P illustrates the proportion of XOR gates within a circuit. This approach provided the first secure two-party evaluation of the Advanced Encryption Standard (AES) circuit using Fairplay compiler [43] (§1.4.1), considered as a highly complex (around 30,000 AND and XOR gates) function (also with some potential applications), taking around 20 minutes to compute and requiring 160 circuits to obtain a 2^{-40} cheating probability.

Another method with the aim of improving efficiency and scalability of garbled circuit was proposed in 2011 [33] that provides a flexible framework for faster secure two-party computation. In this approach, it is not necessary to generate and store the entire garbled circuit in memory and allows users to build and evaluate the circuit modularly. Therefore, very complex circuits can be evaluated, generated and debugged. Also, they could get improve in efficiency and scalability by pipelining the process of circuit generation and evaluation. Thus, this simple pipelining approach result in minimizing circuit size and scalability of the garbled circuit, has led to the development of several complex protocols that make evaluation faster than previous work. E.g., the total number of non-free XOR gates for the entire AES-128 computation was 9280 and the overall time was 0.2 seconds that was 16 times faster than the best previous results.

In 2012, Kreuter et al.[38] focused on parallel implementations design based on garbled circuits to be run on CPUs with many cores. Their proposed circuit consisting of about 6 billion gates; and they could implement running this on 512 cores of powerful cluster computer. They used better optimized circuit compiler than Fairplay and previous method [55]; with several new optimizations such as pipelining, which was suitable for more complicated circuits to reduce the computation time. The time to compute AES was reduced to 1.4 second per block, because of beginning transmission of the GC across the network, while the rest of the circuit is still being generated.

Another research group in 2013 investigated on using consumer-grade *Graphics processing unit* (GPUs) to achieve similar levels of parallelism [28]. They used OT extension techniques [34, 3] to design their GPU-specific protocol and achieve comparable efficiency to the cluster computing implementation, which results in reducing the number of gates in implementation of the AES to 50,000.

The TinyGarble method is another method, introduced by Ebrahim M.Songhori and Ahmad-Reza Sadeghi et al.[69] in 2015. TinyGarble[8] illustrates a high degree of compactness and scalability while using a sequential circuit description for garbled circuit (GC) protocol. It can compress the memory footprint of garbling operation while resulting in fewer cache misses and less CPU cycles. TinyGarble introduced new techniques based on a sequential circuit for synthesizing and optimizing of garbled circuit (GC) protocol, which result in minimizing the number of non-XOR gates and improving computation and communication time. The most significant advantage of TinyGarble is describing the function in a compact format as a sequential logic instead of combinational format (i.e., user can compress the 1024-bit addition function into only a 1-bit adder).Also, the total number of gates using this method for the entire AES-128 computation was 2588 and number of non-XOR gates was 576, which is the best result until now. We can say that TinyGarble introduces the concept of sequential circuits that consist of circuits and loops, where at execution time the loops are unrolled resulting in a combinational circuit that is evaluated with Yao's garbled circuit protocol.

[8]https://github.com/esonghori/TinyGarble

Chapter 2

Introduction to Quantum Gates

"When we get to the very, very small world (say circuits of seven atoms) we have a lot of new things that would happen that represent completely new opportunities for design. Atoms on a small scale behave like nothing on a large scale, for they satisfy the laws of quantum mechanics. So, as we go down and fiddle around with the atoms down there, we are working with different laws, and we can expect to do different things. We can manufacture in different ways. We can use, not just circuits, but some system involving the quantized energy levels, or the interactions of quantized spins." – Richard P. Feynman

This chapter consists of four sections. In the first section, we will discuss about some of the key aspects of quantum mechanics needed for quantum computation (§2.1). This section introduces the basic definitions and notations of quantum mechanics and their characteristics like quantum states and quantum bit (see §2.1.1 and §2.1.2). We will also give some feeling about mathematical formalisms needed to work with quantum computation. Quantum key distribution will be describe in §2.2 and section §2.3 gives the background on reversible and quantum circuits required for this dissertation. Following these sections, we have explained the simulation tool for the design of the quantum circuit in §2.4.

2.1 Quantum Mechanics

This section discusses how quantum mechanics can be used to perform computations and how these computations make a quantum computer different from a conventional computer. For more information about quantum mechanics, see [13, 23]. In the early 1980's [26], Richard Feynman observed that computation, in general, could be done more efficiently if we use quantum mechanical effects. Considering on basic principles of quantum mechanics help us to explain where the power of quantum computer comes from.

In quantum systems, because of the exponential increase in computational space and the number of processors, and hence an exponential increase in the amount of physical space needed, we can have an exponential decrease in the time required for computation, programming, and complexity. Therefore, in quantum systems, we can have an exponential increase in parallelism with the size of the system [16]. According to this characteristic, quantum computing is separated from conventional computing.

2.1.1 Quantum State

In quantum computing, Information is encoded in the form of quantum states that are described in terms of vectors or in the more compact *bra/ket* notation $|\rangle$ invented by Dirac [17].

2.1.2 Quantum Bit (Qubit)

In classical computing, bit is the basic element used to store information, which can be in one of the two states of 0 or 1. In quantum computing, this basic element is called a *quantum bit (qubit).* Qubit is denoted by Dirac notation, which is a unit vector in a two-dimensional complex vector space that indicated by $\{|0\rangle,|1\rangle\}$. $|0\rangle$ and $|1\rangle$ could correspond to the spin-up and spin-down states of an electron. In contrast to a classical bit, a qubit is able to be in two quantum states containing 0, 1 at the same time (often called *superposition*). The quantum state of a qubit as a superposition of the two quantum basis states $|0\rangle$ and $|1\rangle$ is shown in Equation 2.1.

$$|\psi\rangle = \alpha\ |0\rangle + \beta\ |1\rangle \qquad (2.1)$$

α and β are complex numbers such that $|\alpha|^2 + |\beta|^2 = 1$. Superposition state is measured according to the basic $\{\ |0\rangle,|1\rangle\}$. The value $|0\rangle$ is measured with a probability of $|\alpha|^2$ and the result $|1\rangle$ is obtained with a probability of $|\beta|^2$. As shown

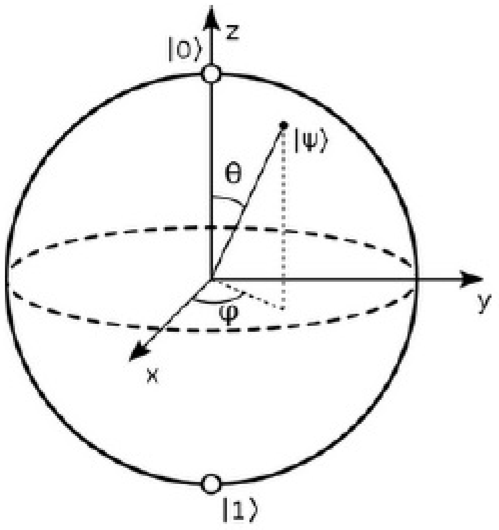


Figure 2.1: The qubit state as a Bloch sphere presentation [62].

in Equation 2.2, each unit vector illustrates the state of qubit in a two-dimensional complex vector space [49, 62].

$$|\psi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \quad , \quad |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad , \quad |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2.2}$$

The state of a quantum bit (qubit) can also be a point on the surface of a sphere called Bloch sphere [49] as shown in Figure 2.1 and can also be written as shown in Equation 2.3.

$$|\psi\rangle = \cos\tfrac{\theta}{2}\ |0\rangle + e^{i\varphi}\sin\tfrac{\theta}{2}|1\rangle \tag{2.3}$$

2.1.3 Quantum Key Distribution

In 1984, Bennet and Brassard [9, 8] considered the first quantum key distribution process, in which private keys on insecure channels can be transferred by sequences of single qubits.

Supposing that Alice and Bob want to agree on a secret key and communicate with each other privately. So, they are connected by the open classical channel and a uni-directional quantum channel that can be accessible by Eve, who wants to listen to their conversation. The process is illustrated Figure 2.2.

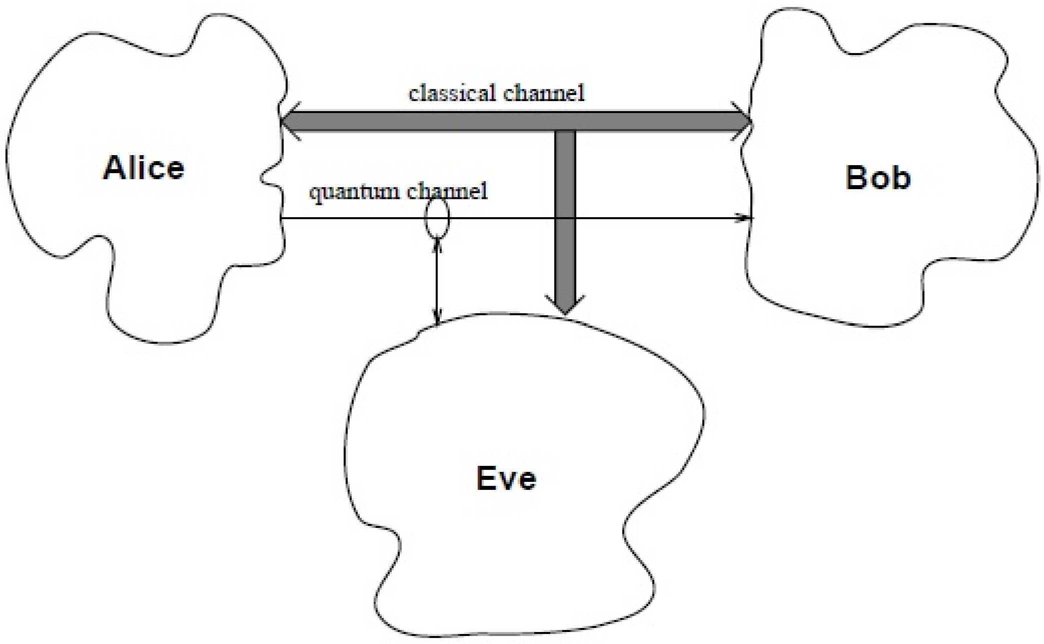


Figure 2.2: Quantum key distribution process [58].

Alice, through the quantum channel, sends photons to Bob, who can measure the quantum state. Eve also tries to measure the state of these photons and resend them to Bob. To establish a secret key, Alice sends a sequence of bits to Bob and encodes each bit in the quantum state of a photon that can be measured by Bob. Alice and Bob can identify those bits, which they have agreed for sending, receiving and using these bits as the key. The listener (Eve) can measure the state of this transmitted photon and resend new photons to Bob. Thus, Eve can use the wrong basis and resend the bit with this wrong basis. So, any listener on the quantum channel can use it for introducing a high error rate that Alice and Bob could detect by communicating their keys through the channel. Other proposed techniques for considering the quantum effects for key distribution have been introduced in [8, 22, 42].

2.2 Quantum Logic

One import fact is that quantum transformations are unitary and quantum gates inherently reversible. So, In this section at first, we have discussed about the reversibility property in designing quantum logics and quantum circuits. Finally, some common quantum(reversible) gates that are used in this thesis are introduced.

2.2.1 Reversible Computing

In 1973 Bennet [7] showed that energy dissipation problem of VLSI circuits can be circumvented by using reversible logic.

Quantum logic gates are inherently reversible and quantum circuits are built based on reversible logic circuits. A reversible logic circuit is realized by a cascade of reversible gates. A gate that implements one to one mapping between n inputs and n outputs are called a $n{\times}n$ reversible logic gate that can be represented as shown in Equation (2.4), where $\mathrm{I_v}$ and $\mathrm{O_v}$ are the input and output vector.

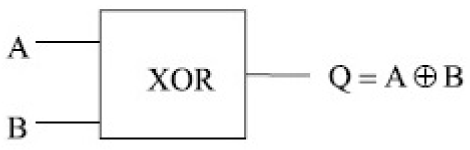
$$\mathrm{I_v} = (x_1,x_2,\ x_3,....x_\mathrm{n}) \quad , \quad \mathrm{O_v} = (y_1,y_2,\ y_3,....y_\mathrm{n}) \tag{2.4}$$

The reversible logic gate must have the same number of inputs and outputs, and for each input pattern there must be a unique output pattern [7]. Reversible logic circuits avoid energy loss by "un computing" the computed information using recycling the energy in the system [7].

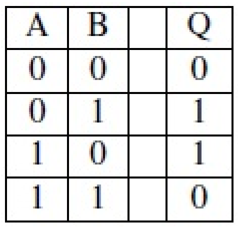
Synthesis of the quantum or reversible logic circuits in compared to the synthesis of the traditional irreversible logic circuits has two restrictions that should be mentioned [29, 54]:

- In the reversible logic, the fan-out of each signal is equal to one.
- Feedback from gate outputs to inputs is not permitted.

A completely or incompletely-specified irreversible function can be embedded into a reversible function by adding extra inputs/outputs. The extra input bits (qubits) are called **constant inputs** and the extra output bits (qubits) are called **garbage outputs.** As an example, consider conventional irreversible XOR gate and its truth table shown in Figure 2.3, and reversible XOR gate and its truth table (P as a garbage output) shown in Figure 2.4.



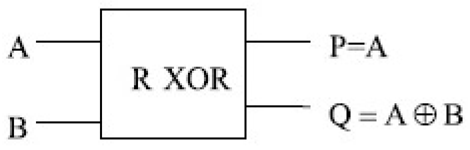
(a) Conventional XOR gate.



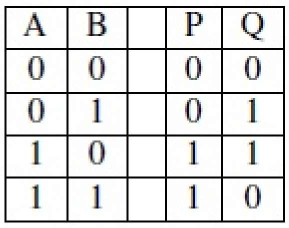
A	B		Q
0	0		0
0	1		1
1	0		1
1	1		0

(b) XOR gate.

Figure 2.3: Conventional XOR gates (gate and truth table).



(a) Reversible XOR gate.



A	B		P	Q
0	0		0	0
0	1		0	1
1	0		1	1
1	1		1	0

(b) R-XOR gate.

Figure 2.4: Reversible XOR gates (gate and truth table).

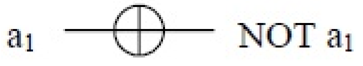


Figure 2.5: Quantum NOT gate.

2.2.2 Quantum Gates

Classical circuits are composed of logic "*gates*" connected by wires, information is transmitted through these wires as electric power. Consequently, quantum circuits consist of a set of "*quantum gates*" to inputs and kept qubits as an information. Quantum gates are also reversible because quantum transformations should be unitary. We here introduce some quantum(reversible) gates that are used in this work.

- **NOT Gate**: A 1 * 1 quantum NOT gate is similar to the conventional NOT gate. It has one input and one output and inverts the input values. As shown in Figure 2.5, circuit representation of this gate in the middle is symbolised by the $\oplus$ sign.

- **CNOT Gate**: A 2 * 2 quantum gate, which is closely related to the NOT gate, is the two-bit Controlled-NOT (or *C-NOT*) gate or Feynman gate [27]. The C-NOT gate performs a NOT on the second input if and only if the first input

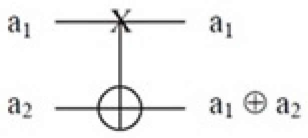


Figure 2.6: Quantum XOR gate.

is 1. Note that the CNOT gate, also called XOR gate, performs an exclusive OR operation between the two input bits, as depicted in Figure 2.6.

- **Toffoli Gate**: A 3 * 3 Toffoli quantum gate [73] is shown in Figure 2.7. It is also denoted as controlled-controlled-not (or *CC-NOT)* gate, since the third output is the inverse of the third input if and only if the first two inputs are equal to 1. A Toffoli gate is universal, i.e., for any reversible Boolean function *f* there exists a circuit containing only Toffoli gates that represents *f*.

  Toffoli gates are important blocks in quantum circuits. In fact, a large number of reversible logic synthesis methods use $N \times N$ Toffoli gates as TOFn (x_1,x_2, x_3,....x_n) where x_n is the target line and the first *N-l* lines are controls [45]. The main reason for popularity of Toffoli gates over the other gates is their completeness and relativeness in using them. TOF1(x_1) is the special case where there are no control inputs, so x_1 is always an inverter, i.e., It is a NOT gate. TOF2(x_1,x_2) has been termed a Feynman or controlled NOT gate (CNOT). TOF3(x_1,x_2,x_3) is often referred to simply as a Toffoli gate.

  A *quantum gate library* contains a functionally complete set of reversible gates. A set containing NOT and CNOT gates are not functionally complete, i.e., there exist functions that cannot be computed by these gates. Adding Toffoli gates to the previous set we get the NCT library, which is one of the most common quantum libraries. Since Toffoli gate is universal, we can represent any reversible Boolean function through the NCT library.

- **Peres Gate**: A 3 * 3 Peres quantum gate Peres gate (PG) [53], also known as New Toffoli Gate (NTG) combining Toffoli gate and Feynman gate. Since the Peres gate (PG) quantum cost [1]is less than Toffoli gate (TG) the Peres gate is

[1]Quantum Cost (QC):The number of (1*1) or (2*2) reversible gates used in the circuit.

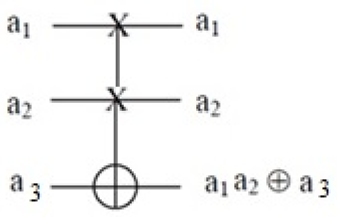


Figure 2.7: Quantum Toffoli gate.

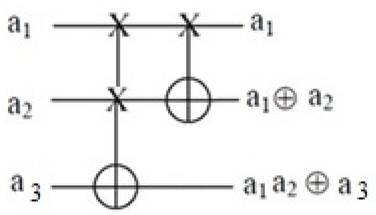


Figure 2.8: Quantum Peres gate.

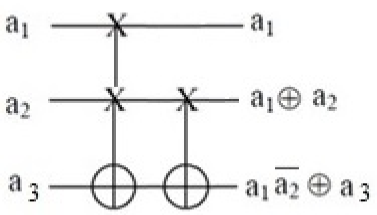


Figure 2.9: Quantum TR gate.

used every- where instead of the Toffoli gate [45]. This gate is shown in Figure 2.8.

- **TR Gate**: The TR gate [72] is the inverse of the Peres gate and its operation on the qubits and its quantum implementation are illustrated in Figure 2.9.

2.3 RevKit: Tool for the Design of Quantum Circuits

The open source toolkit RevKit[2] [66] is a tool for synthesis, optimization, simulation, verification and testing of quantum (and reversible) circuits [76]. Besides it provides elaborated methods for synthesis, optimization, and verification of basic functionality like parsers and export functions. One important property of this tool

[2]http://revkit.org/

is its extensibility, meaning that it can be re-implemented in order to modify or improve the given algorithms. So, RevKit is a extendable framework with significant number of approaches and algorithms, but can easily provide the addition of new methods and algorithms. In this thesis, we used RevKit for minimizing the Boolean functions in the quantum framework, and for counting the number of XOR and non-XOR gates in the obtained quantum circuits.

Example: For instance, considering the quantum implementation of the simple function *f=(a+b)⊕c*, in RevKit that implement this function as following descriptions :

```
revkit> expr [{ab}c]
revkit> convert −expr_to_spec
revkit> exs
revkit> ps –c
```

then we can have these results for number of gates and number of qubits:
Lines: 4 *(1 line is belongs to constant input and 3 lines for inputs)*
Gates: 4
Logic qubits: 4

We can write simple expressions in RevKit using **expr**[3] command. The expressions only support binary operators. It should be considered that for binary AND we use ( ), for XOR we use [ ], for OR we use { }, for NOT we use!.Therefore, we have [{ab}c] for defining *f= (a+b)⊕c* function. Afterwards, we can call several synthesis algorithms[4], e.g., **exs**, tbs, rms, and dbs. Transforming expression to quantum specification (expr > spec) is an alias[5]. So, we need to set **convert −expr_to_spec** to use alias. Using **ps** gives us statistical information about the function. Then we can convert the expression into a reversible circuit using (it will automatically embed). Also, we can see the following circuit in output (see Figure 2.10).

In this simple example we have that the quantum implementation of the given

[3]https://msoeken.github.io/cirkit_doc.html#datastructuresexpressions
[4]https://msoeken.github.io/cirkit_doc.html
[5]https://msoeken.github.io/cirkit_doc.html#getting-started-aliases

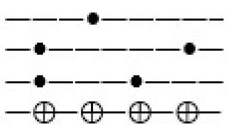


Figure 2.10: Quantum Implementation of function *f=(a+b)*$\oplus c$ using RevKit

Commands	**Description**
expr	Expressions for defining Boolean functions
Spec	Quantum specification
ps	Print statistical information about the function
convert	Convert each data structure to another ones

Table 2.1: RevKit Commands

function has 3 CNOT (XOR) gates and 1 Toffoli gate. Table 2.1 presents some typical RevKit commands with their descriptions.

Chapter 3

Boolean Function Representation

Boolean Function representation in electronic design automation (EDA) tools is the main way to design efficient hardware. Logic representation, on the one hand, try to have the fewest number of primitive elements (literals, nodes, etc.) in order to implement with a small memory footprint. On the other hand, logic presentation must be simple enough to run and execute. We use standard representations for Boolean functions which are particularly useful for SMC protocols. In this chapter, we focus on Boolean function representation as directed acyclic graphs (DAGs) and binary decision diagrams (BDDs) and we will also explain Multiple-Valued logic (MV).

Boolean Function Synthesis. Any Boolean function can be represented using an expression containing AND, OR and NOT operations. Boolean functions can also take in multiple Boolean variables. *"0'* represents *"false"* and *"1'* represents *"True"*. Also, it can be represented using an expression cntaining only NAND operations.

Boolean Function Representation. Boolean functions can be represented and manipulated in many ways. We are going to see logic function representation using truth table (TT) method and Boolean expression forms of DNF and CNF below.

3.1 Truth Tables

The first definition of truth table (TT) was introduced by *Ludwig Wittgenstein* and published in his *Tractatus Logico-Philosophicus (TLP)* book. A truth table (TT) is a mathematical table prepared from the specific Boolean function, which sets the functional values as Boolean expressions. Consider an n-variable function:

$$f(x_1,\ldots,x_n), \qquad (x_i=0,1, \quad i=1,\ldots,n)$$

Truth table (TT) consists of one column for each input variables (*minterms*) and one column showing possible results of operation for those values. Each of the 2^n possible combinations of the n variables called *minterms*, and a function can be represented by all the minterms that appear their evaluation as 1.

3.2 DNF and CNF forms

Every Boolean function can be written as a Boolean expression forms of Disjunctive Normal Form (DNF) and Conjunctive Normal Form (CNF).

Let a term be the disjunction ("or") of a collection of variables. A Boolean expression is in disjunctive normal form (DNF) if the variables within each term are ANDed together, and the terms are ORed together. We can say that these Boolean expressions are in *sum-of-product* (SOP) form, meaning that they are the sum (OR) of a set of products (AND). (SOP) form [44] is one of the first representation form for logic functions that was taken by PLA technology [59].

Another two-level Boolean Function synthesis have been proposed as Conjunctive Normal Form (CNF). A Boolean expression is in CNF if the variables within each term are ORed together, and the terms are ANDed together. A CNF expression is *satisfiable* if there exists an assignment of variables for which the expression is true. For example, *(a or b) and (not c or d)* is true if *a and d* are true. CNF form is defined also as *product-of-sum* (POS) or EX-SOP [68]. POS form is exactly opposite to the SOP form, meaning that all of the variables are ORed and all of these sum terms are ANDed. In these two-level logic representation, reducing the number of *"products"* in SOP or *"sum"* in POS result in logic optimization. An exact optimization tool for these two-level Boolean logic present by ESPRESSO tool. These Boolean logic syntheses are suitable for simple and small sized functions, thus for the more complicated function, we can move to DAG logic representation or (multilevel logic) and BDDs [39].

3.3 Binary Decision Diagrams (BDDs)

A binary decision diagram (BDD) is a rooted directed acyclic graph (DAG) [13] as a multilevel logic synthesis. The directed acyclic graph does not contain cycle, and every directed edge of the graph (wires) connects nodes, which correspond to logic functions (gates). In a DAG, a node is a starting node if it has no ingoing edges, and a node is a terminal node if it has no outgoing edges. Since optimization process runs on the entire DAG, it cannot be easy without having bounds on their nodes' functionality. Moreover, memory footprint for each node should be large because of increase in the representation size. Thus, a more practical and simple construct of DAGs is considered as binary decision diagrams (BDDs) [39] where all diagrams' nodes realize the same function.

Graphical representation for function as binary decision diagrams (BDDs) was introduced at first by Lee [39] in 1959 and it was refined and studied more in other articles [15, 1, 18]. It can also be used to present circuits. A binary decision diagram (BDD) is a directed acyclic graph (DAG) [13] with the following properties:

- All the nodes can be terminal or non-terminal.
- All the terminal nodes are labeled with 0 or 1 and have no outgoing edges.
- All the non-terminal nodes are labeled with a Boolean variable.
- All the non-terminal nodes have two outgoing edges labeled with a 0 and a 1 (one drawn with a dashed line and one drawn with a solid line.)

We can simplify these steps to the following BDD where the function is $f=(A \oplus B)C \oplus AB$, and (A,B,C) are all the variables in the function. (see Figure 3.1)

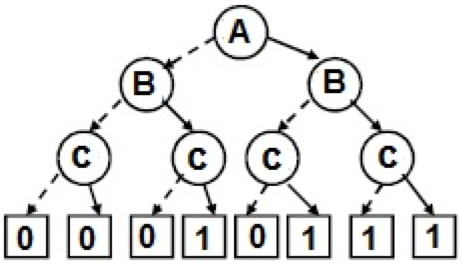


Figure 3.1: Binary decision diagram (BDD) of the function *f=(A⊕B)C⊕AB*.

The size of BDD is defined as the number of variables or (non-terminal nodes), which act as 2:1 multiplexers. It can be used to improve efficiency in various areas of EDA such as verification, testing, optimization, automated reasoning, etc. [44]. Because of increasing price according to BDD size, it can be supported by some optimization algorithms [77]. A BDD that has an ordering variables (x_1,x_2, x_3,....x_n) is *ordered* BDD (OBDD), and BDD with removed shares and redundant nodes is *reduced* BDD (RBDD). A BDD with both reduced and ordered version is called a *reduced ordered* BDD (ROBDD).

3.3.1 OBDD

An ordered BDD (OBDD) is a BDD that has an ordering variable (x_1,x_2, x_3,....x_n). The BDD has the ordering (x_1,x_2, x_3,....x_n) if:

- All the labels of BDD are in (x_1,x_2, x_3,....x_n).
- For any x_i followed by x_j in a path of BDD graph, we have i < j in the variable ordering.
- The OBDD (with variables (x_1,x_2, x_3,....x_n) has n+1 levels: l_1, ..., l_n, l_{n+1} such that:
  - l_{n+1} is the level containing the terminal nodes (0 and 1).
  - l_i (1<i<n) is the level containing variable x_i.

It is to be noted that two OBDDs, which describes the same function but have different orderings, can be significantly different.

3.3.2 ROBDD

Reduced ordered BDD has the following properties:

- Removal of redundant nodes (Nodes that have outgoing edges pointing to the same node can be removed).

- ROBDDs have at most two terminal nodes, labeled with 0 and 1.

The ROBDD (reduced ordered BDD) of function f with ordering O is unique. In other words, if B and B' be two ROBDDs with the same ordering (x_1,x_2, x_3,....x_{n}), If B and B' represent the same function, then B and B' have the same structure.

3.3.3 CUDD: Tool to Build BDD Function Representation

CUDD[1] (Colorado University Decision Diagram) package [67], written by Fabio Somenzi's research group, is one of the most complete packages with a C/C++ library for creating binary decision diagrams (BDDs), zero-suppressed BDDs (ZDD) [47] and algebraic decision diagrams (ADD) [4]. It has a superabundance of BDD functionality with the aim of providing an efficient and best-maintained BDD package in terms of memory and computation that is freely available.
For implementing a BDDs package, we need to define :

- Nodes;

- An item that contains the overal BDD information;

- Basic algorithms;

Node is the basic item of a BDD. In CUDD, a node is represented by the struct ***DdNode***, which contains all the information necessary to correctly manipulate BDDs like variables, number of variables, number of nodes, unique tables and etc. It can also contain ***Index*** which can refer to index of variable in which the node belongs to ***Ref*** for the number of references. When we create BDDs, we have to reference it to compute BDDs.

The ***DdManager*** is the central data structure of CUDD that must be created before calling any CUDD function and it needs to be passed to every CUDD function.

[1]http://www.ecs.umass.edu/ece/labs/vlsicad/ece667/links/bdd.html

As an example, BDD implementation of Half-Adder in CUDD [67] is shown in below:

```
DdNode*   x1 = Cudd-bddIthVar (manager, 0);
DdNode*   x2 = Cudd-bddIthVar (manager, 1);
DdNode*   and1;
and1 = Cudd-bddAnd (manager, x1, Cudd-Not(x2));
Cudd-Ref (and1);
DdNode*   and2;
and2 = Cudd-bddAnd (manager, Cudd-Not(x1), x2);
Cudd-Ref (and2);
DdNode*   sum;
sum = Cudd-bddOr (manager, and1, and2);
Cudd-Ref (sum);
Cudd-RecursiveDeref (manager, and1);
Cudd-RecursiveDeref (manager, and2);
```

3.4 Multiple-Valued Logic (MV)

In the field of circuit design, *Multiple Valued Logic (MVL)* [10, 11, 20, 46, 57, 70, 75] is a direct generalization of the standard Boolean logic, where the classical Boolean domain $B = \{0,1\}$ is replaced by $P = \{0,1,\ldots,|P|-1\}$ with $|P|>1$.

Multiple valued networks are in general more compact in area than the corresponding Boolean circuits [12, 36].

Let P_i be the finite subset of natural numbers $P_i = \{0,1,\ldots,|P_i|-1\}$ with $|P_i|>1$. A multiple valued logic is a generalization of the classical Boolean logic, as described below:

Definition 1: *A multi-valued variable x_i is a variables that takes on values from P_i.*

Definition 2: *A multi-valued function f is a function such that:*

$$F : \{P_1, P_2, \ldots P_n\} \rightarrow P_f.$$

In particular, when $P_1 = P_2 = ... = P_n = P_f = P$ we have that:

$$F : P^n \rightarrow P.$$

Note that, in this case, there are P^{P^n} possible different MVL functions.

A MVL circuit is a circuit composed by MVL gates. MVL gates are a direct generalization of standard Boolean gates. Of course, the number of two-input gates grows exponentially with the dimension of P. Therefore, while the number of two-input Boolean gates is 16, the number of two-input MVL gates g : $P^2 \rightarrow P$ is P^{P^2} (e.g., 19,683 for $P = 3$).

The objective of multiple-valued logic optimization is to reduce the size of the algebraic expression (e.g., a SOP form) representing a MVL function. As in the Boolean domain, this algebraic expression optimization directly implies the minimization of the corresponding MVL circuit. Minimization techniques have been studied since the late nineteen-sixties, and a large variety of two-level (e.g., Espresso-MV [59]) and multi-level (e.g., MVSIS [30]) optimization tools have been proposed.

The main advantages of multiple-valued logic are reduction of wiring complexity and reduction of the number of interconnections required to implement logic functions [75]. The power dissipation also depends on interconnection complexity; thus, low power dissipation can be reached using MVL [24]. Designing of different combinational circuits is another gaining of using multiple-valued logic [19]. However, in circuit design, the good potential advantages of MVL are mitigated by the difficulty of MVL circuit realization. Fortunately, in our context we do not exploit a hardware MVL circuit, but only its algebraic description. Therefore, we can completely bypass the technology issues that are the typical practical problem in the CAD context.

Moreover, for high level design it is natural to think of multiple valued variables, rather than Boolean ones. During the design process, the problem described with multiple valued variables are then transformed in a Boolean problem. This phase, called *encoding*, is particularly critical and potentially onerous. The encoding is a hard problem, especially for large circuits since it is difficult to correlate an encoding decision with the final logic optimization of the circuit [12]. Therefore, generally,

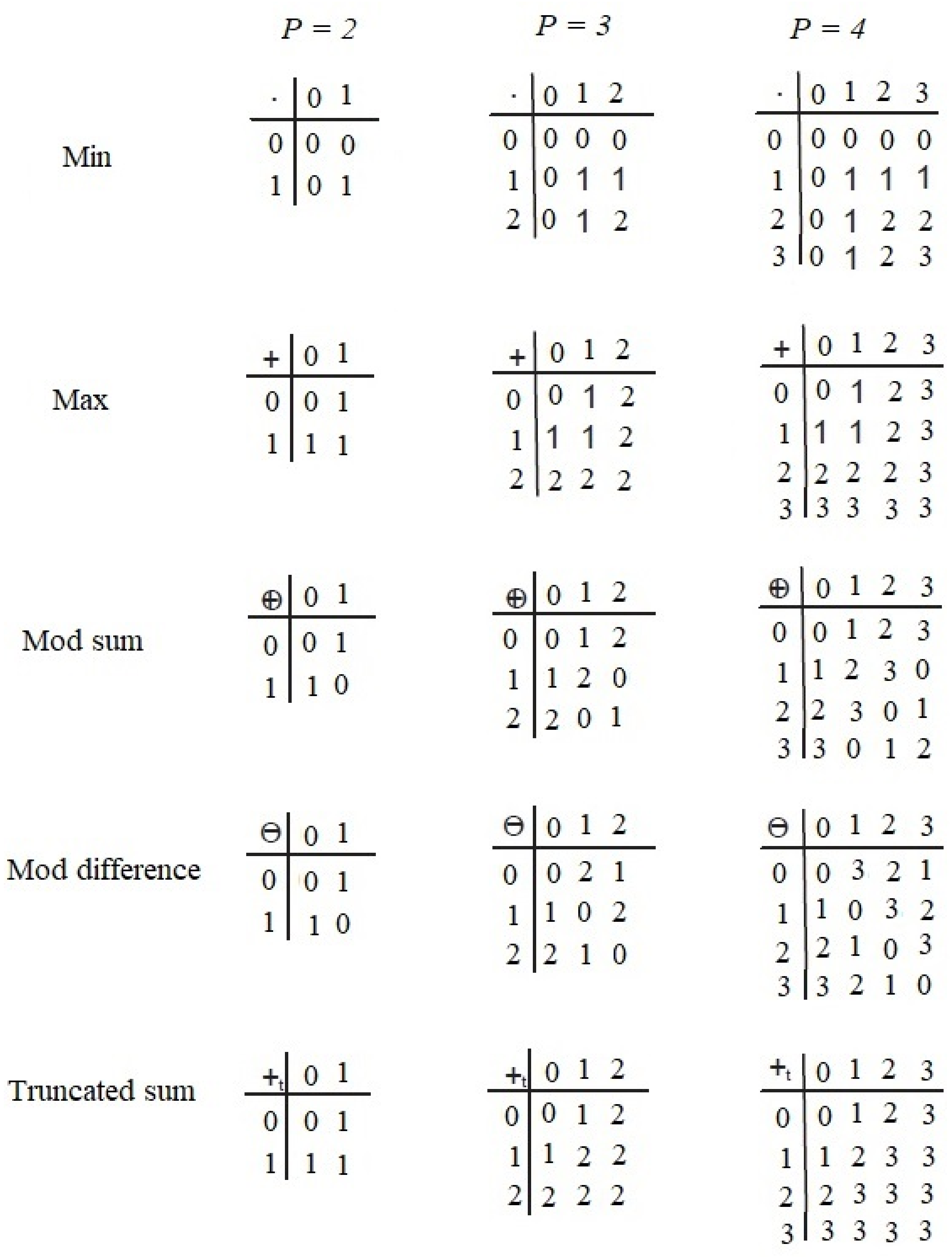
Name	**Operation**	**Deffinition**
Min	x AND y	if $x<y$ equal to x, otherwise equal to y
Max	x OR y	if $x>y$ equal to x, otherwise equal to y
Mod-sum	x XOR y	*(x+y)* mod p
Mod-difference	$x \ominus y$	*(x-y)* mod p
Truncated sum	$x + y$	*min(p* $-$ 1*, sum(x, y))*

Table 3.1: Two-place MVL operators.

encoding is done at the beginning without any specific criterion, and then Boolean logic synthesis is applied to an arbitrary encoded circuit [12]. This is an evident disadvantage for the optimization of the final circuit, due to the Boolean nature of the standard circuits. Clearly, the realization of a MVL circuit would directly solve the problem.

Binary MVL functions are known as two-place functions [46]. MVL two-place operators, *"min"*, *"max"*, *"mod-sum"*, *"runcated sum"* and *"mod-difference"*, are shown and defined in Table 3.1. In particular, the standard Boolean AND gate is the *minimum* gate (i.e., $x.y$ = min(x,y)), OR gate can be generalized to the *maximum* (i.e., $x + y$ = max(x,y)) or to the *truncated sum* gate (i.e., $x{+_t}y$=min($|P|$-1, sum(x, y))), the XOR gate can be generalized to the *mod-sum* (i.e., x⊕ y = (x+y) mod $|P|$) or to the *mod difference* (i.e., x ⊖ y = (x-y) mod $|P|$).

Two-place (Binary MVL) functions of *"min"* operator, *"max"* operator, *"mod-sum"* operator, *"Mod-difference"* operator and *"Truncated sum"* operator for P=2,3,4 (P is Boolean Domain) are illustrated In Figure 3.2. They can be extended also for P=5,6,.. values. We have 16 elements for P=4 and 32 for P=5. Thus, we have 2^P elements for p-values in Binary MVL functions.



Min

$P = 2$

$\cdot$	0	1
0	0	0
1	0	1

$P = 3$

$\cdot$	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

$P = 4$

$\cdot$	0	1	2	3
0	0	0	0	0
1	0	1	1	1
2	0	1	2	2
3	0	1	2	3

Max

$P = 2$

+	0	1
0	0	1
1	1	1

$P = 3$

+	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

$P = 4$

+	0	1	2	3
0	0	1	2	3
1	1	1	2	3
2	2	2	2	3
3	3	3	3	3

Mod sum

$P = 2$

$\oplus$	0	1
0	0	1
1	1	0

$P = 3$

$\oplus$	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

$P = 4$

$\oplus$	0	1	2	3
0	0	1	2	3
1	1	2	3	0
2	2	3	0	1
3	3	0	1	2

Mod difference

$P = 2$

$\ominus$	0	1
0	0	1
1	1	0

$P = 3$

$\ominus$	0	1	2
0	0	2	1
1	1	0	2
2	2	1	0

$P = 4$

$\ominus$	0	1	2	3
0	0	3	2	1
1	1	0	3	2
2	2	1	0	3
3	3	2	1	0

Truncated sum

$P = 2$

$+_t$	0	1
0	0	1
1	1	1

$P = 3$

$+_t$	0	1	2
0	0	1	2
1	1	2	2
2	2	2	2

$P = 4$

$+_t$	0	1	2	3
0	0	1	2	3
1	1	2	3	3
2	2	3	3	3
3	3	3	3	3

Figure 3.2: Truth tables for two-input Boolean and MVL operators.

Chapter 4

Secure Multi-Party Computation Exploiting Quantum Gates

Recall that our approach aims to reduce the number of non-XOR gates, which directly result in reducing the number of interactions between the parties and OT at runtime, reducing then the computation and communication costs. In this chapter, we consider the design of circuits using *quantum gates* [73, 53, 27] instead of traditional ones. The goal is to design a quantum garbled circuit (Q-GC) equivalent to the original Boolean circuit and computing the same function, so that it is possible to increase the number of XOR gates, and reduce at the same time the communication and computation overhead required by the evaluation of non-XOR gates, thus requiring less number of interactions.

For this reason, in this chapter we consider *standard garbling of circuits* in section §4.1 and *quantum implementation of the garbled circuit (GC)* in section §4.2. Then, we compare our results in the number of non-XOR gates in section §4.3.We show the design Q-GCs for the ***Millionaires' Problem***, which is the problem of determining which party has the greatest input, and for the ***32-bit adder***, that is the computation of the sum of the private inputs. In both cases, we compare the resulting circuits with the original Boolean circuits used in GC and show the improvements achieved in terms of reduced communication time.

4.1 Standard Garbling of Circuits

In this section, we consider two simple examples, one for the design of the GC computing the comparison function used in the solution of the Millionaires' Problem (see §4.1.1), and the other for the GC implementing a 32-bit adder (see §4.1.2). In the first case, we refer to the solution provided by Fairplay, while in the second case we start from the circuit returned by the CBFS-MPC framework.

4.1.1 Garbling the Comparison (Millionaires) Circuits

We report here the SFDL programs used to define 4-bit input for the Millionaires' problem and the corresponding results obtained in SHDL outputs (see §4.1.1.1). For the sake of brevity, we report also the results obtained in the 8-bit case (see §4.1.1.2).

4.1.1.1: Garbling the Comparison 4-bit (Millionaires) Circuits

We present the comparison of two 4-bit integers between AliceInput and BobInput. The **SFDL** programming of 4-bit Millionaires is shown in below.

```
program Millionaires {
typeint = Int 4; // 4-bit integer
typeAliceInput = int;
typeBobInput = int;
typeAliceOutput = Boolean;
typeBobOutput = Boolean;
type Output = struct {AliceOutputalice, BobOutput bob};
type Input = struct {AliceInputalice, BobInput bob};

function Output output(Input input) {
output.alice = (input.alice>input.bob);
output.bob = (input.bob>input.alice);}
}
```

And get the following **SHDL** results as outputs:

```
0 input //output$input.bob$0
1 input //output$input.bob$1
2 input //output$input.bob$2
3 input //output$input.bob$3
4 input //output$input.alice$0
5 input //output$input.alice$1
6 input //output$input.alice$2
7 input //output$input.alice$3
8 gate arity 2 table [1 0 0 0] inputs [4 5]
9 gate arity 2 table [0 1 1 0] inputs [4 5]
10 gate arity 2 table [0 1 0 0] inputs [8 6]
11 gate arity 2 table [1 0 0 1] inputs [8 6]
12 gate arity 2 table [1 0 0 1] inputs [10 7]
13 gate arity 2 table [0 0 0 1] inputs [4 0]
14 gate arity 3 table [0 0 0 1 0 1 1 1] inputs [13 9 1]
15 gate arity 3 table [0 0 0 1 0 1 1 1] inputs [14 11 2]
16 gate arity 2 table [0 1 1 0] inputs [12 3]
17 gate arity 2 table [0 1 1 0] inputs [15 16]
18 output gate arity 1 table [0 1] inputs [17] //output$output.alice$0
19 gate arity 2 table [1 0 0 0] inputs [0 1]
20 gate arity 2 table [0 1 1 0] inputs [0 1]
21 gate arity 2 table [0 1 0 0] inputs [19 2]
22 gate arity 2 table [1 0 0 1] inputs [19 2]
23 gate arity 2 table [1 0 0 1] inputs [21 3]
24 gate arity 2 table [0 0 0 1] inputs [0 4]
25 gate arity 3 table [0 0 0 1 0 1 1 1] inputs [24 20 5]
26 gate arity 3 table [0 0 0 1 0 1 1 1] inputs [25 22 6]
27 gate arity 2 table [0 1 1 0] inputs [23 7]
28 gate arity 2 table [0 1 1 0] inputs [26 27]
29 output gate arity 1 table [0 1] inputs [28]
```

From these results we can conclude that the number of **non-XOR** gates is equal to **10** (shown in bold in the previous list).

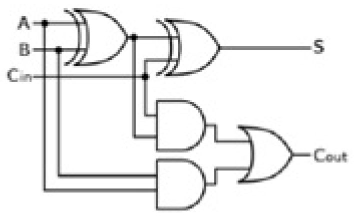


Figure 4.1: Classical adder.

4.1.1.2: Garbling the Comparison 8-bit (Millionaires) Circuits

In case of 8-bit input, the SFDL program is similar (except in the number of bits) but as output circuit we get a SHDL list containing **26 non-XOR** gates.

4.1.2 Garbling the adder circuits

Let us consider the case of the classic full adder, whose Boolean function is:

$$S = A \oplus [B \oplus C_{IN}]$$

where S is the sum, A and B are the summands and C_{IN} is the incoming carry. The adder is used as a basic block in many other problems. Here we assume that Alice randomizes the adder's inputs and encrypts the truth-table (as explained in the introduction) and the two parties wish to execute the addition over their private values A (held by Alice) and B (held by Bob) to obtain the public result A+B without disclosing their inputs to each other. Figure 4.1 shows the classical adder circuit. Applying the CBFS-MPC library (§1.4.2), the GC for a 32-bit adder [1] contains **127 AND** gates, **61 XOR** gates, and **187 INV** gates and the GC for a 64-bit adder [2] contains **265 AND** gates, **115 XOR** gates, and **379 INV** gates.

4.2 Quantum Implementation of Garbled Circuit (GC)

In this section, we show the quantum version of GC with the aim of reducing non-XOR gates, which directly results in reduced OT and computation and communi-

[1]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/adder_32bit.txt
[2]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/adder_64bit.txt

cation costs. In the section at first we define metrics to measure the efficiency of our models and our idea for generating quantum-garbled circuit, then in §4.2.1 we consider on the quantum garbled Boolean circuit (Q-GBC) for Millionaires' Problem in §4.2.2 and Q-GBC for 32-bit adder in §4.2.3.

4.2.1 The Quantum Metrics

We use the following metrics to measure the efficiency of our models and our idea for generating quantum-garbled circuit. The metric implies a partition on the set of gates:

- Quantum XOR gates (e.g., NOT and CNOT)
- Quantum non-XOR gates (e.g., CCNOT)

In particular, while quantum XOR gates are gates that contain only XORs, quantum non-XOR gates are the gates that contain at least one non-XOR.

As mentioned before, a quantum CNOT gate consists of just 1 quantum XOR gate, and the quantum Toffoli or (CCNOT) gate contains just 1 non-XOR (AND). We also consider NOT gate as a XOR gate, since NOT A is equivalent to A XOR 1. In the proposed metric, the cost of a quantum non-XOR gates is the number of non-XORs that it contains. In particular, the CCNOT gate has a cost of 1. Any quantum XOR gate has no cost in the metric.

4.2.2 Q-GBC for the Millionaires' Problem

In the Millionaires' problem, previously described (§ 1.1.4), two millionaires want to securely compare their private input values (money) without revealing more information than output result to the other party. This function is indeed a comparison function [A>B].

The comparison circuit is a synthetic circuit that compares two n-bit numbers and determines larger than, equal to, or less than the other. Note that comparator [A>B] can ba based on subtraction algorithm [51, 61]. Subtractor performs subtraction of minuend A, subtrahend B, and borrow bit that generates difference bit and carry out in output. The carry and difference obtained is compared with to get data is greater or smaller. Thus, the comparison circuit for two N-bit value is a chain of N-bit subtractors.

4.2.2.1: 4-bit quantum comparator circuit

We use the quantum comparator circuit proposed in [61]. As shown in Figure 4.2, the comparator is designed exploiting quantum TR gate [72], where a pair of TR gates is needed for constructing a full subtractor.

A subtractor computes the difference bits and carry-out bits. Q0-Q3 indicate different bits and R0-R3 indicate the carry-out bits. If Cout (carry) is “1”, then A < B. If Cout does not occur and the difference is zero, then A = B. If Cout does not occur, but the difference is not zero then A > B.

Quantum implementation of TR gate in RevKit is shown in below:

```
revkit> expr (a!b)
revkit> convert −expr_to_spec
revkit> exs
runtime: 0.04 secs

revkit > ps -c
Lines: 3
Gates: 2
Logic qubits: 4
revkit > print -c
```

Then we get these results for each TR gate as shown in Figure 4.3.

According to RevKit result, we can say that every TR gate has 1 CNOT (XOR) gate and 1 Toffoli (non-XOR) gate. Thus, we can conclude that for the implementation of a 4-bit comparison circuit, we need **8 non-XOR** gates.

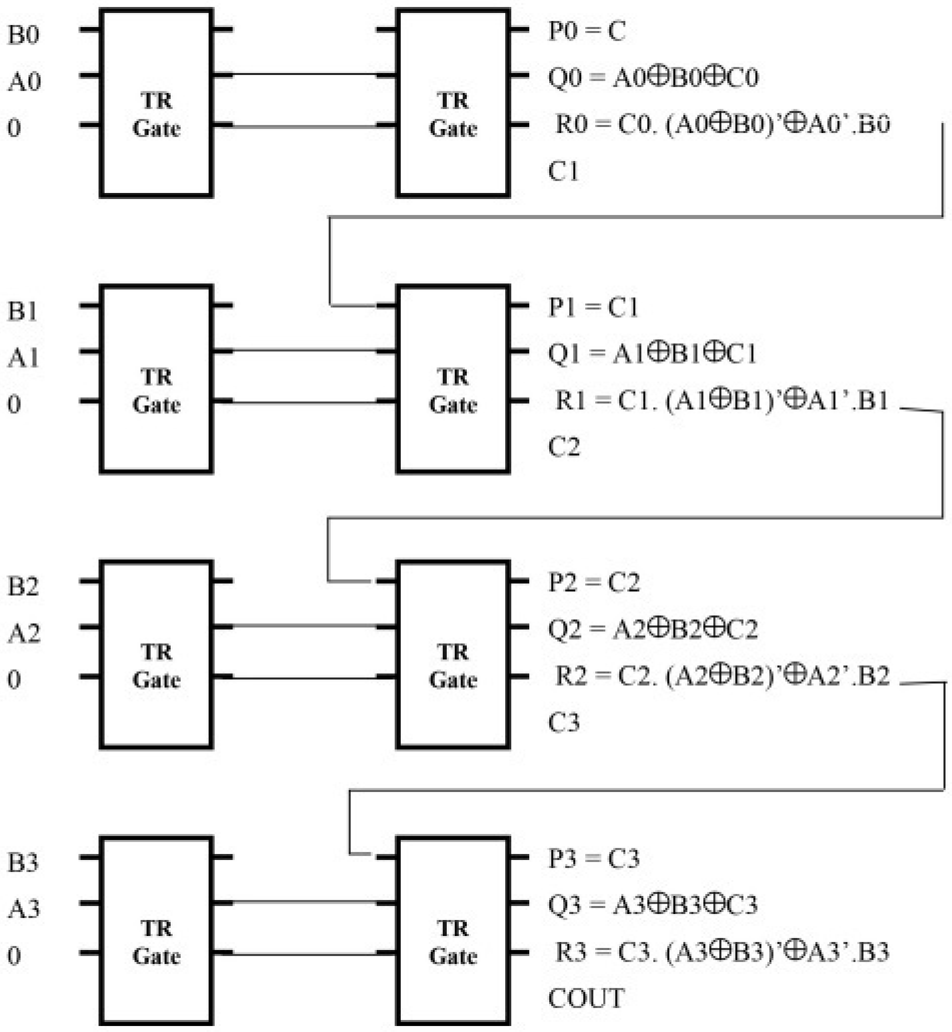


Figure 4.2: Quantum comparator [61].

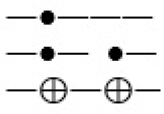


Figure 4.3: Quantum implementation of TR gate using RevKit.

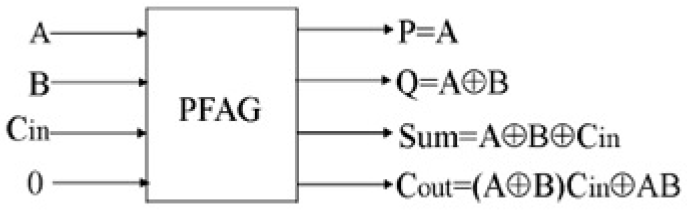


Figure 4.4: Quantum Peres Full Adder gate (for 1-bit adder).

4.2.2.2: 8-bit quantum comparator circuit

To implement 8-bit quantum comparator circuit, we can use the same circuit described in section §4.2.2.1 (Figure 4.2) and extend it to a 8-bit comparator circuit. Therefore, the 8-bit Millionaires' problem solved with quantum gates has 16 TR gates and **16 non-XOR** gates.

4.2.3 Q-GBC for the 32-bit adder

We use Peres Full Adder gate (PFAG) as a quantum full adder [35] shown in Figure 4.4 The PFAG can be implemented by two quantum Peres gate (PG) [53] as shown in Figure 4.5. Also, the quantum Implementation of PG in RevKit is shown in below and Figure 4.6:

```
revkit> expr [(ab)c]
revkit> convert −expr_to_spec
revkit> exs
run-time: 0.04 secs
revkit> ps -c
Lines: 4
Gates: 2
T-depth: 3
T-count: 7
Logic qubits: 4
revkit> print -c
```

Then, according to RevKit result, we can say that every PG has 1 CNOT (XOR) gate and 1 Toffoli (non-XOR) gate. Moreover, a 32-bit adder is a chain of 32 PFAG,

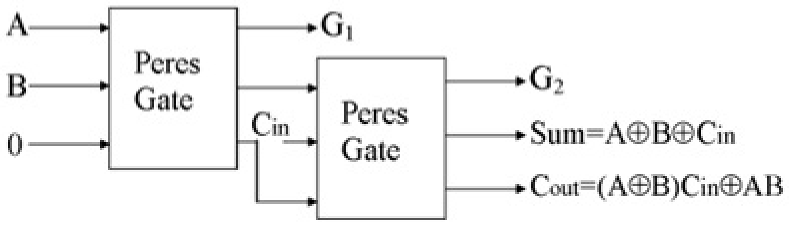


Figure 4.5: Implementation of PFAG using Peres gates (for 1-bit adder).

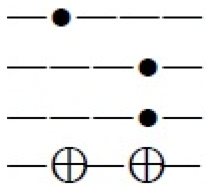


Figure 4.6: Quantum Implementation of PR gate using RevKit (for 1-bit adder).

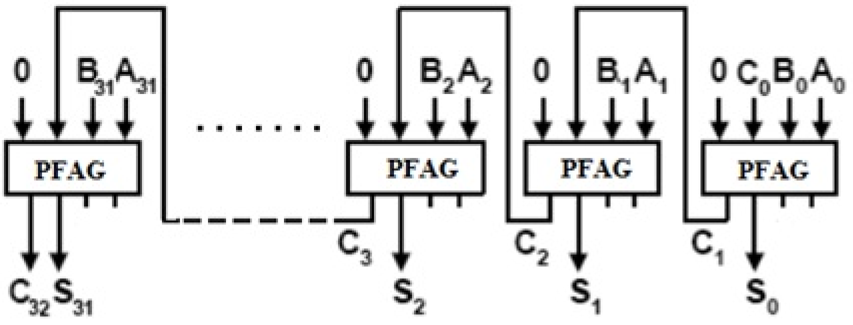


Figure 4.7: Quantum adder 32-bit.

as shown in Figure 4.7. Thus, we can conclude a 32-bit adder can be implemented as a GC with quantum gates requiring just **64 non-XOR** gates.

4.2.4 Q-GBC for the 64-bit adder

To implement quantum garbled circuit implementation of 64-bit adder circuit, we can use the same circuit described in section §4.2.3 and extend it to a 64-bit adder circuit as shown in Figure 4.8.

As it is possible to observe in Figure 4.5, every PFAG consists of two Peres gates (PG) and every PG has 1 CNOT (XOR) gate and 1 Toffoli (non-XOR) gate. Thus; PFAG has 2 non-XOR gates and chain of 64 PFAG concludes **128 non-XOR** gates

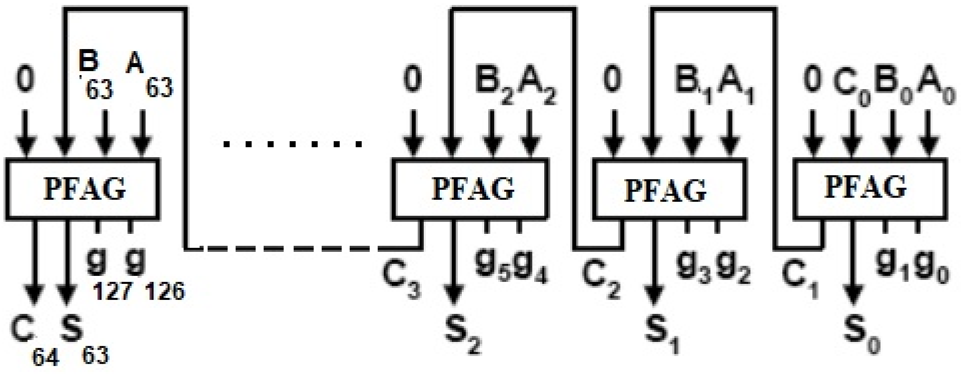


Figure 4.8: Quantum adder 64-bit.

Version	Milionaire 4-bit	Milionaire 8-bit	Adder 32-bit	Adder 64-bit
GC	10	26	127	265
Quantum GC	8	16	64	128

Table 4.1: Comparison of our circuit in case of number of non-XOR gates.

in 64-bit adder.

4.3 Discussion and Comparison of Results

We have put forward the idea of decreasing the computational cost of SMC by reducing the number of non-XOR gates using quantum gates. Our approach is innovative since it is one of the first attempts of using quantum gates for the design of GCs. We validate the approach showing its applicability in two classical SMC examples, the Millionaires' problem, 32bit adder and 64bit adder.

As it is possible to observe in Table 4.1, where we have compared the proposed quantum GC design with classical GC in terms of the proposed metric (i.e., the number of non-XOR gates), we have a reduced number of non-XOR gates in all the examined cases. The results are encouraging, since we report a reduction of the number of non-XOR gates of about **20%** in *4-bit Millionaires' problem*, about **36%** in *8-bit Millionaires' problem*, about **49%** in *32-bit adder* and about **51.6%** in *64-bit adder*. This result can be intuitively explained by the fact that XORs are very common in quantum circuits.

Chapter 5

BDDs for Secure Multi-Party Computation

The aim of this chapter is to define BDD-based function representation method to find classical representations can be equal to XOR function and can be transformed to XORs gates without altering the final output. The idea is to identify gates that are very similar to XOR gates to reduce the number of non-XOR gates in the circuit, which can result in a reduced number of interactions between the parties, and therefore in a more efficient secure computation.

5.1 Using BDDs for Multi-Party Computation

When we have a complex circuit, we want to try to transform some of the contained gates to XOR gates. Now the ***first question*** is that:

"Can we transform some non-XOR gates
of the circuit in XOR gates?"

To ***answer*** of this question, we try to find the similarity of other gates and XOR gates. This means that we are looking for gates that differ with a XOR gate only for one output. If the corresponding input never occurs to this gates, it means that the gate can operates as XOR gate. For example, the similarity of some functions with XOR is shown below:

A	B	A XOR B	A OR B
0	0	0	0
0	1	1	1
1	0	1	1
1	**1**	"0"	"1"

Table 5.1: Showing similarity of functions *f=A+B* and *f=A⊕B* using their truth tables.

A	B	A⊕B	A.B	(A⊕B)+(A.B)	(A⊕B)⊕(A.B)
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	0	1	1
1	**1**	0	1	1	1

Table 5.2: Showing similarity of function *f=(A⊕B)+(A.B)* with *f=(A⊕B)⊕(A.B)*.

Example: Similarity of *f=A+B* and *f=A⊕B*

As shown in Table 5.1, gate *"OR"* can be transformed to *"XOR"* gate if the inputs *A="1"* and *B="1"* never occur. In this case the *"OR"* gate can be transformed in a *"XOR"* gate.

Example: Similarity of *f=(A⊕B)+(A.B)* with *f=(A⊕B)⊕(A.B)*

For example, consider the simple function *f=(A⊕B)+(A.B)*. Note that, the inputs to the *"OR"* gate (+) never have the configuration *"1""1"* (i.e., *"(A⊕B)"* and *"(A.B)"* can not be "1" at the same time. (As shown in Table 5.2 and Figure 5.1)). For this reason, the function *f=(A⊕B)+(A.B)* can be transformed to the *f=(A⊕B)⊕(A.B)*.

Let us now consider any circuit and try to find other classical representation that equal to XOR. It is important to note that we do not want to change the circuit, we just want to find gates that are not XORs but work like XORs, and therefore they can be used as XOR gates.

$$\underbrace{(A\ AND\ B)}_{1} + \underbrace{(A\ XOR\ B)}_{0}$$

Figure 5.1: Different results with input (1,1) to both sides of OR gate.

A	B	A XOR B	A Nand B	A AND B’	A’ AND B	A OR B
0	0	**0**	“1”	0	0	0
0	1	**1**	1	“0”	1	1
1	0	**1**	1	1	“0”	1
1	1	**0**	0	0	0	“1”

Table 5.3: Truth table of similar functions to XOR gate.

Finding other classical gates of the circuit similar to XOR gate is possible when the input configuration on which they differ from a XOR never occurs. Thus, we can say that these gates can work as XOR gate and can be transformed to XOR function. Some of the classical functions that differ from the XOR function in just one case are shown in Table 5.3.

The ***second question*** is that:

"How can we check if a input configuration never occurs in a gate of a given circuit?"

For this reason, we used BDDs method to find possible classical representation that can be transformed to *XOR*, which is discussed in next section.

The main idea is based on the BDD representation of the subfunctions corresponding to the inputs of the gate we want to transform in a *XOR*. For example consider in $f = f_1 + f_2$. If the inputs f_1=*"1"* and f_2=*"1"* do not occur, the *"OR"* gate can be transformed to the *"XOR"* gate. To perform this check, we need to compute the intersection between f_1 and f_2. If the intersection is empty, this means that the input *"1" "1"* never occurs.

In general, if we have one of the gate, (gate in Table 5.3 similar to the XOR gate), we can check *A Nand B* performs the intersection of *A' AND B'* (the input is *"A=0""B=0"*). *A AND B'* performs the intersections between *A' AND B* (the input is *"A=0""B=1"*) and so on. In order to perform these intersections we can use BDDs representing the subfunctions.

5.1.1 Using BDDs to Find Similar Classical Representation to XOR

Finding the rule to understand whether the different input items in each gate happened or not is very simple if we use ***array of BDD***. Considering BDD of each item helps us to find a solution. We can say that this different item would never happen if its BDD were equal to ***"0" (Zero)***. We clarify this method using the BDDs for 32-bit adder as discussed below.

5.1.1.1: BDDs for 32-bit Adder:

We have presented all the subfunctions of 32-bit adder[1] according to CBFS-MPC tool (see §4.1.2) using several BDDs as shown in Figure 5.2.

As we have seen before, the number of inputs' wires are 32 and the number of gates is 439. The first 64 elements in our BDDs array consist of our inputs, which for each element we have BDD (see Figure 5.2). From 64 to 438 we have BDDs of gates and the last 33 BDD (from 406 to 438) we have BDDs of outputs.

Now, the main question is:

"How can we find the gate that can be replaced with a XOR gate? and how can we find whether their particular case can happen or not?"

The answer to this question can be found on the BDDs. The gate can work as XOR gate and can be replaced with XOR gate if its BDD was equal to *"0"*, which means this particular case did not happen at all. In this particular case, we have just AND, NOT and XOR gates. We try to transform these AND, NOT gates to XOR gate.

[1]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/adder_32bit.txt

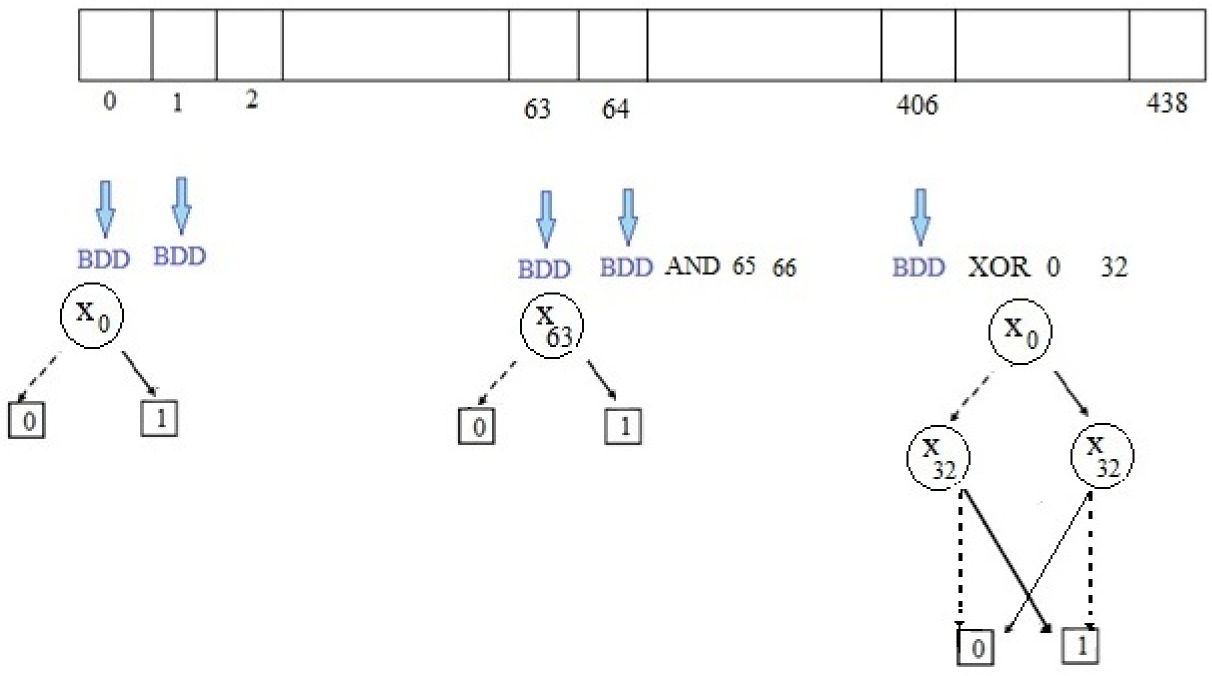


Figure 5.2: Array of BDDs for 32-bit adder.

For example, let us consider this case:

2 1 94 95 92 AND \ *And gate with 2 inputs, 1 outputs , inputs:(94 , 95) , output:92*

1 1 92 89 INV \ *NOT gate with 1 input, 1 output , input:92 , output:89*

That means: 89 = ! (94 AND 95)

Let us try to see if (94 AND 95) happened or not. We should see its BDD, if it is equal to *"0"*, we can conclude that this case can be worked as XOR gate and can be replaced with XOR gate.

Implementation of our method:

We used ***CUDD*** for making array of BDDs and examined our method for 32-bit adder and 64-bit adder[2] functions, Data Encryption Standard (DES)[3] and MD5[4] which can be performed also for other functions. Some important notes of this coding (in case of 32-bit adder), which can be similar for other functions are explained

[2]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/adder_64bit.txt

[3]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/DES-expanded.txt

[4]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/md5.txt

below:

- At first, we have to define number of inputs (variables) equal to "64" (x_0-x_{63}), number of outputs equal to "33" and number of gates equal to "439". We constructed BDDs for these variables and gates as shown in Figure 5.2. We should also define name of inputs as a string to perform XOR, AND or INV and number of input(s) in gates that can be "1" for INV and "2" for AND or XOR (as you seen before, in 32-bit adder function, we have just 3 types of gates; AND, INV and XOR).
- We can allocate the place for gates in this array in which each element is BDDs node; meaning that we have BDDs for each node that should be allocated.
- When we create BDDs for each node, we have to reference it to compute BDDs and confirm that BDDs is done.

  Note: When we reading the circuit, we just count the number of times that it is appear. What we have to do is that everytime a gate is used by another gate, we increment the reference of this gate. So, we can find how many times this gate is used.
- We try to replace AND gate with XOR gate, because the cost of INV is "0" and it is not necessary to change it to XOR gate.
- The important note is that, when we found that AND gate can be transformed to XOR gate, it is necessary to check whether it is used somewhere else or not. We can not perform this transform if it is used for more than one time in the network. So, we have to be sure that nobody else used this gate. For this reason, we should use another array to count how many times this gate is used. This is a vector that called *"references"*. If *reference* is equal to "0" means that this gate can be removed; otherwise, it means that this gate is used some where else. Then, references array help us to check whether we have gained or not.
- We should also keep the order of lines (e.g., 32-bit adder[5] or 64-bit adder[6]). So, in process of making AND gate transform to XOR gate, we would like just a

[5]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/adder_32bit.txt
[6]https://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/adder_64bit.txt

Methods	**32-bit Adder**				
	#**NCycle**	#ANDs	#XORs	#INVs	#**Gain**
CBFS-MPC	**-**	127	61	187	-
CUDD (BDDs)	**300**	125	63	187	**2**

Table 5.4: Comparison of our gaining for 32-bit Adder in BDDs method.

Methods	**64-bit Adder**				
	#**NCycle**	#ANDs	#XORs	#INVs	#**Gain**
CBFS-MPC	**-**	265	115	379	-
CUDD (BDDs)	**500**	253	127	379	**12**

Table 5.5: Comparison of our gaining for 64-bit Adder in BDDs method.

make copy exactly the same and doing changes; nothing else.

- BDDs of these functions was too big, because of this we used BDD_Init to say ”stop” and get a gain. (e.g., ”stop” sfter position 300 for 32-bit adder and 500 for 64-bit adder)

5.1.2 BDDs Method for Adders

We have applied and compared the proposed BDDs method with previous implementation of garbled Boolean circuit using CBFS-MPC (§4.1.2) for 32-bit adder (Table 5.4) and 64-bit adder (Table 5.4) and get following results in the number of non-XOR gates.

5.1.3 BDDs Method for Data Encryption Standard (DES)

Data Encryption Standard (DES) [74] is a symmetric-key method of data encryption that was published by the National Institute of Standards and Technology (NIST) in the early 1970. It was the first encryption algorithm adapted by the U.S.government for public exposure and also by industries such as financial services, where strong encryption are highly needed. The simplicity of DES was considered for using it in wide variety of embedded systems, smart cards, SIM cards and network devices requiring encryption like modems and routers.

Data Encryption Standard (DES) works by using the same key to encrypt and decrypt a message, so both the sender and the receiver should know and use the same private key. cryptographic key and algorithm are applied to a block of data which the block size in DES algorithm is 64 bits. DES takes a fixed-length block of the message (plaintext) and transforms it through a series of permutation and substitution into another bit-string (ciphertext) with same length. Encryption of a block of the message also takes place in 16 rounds.

DES uses a 64-bit key to customize the transformation; however, only 56 of these are actually used by the algorithm, but eight of those bits are used for parity checks. Decryption can performed by those who know the particular key used to encrypt.

Using BDDs method for DES algorithm both with key expanded and no key expansion in different time-cycles results very good improving in the number of non-XOR gates as shown in Table 5.6 and Table 5.7.

5.1.4 BDDs Method for MD5 Cryptographic Function

The MD5 algorithm is a one-way cryptographic function that accepts a message of any length as input and returns a fixed-length 128-bit digest value as output. The *message digest* output is sometimes also called the "hash" or "fingerprint" of the input.

MD5 was designed by well-known cryptographer Ronald Rivest in 1991 used in many situations where a potentially long message needs to be processed. The most common application of MD5 is the creation and verification of digital signatures.

As it is possible to observe in Table 5.8, using BDDs method for MD5 function can get very good results in the number of non-XOR gates in different time-cycles.

Methods	**DES (Key Expanded)**				
	#**NCycle**	#ANDs	#XORs	#INVs	#**Gain**
CBFS-MPC	-	18175	1351	10875	-
CUDD (BDDs)	500	18170	1356	10875	5
CUDD (BDDs)	600	18148	1378	10875	27
CUDD (BDDs)	650	18125	1401	10875	50
CUDD (BDDs)	800	18048	1478	10875	127
CUDD (BDDs)	900	18021	1505	10875	154
CUDD (BDDs)	1000	17980	1546	10875	195
CUDD (BDDs)	1500	17861	1665	10875	314
CUDD (BDDs)	1700	17790	1736	10875	385
CUDD (BDDs)	1800	17751	1775	10875	424
CUDD (BDDs)	1900	17734	1792	10875	441

Table 5.6: Comparison of our gaining for DES (Key Expanded) in BDDs method.

Methods	**DES (No Key Expansion)**				
	#**NCycle**	#ANDs	#XORs	#INVs	#**Gain**
CBFS-MPC	-	18124	1340	10849	-
CUDD (BDDs)	500	18122	1342	10849	2
CUDD (BDDs)	600	18108	1356	10849	16
CUDD (BDDs)	650	18086	1378	10849	50
CUDD (BDDs)	1000	17910	1554	10849	214
CUDD (BDDs)	900	18021	1505	10875	154
CUDD (BDDs)	1500	17799	1665	10849	325
CUDD (BDDs)	1600	17752	1712	10849	372
CUDD (BDDs)	1700	177170	1747	10849	407
CUDD (BDDs)	1800	17689	1775	10849	435
CUDD (BDDs)	1900	17680	1784	10849	444

Table 5.7: Comparison of our gaining for DES (No Key Expanded) in BDDs method.

Methods	**MD5**				
	#**NCycle**	#ANDs	#XORs	#INVs	#**Gain**
CBFS-MPC	-	29084	14150	34627	-
CUDD (BDDs)	700	29083	14151	34627	1
CUDD (BDDs)	800	29079	14155	34627	5
CUDD (BDDs)	850	29078	14156	34627	6
CUDD (BDDs)	900	29077	14157	34627	7
CUDD (BDDs)	1000	29075	14159	34627	9
CUDD (BDDs)	1100	29072	14162	34627	12
CUDD (BDDs)	1200	29068	14166	34627	16
CUDD (BDDs)	1300	29067	14167	34627	17

Table 5.8: Comparison of our gaining for MD5 in BDDs method.

5.1.5 Discussion and Comparison of Experimental Results

We have compared the proposed BDDs method with previous implementation of garbled Boolean circuit using CBFS-MPC (§4.1.2). We can conclude that we have a reduced number of non-XOR gates (AND gates) in *32-bit adder* of about **1.6%**, in *64-bit adder* of about **4.6%**, in *MD5* of about **0.05%** and in *DES* of about **2.5%**.

Chapter 6

Multiple -Valued Logic for Secure Multi-Party Computation

As mentioned before, Secure Multi-party Computation (SMC) protocols enable two or more parties to compute collaboratively generic functions while keeping secret their inputs, sharing only the final result. To achieve this goal, a technique relying on the design of Garbled Circuits (GC) has been firstly proposed by Yao. Garbled circuits are Boolean circuits that can be evaluated using a distributed protocol for computing the result for each gate, till computing the output values. According to this method, standard function f is encoded in a Boolean function f_B that can be represented by a Boolean circuit. Starting from their input values, the parties interact to compute the final result by exchanging some encrypted information in order to evaluate the output of each Boolean gate in the circuit. For this reason, the cost of the secure two-party computation protocol is generally proportional to the number of logic gates in the Boolean circuit. To avoid (or limit) the encoding of the function to be securely computed, and in order to obtain a more compact circuit description, in this section we study the generalization of the classical Yao's protocol in the MVL context. We show also how in this context it is possible to extend some of the techniques to improve the evaluation of multiple valued gates, having no communication costs.

6.1 Multiple Valued Yao's Protocol

We will explore alternative GC representations, focusing on Multiple Valued Logic ones and analyze the deployment of Multiple Valued Logic techniques for the design of GC. Yao's Garbled Circuit construction is composed of two phases: Garbling and Evaluation, which are distinctly executed by the two parties. During the Garbling phase, Alice converts a circuit into a garbled circuit, while Bob performs the Evaluation phase taking in input the garbled circuit, executing some interactions with Alice, and finally computing the output value.

Here we consider two parties, Alice and Bob that want to evaluate the Multiple-Valued Logic (MVL) (§3.4) instead of the Boolean logic gate, which result in reduced wiring complexity and number of interactions required to implement logic functions of Garbled Circuits.

In order to generalize Yao's construction, let us consider a circuit composed of a single multiple valued gate with two input wires, w_1 and w_2, and one output wire w_1. Let x_1 denote the input multiple-valued value known only to Alice, and x_2 the input multiple-valued value known only to Bob.

Consider the set $P = \{0, 1, \ldots, p-1\}$ with $p = |P| > 1$ and the multiple-valued gate $G : P^2 \to P$. To proceed with the computation, for each wire w_i Alice generates p randomly selected different cryptographic keys, one for the input value 0 denoted by k_i^0, one for the input value 1, k_i^1, …, one for the input value $p-1$, k_i^{p-1}.

The keys are used as input to a selected encryption algorithm, denoted as $E_{k_1,k_2,\ldots,k_{p-1}}(m)$. Using those keys, Alice can compute the garbled truth table for the function computed by the gate, where each entry is obtained using a combination of the input keys corresponding to the possible input values, and contains the encryption of the corresponding output value of the gate. Note that a truth table for a gate $G : P^2 \to P$ has $|P|^2$ entries. For example, while a truth table in the Boolean domain has 4 Boolean entries, in the case of $|P| = 3$ the truth table has 9 multiple valued values. XOR Multiple-Valued Logic (MVL) (i.e., p=3) according to (*"mod-sum"* operator) for two parties with two input wires x_1 and x_2, is shown in in Table 6.1.

x	y	z (MV XOR)
0	0	0
0	1	1
0	2	2
1	0	1
1	1	2
1	2	0
2	0	2
2	1	0
2	2	1

Table 6.1: Truth table of MV XOR (mod-sum) operation for p=3 in Garble Circuit.

To generate the garbled Boolean circuit Alice selects and associates random keys to each input and output wire and for each possible value. In our MV XOR gate, she generates the random keys $k^0{}_x$, $k^1{}_x$, $k^2{}_x$ for input wire x and $k^0{}_y$, $k^1{}_y$, $k^2{}_y$ for input wire y and $k^0{}_z$, $k^1{}_z$, $k^2{}_z$ for output wire z as shown in Figure 6.1. Alice encrypts the truth table of the gate using those random keys and sends the garbled truth table to Bob. The resulting garbled truth table is shown in Table 6.2. Once Alice has computed the garbled values for all the entries of the table, she can send a permutation of the truth table, together with the keys corresponding to her inputs.

Notice that the knowledge of the keys, doesn't allow Bob to learn anything on the input values. At this point, Bob needs, from Alice, the keys corresponding to her own input values without disclosing them to Alice. For this purpose, the party can engage in an Oblivious Transfer protocol (OT), allowing Bob to learn the keys corresponding to his inputs.

In general, the GC construction relies on $1-out-of-2-$OT protocol between a sender and a reciever. The sender Alice has two secret values v_0 and v_1, and the receiver has a secret bit i. At the end of the protocol Bob learns v_i, but nothing about v_{i-1}, while Alice doesn't learn anything about the selection bit i. OT protocol is a widely studied cryptographic primitive, with different variants and implementa-

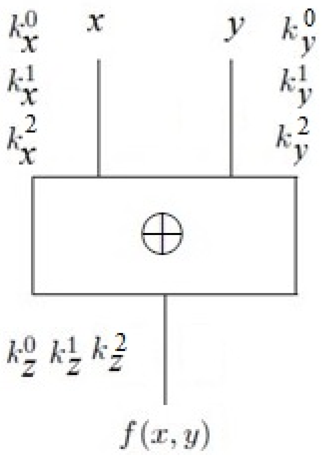


Figure 6.1: MV XOR gate (mod-sum) with its corresponding wire keys.

x	y	
0	0	$\mathrm{E}k^0{}_x(\mathrm{E}k^0{}_y(k^0{}_z))$
0	1	$\mathrm{E}k^0{}_x(\mathrm{E}k^1{}_y(k^1{}_z))$
0	2	$\mathrm{E}k^0{}_x(\mathrm{E}k^2{}_y(k^2{}_z))$
1	0	$\mathrm{E}k^1{}_x(\mathrm{E}k^0{}_y(k^1{}_z))$
1	1	$\mathrm{E}k^1{}_x(\mathrm{E}k^1{}_y(k^2{}_z))$
1	2	$\mathrm{E}k^1{}_x(\mathrm{E}k^2{}_y(k^0{}_z))$
2	0	$\mathrm{E}k^2{}_x(\mathrm{E}k^0{}_y(k^2{}_z))$
2	1	$\mathrm{E}k^2{}_x(\mathrm{E}k^1{}_y(k^0{}_z))$
2	2	$\mathrm{E}k^2{}_x(\mathrm{E}k^2{}_y(k^1{}_z))$

Table 6.2: Initial garbled circuit table for MV XOR gate.

tion, whose robustness has been considered under different security models. In the Yao extension, we consider a $1 - out - of - n$-OT protocol that can be defined as a natural generalization of a $1 - out - of - 2$ OT, where the sender has n values, and the receiver has an index i, corresponding to the value he owns and for which she wishes to receive the i-th key, without the sender learning i. At the end, the OT protocol allows Bob to retrieve the right key corresponding to her input value for the gate, while Alice doesn't learn anything about the selected input. Since now Bob knows the two keys, he can decrypt the right entry in the table, and retrieve the key of the output value.

In general, for a circuit composed of multiple gates, Alice should compute garbled values for all the input wires and use them for computing the truth table for each multiple-valued gate. Then she should send all the truth tables, and all her input values to Bob, which can invoke the OT protocol for each needed input value to the circuit. Once retrieved all the values, Bob can compute the output keys for all the gates of the circuit.

At the end of the protocol, Bob has generated the key k of the final output and sends it to Alice. The party Alice can now send to Bob the value, corresponding to k, of the output of the entire function. This value is the final result of the computation.

6.2 Improved Evaluation for Multiple Valued Gates

In [37], Kolesnikov and Schneider presented an optimisation, which allows the evaluation of XOR gates for free, avoiding any interaction between the two parties for such gates, i.e. there is no need to compute and send the garbled tables for the XOR gates. The optimisation requires that there is a global random value R known only to one party, such that for all garbled wires w_i it holds that $k_i^1 = k_i^0 \oplus R$, i.e. the garbled value corresponding to 1 for a wire, is determined by XOR-ing the garbled 0 value with the random quantity R. In this way, computing the output value for a XOR gate amounts to compute the value resulting by the XOR of the two input values. Security of this solution has been proved in different context in [37, 55] under different assumptions.

Here we show how also in the case of multiple valued gates, these improved evaluation techniques can be easily extended. Specifically, we show how the SUM gate G, in the case of three-valued logic can be can be evaluated without communication between the parties (the reasoning holds for gates in the case of P values).

Let G have two input wires W_a and W_b and output wire W_c. Garble the wire values as follows. Randomly choose w_a^0, $w_b^0, R_1, R_2 \in_R \{0,1,2\}^N$, with the following properties for R_1 and R_2:

$$R_1 \oplus R_2 = 0$$

,

$$R_1 \oplus R_1 = R_2$$

,

$$R_2 \oplus R_2 = R_1$$

. Set $w_c^0 = w_a^0 \oplus w_b^0$, and $\forall i \in (a,b,c) : w_i^1 = w_i^0 \oplus R_1$ and $w_i^2 = w_i^0 \oplus R_2$

It is easy to see that the garbled gate output is simply obtained by summing the garbled gate inputs:

$$w_c^0 = w_a^0 \oplus w_b^0 = (w_a^0 \oplus R_1) \oplus (R_2 \oplus w_b^0) = w_a^1 \oplus w_b^2 = \\ (w_a^0 \oplus R_2) \oplus (R_1 \oplus w_b^0) = w_a^2 \oplus w_b^1$$

$$w_c^1 = w_c^0 \oplus R_1 = w_a^0 \oplus (w_b^0 \oplus R_1) = w_a^0 \oplus w_b^1 = (w_a^0 \oplus R_1) \oplus \\ \oplus w_b^0 = w_a^1 \oplus w_b^0 = (w_a^0 \oplus R_1 \oplus R_1) \oplus (R_2 \oplus w_b^0) = w_a^2 \oplus w_b^2$$

$$w_c^2 = w_c^0 \oplus R_2 = w_a^0 \oplus (w_b^0 \oplus R_2) = w_a^0 \oplus w_b^2 = (w_a^0 \oplus R_2) \oplus \\ \oplus w_b^0 = w_a^2 \oplus w_b^0 = (w_a^0 \oplus R_2 \oplus R_2) \oplus (R_1 \oplus w_b^0) = w_a^1 \oplus w_b^1$$

We postpone here the security proofs for this approach.

Chapter 7

Conclusions

Two-party Secure Multi-party Computation (SMC) protocols have performed collaborative computation of generic function f of their respective inputs x_1 and x_2, between two or more parties who do not want to disclose the input values they own and share only the computed result. In the general formulation, two parties want to compute a function on their respective inputs, while maintaining the privacy of the inputs. Secure multi-party computation protocols can have high communication cost as the complexity of the computation grows. Recently, the question to improve efficiency in secure multi-party computation and its potential has gained much interest. Significant improvements in efficiency have been achieved and a number of SMC-based solutions to a number of problems such as private auctions, tax-fraud detection, email filtering and others have been delivered. In this thesis, we discussed about the idea of decreasing the communication complexity and computational cost of secure multi-party computation (SMC).

One of the proposed approaches to solve the SMC problem relies on the design of Garbled Circuit (GC), proposed by Yao (§1.2). The Garbled Circuit (GC) protocol has been introduced as a method for addressing two-party secure computations for the evaluation of the input function *(f)* represented as a Boolean Circuit, which is based on the encryption of the input and intermediate values so that only the final result is shared among the parties. The use of garbling for secure multi-party shared computation has seemed to be widely studied in the field of secure computing and its application. Since the execution of this protocol requires interaction between the collaborating parties, the total cost and run-time interaction between parties increases linearly with the number of gates, and can be huge for complex GC. So, reducing the circuit size and the number of gates is important to reduce the overall communication cost and the number of operations for the evaluation of GC.

Free-XOR technique (§1.3.1), proposed by Kolesnikov and Schneider [37], is one of the possible ways to improve the performance of garbled circuit (GC). According to this idea, secure evaluation of XOR gates does not need the transfer of garbled tables and does not require garbling of XOR gates. So, replacing costly non-XOR gates with some *free*-XOR gates allows us to have more efficient secure computation.

In this dissertation, we have explored the idea of decreasing communication complexity and communication cost of SMC and reduced its computation time and the number of operations based on reducing the number of non-XOR gates to improve efficiency of the Garbled Circuit construction, which means reducing the circuit cost and the gates for interaction required in the Boolean Circuit. We focus on reducing the number of interactions between parties at runtime, which results in reduced communication cost and communication time of Secure Multi-party Computation (SMC) by reducing the number of non-XOR gates since XOR gates have no cost for the execution of the secure computation protocol. This thesis is proposing to use following three approaches (exploiting Quantum gates, BDDs and MVL) to reduce gate complexity and cost in realization of garbled circuit and improve computational time and communication cost of SMC and validate these approaches showing its applicability in classical SMC examples, the Millionaires' problem, adders, DES and MD5.

In chapter 4, we have discussed the possibility to construct Garbled Circuit using quantum gates (QG), observing that in some cases, the quantum GC requires a lower number of non-XOR gates with respect to the corresponding classical GC implementations. Having fewer non-XOR gates results in a reduced number of interactions between the parties at runtime, reducing the communication cost and improving the overall efficiency of the execution of the SMC protocol. Our approach is innovative since it is one of the first attempts of using quantum gates for the design of GCs. Our experimental results show that reduction of the number of non-XOR gates was about 20% of 4-bit Millionaires' problem, about 36% of 8-bit Millionaires' problem, about 51.6% of 64-bit adder, and about 49% in the 32-bit adder. This result can be intuitively explained by the fact that XORs are very common in quantum circuits.

In chapter 5, we used a Binary Decision Diagram structure (BDD) for our improving proposed methods. Using BDDs helps us to find similarity of other function representations to XOR operation, aiming at transforming some non-XOR gates of the circuit in XOR gates to reduce the number of non-XOR gates in GC construction, which can result in reducing the communication costs and circuit cost. Our experimental results show that reduction of the number of non-XOR gates was about 1.6% of 32-bit adder, 4.6% of 64-bit adder, 0.05% of MD5 function and 2.5% of DES.

In chapter 6, we generalize Yao's secure two-party computation protocol to multiple-valued logic and we consider a general multiple-valued function f that is represented by a multiple-valued circuit composed by multiple-valued gates and discussing their impact on the overall computation and communication costs. We pursue the idea of using Multiple Valued Logic for the synthesis of garbled circuits, showing an extension of the classic Yao protocol for the evaluation of multiple valued gates. In this chapter, we show how these improved evaluation techniques can be readily extended to the case of MVL gates.

7.1 Future Work

In the future, we intend to investigate several directions as follows:

- Possible future research on novel logic representations with support of XOR gates, i.e., the XOR-AND-Inverter Graphs (XAIGs). Using XOR based synthesis and implementing on XAIG based rewriting algorithm in optimization tool (e.g., ABC) to improve efficiency of GC.

- Investigate some combinational network and converting Boolean circuit to cell-based combinational circuit with suitable elementary cells whose properties can be converted in a quantum version. Thus, we can drive these properties from elementary cell for the whole garbled circuit (GC).

- Developing security proofs for our proposed protocols, as well as a full set of experimental prototypes to compare the size and the performance of the multiple valued (MV) circuits with respect the original circuits.
- We plan to work on different methods and techniques for the minimization of non-XOR gates, by using tools and design techniques usually deployed in synthesis for achieving different goals.
- Working on different methods for the optimization of the synthesis of garbled circuit relying on quantum gates (QG) and MVL gates, trying to apply the methodology to more complex case studies.
- Designing quantum GC directly from the definition of the input function, and not from the definition of a GC.

Bibliography

[1] Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on computers*, (6):509–516, 1978.

[2] David W Archer, Dan Bogdanov, Benny Pinkas, and Pille Pullonen. Maturity and performance of programmable secure computation. *IEEE Security & Privacy*, 14(5):48–56, 2016.

[3] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 535–548. ACM, 2013.

[4] R Iris Bahar, Erica A Frohm, Charles M Gaona, Gary D Hachtel, Enrico Macii, Abelardo Pardo, and Fabio Somenzi. Algebric decision diagrams and their applications. *Formal methods in system design*, 10(2-3):171–206, 1997.

[5] Mauro Barni, Pierluigi Failla, Vladimir Kolesnikov, Riccardo Lazzeretti, Ahmad-Reza Sadeghi, and Thomas Schneider. Secure evaluation of private linear branching programs with medical applications. In *ESORICS*, volume 5789, pages 424–439. Springer, 2009.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[7] Charles H Bennett. Logical reversibility of computation. *IBM journal of Research and Development*, 17(6):525–532, 1973.

[8] Charles H Bennett. Quantum cryptography using any two nonorthogonal states. *Physical review letters*, 68(21):3121, 1992.

[9] Charles H Bennett and Gilles Brassard. Quantum public key distribution reinvented. *ACM SIGACT News*, 18(4):51–53, 1987.

[10] Anna Bernasconi and Valentina Ciriani. Autosymmetric multiple-valued functions: Theory and spectral characterization. In *Multiple-Valued Logic (ISMVL), 2011 41st IEEE International Symposium on*, pages 10–15. IEEE, 2011.

[11] Anna Bernasconi and Valentina Ciriani. Autosymmetric and dimension reducible multiple-valued functions. *Journal of Multiple-Valued Logic & Soft Computing*, 23, 2014.

[12] Robert K Brayton and Sunil P Khatri. Multi-valued logic synthesis. In *VLSI Design, 1999. Proceedings. Twelfth International Conference On*, pages 196–205. IEEE, 1999.

[13] Robert K Brayton, Richard Rudell, Alberto Sangiovanni-Vincentelli, and Albert R Wang. Mis: A multiple-level logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):1062–1081, 1987.

[14] Justin Brickell, Donald E Porter, Vitaly Shmatikov, and Emmett Witchel. Privacy-preserving remote diagnostics. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 498–507. ACM, 2007.

[15] Randal E Bryant. Graph-based algorithms for boolean function manipulation. *Computers, IEEE Transactions on*, 100(8):677–691, 1986.

[16] David Deutsch and Richard Jozsa. Rapid solution of problems by quantum computation. In *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 439, pages 553–558. The Royal Society, 1992.

[17] Paul Adrien Maurice Dirac. A new notation for quantum mechanics. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 35, pages 416–418. Cambridge University Press, 1939.

[18] Rolf Drechsler and Detlef Sieling. Binary decision diagrams in theory and practice. *International Journal on Software Tools for Technology Transfer (STTT)*, 3(2):112–136, 2001.

[19] Elena Dubrova. Multiple-valued logic in vlsi: Challenges and opportunities. In *Proceedings of NORCHIP*, volume 99, pages 340–350, 1999.

[20] Elena Dubrova. Multiple-valued logic synthesis and optimization. *Logic synthesis and verification*, pages 89–114, 2002.

[21] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. Scapi: The secure computation application programming interface. *IACR Cryptology EPrint Archive*, 2012:629, 2012.

[22] Artur K Ekert, John G Rarity, Paul R Tapster, and G Massimo Palma. Practical quantum cryptography based on two-photon interferometry. *Physical Review Letters*, 69(9):1293, 1992.

[23] Zekeriya Erkin, Martin Franz, Jorge Guajardo, Stefan Katzenbeisser, Inald Lagendijk, and Tomas Toft. Privacy-preserving face recognition. In *International Symposium on Privacy Enhancing Technologies Symposium*, pages 235–253. Springer, 2009.

[24] Daniel Etiemble and Michel Israel. Comparison of binary and multivalued ics according to vlsi criteria. *Computer*, 21(4):28–42, 1988.

[25] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Communications of the ACM*, 28(6):637–647, 1985.

[26] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[27] Richard P Feynman. Quantum mechanical computers. *Optics news*, 11(2):11–20, 1985.

[28] Tore Kasper Frederiksen, Thomas P Jakobsen, and Jesper Buus Nielsen. Faster maliciously secure two-party computation using the gpu. In *International Conference on Security and Cryptography for Networks*, pages 358–379. Springer, 2014.

[29] E Fredkin and T Toffoli. Conservative logic int. 3. *Theor. Phys. 21 219*, 253, 1982.

[30] M Gao, J-H Jiang, Y Jiang, Y Li, A Mishchenko, S Sinha, T Villa, and R Brayton. Optimization of multi-valued multi-level networks. In *Multiple-Valued Logic, 2002. ISMVL 2002. Proceedings 32nd IEEE International Symposium on*, pages 168–177. IEEE, 2002.

[31] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 218–229. ACM, 1987.

[32] Wilko Henecka, Ahmad-Reza Sadeghi, Thomas Schneider, Immo Wehrenberg, et al. Tasty: tool for automating secure two-party computations. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 451–462. ACM, 2010.

[33] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, volume 201, 2011.

[34] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Crypto*, volume 2729, pages 145–161. Springer, 2003.

[35] Md Saiful Islam, MM Rahman, Zerina Begum, and Mohd Z Hafiz. Low cost quantum realization of reversible multiplier circuit. *Information technology journal*, 8(2):208–213, 2009.

[36] Atul K Jain, Ron J Bolton, and Mostafa H Abd-El-Barr. Cmos multiple-valued logic design. i. circuit implementation. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(8):503–514, 1993.

[37] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. *Automata, Languages and Programming*, pages 486–498, 2008.

[38] Benjamin Kreuter, Abhi Shelat, and Chih-Hao Shen. Billion-gate secure computation with malicious adversaries. In *USENIX Security Symposium*, volume 12, pages 285–300, 2012.

[39] Chang-Yeong Lee. Representation of switching circuits by binary-decision programs. *Bell Labs Technical Journal*, 38(4):985–999, 1959.

[40] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1(1):5, 2009.

[41] Yehuda Lindell, Benny Pinkas, and Nigel P Smart. Implementing two-party computation efficiently with security against malicious adversaries. In *International Conference on Security and Cryptography for Networks*, pages 2–20. Springer, 2008.

[42] Hoi-Kwong Lo and Hoi Fung Chau. Unconditional security of quantum key distribution over arbitrarily long distances. *science*, 283(5410):2050–2056, 1999.

[43] Dahlia Malkhi, Noam Nisan, Benny Pinkas, Yaron Sella, et al. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, volume 4. San Diego, CA, USA, 2004.

[44] Giovanni De Micheli. *Synthesis and optimization of digital circuits.* McGraw-Hill Higher Education, 1994.

[45] D Michael Miller, Dmitri Maslov, and Gerhard W Dueck. A transformation based algorithm for reversible logic synthesis. In *Design Automation Conference, 2003. Proceedings*, pages 318–323. IEEE, 2003.

[46] D Michael Miller and Mitchell A Thornton. Multiple valued logic: Concepts and representations. *Synthesis lectures on digital circuits and systems*, 2(1):1–127, 2007.

[47] Shin-ichi Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *Proceedings of the 30th international Design Automation Conference*, pages 272–277. ACM, 1993.

[48] Moni Naor, Benny Pinkas, and Reuban Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conference on Electronic commerce*, pages 129–139. ACM, 1999.

[49] Michael A Nielsen and Isaac Chuang. Quantum computation and quantum information, 2002.

[50] Margarita Osadchy, Benny Pinkas, Ayman Jarrous, and Boaz Moskovich. Scifi-a system for secure face identification. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 239–254. IEEE, 2010.

[51] Chandrahash Patel and CS Veena. Comparator design using full adder. *International Journal of Research in Engineering and Technology*, 3(07):365–368, 2014.

[52] Rohit Pathak and Satyadhar Joshi. Secure multi-party computation using virtual parties for computation on encrypted data. *Advances in Information Security and Assurance*, pages 412–421, 2009.

[53] Asher Peres. Reversible logic and quantum computers. *Physical review A*, 32(6):3266, 1985.

[54] Marek Perkowski, Lech Jozwiak, Pawel Kerntopf, Alan Mishchenko, Anas Al-Rabadi, Alan Coppola, Andrzej Buller, Xiaoyu Song, Svetlana Yanushkevich, Vlad P Shmerko, et al. A general decomposition for reversible logic. 2001.

[55] Benny Pinkas, Thomas Schneider, Nigel P Smart, and Stephen C Williams. Secure two-party computation is practical. In *Asiacrypt*, volume 9, pages 250–267. Springer, 2009.

[56] Michael O Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[57] David A. Rich. A survey of multivalued memories. *IEEE Transactions on Computers*, (2):99–106, 1986.

[58] Eleanor Rieffel and Wolfgang Polak. An introduction to quantum computing for non-physicists. *ACM Computing surveys*, 32(3):300–335, 2000.

[59] Richard L Rudell and Alberto Sangiovanni-Vincentelli. Multiple-valued minimization for pla optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):727–750, 1987.

[60] Ahmad-Reza Sadeghi, Thomas Schneider, and Immo Wehrenberg. Efficient privacy-preserving face recognition. In *ICISC*, volume 9, pages 229–244. Springer, 2009.

[61] Subramanian Saravanan, Ila Vennila, and Sudha Mohanram. Design and implementation of an efficient reversible comparator using tr gate. *Circuits and Systems*, 7(09):2578, 2016.

[62] Zahra Sasanian. *Technology mapping and optimization for reversible and quantum circuits.* PhD thesis, 2012.

[63] Maryam Sepehri, Stelvio Cimato, and Ernesto Damiani. A multi-party protocol for privacy-preserving range queries. In *Workshop on Secure Data Management*, pages 108–120. Springer, 2013.

[64] Maryam Sepehri, Stelvio Cimato, and Ernesto Damiani. Privacy-preserving query processing by multi-party computation. *The Computer Journal*, 58(10):2195–2212, 2014.

[65] Rashid Sheikh, Beerendra Kumar, and Durgesh Kumar Mishra. A distributed k-secure sum protocol for secure multi-party computations. *arXiv preprint arXiv:1003.4071*, 2010.

[66] Mathias Soeken, Stefan Frehse, Robert Wille, and Rolf Drechsler. Revkit: An open source toolkit for the design of reversible circuits. In *International Workshop on Reversible Computation*, pages 64–76. Springer, 2011.

[67] Fabio Somenzi. Cudd: Cu decision diagram package-release 1.0. 4. *Tech. Rep.*, 1995.

[68] Ning Song and Marek A Perkowski. Exorcism-mv-2: minimization of exclusive sum of products expressions for multiple-valued input incompletely specified functions. In *Multiple-Valued Logic, 1993., Proceedings of The Twenty-Third International Symposium on*, pages 132–137. IEEE, 1993.

[69] Ebrahim M Songhori, Siam U Hussain, Ahmad-Reza Sadeghi, Thomas Schneider, and Farinaz Koushanfar. Tinygarble: Highly compressed and scalable sequential garbled circuits. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 411–428. IEEE, 2015.

[70] Bernd Steinbach, Marek A Perkowski, and Christian Lang. Bi-decompositions of multi-valued functions for circuit design and data mining applications. In

Multiple-Valued Logic, 1999. Proceedings. 1999 29th IEEE International Symposium on, pages 50–58. IEEE, 1999.

[71] Craig Stuntz. What is homomorphic encryption, and why should i care? *Craig Stuntz Weblog, March*, 18, 2010.

[72] Himanshu Thapliyal and Nagarajan Ranganathan. A new design of the reversible subtractor circuit. In *Nanotechnology (IEEE-NANO), 2011 11th IEEE Conference on*, pages 1430–1435. IEEE, 2011.

[73] Tommaso Toffoli. Reversible computing. *Automata, Languages and Programming*, pages 632–644, 1980.

[74] Walter Tuchman. Internet besieged. chapter A Brief History of the Data Encryption Standard, pages 275–280. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1998.

[75] Zvonko G Vranesic and Kenneth C Smith. Engineering aspects of multi-valued logic systems. *Computer*, 7(9):34–41, 1974.

[76] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W Dueck, and Rolf Drechsler. Revlib: An online resource for reversible functions and reversible circuits. In *Multiple Valued Logic, 2008. ISMVL 2008. 38th International Symposium on*, pages 220–225. IEEE, 2008.

[77] Congguang Yang and Maciej Ciesielski. Bds: A bdd-based logic optimization system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21(7):866–876, 2002.

[78] Andrew C Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[79] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.