Protocols for unconditionally computationally secure circuit computation obfuscation

Yu, Yu

2007

Yu, Y. (2007). Protocols unconditionally computationally secure circuit computation obfuscation. Doctoral thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/2491

https://doi.org/10.32657/10356/2491

Nanyang Technological University

Protocols for Unconditionally and Computationally Secure Circuit Computation and Obfuscation

Yu Yu

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2007

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

Date: August 17, 2007

Yu Yu

Acknowledgements

I would like to thank my supervisor Dr. Jussipekka Leiwo, without whom I would not be where I am today.

I would also like to thank my co-supervisor Dr. Benjamin Premkumar for all the great help in mathematics.

I would especially thank my family for their timeless patience and constant supports.

Last but not the least, I would like to thank the anonymous reviewers for their helpful comments and suggestions.

List of Abbreviations and Symbols

Abbreviations

CED	Computing with Encrypted Data
CEF	Computing with Encrypted Functions
CTO	Circuit Topology Obfuscation
IND-CCA1	Semantic Security under Nonadaptive Chosen-ciphertext Attacks
IR	Intermediate Results
LFSR	Linear Feedback Shift Register
OT	Oblivious Transfer
PH	Privacy Homomorphism
PPT	Probabilistic Polynomial-Time algorithm
PRG	Pseudo-Random Generator
SCC	Secure Circuit Computation
SI	Sensitive Information
XOR	eXclusive OR

Symbols

$\mathcal{L}$	A linear feedback shift register
$\mathcal{G}$	A pseudo-random generator function
Ω	Basis (the set of gate functions for a circuit)
C	Boolean circuit
d	Circuit depth (the longest length from an input to an output)
x	Circuit input
s	Circuit size (the number of gates in a circuit)
$\rightarrow$	Implies
t	Security parameter
$\cdot$	String concatenation operator or the multiplication sign
$\lvert\ \rvert$	The bit length of a string (e.g. $\lvert 010\rvert$=3)
#()	The cardinality of a set (e.g. $\#(\phi)$=0)
$w()$	The hamming weight of a binary string (e.g. $w(101101)$=4)
$T()$	The topology information of a circuit (e.g. $T(C)$)

Table of contents

List of Figures

List of Tables

Abstract

We focus on protocols that enable secure computation and obfuscation of Boolean circuits under reasonable scenarios, where an (infinitely powerful or computationally bounded) adversary obtains nothing substantial from computing or eavesdropping on a circuit. We classify the protocols by the information they are intended to hide, and provide their respective literature review as well as the relations among them.

We discuss the possibility of transforming polynomial-time programs into polynomial size circuits and how to efficiently design a compiler that inputs user-written C-style code and outputs the corresponding circuit format. Then, we construct a Yao-style circuit encryption scheme, based on which we build a one-time-use secure circuit computation (SCC) protocol, and prove that the protocol only reveals circuit topology to a polynomial-time adversary. Nevertheless, we show that circuit topology may sometimes disclose sensitive information regarding the circuit and we need to apply circuit topology obfuscation (CTO) to it to solve the problem. Hence, combined with CTO, the improved SCC protocol discloses nothing substantial.

CTO is central to SCC protocols as well as securing circuit from passive reverse

engineers. As the overhead of CTO can be in terms of either circuit depth or circuit size, we propose two efficient provably and unconditionally secure CTO approaches, one depth-efficient and the other size-efficient, with both incurring at most poly-logarithmic overheads. The depth-efficient approach is built on universal circuits and a hard-wiring technique that can also be used to improve the symmetric two-party computation protocol. The size-efficient approach is based on a hierarchial model that periodically re-shuffles intermediate results such that a circuit topology observer learns nothing.

The foregoing one-time-use protocol requires changing encryption key for each round of computation, which reduces its efficiency and practicability. Based on the above-mentioned CTO algorithms, we propose two distinct reusable SCC protocols and prove their security under the respective assumptions, where the former protocol is constructed with algebraic privacy homomorphism (PH) and the latter with stateful gates. The former protocol is more of theoretic significance than having practical value. We discuss the security of algebraic PH in the black-box model and show that algebraic PH is equivalent to a two-round reusable SCC protocol that reveals nothing substantial to an intended adversary. In contrast, the latter protocol can be implemented in practice to defeat passive or even active adversaries as long as memories of stateful gates are safely protected. We minimize the size of memory to t (the security parameter of the stream cipher) bits per gate and prove that the protocol is computationally secure.

Chapter 1

Introduction

1.1 Motivation

As a procedure for determining outputs by functions on different inputs, computation can be formalized using different models to be carried out mechanically, no matter how complicated it is, by pre-programmed software or hardware. Of all the existing models of computation, Boolean circuit is a simple yet effective one with significant importance to computational theory and it can be easily implemented either by hardware or by software.

Normally, to compute a Boolean circuit, the circuit and its input should be revealed to the party performing the computation. However, there are scenarios where we hope to keep the circuit or the computation transcript (i.e. inputs, intermediate results and outputs) secret. Furthermore, the implementation of a circuit (e.g. a printed circuit board) may be subject to various forms of attacks such as timing attacks, reverse engineering and eavesdropping that jeopardize the privacy of the computation.

With protocols for secure circuit computation and obfuscation, circuits can be computed or engineered in a way that some certain information regarding the circuit computation is well (provably) protected. However, so far most protocols are for secure two-party/multiparty computation, which is restricted to two or more symmetric parties (each in possession of a private input) computing a common circuit against each other. Moreover, those protocols hide only partial information (regarding the circuit computation). Another weakness is that most protocols are for one-time use only, namely, randomness must be refreshed for each round of computation. Therefore, existing protocols may not work well in our asymmetric setting (which we refer to as SCC), which calls for more efficient provably secure protocols.

Privacy has now been a growing concern of our daily life and the applications of SCC protocols include but are not limited to the following:

- Privacy Information Retrieval. This allows a client to retrieve the i-th record from the database of a remote server in such a private manner that i is not revealed to the server [1]. In a more demanding case [2], the client may want to perform a keyword search without disclosing to the server the keyword and returned results. All these problems can be reduced [3] to secure circuit computation and hence solved by SCC protocols, namely, the client represents its query as input x and the server represents the database as a circuit C and they execute a SCC protocol so that the client obtains $C(x)$ privately.

- Private Electronic Voting. This simulates the paper vote in the real world

such that voters participate in the election privately while ensuring that no fraud is committed (see e.g. [4, 5, 6]). An electronic voting scheme can also be represented using Boolean circuit, namely, each voter holds an input x to the counter circuit C which calculates the result of election.

- Secure Mobile Agents. In some cases, portable weak power devices (e.g. smart cards) cannot perform advanced computation and have to utilize the computing service provided by untrustworthy workstations at the cost of computation privacy (see Section 3.1.1 for a detailed discussion). With SCC protocols, a mobile agent can let a untrustworthy party compute a circuit such that nothing substantial is revealed to the party.

1.2 Objective

We briefly depict the problem of SCC between two asymmetric parties as follows: Alice, holding a Boolean circuit C, wants Bob to compute C on any input(s) she gives in such an oblivious manner that sensitive information is not revealed to Bob and final result is sent to Alice. This is achieved by executing a protocol between Alice and Bob. To devise such a protocol, the problem must be further specified with the following parameters and requirements:

- Sensitive information to be protected against Bob. We can define it as circuit C, inputs, intermediate results, outputs, or any combination of them.

- Computational capacity. Alice can be either computationally bounded or infinitely powerful, so can Bob.

- Adversarial model. Bob can be either semi-honest [7, Section 7.2.2] (also known as curious-but-honest) or malicious [7, Section 7.2.3]. A semi-honest adversary does not deviate from the protocol except that he attempts to learn the sensitive information from his message transcript. In most cases, it is more difficult for an adversary to deviate from the protocol (to gain any benefit) than merely recording all the intermediate results for future analysis. Furthermore, the recording/logging capability is available for some computing devices and the records/logs can be used by those least malicious adversaries (e.g. administrators) who are merely curious. The semi-honest model can also be considered as that both parties are honest and there are eavesdropping adversaries (who can intercept all messages transmitted and one party's private inputs) on the communication channel. A malicious adversary can tamper with the protocol execution (i.e. deviate from the predefined protocol) actively to gain additional information he is not supposed to learn. A protocol in the malicious model shall be able to detect the malicious behavior with an overwhelming probability.

- Round complexity, which is defined as the number of communication rounds during protocol execution. It would be desirable for a protocol to have a constant number of rounds as its round complexity does not depend on any

factor (security parameter, circuit size, circuit depth etc.) and hence scales up well.

- The benefits Alice gains from executing the protocol. We note that there is a trivial protocol, namely, Alice computes C herself and Bob does nothing, but in this case Alice takes no advantage of Bob. Likewise, if Alice spends more efforts on carrying out a protocol than computing the circuit herself, then the protocol is also trivial. Therefore, a protocol is non-trivial only if by executing it as much as possible the workload of Alice can be shifted to Bob in a secure way (as in requirement 1). When speaking of a protocol hereafter, we refer to a non-trivial one.

- The overhead for Bob to carry out the protocol rather than to directly compute C. This is also used to measure the efficiency of the protocol.

In this thesis, we discuss protocols where sensitive information is defined as circuit, inputs, intermediate results and the final outputs. We mainly focus on such an asymmetric setting where (1) polynomial-time Alice is computationally weaker than Bob, who is also polynomial-time or even infinitely powerful; (2) the protocol should guarantee that Bob does not cheat Alice but not necessarily the converse as Alice is weaker and is holding all the secrets. In most chapters, we first introduce a protocol for defeating semi-honest Bob, then followed by the extended protocol against malicious Bob. We also hope to minimize the round complexity to two rounds only (thus round-optimal), namely, Alice sends to Bob a prepared (encrypted

or obfuscated) format of C, which Bob computes to send the result back to Alice. Apparently, protocols should be non-trivial to let Alice use Bob's computing power and the overhead incurred on Bob should be as little as possible.

1.3 Thesis Structure

The remainder of the thesis is organized as follows: Chapter 2 provides some background knowledge as well as a review of the literature. In Chapter 3, we discuss the possibility and feasibility of converting polynomial-time programs to polynomial-size circuits, then we present an SCC protocol with garbled circuits and show its inherent weaknesses such as non-reusable keys and information disclosure due to circuit skeleton, where we propose the notion of circuit topology obfuscation (CTO) to overcome the latter weakness. In Chapter 4, we compare two approaches for CTO, namely, obfuscation using universal circuits and hierarchical obfuscation, and exemplify their respective advantages (e.g. the former approach can be used to improve the secure two-party computation protocol). In Chapter 5, we present a reusable SCC protocol based on algebraic PH, and also provide an alternative hardware implementation that is computationally secure against probing attacks. Finally, Chapter 6 draws the conclusion and highlights future work.

1.4 Thesis Contribution

The main contribution of the thesis can be summarized as follows:

1. We examine the security of Yao's two-party computation protocol in our asymmetric setting and give formal proofs that the protocol can hide everything except the circuit topology. As the circuit topology may also disclose sensitive information, we propose the notion of CTO and show its effects on improving the secure two-party computation protocol. We also contribute an efficient and powerful compiler which allows the conversion of user-written C-style source code to Boolean circuits to facilitate circuit computation protocols.

2. We propose two CTO algorithms, one depth-efficient and the other size-efficient. The former is inspired by universal circuits that incur only a logarithmic overhead on circuit depth while the latter is based on a sequential and hierarchial obfuscation technique with a poly-logarithmic overhead on circuit size. We prove that both approaches are capable of hiding circuit topology from unbounded adversaries. Those CTO algorithms are not restricted to our scenario but can be used in any secure computation protocol to prevent information disclosure due to circuit topology.

3. We propose an SCC protocol that allows computing a circuit repeatedly (on different inputs) without necessarily changing the encryption key. We propose a uniform representation of gate functions with which gates can be computed in

their encrypted formats without revealing their functions, inputs and outputs. Then, we show the equivalence of 2-round reusable SCC protocol and algebraic PH.

4. We propose a practical protocol that enables secure implementation of circuits against probing attacks. This is achieved by introducing stateful gates and exploring the properties of the linear feedback shift register (LFSR) sequence. The overhead is a t-bit memory and no more than $2t$ XOR taps per gate with t being the security parameter.

Chapter 2

Background

In this chapter, we give some background information such as Boolean circuits, an introduction to cryptography, a literature review of SCC protocols and the relationships among them.

2.1 Boolean Circuits

Informally, a standard Boolean circuit is a directed acyclic graph with three types of labeled nodes: inputs, gates and outputs. Inputs are the sources of the graph (i.e. nodes with fan-in 0) and are labeled with Boolean variables. Outputs are the sinks of the graph (i.e. nodes with fan-out 0) and carry the values of the circuit output. Gates are nodes labeled with Boolean functions such as AND ($\wedge$), OR ($\vee$), and NOT ($\neg$) (the fan-in of "$\neg$" is 1). Circuit size is defined as the number of gate nodes in the graph and circuit depth is the number of nodes in the longest path from an input node to an output node. As depicted in Figure 2.1, if we number all the input nodes, gate nodes and output nodes, then all the wires are numbered as well. That is, a node is assigned the same number as the wire carrying its result. A node

(e.g. node 2 in Figure 2.1) may have more than one outgoing wires (i.e., fan-out is larger than 1), but those wires are considered as the same since they carry the same result.

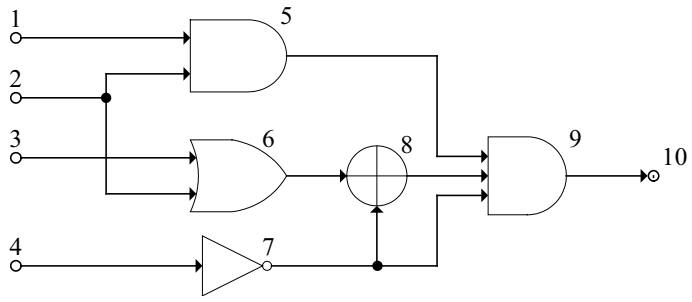

Figure 2.1: A Boolean circuit C with nodes sequentially numbered.

2.1.1 Assumptions

A circuit C with m outputs can be viewed as m 1-output subcircuits (Boolean functions) $C_1, \cdots, C_m$, where their inputs coincide with C's and the output of each C_i $(1 \leq i \leq m)$ equals to the i-th output of C. Thus, for simplicity, we assume hereafter that C has n inputs, s gates, and only 1 output.

We also assume in this thesis that each gate (of a circuit) has fan-in 2 since any k-bounded fan-in circuit can be simulated by a 2-bounded fan-in circuit with an expansion factor of $O(k)$ in circuit size and $O(\log k)$ in circuit depth. In addition, we assume that the functions of gates are drawn from the basis

$$\Omega = \{g \mid g : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}\} \ . \tag{2.1}$$

In other words, a gate g can be either a basic gate (e.g. $g(a,b)=a\vee b$), a non-trivial gate (e.g. $g(a,b)=a\wedge\bar{b}$) or even a degenerate gate (e.g. $g(a,b)=a$), where a degenerate gate takes as inputs two wires and outputs one of their results or a Boolean constant (i.e. neither of the inputs), and a non-trivial gate is internally made of several basic gates. We note that NOT gate (gate of fan-in 1) is not necessary as it can be integrated into its adjacent gate, e.g., $g(a,b)$ followed by a NOT gate can be simplified as $g'(a,b)= g(a,b)\oplus 1$. As the basis Ω has 16 elements, we can efficiently represent a gate function using a truth table, namely, the function of $g(a,b)$ is written in the fixed order of

$$[g(0,0),\ \ g(0,1),\ \ g(1,0),\ \ g(1,1)]\ . \tag{2.2}$$

2.1.2 Definitions

We can write input x, circuit C, intermediate results (IR) and output $C(x)$ as follows:

$$\begin{aligned} & v_1, \cdots, v_n \\ & v_{n+1} = g_{n+1}(v_{a_1}, v_{b_1}) \\ & \vdots \\ & v_{n+s} = g_{n+s}(v_{a_s}, v_{b_s}) \end{aligned} \tag{2.3}$$

where v_i denotes the result of node i during the computation of C on input $x{=}v_1\cdots v_n$, $IR{=}v_{n+1}\cdots v_{n+s-1}$, $C(x){=}v_{n+s}$, $a_i < b_i < n+i$ for all $1 \le i \le s$, and hence C can

be defined (or encoded) as

$$(n, s, 1), (a_1, b_1, g_{n+1}), (a_2, b_2, g_{n+2}), \ \cdots, \ (a_s, b_s, g_{n+s}) \ . \tag{2.4}$$

We also define the graph representation, namely the circuit topology of C (denoted by $T(C)$) as

$$(n, s, 1), (a_1, b_1), (a_2, b_2), \ \cdots, \ (a_s, b_s) \ . \tag{2.5}$$

In other words, given $T(C)$, we can construct the same directed acyclic graph as C. The only difference is that the graph is unlabeled (i.e. no function is assigned to any gate node).

2.2 Preliminaries to Cryptography

In this section, we introduce some basic concepts, notions and proof techniques in cryptography that are related to or used in our work.

2.2.1 Encryption Schemes and Privacy Homomorphism

Basically, a fixed-length encryption scheme (i.e. block cipher) is a triplet (G, E, D) satisfying the following conditions (see also [7, Definition 5.1.1]):

1. On input 1^t, probabilistic key generator G outputs a pair of strings (e, d), where 1^t is the security parameter, e is the encryption key and d is the de-

cryption key.

2. There exists a polynomially bounded function $\ell : \mathbb{N} \to \mathbb{N}$, called the block length, such that for any pair (e, d) produced by $G(1^t)$ and for any $x \in \{0,1\}^{\ell(t)}$, it holds that

$$\Pr[\ D_d(\ E_e(x)\ ) = x\ ] = 1 \tag{2.6}$$

where E_e denotes the encryption function under encryption key e, D_d denotes the decryption function under decryption key d and the probability is taken over the internal coin tosses of E (E might be probabilistic while D is always deterministic).

The above definition does not distinguish between private-key encryption schemes and public-key ones. In private-key schemes, e and d can be inferred from each other and thus we assume for simplicity the private key k=e=d. In public-key schemes, e is public and it should be computationally infeasible to infer d from e.

For a block cipher (G, E, D), if there exists a polynomial-time algorithm $\boxplus$ such that for any pair of (e, d) produced by $G(1^t)$, the following equality holds

$$D_d(\ \boxplus(\ E_e(x),\ \ E_e(y)\ )\ ) = x + y \tag{2.7}$$

where the algebraic structure of the plaintext space is a ring and "+" denotes addition over the ring, then (G, E, D) is additively homomorphic or we say (G, E, D, $\boxplus$) is an additive PH (see e.g. [8]). Analogously, we can define a multiplicative PH, (G, E, D, $\boxtimes$), using almost the same definition as the additive PH except that (2.7) is replaced by (2.8):

$$D_d(\ \boxtimes(\ E_e(x),\ \ E_e(y)\ )\ ) = x \times y \tag{2.8}$$

where the algebraic structure of the plaintext space is a ring and "$\times$" denotes multiplication over the ring. An example of multiplicative PH is the RSA algorithm [9]. If a PH is both additive and multiplicative, it is called an algebraic PH.

Rivest, Adleman and Dertouzos [8] first proposed the notion of PH in [10], where they gave four PHs but were later shown vulnerable to either ciphertext-only attacks [1] or known-plaintext attacks. Goldwasser and Micali [12] presented an additive PH (the GM crypto-system) with the block length $\ell(t)$=1 (i.e. regardless of the security parameter) and it is semantically secure if the Quadratic Residuosity Assumption holds. More efficient GM crypto-system variants are proposed in [13, 14]. There are also other semantically secure additive PHs (e.g. [15, 16]). Domingo-Ferrer proposed two algebraic PHs [17, 18] claimed to be provably secure, but the cryptanalysis was given in [19, 20, 21]. Sander and Young and Yung [22] proposed

[1] Detailed description about ciphertext-only attacks, known-plaintext attacks and other types of attacks can be found at cryptography textbooks such as [11].

an inefficient but unconditionally secure algebraic PH. Boneh, Goh and Nissim [23] proposed an efficient and provably secure algebraic PH that only supports one multiplication operation.

Feigenbaum and Merritt [24] questioned the existence of secure algebraic PH. Ahituv, Lapid and Neumann [25] showed that any additive PH whose $\boxplus$ is a simple addition function is insecure [2] under chosen-plaintext attacks. Boneh and Lipton [26] showed that any deterministic algebraic PH can be broken in sub-exponential time under reasonable assumptions, which is not surprising in the sense that (stateless) deterministic encryption schemes are not secure in the standard privacy notion [3] [7, Definition 5.2.1].

2.2.2 Pseudo-random Generators and Linear Feedback Shift Registers

A pseudo-random generator (PRG) is a polynomial-time function $\mathcal{G} : \{0,1\}^t \rightarrow \{0,1\}^*$ that expands a t-bit random seed k into a sequence of arbitrary length. In addition, it should be computationally infeasible for any polynomial-time adversary to distinguish between $\mathcal{G}(k)$ and a truly random sequence. PRG can be used to construct a synchronous stream cipher that consists of an encryption function

[2] This result is sometimes falsely taken as that no additive PH can be secure against chosen-plaintext attacks.

[3] This notion refers to semantic security (or equivalently computational indistinguishability). Under (stateless) deterministic encryption, there is a one-to-one correspondence between plaintexts and ciphertexts, and an eavesdropper can recognize known ciphertexts and perform a statistical analysis on a large number of ciphertexts.

$E_k(m)$=$m \oplus \mathcal{G}(k)$ and a decryption function $D_k(c)$=$c \oplus \mathcal{G}(k)$, where random seed k serves as the private encryption/decryption key, m, c and $\mathcal{G}(k)$ are of the same bit length and $\oplus$ is the bitwise XOR operator. The stream cipher can be viewed as approximating the action of the well-known one-time pad [27] except that encryption key k is much shorter than m and it is only secure against polynomial-time adversaries.

There are several classes of PRGs such as linear congruential generators [28], Blum Blum Shub generators [29], block cipher based generators [30] and those constructed with linear feedback shift registers (LFSRs). Of all the PRGs, LFSR-based ones are the fastest and easiest to implement in both hardware and software (they are even faster than natural binary counters).

An LFSR, denoted by $\mathcal{L}$, is a finite state machine that consists of a t-bit inner state $S_i = (s_i, \cdots, s_{i+t-1})$, a $t \times t$ update bit matrix M such that S_{i+1}=$S_i \cdot M$ and a 1-bit output, denoted by $\mathcal{L}[i]$, produced at each state S_i. As shown in Figure 2.2, the inner state of $\mathcal{L}$ is initialized with S_0=$(s_0, \cdots, s_{t-1})$. At each state S_i=$(s_i, \cdots, s_{i+t-1})$, it shifts the inner state by 1 bit to the left with s_i discarded as output (i.e. $\mathcal{L}[i]$=s_i) and it enters the next state S_{i+1}=$(s_{i+1}, \cdots, s_{i+t})$, where

$$s_{i+t} = \sum_{j=0}^{t-1} a_j s_{i+j} \tag{2.9}$$

and a_j indicates whether there is an XOR tap on the $(j+1)^{th}$ memory cell or not

(see Figure 2.2). By properly choosing the feedback vector [4] $(a_0, a_1, \cdots, a_{t-1})$, an LFSR can produce a sequence with the maximal period 2^t-1, namely, $(S_0, S_1, \cdots)$ will iterate all possible states except the state of all zero. Now $S_{i+1}=S_i \cdot M$ can be written as:

$$(s_{i+1}, \cdots, s_{i+t}) = (s_i, \cdots, s_{i+t-1}) \cdot \begin{pmatrix} 0 & 0 & & 0 & a_0 \\ 1 & 0 & \cdots & 0 & a_1 \\ 0 & 1 & & 0 & a_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & a_{t-1} \end{pmatrix} . \tag{2.10}$$

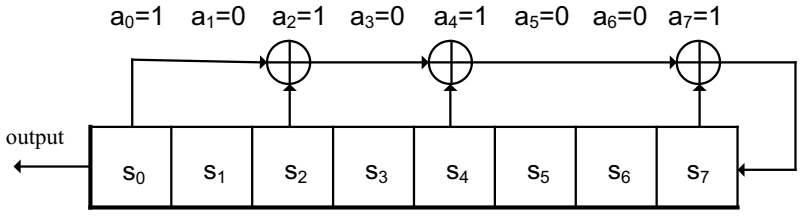

Figure 2.2: An 8-bit LFSR at its initial state $S_0=\{s_0, \cdots, s_7\}$ with feedback vector $a_0 \cdots a_7 = 10101001$.

Due to the ease of construction from simple electronic circuits, long periods, and uniformly distributed outputs (cf. [31] for further details), LFSRs are often used as building blocks for stream ciphers. Although the sequence of LFSR is subject to easy cryptanalysis due to its high linearity, this problem can be solved by introducing non-linearity with any of the following techniques:

[4] A t-bit LFSR produces a sequence of period 2^t-1 (a PN-sequence) if and only if the feedback polynomial (mod 2) $x^t+a_{t-1}x^{t-1}+a_{t-2}x^{t-2} + \cdots + a_0$ is a primitive polynomial [31].

1. Non-linear combination of the outputs of two or more LFSRs, e.g., non-linear filtering generators [32].

2. Irregular clocking of LFSR(s), e.g., shrinking generators [33] constructed with two LFSRs and self-shrinking generators [34] constructed with a single LFSR.

We will use the latter technique as it requires only one LFSR. Moreover, despite some analyses (e.g. [35, 36]), there are as yet no results on the cryptanalysis of either shrinking or self-shrinking generators (see also the comments by the RSA security laboratory [37]). The only drawback is irregular clocking, namely, the output rate of those generators is not constant unless precautions (e.g. buffering the output with a memory) are taken.

2.2.3 Computational Indistinguishability and the Simulation Paradigm

As a concept key to provable security and cryptography, computational indistinguishability [38, Section 3.2.1] is used to describe the similarity of two ensembles with regard to probabilistic polynomial-time algorithms. An ensemble of the form $X = \{ X_t \}_{t\in\mathbb{N}}$ is a sequence of random variables indexed by $\mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers and the length of each X_t is polynomial in t.

Definition 2.1 (Computational Indistinguishability)**.** *Two ensembles,* $X = \{ X_t \}_{t\in\mathbb{N}}$ *and* $Y = \{ Y_t \}_{t\in\mathbb{N}}$ *are computationally indistinguishable if for every proba-*

bilistic polynomial-time decision algorithm D, every positive polynomial $p(\cdot)$, and all sufficiently large t's, it holds that

$$| \ \Pr[D(X_t, 1^t) = 1] - \Pr[D(Y_t, 1^t) = 1] \ | < \frac{1}{p(t)} \tag{2.11}$$

The simulation paradigm is a basic technique to obtain elegant proofs for the security of encryption schemes and cryptographic protocols. For example, to prove that the ciphertext ensemble X generated by an encryption scheme reveals nothing substantial to polynomial-time adversaries, it suffices to show that there exists an efficient algorithm such that given only the same security parameter it can produce ensemble Y which is computationally indistinguishable to X.

2.3 Literature Review

We provide a classification of SCC protocols and the relationships among them, followed by a literature review and an introduction to our work.

2.3.1 A Taxonomy of SCC Protocols and Their Relations

In the problem of SCC, Alice holds circuit C and input x, and wants Bob to compute $C(x)$ for her such that sensitive information (SI) is hidden from Bob, where SI can

be defined as a nonempty subset of

$$\{x,\ C,\ IR,\ C(x)\}\ . \tag{2.12}$$

If we denote by $\mathcal{P}(SI)$ the SCC protocol capable of hiding SI (and SI only) from Bob, then there are at most 2^4-1 possible types of protocols of $\mathcal{P}(SI)$. Due to the following restrictions:

- SI must contain either x or C or both, namely, $\{x, C\} \bigcap SI \neq \phi$, otherwise, the protocol is not possible as Bob can infer IR and $C(x)$ from x and C.
- If $C \in SI$, then $IR \in SI$. It does not make too much sense that Bob learns IR without any knowledge of C.
- If $C \notin SI$ and $IR \notin SI$, then $C(x) \notin SI$. In most cases, $C(x)$ can be determined by C and IR.

we only consider those meaningful $\mathcal{P}(SI)$'s with their relations to each other shown in Figure 2.3 which is a directed graph and is justified by Proposition 2.1 and Proposition 2.2.

Proposition 2.1. *If there exists a non-trivial $\mathcal{P}(SI_1)$ and $SI_2 \subseteq SI_1$, then there exists a non-trivial $\mathcal{P}(SI_2)$ which takes one more round than the $\mathcal{P}(SI_1)$ does.*

Proof. $\mathcal{P}(SI_2)$ can be obtained by carrying out $\mathcal{P}(SI_1)$ and then revealing SI_1-SI_2 to Bob. □

Proposition 2.2. *If there exists a non-trivial $\mathcal{P}(\{C, IR\})$, then there exists a non-trivial $\mathcal{P}(\{x, C, IR, C(x)\})$ which takes the same number of rounds as $\mathcal{P}(\{C, IR\})$ does.*

Proof. $\mathcal{P}(\{x, C, IR, C(x)\})$ can be constructed as follows: Alice randomly selects a pair of encryption and decryption keys (e, d) and prepares a circuit C' that takes $E_e(x)$ as input, decrypts $E_e(x)$ to get x, computes $C(x)$ and produces $E_e(C(x))$ as output. Then Alice executes $\mathcal{P}(\{C', IR'\})$ with Bob and finally decrypts $E_e(C(x))$ to get $C(x)$. □

We note that " $\rightarrow$" is a transitive relation. That is, if in Figure 2.3 there is a directed path from $\mathcal{P}(SI_1)$ to $\mathcal{P}(SI_2)$, then it holds that $\mathcal{P}(SI_1)\rightarrow\mathcal{P}(SI_2)$.

2.3.2 Circuit Obfuscation ($\mathcal{P}(\{C, IR\})$)

Circuit obfuscation can be viewed as a 2-round $\mathcal{P}(\{C, IR\})$: Alice has circuit C and input x, and wants Bob to compute $C(x)$ for her in such a manner that Bob

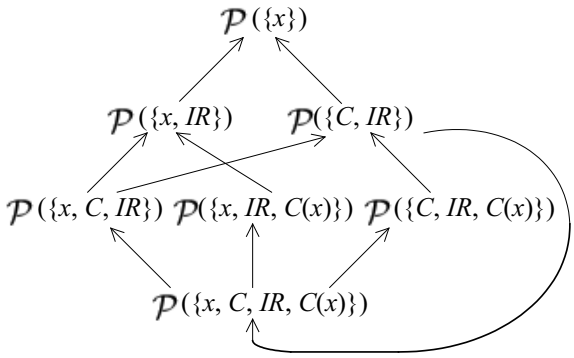

Figure 2.3: The protocol relation diagram, where $\mathcal{P}(SI_1)\rightarrow\mathcal{P}(SI_2)$ denotes that the existence of non-trivial $\mathcal{P}(SI_1)$ implies that of non-trivial $\mathcal{P}(SI_2)$.

learns nothing substantial about C and IR. Thus, Alice transforms C into a circuit C', sends C' together with x to Bob, and Bob computes and sends $C'(x)$ back to Alice, where C' satisfies the following conditions:

1. (function): C' computes the same function as C does, i.e., for any valid input x, it holds that $C'(x)$=$C(x)$.

2. (overhead): Compared with C, C' is increased by at most a polynomial factor in circuit size and depth.

3. (privacy [5]): For any x, (x, C', IR', $C'(x)$) reveals nothing substantial about C and IR to computationally bounded Bob, where IR and IR' are respectively the intermediate results of C and C' on input x.

If the above 2-round $\mathcal{P}(\{C, IR\})$ exists, C is obfuscatable. Unfortunately, Barak et al. [39] have proved that universal circuit obfuscators do not exist, where by "universal" they do not rule out that some classes of circuits are obfuscable under cryptographic assumptions [40]. Thus, for most circuits, there does not exist such a non-trivial 2-round protocol for $\mathcal{P}(\{C, IR\})$.

2.3.3 Computing with Encrypted Data ($\mathcal{P}(\{x, IR, C(x)\})$)

Computing with Encrypted Data (CED) [41] is a 2-round $\mathcal{P}(\{x, IR, C(x)\})$: Alice has circuit C and input x, and wants Bob to compute $C(x)$ for her such that Bob

[5]The privacy condition is equivalent to the black-box property [39]: Anything that can be efficiently computed from C can be efficiently computed given oracle access to C.

learns nothing substantial about x, IR and $C(x)$. Therefore, Alice encrypts x to produce $y{=}E_k(x)$ and sends y and C to Bob. Bob computes C on y to get $C(E_k(x))$, which is then sent back to Alice and decrypted to obtain $C(x)$. Note that here for brevity $C(E_k(x))$ is not a rigorous notation as it does not denote the result of feeding $E_k(x)$ to C, but refers to an encrypted message of $C(x)$. More details about CED can be found in [41].

If the above 2-round $\mathcal{P}(\{x, IR, C(x)\})$ exists, C is computable with encrypted data. The major difficulty of CED is how Bob can obliviously obtain $E_k(C(x))$ from $E_k(x)$ and C without knowing the encryption and decryption keys. It has been shown that C is computable with encrypted data when C computes a polynomial function and E_k is algebraically homomorphic [24]. Moreover, Abadi, Joan Feigenbaum and Kilian [41] show that the discrete logarithm problem and the primitive root problem are also computable with encrypted data against unboundedly powerful Bob. However, it is yet unknown whether or not 2-round $\mathcal{P}(\{x, IR, C(x)\})$ exists for a generic circuit C.

2.3.4 Computing with Encrypted Functions ($\mathcal{P}$ ({ C, IR, $C(x)$ }))

Computing with Encrypted Functions (CEF) [42] is a 2-round $\mathcal{P}(\{C, IR, C(x)\})$: Alice has a circuit C and wants Bob to compute $C(x)$ for her without revealing anything substantial more than input x to Bob. Thus, Alice encrypts C and sends

to Bob $E_k(C)$ and x. Bob computes $E_k(C)$ on x to get $E_k(C(x))$, which is then sent back to Alice and decrypted to obtain $C(x)$.

If the above 2-round $\mathcal{P}(\{C, IR, C(x)\})$ exists, C is CEF computable. Sander and Tschudin [43] show that polynomial functions are CEF computable by using additive PH, where polynomials are hidden by encrypting their coefficients and only their orders are revealed to Bob. Inspired by the McEliee public-key cryptosystem [44], Loureiro [45] propose a CEF protocol for generic circuits, but it is not practical as the size of encrypted circuit blows up exponentially. To the best of our knowledge, no efficient $\mathcal{P}(\{C, IR, C(x)\})$ has been proposed.

2.3.5 Yao's Two-party Computation Protocol ($\mathcal{P}(\{x, IR\})$)

Yao's secure two-party computation protocol [46] can be loosely regarded as a variant of multiple-round $\mathcal{P}(\{x, IR\})$: Alice and Bob with their private inputs x and y respectively, want to compute $C(x, y)$ for a publicly known circuit C without disclosing their private inputs to each other. Thus, Alice sends to Bob the encrypted format of C and x, namely $E_k(C)$ and $E_k(x)$. Bob then executes a number of 1-out-of-2 oblivious transfers (OTs) [47, 48] in parallel with Alice such that Bob gets y encrypted without revealing y to Alice. Finally, Bob computes $E_k(C)$ on $E_k(x)$ and $E_k(y)$ to get $E_k(C(x,y))$, which is decrypted by Alice to obtain $C(x,y)$.

We use RSA encryption as an example that how a 1-out-of-2 OT can be constructed. Alice have two messages m_0 and m_1 and Bob has a bit b. Bob wants to

learn m_b without disclosing b to Alice while Alice wants to ensure that Bob learns only one message (m_b). Thus, they engage in the following steps:

1. Alice prepares RSA parameters: modulus N, public key e, and private key d, picks two random messages r_0 and r_1, and sends N, e, r_0, and r_1 to Bob.

2. Bob chooses a random message r, encrypts it, and sends $E_e(r)+r_b$ to Alice.

3. Alice decrypts $E_e(r) + r_b - r_0$ and $E_e(r) + r_b - r_1$ respectively and sends $D_d(E_e(r) + r_b - r_0) + m_0$ and $D_d(E_e(r) + r_b - r_1) + m_1$ to Bob.

4. Bob knows which message he wants (b) and subtracts it with r to obtain m_b.

If Alice and Bob are both semi-honest, the above protocol is constant-round as required by OT. Otherwise, it may take more rounds since zero knowledge proof is needed to force honest behaviors. Goldreich, Micali and Wigderson [49] improved the protocol by detailing how to do circuit encryption and extending it to the multi-party case. Yao's protocol is applicable to generic circuits and, if properly constructed, can be proved to be computationally secure [50, 7]. However, it has some limitations such as non-reusable keys and information disclosure of C, which reduce its practicability (see the detailed discussion in Chapter 3).

2.3.6 Our Work ($\mathcal{P}(\{x, C, IR, C(x)\})$)

As surveyed in this chapter, existing protocols are either restricted to certain classes of circuits or not efficient or private enough. In this thesis, we concentrate on a

2-round $\mathcal{P}(\{x,\ C,\ IR,\ C(x)\})$: Alice has a circuit C and wants Bob to compute $C(x)$ for her without revealing Bob anything substantial. Thus, Alice encrypts C and x and sends to Bob $E_k(C)$ and $E_k(x)$. Bob computes $E_k(C)$ on $E_k(x)$ to get $E_k(C(x))$, which is then sent back to Alice and decrypted to obtain $C(x)$. By Proposition 2.1, if such a protocol exist, then it implies other protocols that hide any partial information regarding the computation by appending one more round.

Chapter 3

A One-time-use SCC Protocol for $\mathcal{P}(\{x, C, IR, C(x)\})$

In this chapter, we construct a protocol for $\mathcal{P}(\{x, C, IR, C(x)\}$ with garbled circuits (in combination with a simple CTO technique) and prove that it is computationally secure as long as C is computed on only one input x under each encryption key. The protocol can be used to help weak-power devices to utilize external computing power in such a private way that nothing substantial is disclosed. For a complicated problem, it is not feasible to construct the corresponding circuit by hand, so we detail in Appendix A how we design a compiler that takes user-written C-style code as input and generates a Boolean circuit computing the same function.

3.1 Protocol Preparation

3.1.1 Application Scenario of the One-time-use Protocol

Modern smart cards can do basic cryptographic computation such as 3DES, AES or even RSA with cryptographic co-processors. However, they lack powerful processors and sufficient memory for more advanced applications. For example, some smart cards are using fingerprint verification to authenticate their cardholders as it is safer than the traditional personal identification number (PIN) verification due to the fact that fingerprints are distinctive and can hardly be borrowed. However, unlike PIN verification, fingerprint verification involves fingerprint image processing, which is computationally intensive and hence it cannot totally be done on smart cards. In general, there are three ways to achieve a fingerprint verification:

1. **Match-off-card (a.k.a. template-on-card)**. A fingerprint image is pre-stored (during the enrollment phase) on the smart card as a template. During the verification, a powerful workstation samples the fingerprint of the cardholder and compares it against the template, which is read from the smart card.

2. **Match-on-card** [51]. During the verification, a workstation scans the fingerprint of the cardholder and conducts heavy pre-processing work on it to produce a compact format (called minutiae). Then the compact data is sent to the smart card, where the matching is performed between it and the tem-

plate.

3. **System-on-card** [52]. A smart card is integrated with a fingerprint scanner and is powerful enough to perform the fingerprint verification all by itself.

In terms of privacy, system-on-card would be optimal but currently only prototypes are available and none of them meets the requirements of the ISO specification [53], namely thickness and torsion tests. Both match-off-card and match-on-card assume a workstation willing to provide computing service, which leads to a potential privacy breach. That is, in the match-off-card scenario, the workstation knows the template of the cardholder as it is required for the verification. Match-on-card is more secure than match-off-card in that the template never goes to the workstation. Nevertheless, the workstation can still learn information regarding the template indirectly by storing those samples that successfully match the template.

Therefore, under some circumstances, weak-power devices have to utilize external computing power at the cost of their privacy. The scenario can be described as follows: Alice, a weak-power device (e.g. a smart card), has a private program p. When roaming around the neighborhood, she would capture some private input x but has to ask her neighbor Bob to compute $p(x)$. In addition, before leaving home, Alice could let her home PC to do some intensive pre-computation, e.g., to transform p into circuit C (to invoke an SCC protocol). Note that the PC cannot be easily carried away due to its size and weight, neither can it compute $p(x)$ in advance since x is obtained after Alice leaves home. Given that Bob can be semi-honest or even

malicious, how can an SCC protocol be carried out such that Alice obtains $p(x)$ correctly while Bob learns nothing substantial?

3.1.2 Transforming Programs into Boolean Circuits

The most significant difference between programs and circuits, in terms of computability, is that programs support variable-length inputs and outputs whereas circuits deal with only fixed-length inputs and outputs. In other words, to simulate a generic program p that takes arbitrary-length inputs, we need a family of circuits $\{C_n\}_{n\in\mathbb{N}}$, where each C_n takes n-bit inputs. Nevertheless, in practice, many programs only deal with fixed-length inputs to produce fixed-length outputs. For example, a fingerprint verification program takes as input a fingerprint image of fixed resolution and gray-scale depth, compares it with the template, and produces a 1-bit output indicating whether the matching is successful or not. Thus, we assume hereafter that program p only takes fixed-length inputs and outputs.

Informally, a program is a sequence of instructions that can be executed on a computing device. According to the von Neumann architecture (the most prevailing computer architecture), a computing device consists of a register (accumulator), a memory and a CPU that can perform the following instructions [54]:

- **Load.** Copy the value of a memory location to the register.
- **Store.** Store the value of the register to a memory location.
- **Add.** Perform addition and store the sum into the register.

- **Complement.** Flip the value of the register.
- **Jump.** Perform an unconditional branching.
- **JumpZ.** Perform a conditional branching based on the value of the register.
- **Halt.** Halt the execution of the program.

Present computers may have some enhanced instructions (e.g. subtraction, multiplication and division) and more registers to facilitate execution, but the instructions listed above are sufficient in the sense that they can efficiently simulate any other instruction. We establish the possibility of transforming polynomial-time programs into polynomial-size circuits with the following two steps:

1. A program is polynomial-time computable if there is a polynomial $poly$ such that for any n-bit input, it will halt within $poly(n)$ instructions. It has been shown that each of the above instructions can be simulated by a Turing machine in polynomial steps and consequently problems solvable by a (fixed-length-input-output) polynomial-time program can also be solved by a Turing machine in polynomial time [54, Theorem 1.3].

2. Every polynomial-time Turing machine running on n-bit inputs can be simulated by a $poly(n)$-size Boolean circuit [55]. Note that a program can take inputs of various lengths whereas a Boolean circuit only takes fixed-length inputs. Thus, a polynomial-time program would functionally equivalent to a family of Boolean circuits with size being polynomial in the problem size (i.e.

input length n). For example, the computation time of a program that bubble sorts $n/8$ 8-bit-long integers shall be of order $O(n^2)$ and accordingly there are a family of $O(n^2)$-size Boolean circuits $\{C_n\}_{n/8\in\mathbb{N}}$, where each C_n takes $n/8$ 8-bit-long integers and produces the sorted integers as output.

Although theoretically possible, we still need an automated tool to facilitate the conversion of polynomial-time programs to polynomial-size circuits. Malkhi et al. [56] have implemented a compiler that inputs programs and outputs corresponding Boolean circuits, where the programs are written in C-style code except the following major differences:

- Integers can be declared as any bit long (10-bit, 50-bit, 100-bit etc.). This is different from most programming languages which have pre-defined length (e.g. 16-bit and 32-bit) for integers.
- No pointers and variable array indexes are allowed.
- The "for" statement (e.g. "for $i = c1$ to $c2$ do") should have a predictable (at compile time) number of iterations (i.e. $c2-c1$ is a constant).

As detailed in Appendix A, we develop a compiler independently and our compiler is more powerful than Malkhi et al's in that it supports advanced arithmetic operations (e.g. multiplication, truncating division, rounding division and modular arithmetic) and statements (e.g. the "Switch" statement). Such a compiler is sufficient for most polynomial-time programs because we can find an upper bound (i.e. the worst case)

for a loop whose number of iterations cannot be determined at compile time. For example, Figure 3.1.a is the C code for the bubble sort problem, where variable $HaveSwapped$ is used to prevent redundant loops and improve efficiency. Our code, as shown in Figure 3.1.b, cannot use $HaveSwapped$ as the compiler does not allow "break", which is unpredictable at compile time. In addition, we also use A_i instead of $A[i]$. Thus, we cannot index A by changing the value of i, which means that the two loops in Figure 3.1.b are macros and will be unrolled by the compiler. Table 3.1 illustrates the numbers of Boolean gates needed to simulate basic operations between two operands. For a polynomial-time program, the number of basic operations after unrolling all loops is a polynomial $poly$ (in input length) and the number of gates needed for each basic operation is also bounded by another polynomial $poly'$ (see Table 3.1). Therefore, the resulting Boolean circuit is polynomial-size (i.e. $poly \times poly'$). For instance, after unrolling the two loops in Figure 3.1.b, we have

$$(n-1) + (n-2) + \cdots + 1 = (n-1)n/2 \tag{3.1}$$

code blocks with each block simulated by

$$1 + (4L - 3) + 3 \times L = 7L - 2 \tag{3.2}$$

gates. Thus, the number of gates in the corresponding circuit is no more than $(7/2) \cdot n^2 L$. As the length of the program input (i.e., $A_0, \cdots, A_{n-1}$) is $n \cdot L$, it follows

that $(7/2)\cdot n^2 L$ is a polynomial in $n\cdot L$.

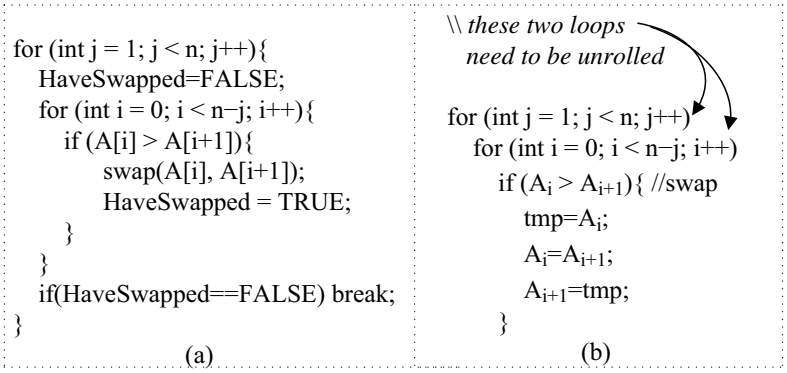

Figure 3.1: (a) Bubble sort in C code. (b) Bubble sort in C-style code for our compiler.

Table 3.1: Numbers of gates (generated by our compiler) to simulate basic operations between L-bit-long operands A and B in cases of fan-in bounded by 2 and that bounded by 3.

operation	no. of gates with fan-in upper bounded by 2	no. of gates with fan-in upper bounded by 3
$A\pm B$ (addition / substraction)	$5L-3$	$2L$
$A\times B$ (multiplication)	$6L^2-8L+3$	$3L^2$
$A\&B$, $A\|B$, $A\oplus B$ (bitwise operation)	L	L
$A:=B$ (value assignment within "IF")	$3L$	L
$A>B$, $A\geq B$, $A<B$, $A\leq B$ (comparison)	$4L-3$	L
$A=B$, $A\neq B$ (comparison)	$2L-1$	L
entering an "IF" block	1	1

3.2 A Preliminary One-time-use SCC Protocol

Now that we can produce circuits with a dedicated compiler, it remains to introduce the one-time-use protocol that enables secure computation of Boolean circuits. In

this section, we present a preliminary protocol constructed with Yao's garbled circuits. As Yao gave no detail on the construction, Goldreich et al. [49], Rogaway [50], Beaver [57], Naor et al. [58] and Lindell et al. [59] each proposed a "garbled circuit construction" variant, but only Rogaway and Lindell et al. gave rigorous [1] proofs for the security. We propose a garbled circuit construction method based on PRGs and prove that only the topology of C (i.e. $T(C)$) is revealed to Bob.

3.2.1 An Overview of the Preliminary Protocol

The protocol can be summarized as follows:

Protocol 3.1 (The preliminary SCC protocol)**.**

1. *(Pre-computation): On behalf of Alice, Alice's home PC randomly selects a key k, garbles C to get $E_k(C)$ and produces the corresponding input encryption and output decryption table Tab, where Tab will be securely stored by Alice and $E_k(C)$ is revealed to Bob.*

2. *(Secure Computation): When roaming round the neighborhood, Alice gets x, encrypts it (by looking it up in Tab) and sends the encrypted input $E_k(x)$ to Bob. Bob computes $E_k(C)$ on $E_k(x)$ to obtain $E_k(C(x))$.*

3. *(Result Validation): Upon receiving $E_k(C(x))$ from Bob, Alice checks the genuineness (i.e. whether Bob cheats or not) of the result and decrypts it to get $C(x)$, which are done with the aid of Tab.*

[1] Tate and Xu claimed that Rogaway's construction and proof are slightly flawed [60].

The above can be viewed as a 2-round protocol if $E_k(C)$ and $E_k(x)$ are sent to Bob as a single message.

3.2.2 Pre-computation

For an n-input-s-gate-1-output circuit C encoded as (see Section 2.1.2 for detailed encoding scheme)

$$(n, s, 1), (a_1, b_1, g_{n+1}), (a_2, b_2, g_{n+2}), \ \cdots \ , \ (a_s, b_s, g_{n+s}) \ , \tag{3.3}$$

Alice's home PC uniformly and randomly selects a string

$$k = W_1^0 W_1^1 c_1 W_2^0 W_2^1 c_2 \cdots W_{n+s-1}^0 W_{n+s-1}^1 c_{n+s-1} W_{n+s}^0 W_{n+s}^1 \tag{3.4}$$

from $\{0,1\}^{|k|}$, where for $1 \leq i \leq n$ (resp., $n+1 \leq i < n+s$) $W_i^0 W_i^1 c_i$ corresponds to input (resp., intermediate-gate) node i, $W_{n+s}^0 W_{n+s}^1$ is with the last gate node carrying the output, for $1 \leq i < n+s$ and $0 \leq b \leq 1$, $|W_i^b| = fo_i \times t$, $|c_i| = fo_i$ (c_{n+s} is empty as $fo_{n+s}=0$), $|W_{n+s}^b| = t$, fo_i is the fan-out of node i and t is the security parameter. For each g_{n+i} ($1 \leq i \leq s$) represented using truth table $[g_{n+i}(0,0), g_{n+i}(0,1), g_{n+i}(1,0), g_{n+i}(1,1)]$, its truth table is first transformed into

$$[W_{n+i}^{g_{n+i}(0,0)} \cdot {}^{g_{n+i}(0,0)}c_{n+i}, \ W_{n+i}^{g_{n+i}(0,1)} \cdot {}^{g_{n+i}(0,1)}c_{n+i}, \ W_{n+i}^{g_{n+i}(1,0)} \cdot {}^{g_{n+i}(1,0)}c_{n+i}, \ W_{n+i}^{g_{n+i}(1,1)} \cdot {}^{g_{n+i}(1,1)}c_{n+i}] \ , \tag{3.5}$$

where "·" denotes string concatenation and c_{n+i}^{b} denotes a flipped string obtained by XORing every bit of string c_{n+i} with bit b. Suppose that gate g_{n+i} is the p-th ($1 \leq p \leq fo_{a_i}$) gate that takes node a_i as input and the q-th ($1 \leq q \leq fo_{b_i}$) gate that takes node b_i as input, then the above table (see (3.5)) is further encrypted and permuted using $(W_{a_i}^0[p], W_{a_i}^1[p], c_{a_i}[p])$ and $(W_{b_i}^0[q], W_{b_i}^1[q], c_{b_i}[q])$, where $W[j]$ denotes the j-th t-bit-substring of W and $c[j]$ is the j-th bit of c. The encryption is done by XORing each $W_{n+i}^{g_{n+i}(\alpha,\beta)} \cdot c_{n+i}^{g_{n+i}(\alpha,\beta)}$ in (3.5) with $X_{c_{b_i}[q]\oplus\beta}^{\alpha} \oplus Y_{c_{a_i}[p]\oplus\alpha}^{\beta}$, where

$$\begin{aligned} \mathcal{G}(W_{a_i}^{\alpha}[p]) = & \underbrace{x_1 \ldots x_{fo_i(1+t)}}_{X_0^{\alpha}} \underbrace{x_{fo_i(1+t)+1} \ldots x_{fo_i(2+2t)}}_{X_1^{\alpha}} , \\ \mathcal{G}(W_{b_i}^{\beta}[q]) = & \underbrace{y_1 \ldots y_{fo_i(1+t)}}_{Y_0^{\beta}} \underbrace{y_{fo_i(1+t)+1} \ldots y_{fo_i(2+2t)}}_{Y_1^{\beta}} , \end{aligned} \tag{3.6}$$

$0 \leq \alpha, \beta \leq 1$, and $\mathcal{G}$ is a PRG function. After encryption, permute the resulting table as follows:

$$[W_{00}^{'}, W_{01}^{'}, W_{10}^{'}, W_{11}^{'}] \rightarrow [W_{\pi_i(0,0)}^{'}, W_{\pi_i(0,1)}^{'}, W_{\pi_i(1,0)}^{'}, W_{\pi_i(1,1)}^{'}] , \tag{3.7}$$

where $\pi_i(\alpha, \beta) = (\alpha \oplus c_{a_i}[p]) \cdot (\beta \oplus c_{b_i}[q])$. In this way, the truth table of g_{n+i} is garbled (i.e. encrypted and permuted) and we denote the garbled table by $E_k(g_{n+i})$. Finally,

Alice sends the encrypted circuit

$$E_k(C) = (n, s, 1), (a_1, b_1, E_k(g_{n+1})), (a_2, b_2, E_k(g_{n+2})), \ \cdots, \ (a_s, b_s, E_k(g_{n+s})) \tag{3.8}$$

to Bob, and stores the input encryption and output decryption table

$$Tab = W_1^0 W_1^1 c_1 W_2^0 W_2^1 c_2 \cdots W_n^0 W_n^1 c_n W_{n+s}^0 W_{n+s}^1 \tag{3.9}$$

by herself.

3.2.3 Secure Computation and Result Validation

When outside, Alice captures $x{=}v_1 \cdots v_n$, encrypts it by looking it up in Tab and sends $E_k(x){=}W_1^{v_1} \overset{v_1}{c_1} \cdots W_n^{v_n} \overset{v_n}{c_n}$ to Bob. Upon receiving $E_k(x)$, Bob computes $E_k(C)$ as follows: For each gate g_{n+i} ($1{\leq}i{\leq}s$), Bob picks out the $\overset{v_{a_i}}{c_{a_i}}[p]{\cdot}\overset{v_{b_i}}{c_{b_i}}[q]$-th (00-th is the first, 01-th is the second and so on) item from $E_k(g_{n+i})$, XORs it with $(X^{v_{a_i}}_{c_{b_i}[q]\oplus v_{b_i}} \oplus Y^{v_{b_i}}_{c_{a_i}[p]\oplus v_{a_i}})$ to obtain $W_{n+i}^{g_{n+i}(v_{a_i}, v_{b_i})} \cdot \overset{g_{n+i}(v_{a_i}, v_{b_i})}{c_{n+i}}$, where we note that $\overset{v_{a_i}}{c_{a_i}}[p] = c_{a_i}[p] \oplus v_{a_i}$, $\overset{v_{b_i}}{c_{b_i}}[q] = c_{b_i}[q] \oplus v_{b_i}$ and $v_{n+i}{=}\ g_{n+i}(v_{a_i}, v_{b_i})$. Finally, Bob sends the result $W_{n+s}^{v_{n+s}}$ to Alice. Alice compares it against $W_{n+s}^0 W_{n+s}^1$ to determine whether $v_{n+s}{=}0$, or $v_{n+s}{=}1$, or that Bob cheats.

3.2.4 Proofs for the Security of the Protocol

Loosely speaking, the protocol is computationally secure against Bob if it meets the following conditions:

1. (correctness): In the semi-honest adversarial model, Alice correctly obtains the result after executing the protocol with Bob.

2. (privacy with respect to semi-honest Bob): In the semi-honest adversarial model, Bob learns nothing substantial from the protocol transcript.

3. (privacy with respect to malicious Bob): If Bob deviates from the protocol by sending Alice a fake result, then Alice can detect it with a probability only negligibly less than 1.

Theorem 3.1 (correctness with respect to semi-honest behavior)**.** *Let C be the circuit to be computed and x be its input, then the preliminary SCC protocol (Protocol 3.1) correctly computes C and x in the semi-honest model.*

Proof. In the semi-honest model, neither party deviates from the described protocol. Hence, it suffices to prove the correctness of computing $E_k(C)$. For each gate g_{n+i} ($1 \leq i \leq s$), the $r \cdot s$-th item of $E_k(g_{n+i})$ is

$$X_s^{c_{a_i}[p] \oplus r} \oplus Y_r^{c_{b_i}[q] \oplus s} \oplus (W_{n+i}^{g_{n+i}(c_{a_i}[p] \oplus r, c_{b_i}[q] \oplus s)} \cdot {}^{g_{n+i}(c_{a_i}[p] \oplus r, c_{b_i}[q] \oplus s)}_{C_{n+i}}) \ . \tag{3.10}$$

According to the protocol, Bob will pick out the $(c_{a_i}[p] \oplus v_{a_i}) \cdot (c_{b_i}[q] \oplus v_{b_i})$-th item

from $E_k(g_{n+i})$ and XOR it with $X^{v_{a_i}}_{c_{b_i}[q]\oplus v_{b_i}} \oplus Y^{v_{b_i}}_{c_{a_i}[p]\oplus v_{a_i}}$. Since the following equation holds when $r{=}c_{a_i}[p]\oplus v_{a_i}$ and $s{=}c_{b_i}[q]\oplus v_{b_i}$,

$$X_s^{c_{a_i}[p]\oplus r} \oplus Y_r^{c_{b_i}[q]\oplus s} = X^{v_{a_i}}_{c_{b_i}[q]\oplus v_{b_i}} \oplus Y^{v_{b_i}}_{c_{a_i}[p]\oplus v_{a_i}} \quad . \tag{3.11}$$

the resulting value is $W_{n+i}^{g_{n+i}(v_{a_i},v_{b_i})} \cdot {}^{g_{n+i}(v_{a_i},v_{b_i})}_{\quad c_{n+i}}$. Therefore, Bob can correctly compute each gate to produce output $W_{n+s}^{g_{n+s}(v_{a_s},v_{b_s})}$, which is decrypted by Alice to get $C(x){=}v_{n+s}{=}g_{n+s}(v_{a_s}, v_{b_s})$. □

As Bob receives $E_k(C)$ and $E_k(x)$, the protocol is not able to hide the circuit topology $T(C)$ from him. Thus, we define the privacy in the semi-honest model as in Definition 3.1 (see its analogue in [7, Definition 7.2.1]).

Definition 3.1 (privacy with respect to semi-honest behavior)**.** *Let C be the circuit to be computed and x be its input, denote by $VIEW_{C,x}$ the view of Bob during execution of the preliminary protocol (Protocol 3.1) on C and x , then Bob learns nothing substantial other than $T(C)$ if there exists a probabilistic polynomial-time algorithm (PPT) S such that the following two ensembles indexed by security parameter t are computationally indistinguishable, namely*

$$\{S(\ T(C)\ )\}_{t\in\mathbb{N}} \stackrel{c}{\equiv} \{VIEW_{C,x}\}_{t\in\mathbb{N}} \ , \tag{3.12}$$

where

$$VIEW_{C,x} = (E_k(C), E_k(x)) = (T(C), E_k(g_{n+1}), \cdots, E_k(g_{n+s}), W_1^{v_1}\ {}^{v_1}c_1, \cdots, W_n^{v_n}\ {}^{v_n}c_n). \tag{3.13}$$

Note that in each $E_k(g_{n+i})$ with $1 \leq i \leq s$, there are four encrypted items but only one (i.e. the ${}^{v_{a_i}}c_{a_i}[p] \cdot {}^{v_{b_i}}c_{b_i}[q]$-th) is selected and decrypted by Bob during the protocol execution. We call the selected item on-path and the other three off-path. Similarly, for each input bit, there are two strings that represent 0 and 1 respectively, and the one representing the value of the input bit is called on-path input string and the other off-path.

Lemma 3.1. *If we replace every $E_k(g_{n+i})$ $(1 \leq i \leq s)$ in $VIEW_{C,x}$ with $E_k(g'_{n+i})$ to produce*

$$VIEW' = (T(C), E_k(g'_{n+1}), \cdots, E_k(g'_{n+s}), W_1^{v_1}\ {}^{v_1}c_1, \cdots, W_n^{v_n}\ {}^{v_n}c_n). \tag{3.14}$$

where each $E_k(g'_{n+i})$ is constructed by replacing all the three off-path items with uniformly distributed random strings of the same length, then the following ensembles are computationally indistinguishable, namely

$$\{VIEW'\}_{t \in \mathbb{N}} \stackrel{c}{\equiv} \{VIEW_{C,x}\}_{t \in \mathbb{N}} \tag{3.15}$$

assuming the existence of PRGs.

Proof. We construct a hybrid walk (see similar techniques in [50, 60]) for the above equality:

$$\{VIEW'\}_{t\in\mathbb{N}} = \{VIEW'_0\}_{t\in\mathbb{N}} \rightarrow \{VIEW'_1\}_{t\in\mathbb{N}} \rightarrow \cdots \rightarrow \{VIEW'_s\}_{t\in\mathbb{N}} = \{VIEW_{C,x}\}_{t\in\mathbb{N}} \tag{3.16}$$

where $VIEW'_j$ $(0{\leq}j{\leq}s)$ is defined as

$$(T(C), E_k(g'_{n+1}), \cdots, E_k(g'_{n+s-j}), E_k(g_{n+s-j+1}), \cdots, E_k(g_{n+s}), W_1^{v_1}\ {}^{v_1}c_1, \cdots, W_n^{v_n}\ {}^{v_n}c_n). \tag{3.17}$$

For contradiction, we assume that (3.15) does not hold. Then, due to the transitivity of computational indistinguishability [50, Proposition 4.2.1], there is at least one discontinuous point in the constructed walk such that $\{VIEW'_h\}_{t\in\mathbb{N}}$ and $\{VIEW'_{h+1}\}_{t\in\mathbb{N}}$ $(0{\leq}h{\leq}s{-}1)$ are polynomial-time distinguishable. By definition, $VIEW'_{h+1}$ differs to $VIEW'_h$ only in that $E_k(g_{n+s-h})$ is replaced by $E_k(g'_{n+s-h})$. Without loss of generality [2], we assume that g_{n+s-h} is an XOR gate, $c_{a_{s-h}}[p]{=}1$, $v_{a_{s-h}}{=}0$, $c_{b_{s-h}}[q]{=}0$ and $v_{b_{s-h}}{=}1$, then the difference between $E_k(\ g_{n+s-h}\ )$ and $E_k(\ g'_{n+s-h}\ )$ can be tabulated as in Table 3.2. By the definition of PRG, the ensembles $\{E_k(g_{n+s-h})\}_{t\in\mathbb{N}}$ and $\{E_k(g'_{n+s-h})\}_{t\in\mathbb{N}}$ are computationally indistinguishable. The random seeds producing X_0^1, X_1^1, Y_0^0 and Y_1^0 either reside in the off-path items of gates preceding g_{n+s-h} or are off-path input strings not revealed to Bob. In addition,

[2] The function of g_{n+s-h} and the values of $c_{a_{s-h}}$, $v_{a_{s-h}}$, $c_{b_{s-h}}$ and $v_{b_{s-h}}$ have no effect on proving the lemma.

for both $VIEW'_{h+1}$ and $VIEW'_h$, all the off-path items of gates preceding g_{n+s-h} have been replaced with uniformly random strings. Therefore, none of X_0^1, X_1^1, Y_0^0 and Y_1^0 is correlated with any other part of $VIEW'_{h+1}$ and $VIEW'_h$, and it follows that $\{VIEW'_h\}_{t\in\mathbb{N}}$ and $\{VIEW'_{h+1}\}_{t\in\mathbb{N}}$ are also computationally indistinguishable, which is a contradiction. □

Table 3.2: A comparison between $E_k(g_{n+s-h})$ and $E_k(g'_{n+s-h})$.

Item index	$E_k(g_{n+s-h})$	$E_k(g'_{n+s-h})$
00	uniform random string	$X_0^1 \oplus Y_0^0 \oplus (W^1_{n+s-h} \cdot \overset{1}{c}_{n+s-h})$
01	uniform random string	$X_1^1 \oplus Y_0^1 \oplus (W^0_{n+s-h} \cdot \overset{0}{c}_{n+s-h})$
10	uniform random string	$X_0^0 \oplus Y_1^0 \oplus (W^0_{n+s-h} \cdot \overset{0}{c}_{n+s-h})$
11	$X_1^0 \oplus Y_1^1 \oplus (W^1_{n+s-h} \cdot \overset{1}{c}_{n+s-h})$	$X_1^0 \oplus Y_1^1 \oplus (W^1_{n+s-h} \cdot \overset{1}{c}_{n+s-h})$

Lemma 3.2. *There exists a PPT S such that the following two ensembles are identically distributed, namely*

$$\{S(\ T(C)\ )\}_{t\in\mathbb{N}} \equiv \{VIEW'\}_{t\in\mathbb{N}}\ . \tag{3.18}$$

Proof. With $T(C)$, S can simulate $W_1^{v_1}\overset{v_1}{c}_1,\cdots,W_n^{v_n}\overset{v_n}{c}_n$, $E_k(g'_{n+1}),\cdots,E_k(g'_{n+s})$ as follows: For each $1\leq i\leq n+s$, it produces a uniformly distributed random string pair (W'_i, c'_i) with $|W'_i|=|W_i^{v_i}|$ and $|c'_i|=|\overset{v_i}{c}_i|$. For each $1\leq i\leq s$, simulate the on-path item (i.e. the $c'_{a_i}[p]\cdot c'_{b_i}[q]$-th) of $E_k(g'_{n+i})$ with $X_{c'_{b_i}[q]}\oplus Y_{c'_{a_i}[p]}\oplus(W'_{n+i}\cdot c'_{n+i})$ and other three off-path items with uniformly distributed random strings of the same length. As each $W'_i\cdot c'_i$ is identically distributed to $W_i^{v_i}\overset{v_i}{c}_i$ and the on-path items are simulated

analogously, the conclusion follows. □

Theorem 3.2 (privacy with respect to semi-honest behavior)**.** *Let C be the circuit to be computed and x be its input, assume that Bob is semi-honest and computationally bounded, then for any x and C, the execution of the preliminary protocol (Protocol 3.1) reveals nothing substantial other than $T(C)$ to Bob assuming the existence of PRGs.*

Proof. With Definition 3.1, Lemma 3.1 and Lemma 3.2, the conclusion is straightforward.

□

Theorem 3.3 (privacy with respect to malicious behavior)**.** *Let C be the circuit to be computed and x be its input, assume that Bob cheats Alice by feeding her a fake result during the execution of Protocol 3.1, then for every positive polynomial $p(\cdot)$ and all sufficiently large t's, it holds that*

$$\Pr[\textit{Alice detects cheating}] > 1 - \frac{1}{p(t)} \tag{3.19}$$

assuming the existence of PRGs.

Proof. As Alice will verify the result by comparing it against W_{n+s}^0 and W_{n+s}^1, the cheating is successful only if Bob can give Alice $W_{n+s}^{v_{n+s}\oplus 1}$ instead of $W_{n+s}^{v_{n+s}}$. Due to Theorem 3.3, Bob learns only $T(C)$ during the protocol execution and the probability of faking $W_{n+s}^{v_{n+s}\oplus 1}$ is $1/2^t$, namely, the conclusion holds. □

3.3 An Improved SCC Protocol with Enhanced Privacy

We have presented an SCC protocol that hides everything but $T(C)$ from computationally bounded Bob. In this section, we show that $T(C)$ may also reveal sensitive information and we improve the protocol by introducing CTO.

3.3.1 Information Disclosure Due to $T(C)$

In some cases, $T(C)$ carries sensitive information and may even reveal C, intermediate results and $C(x)$. Let us see a simple example: C computes such a function that takes a 3-bit unsigned integer x as input, compares it against y and outputs 1 iff $x>y$ (otherwise 0), where y is also a 3-bit unsigned integer hardwired in C as a constant. Assume that Bob has a prior knowledge of C except for the value of y, and that the binary representations of x and y are $x_3x_2x_1$ and $y_3y_2y_1$ respectively with x_3 and y_3 the most significant bits, then, as shown in Table 3.3, $T(C)$ differs when y changes. If y is even, C has two gates, otherwise, it needs at most one gate. Thus, if Bob finds that C has only one gate, he can infer that y is an odd integer (i.e. y_1=1).

A solution is to obfuscate C such that the resulting C' computes the same function as C does while $T(C')$ is independent of the internal private template y. For

instance, we obfuscate the circuit C in Table 3.3 into C' as follows:

$$\begin{aligned}
&\text{inputs: } v'_1 = x_1, v'_2 = x_2, v'_3 = x_3 \\
&v'_4 = g_4(v'_1, v'_2) : [0,\ \ \bar{y_2},\ \ \bar{y_2} \wedge \bar{y_1},\ \ \bar{y_2} \vee \bar{y_1}] \\
&v'_5 = g_5(v'_3, v'_4) : [0,\ \ \bar{y_3},\ \ \bar{y_3},\ \ 1] \\
&\text{output: } v'_5
\end{aligned} \tag{3.20}$$

where gate functions are represented using truth tables. The obfuscated circuit C' outputs 1 iff $x_3x_2x_1 > y_3y_2y_1$ as we can verify that v'_4=1 iff $x_2x_1 > y_2y_1$, and that v'_5=1 iff (x_3>y_3) or (x_3=y_3 and v'_4=1).

3.3.2 A Simple CTO Technique

In the previous section, we illustrated an example of obfuscation that makes the circuit topology reveal nothing regarding the internal private template y, but in a more general case, it may be hard to formalize which information is private. Thus, we define CTO with Definition 3.2, which makes the circuit topology leak nothing substantial.

Table 3.3: Logic of circuit C for different y hardwired in it, where C takes x as input and outputs 1 iff x>y.

y=$y_3y_2y_1$	$C(x)$	y=$y_3y_2y_1$	$C(x)$
000	$x_3 \vee (x_2 \vee x_1)$	100	$x_3 \wedge (x_2 \vee x_1)$
001	$x_3 \vee x_2$	101	$x_3 \wedge x_2$
010	$x_3 \vee (x_2 \wedge x_1)$	110	$x_3 \wedge (x_2 \wedge x_1)$
011	x_3	111	0

Definition 3.2 (Circuit Topology Obfuscation)**.** *Circuit C' has an obfuscated circuit topology of C if the following conditions hold:*

1. ***Correctness***. *C and C' have the same input length (i.e. n) and output length (i.e. 1) and for any valid input $x \in \{0,1\}^n$, $C(x)=C'(x)$ holds.*

2. ***Privacy***. *The circuit topology of C', namely*

$$T(C') = ((n, s', 1), (a'_1, b'_1), (a'_2, b'_2), \cdots, (a'_{s'}, b'_{s'})) \tag{3.21}$$

reveals only $(n,s',1)$ or $(n,s',d',1)$ in an information-theoretic sense, namely, there exists a polynomial-time algorithm such that given any $(n, s', 1)$ or $(n, s', d', 1)$ it produces $T(C')$ efficiently. s' and d' are the size and depth of circuit C' respectively.

Now we introduce a simple CTO technique only for demonstration of its possibility, and we will present more efficient and complicated ones in Chapter 4.

Initially, denote the n inputs of the obfuscated circuit C' by $v'_1, \cdots, v'_n$, where for each $1 \leq j \leq n$, v'_j corresponds to v_j (the result of the j-th node in C). Then, C is obfuscated by induction, namely, suppose that we have $v'_1, \cdots, v'_{n+i}$ with $0 \leq i < s$ and $v'_j = v_j$ for all $1 \leq j \leq n+i$, we generate a subcircuit SC_{n+i} that takes $v'_1, \cdots, v'_{n+i}$ as input and produces output

$$v'_{n+i+1} = v_{n+i+1} = g_{n+i+1}(v_{a_{i+1}}, v_{b_{i+1}}) \ . \tag{3.22}$$

Finally, concatenate all the generated subcircuits to produce C' with v'_{n+s} as the output. As we have not numbered the nodes in the sub-circuits, v'_{n+i} is used only for correspondence with v_{n+i}.

Now we describe the construction of each SC_u. As shown in Figure 3.2, the circuit topology of SC_u is a balanced binary tree, where leaf nodes take v'_1 through v'_u as inputs, root node outputs v'_{u+1}=$g_{u+1}(v'_{a_{u+1}},v'_{b_{u+1}})$, and each level holds the maximal number of nodes except that the highest level keeps its nodes aligned to the left. Two connected paths (e.g., the highlighted paths in Figure 3.2), from respectively wire $v'_{a_{u+1}}$ and wire $v'_{b_{u+1}}$ to wire v'_{u+1}, will converge at some node (root node in the worst case), which we call the convergence node. Thus, we assign function g_{u+1} to the gate that corresponds to the convergence node (e.g., node 2 in Figure 3.2), and assign degenerate functions to the other on-path gates (e.g., nodes numbered 1, 4, 5, 9, 11, 18, 22 in Figure 3.2) such that the output is equal to the on-path incoming wire (e.g., the output of gate node 1 is equal to its left child), and assign arbitrary functions to the rest of the gates.

Correctness. The correctness of obfuscation is implied by that of the subcircuit SC_u. That is, the values of $v'_{a_{u+1}}$ and $v'_{b_{u+1}}$ are copied all the way to the convergence node, where the function v'_{u+1}=$g_{u+1}(v'_{a_{u+1}},v'_{b_{u+1}})$ is computed and v'_{u+1} is copied to the root node and produced as the output of SC_u. Thus, v'_j=v_j holds for all $1\leq j\leq n+s$.

Overhead. As SC_u has size $u-1$ and depth $\lceil \log_2 u \rceil$, the circuit size of C' is

$$s' = \sum_{u=n}^{n+s-1} (u-1) = \frac{(n-1+n+s-2)\cdot s}{2} \tag{3.23}$$

and its depth is

$$\begin{aligned} \sum_{u=n}^{n+s-1} \lceil \log_2 u \rceil < \sum_{u=1}^{n+s} \lceil \log_2 u \rceil \leq \sum_{j=1}^{\lceil \log_2 (n+s) \rceil} j 2^{j-1} \\ \approx (n+s) \log_2 (n+s) \ . \end{aligned} \tag{3.24}$$

Thus, for an n-input-s-gate circuit C with $s \gg n$, the obfuscated circuit C' has size $s^2/2$ and depth $s \cdot \log_2 s$.

Privacy. By Definition 3.2, we give the proof by showing that $T(C')$ can be ef-

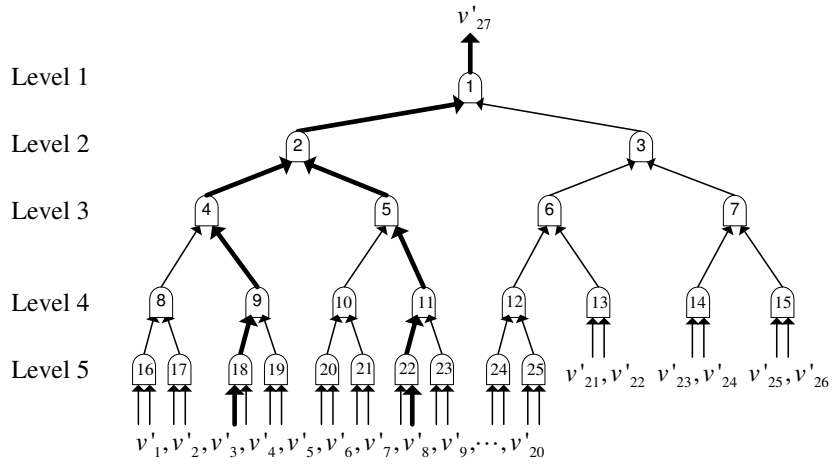

Figure 3.2: An example of how to construct SC_u with u=26, a_{u+1}=5, b_{u+1}=14, and v'_{u+1}=$g_{u+1}(v'_{a_{u+1}}, v'_{b_{u+1}})$.

ficiently computed from $(n,\ s',\ 1)$. First, we obtain s from n, s' and (3.23). Then, for each $u \in \{n, \cdots, n+s-1\}$, we compute $T(SC_u)$ from u and concatenate $T(SC_n)$, $T(SC_{n+1}), \cdots, T(SC_{n+s-1})$ to obtain $T(C')$. Note that $T(SC_u)$ is fully determined by u since $T(SC_u)$ is a balanced binary tree with the nodes on the highest level aligned to the left.

3.3.3 The Improved Protocol

Protocol 3.2 (The improved SCC protocol)**.** *Alice obfuscates C using a CTO algorithm to produce C' and then engages with Bob in Protocol 3.1 on x and C'.*

Theorem 3.4 (correctness and privacy)**.** *Let C be the circuit to be computed and x be its input, and assume that PRGs exist, then the improved SCC protocol (Protocol 3.2) correctly and privately (without revealing anything substantial to Bob) computes C in the semi-honest model, and if Bob cheats Alice by feeding her a fake result, then for every positive polynomial $p(\cdot)$ and all sufficiently large t's, it holds that*

$$\Pr[\textit{Alice detects cheating}] > 1 - \frac{1}{p(t)} \quad . \tag{3.25}$$

Proof. The simple CTO algorithm produces C' as an obfuscated version of C in the sense of Definition 3.2. According to Theorem 3.1, the protocol can correctly compute $C'(x)$ which is equal to $C(x)$ by Definition 3.2. Due to Theorem 3.2, only $T(C')$ is revealed to semi-honest Bob and by Definition 3.2 $T(C')$ discloses nothing more then $(n,s',d',1)$. According to Theorem 3.3, the protocol also defeats malicious

Bob. Therefore, the conclusion follows. □

3.4 Summary of the Chapter

We have shown how to construct one-time-use SCC protocols that help weak-power device Alice utilize untrustworthy Bob's computing power. Alice has to store Tab for input encryption and output decryption which occupies a storage space of

$$2t + \sum_{i=1}^{n} (2t+1) fo_i < F(2t+1)(n+1) \tag{3.26}$$

bits, where F is the maximal fan-out of the input nodes. Considering today's cheap cost of storage, the amount is not large and it does not depend on the circuit size.

As Bob computes an encrypted circuit instead of a plain one, there is an overhead factor

$$O(1) \times r_{C'/C} \times poly(t) \tag{3.27}$$

where $O(1)$ is a constant, $r_{C'/C}$ denotes the circuit size ratio of C' to C in case of sequential computation and the circuit depth ratio of C' to C if circuit is computed in parallel, and $poly(t)$ is the number of steps for PRG $\mathcal{G}$ to generate a masking string for decryption, which is a polynomial in t by definition. Recall that Alice cannot know x in advance and hence her home PC cannot compute $C(x)$ for her. In this sense, the overhead factor is reasonable if we can find more efficient CTO algorithms to minimize $r_{C'/C}$ (refer to the depth-efficient and size-efficient CTO algorithms in

Chapter 4).

Despite the provable security, Protocol 3.1 and Protocol 3.2 still have a major weakness, namely, for each encryption key k, the corresponding $E_k(C)$ can be computed on only one instance of x, otherwise, the privacy results do not hold. For example, suppose that Bob computes $E_k(C)$ on two different inputs $v_1 \cdots v_n$ and $v'_1 \cdots v'_n$, then Bob is able to find the correlation between each pair of v_i and v'_i, namely,

$$v_i \oplus v'_i = \begin{cases} 0, & \text{if } W_i^{v_i} \text{ and } W_i^{v'_i} \text{ are identical} \\ 1, & \text{otherwise} \end{cases} \tag{3.28}$$

Such correlations exist not only for $i \in \{1, \cdots, n\}$, but also for $i \in \{n+1, \cdots, n+s\}$. Therefore, both Protocol 3.1 and Protocol 3.2 are one-time-use. If Alice needs to compute C on two different inputs, we should invoke the protocol twice and prepare two copies of $E_k(C)$ encrypted under different keys. In chapter 5, we will explore reusable (as opposite to one-time-use) SCC protocols that can repeatedly and securely compute a circuit (on different inputs) without necessarily changing encryption key.

Chapter 4

Circuit Topology Obfuscation

In this chapter, we discuss the central role that CTO plays in SCC protocols as well as its applications to practical problems. We propose two approaches: depth-efficient CTO and size-efficient CTO.

4.1 Significance of CTO

As we can observe from Figure 2.3, the sub-graph made up of only $\mathcal{P}(SI)$'s with $C \in SI$ is a connected graph and there is a connected path from any $\mathcal{P}(SI_1)$ with $C \in SI_1$ to any $\mathcal{P}(SI_2)$ with $C \notin SI_2$. In other words, the existence of a hiding-C SCC protocol implies that of any other SCC protocol. In spite of that, there are as yet few hiding-C protocols because it is hard to allow someone to compute a circuit C without telling him how the circuit is wired (i.e. $T(C)$). Therefore, CTO is essential to SCC protocols as it insures that the topology of an obfuscated circuit, $T(C')$, reveals nothing substantial even to infinitely powerful observers while C' preserves the same function as the original circuit.

In addition, CTO can also solve the following problem: Computationally bounded

Alice is about to implement a private function f on a circuit against Bob, who is an infinitely powerful adversary that attempts to recover f by passively observing the topology of its circuit implementation, then how could f be implemented such that Bob learns nothing from the circuit topology? This is not a trivial problem as in practice we can hardly hide the skeleton of a circuit from a reverse engineer, furthermore, circuit topology may carry non-trivial information regarding the circuit for the following reasons:

1. The basis of circuit C usually has a small number of elements, e.g., OR ($\vee$), AND ($\wedge$) and NOT ($\neg$), with each element having a specific range of fan-in, e.g., given the topology of a circuit with basis $\{\vee, \neg\}$, it is a trivial task to recover the circuit by assigning gate nodes of fan-in 1 with "$\neg$" and the rest with "$\vee$".

2. For the sake of efficiency, C is usually designed in a way such that all its subcircuits computing fundamental functions (e.g., addition, multiplication, matrix product) are in the minimized format and an adversary can easily recognize them from their topology, e.g., we can easily identify the circuit of a full-adder given only how the circuit is wired.

Thus, in some cases, given $T(C)$ an adversary could (partially) recover C, and therefore f. With CTO, Alice obfuscates the minimized circuit C to produce C' and implements C' instead of C, where by definition $T(C')$ discloses nothing substantial.

There are two ways to measure the overhead of CTO: (1) the ratio of s' to s; (2)

the ratio of d' to d, where s', s, d' and d are the size of C', the size of C, the depth of C' and the depth of C respectively.

Circuit size corresponds to the amount of hardware required to implement a circuit, and in case that a circuit is computed by single-threaded software it also determines the computation time. Hence, if we want to implement an obfuscated circuit with least amount of hardware, or to minimize the sequential computation time, s'/s would be the major overhead factor to be taken into consideration.

Circuit depth determines the computation time when a circuit is computed by hardware or parallelized software, so if we wish to have a minimal parallel computation time, we consider d'/d as the overhead factor.

4.2 Depth-Efficient CTO

We show that a depth-efficient CTO method is implied by universal circuits and it also improves the secure two-party computation protocol in terms of circuit size.

4.2.1 CTO Using Universal Circuits and Hardwiring Techniques

Valiant [61] proposed universal circuits and showed that there is a universal circuit $C_{d,s}$ of size $O(ds\log s)$ and depth $O(d\log s)$ such that for any circuit C of size s and depth d, we can efficiently compute a string u_C (an encoding of C) such that for any

x: $C_{d,s}(u_C,x) = C(x)$. Based on the above result, we define an algorithm D (see Algorithm 1) that hardwires u_C into $C_{d,s}$ to produce C'. Note that if the hardwiring is done cursorily, then the topology of the resulting circuit may reveal the data that are hardwired into it (see e.g. Table 3.3).

Algorithm 1 Hardwiring u_C into $C_{d,s}$ to produce C'.

Inputs: u_C=$a_1 \cdots a_m$ and $C_{d,s}$, where the first m input-nodes of $C_{d,s}$ correspond to $a_1 \cdots a_m$, the next n correspond to x and gate-nodes are $g_{m+n+1}, \cdots, g_{m+n+s'}$.
Number the input-nodes of C' with $1, \cdots, n$ ($|x| = n$) respectively.
Define a map M such that node j of C' corresponds to node $M(j)$ of $C_{d,s}$.
Let I_i ($1 \leq i \leq m+n+s'$) be the set associated with node i of $C_{d,s}$.
$M(1) \leftarrow m+1, \cdots, M(n) \leftarrow m+n, I_1 \leftarrow \phi, \cdots, I_m \leftarrow \phi, I_{m+1} \leftarrow \{m+1\}, \cdots, I_{m+n} \leftarrow \{m+n\}$, $h \leftarrow n+1$. {Node h is the next node of C' to be generated.}
$v_1 \leftarrow a_1, \cdots, v_m \leftarrow a_m$ and mark $v_{m+1}, \cdots, v_{m+n}$ as unknowns. {v_i can be a constant (I_i=ϕ), itself (I_i=$\{i\}$), a unary or binary function of its preceding unknowns whose indexes are given by I_i.}
for i=$m+n+1$ to $m+n+s'$, consider g_i with input-nodes numbered l_i and r_i **do**
  if (I_{l_i}=$\{u, w\}$ and I_{r_i}=$\{y, z\}$ and $I_{l_i} \cup I_{r_i}$ has at least 3 elements) or (node i carries a circuit-output) **then**
    if $M^{-1}(l_i)$ is undefined **then**
      Represent g'_h (the h-th node of C') according to v_{l_i}, $M(h) \leftarrow l_i$, $h \leftarrow h+1$. {namely, if v_{l_i} is a function of node u and node w, then let g'_h be the same function as v_{l_i} with input-nodes numbered $M^{-1}(u)$ and $M^{-1}(w)$.}
    end if
    if $M^{-1}(r_i)$ is undefined **then**
      Represent g'_h according to v_{r_i}, $M(h) \leftarrow r_i$, $h \leftarrow h+1$.
    end if
    Let g'_h be the same function as g_i with input-nodes numbered $M^{-1}(l_i)$ and $M^{-1}(r_i)$, $M(h) \leftarrow i$, $h \leftarrow h+1$.
    $v_i \leftarrow g_i(v_{l_i}, v_{r_i})$, $I_i \leftarrow \{l_i, r_i\}$.
  else if $I_{l_i} \cup I_{r_i}$=$\{u, w\}$, or $\{u\}$, or ϕ **then**
    $v_i \leftarrow g_i(v_{l_i}, v_{r_i})$ and $I_i \leftarrow I_{l_i} \cup I_{r_i}$.
  end if
end for
Output: Circuit C'.

Theorem 4.1 (Depth-efficient CTO)**.** *Let C be the original circuit with input length*

n, size s and depth d, let D be the hardwiring algorithm as in Algorithm 1, define a CTO algorithm as: On input C, it constructs a universal circuit $C_{d,s}$ for circuits of size s, depth d, input length n and output length 1, efficiently computes the encoding of C, namely u_C, and hardwires u_C into $C_{d,s}$ to obtain C'. Then, it holds that

1. *For any valid input x, $C'(x)$=$C(x)$.*
2. *$T(C')$ discloses nothing more than (n, s, d, 1) in an information-theoretic sense, namely, there is an efficient algorithm such that for any (n, s, d, 1) it can produce the corresponding $T(C')$.*
3. *The size and depth of C' are no larger than $O(ds\log s)$ and $O(d\log s)$ respectively.*

Proof. By the correctness of universal circuits, namely $C_{d,s}(u_C,x) = C(x)$, it remains to prove that $C'(x)$=$C_{d,s}(u_C,x)$ for all x's, which is straightforward as each node h in C' carries the same result as node $M(h)$ in $C_{d,s}$ (see the definition of M in Algorithm 1). As for the privacy condition, given (n, s, d, 1), we can obtain $C_{d,s}$ and $|u_C|$ (the length of u_C), then we invoke D on $C_{d,s}$ and a $|u_C|$-bit all-zero string to obtain C'', where it must hold that $T(C'')$=$T(C')$ as we can see from Algorithm 1 that the topology of the resulting circuit is generated in a way independent of the value of u_C. In other words, $T(C')$ is fully determined and can be efficiently obtained by (n, s, d, 1). Finally, since C' is a hardwired version of $C_{d,s}$, it has at most [1] size

[1] Although the size and depth of C' are given in terms of upper bound, in most cases they would equal to $O(ds\log s)$ and $O(d\log s)$ respectively because the hardwiring algorithm reduces only the input-level gates of $C_{d,s}$ and its impact vanishes quickly as the depth increases.

$O(ds\log s)$ and depth $O(d\log s)$. □

4.2.2 Improving the Two-Party Computation Protocol

With the hardwiring technique, we can improve the secure two-party computation protocol. That is, instead of following the original protocol (see Section 2.3.5), Alice and Bob execute the following protocol:

Protocol 4.1 (The improved two-party computation protocol)**.** *By invoking D (Algorithm 1), Alice hardwires her private input x into C to produce C' and sends the encrypted format $E_k(C')$ to Bob. Bob then executes $|y|$ 1-out-of-2 OTs [47, 48] in parallel* [2] *with Alice such that Bob gets his input y encrypted without revealing y to Alice. Finally, Bob computes $E_k(C')$ on $E_k(y)$ to get $E_k(C'(y))$, which is decrypted by Alice to obtain $C'(y)$.*

Theorem 4.2 (correctness and privacy in the semi-honest model)**.** *Let C be the circuit to be computed and let x and y be the private inputs of Alice and Bob respectively, assume that trapdoor permutations exist, assume that circuits are encrypted and then computed using the method we proposed in Section 3.2.2 and Section 3.2.3, and assume that both Alice and Bob are semi-honest and computationally bounded, then Protocol 4.1 correctly and privately computes C and x, namely, both parties obtain $C(x,y)$ correctly and Alice knows nothing regarding y more than what $(C,x,C(x,y))$ reveals and Bob knows nothing regarding x more than what $(C,y,C(x,y))$ reveals.*

[2] Note that y is encrypted bit by bit independently, so the encryption can be done in parallel using $|y|$ 1-out-of-2 OTs with the same number of rounds as that of a single OT.

Proof sketch. By the correctness of our encrypted circuits (Theorem 3.1), both parties correctly get $C'(y)$, which equals to $C(x,y)$ according to Theorem 4.1. Regarding the privacy for Alice, Bob would learn y and $C'(y)$, which are required by Protocol 4.1, and also $T(C')$ (see Theorem 3.2), which reveals nothing regarding x assuming the existence of PRGs (see Theorem 4.1). As for Bob's privacy, if trapdoor permutations exist, then Bob can get y encrypted by means of OT without revealing it to Alice [7, Proposition 7.3.6] and hence Alice learns only C, x and $C'(y)$. Note that PRGs are implied by trapdoor permutations [13]. □

We can modify any secure two-party computation protocol that works in the semi-honest model (and hence Protocol 4.1) into one that defeats malicious Alice and Bob (see the conversion in [7, Section 7.4]), but in this case the round complexity of the modified protocol would not be constant but proportional to circuit size.

Our protocols (Protocol 4.1 and its modified version for malicious adversaries) are more efficient than the original two-party computation protocol as we significantly reduce the circuit size. Consider a typical two-party computation problem, the Millionaire Problem [62], where two millionaires Alice and Bob want to know who is richer, without revealing their actual wealth x and y to each other. Assume that $x{=}v_n \cdots v_1$ and $y{=}v_{2n} \cdots v_{n+1}$ are both n-bit unsigned integers with v_n and v_{2n} being the most significant bits and that $C(x,y)$ outputs a two-bit result indicating whether $x{<}y$, $x{=}y$ or $x{>}y$, then the original protocol computes C whose optimal format is as follows:

Pseudo code 4.1: Circuit C for the Millionaire Problem using the original protocol

inputs: $v_1, \cdots, v_n, v_{n+1}, \cdots, v_{2n}$

gates:

$v_{2n+1}=g_{2n+1}(v_1, v_{n+1})=v_1 \oplus v_{n+1} \oplus 1$ $\{v_{2n+1}=1 \text{ iff } v_1=v_{n+1}\}$

$v_{2n+2}=g_{2n+2}(v_1, v_{n+1})=\bar{v_1} \wedge v_{n+1}$ $\{v_{2n+2}=1 \text{ iff } v_1<v_{n+1}\}$

for i = 2 to n

$v_{2n+5i-7}=g_{2n+5i-7}(v_i, v_{n+i})=v_i \oplus v_{n+i} \oplus 1$ $\{v_{2n+5i-7}=1 \text{ iff } v_i=v_{n+i}\}$

$v_{2n+5i-6}=g_{2n+5i-6}(v_{2n+5i-8}, v_{2n+5i-7})=v_{2n+5i-8} \wedge v_{2n+5i-7}$

$v_{2n+5i-5}=g_{2n+5i-5}(v_i, v_{n+i})=\bar{v_i} \wedge v_{n+i}$ $\{v_{2n+5i-5}=1 \text{ iff } v_i<v_{n+i}\}$

$v_{2n+5i-4}=g_{2n+5i-4}(v_{2n+5i-9}, v_{2n+5i-7})=v_{2n+5i-9} \wedge v_{2n+5i-7}$

$\{v_{2n+5i-4}=1 \text{ iff } v_i \cdots v_1=v_{n+i} \cdots v_{n+1}\}$

$v_{2n+5i-3}=g_{2n+5i-3}(v_{2n+5i-6}, v_{2n+5i-5})=v_{2n+5i-6} \vee v_{2n+5i-5}$

$\{v_{2n+5i-3}=1 \text{ iff } v_i \cdots v_1<v_{n+i} \cdots v_{n+1}\}$

end for

outputs: v_{7n-4} and v_{7n-3} .

Thus, the number of nodes in C is $7n-3$ (see also [56, Table 1] for a similar result of 254 when n=32). With our protocol, $x=a_n \cdots a_1$ is treated as a constant and is hardwired (using Algorithm 1) into C to produce C', which takes only $y=v_n \cdots v_1$ as input and has $(3n-2)$ nodes:

Pseudo code 4.2: Circuit C' for the Millionaire Problem using the

improved protocol

inputs: $v_1, \cdots, v_n$

gates:

v_{n+1}=$g_{n+1}(v_1, v_2)$=1 iff a_2a_1=v_2v_1,

v_{n+2}=$g_{n+2}(v_1, v_2)$=1 iff $a_2a_1 < v_2v_1$

for i = 3 to n

v_{n+2i-3}=$g_{n+2i-3}(v_i, v_{n+2i-5})$=$v_{n+2i-5} \wedge (\bar{a}_i \oplus v_i)$ $\{v_{n+2i-3}$=1 iff $a_i \cdots a_1$=$v_i \cdots v_1\}$

v_{n+2i-2}=$g_{n+2i-2}(v_i, v_{n+2i-4})$=$(v_{n+2i-4} \wedge (\bar{a}_i \oplus v_i)) \vee (\bar{a}_i \wedge v_i)$

$\{v_{n+2i-2}$=1 iff $a_i \cdots a_1 < v_i \cdots v_1\}$

end for

outputs: v_{3n-3} and v_{3n-2} .

Therefore, in the case of the Millionaire Problem, we improve the efficiency of the two-party computation protocol by reducing the circuit size to less than half with provable security.

4.3 Size-Efficient CTO

We first introduce two basic tools, permutation circuits and multiplexer circuits and then present a size-efficient CTO algorithm followed by proofs for the security and overhead.

4.3.1 Oblivious Permutation Circuits

As shown in Figure 4.1.(A), the results of n nodes, numbered $1, \cdots, n$ respectively, are used (by subsequent nodes) in an order defined by a permutation (bijection) π:$\{1,\cdots,n\}\rightarrow\{1,\cdots,n\}$, namely, node $\pi^{-1}(1)$ is referred first, node $\pi^{-1}(2)$ second, $\cdots$, and $\pi^{-1}(n)$ last, then for any subsequent node, a circuit topology observer can easily tell which node(s) contributes to its input.

If we could find an efficient subcircuit construction algorithm such that given any n-to-n permutation π:$\{1,\cdots,n\}\rightarrow\{1,\cdots,n\}$, it outputs an n-input-n-output subcircuit C_π (see Figure. 4.1.(B)) satisfying:

- (Correctness). For all $i\in\{1,\cdots,n\}$, the $\pi(i)$-th output of C_π is a copy of its i-th input.
- (Privacy). $T(C_\pi)$ is uniform for any n-to-n permutation π, namely, it reveals nothing about how the n inputs are permuted.

then we insert C_π in between the circuit in Figure. 4.1.(A), and the resulting circuit functions the same as the original one while a circuit topology observer can no longer tell in which order the leftmost n nodes are referred by subsequent ones.

We reduce the problem of generating C_π to a sorting network problem: There are n variables, $x_1, x_2, \cdots, x_n$, which are initialized to $\pi(1), \pi(2), \cdots, \pi(n)$ respectively and each x_i $(1\leq i\leq n)$ is attached to node l_i with initially $l_i=i$. A comparison between x_i and x_j involves the following actions:

1. Generate two gates $g_{L+1}(v_{l_i},v_{l_j})$ and $g_{L+2}(v_{l_i},v_{l_j})$

$$g_{L+1}(v_{l_i}, v_{l_j}) = \begin{cases} v_{l_i}, \text{ if } x_i < x_j \\ v_{l_j}, \text{ otherwise} \end{cases} \qquad g_{L+2}(v_{l_i}, v_{l_j}) = \begin{cases} v_{l_j}, \text{ if } x_i < x_j \\ v_{l_i}, \text{ otherwise} \end{cases} \tag{4.1}$$

   where L was the maximum node number before generating the above gates, v_{l_i} and v_{l_j} denote the results of node l_i and node l_j respectively.

2. Swap the values of x_i and x_j if $x_i > x_j$. Re-attach x_i and x_j to node $L+1$ and node $L+2$ respectively (i.e. $l_i \leftarrow L+1$ and $l_j \leftarrow L+2$) and increment L by 2.

How can $x_1, x_2, \cdots, x_n$ be sorted to an ascending order by performing comparisons in a predetermined manner (independent of the values of $x_1, \cdots, x_n$)? When the sorting is done, C_π is produced as well: the input nodes of C_π are node 1, $\cdots$, node

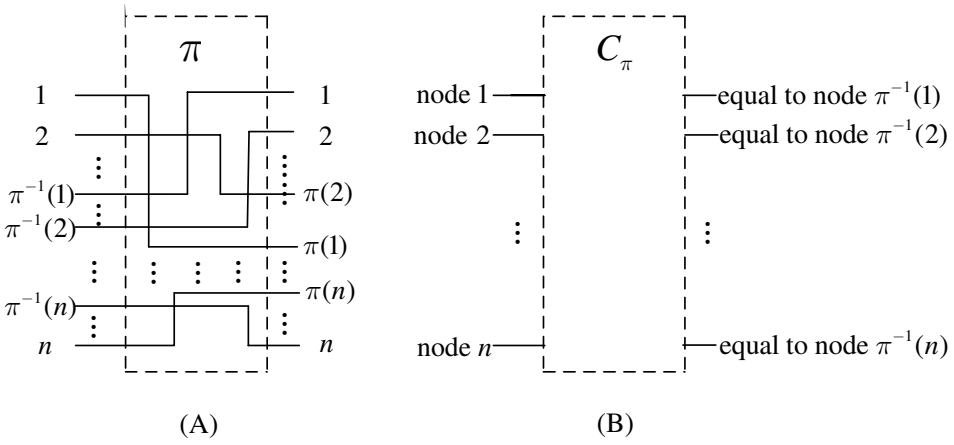

Figure 4.1: (A) The results of n nodes are referred in an order given by permutation π:$\{1, \cdots, n\} \rightarrow \{1, \cdots, n\}$. (B) A permutation subcircuit C_π whose input-output-correspondence is defined by permutation π.

n, gates are generated during comparisons and the fist output is node l_1, the second is node l_2, $\cdots$, the n-th is node l_n.

We verify that the above reduction is right:

1. (Correctness). During each comparison, v_{l_i} and v_{l_j} are swapped (by generating $g_{L+1}(v_{l_i},v_{l_j})=v_{l_j}$, $g_{L+2}(v_{l_i},v_{l_j}) = v_{l_i}$ and letting $l_i=L+1$ and $l_j=L+2$) if and only if x_i and x_j are swapped. Initially, it holds that $x_i=\pi(i)$ and $v_{l_i}=v_i$. By the time x_1, $\cdots$, x_n are sorted, it will hold that $x_{\pi(i)}=\pi(i)$ and hence the $\pi(i)$-th output of the subcircuit is equal to the i-th input, namely, $v_{l_{\pi(i)}}=v_i$.

2. (Privacy). No matter $x_i < x_j$ or not, two gates are generated in the same way in terms of circuit topology. In addition, the comparisons are performed in an order regardless of π and hence the resulting circuit topology is uniform for all n-to-n permutations.

A satisfactory [3] solution for the reduced problem is given by Batcher [64], who uses $O(n{\cdot}\log^2 n)$ comparisons and a depth of $O(\log^2 n)$ to sort n elements in a data-independent way. Since each comparison corresponds to two gates, we can construct an n-to-n permutation subcircuit C_π with size $O(n{\cdot}\log^2 n)$, depth $O(\log^2 n)$ and a uniform topology.

In some cases, we also hope to produce an n_1-input-n_2-output subcircuit with $n_1<n_2$ and the input-output mapping is an injection, namely, there is a permutation

[3] There is an asymptotically better result (also the lower bound), namely, the AKS network [63] with $O(n{\cdot}\log n)$ comparisons and depth $O(\log n)$, but it is impractical as the constant factor hidden by the big-oh notation is extremely large.

π_1:$\{1, \cdots, n_1\} \rightarrow \{a_1, \cdots, a_{n_1}\}$ with $\{a_1, \cdots, a_{n_1}\} \subset \{1, \cdots, n_2\}$ and the $\pi_1(i)$-th output equal to the i-th input. In this case, we construct another arbitrary bijection π_2: $\{n_1{+}1, \cdots, n_2\} \rightarrow (\{1, \cdots, n_2\}{-}\{a_1, \cdots, a_{n_1}\})$ and let permutation $\pi = \pi_1 \cup \pi_2$. We append $n_1{-}n_2$ constant inputs (e.g. 0) to the end of the input list and proceed to generating C_π using π. During the comparison, if a resulting gate has one or two constant inputs (i.e. a degenerate gate), e.g., $g_k(v_i,0)$ or $g_k(0,0)$, write it as $g_k(v_i,v_*)$ or $g_k(v_*,v_\#)$, where $*$ and $\#$ can be arbitrarily chosen. Thus, by padding $(n_2{-}n_1)$ constant inputs, we obtain C_π capable of permuting n_1 inputs into n_2 outputs obliviously.

4.3.2 Oblivious Multiplexer Circuits

Another useful subcircuit for obfuscation is the multiplexer circuit C_{mux} that takes as input the results of n nodes (i.e., $v_1, \cdots, v_n$) and copies one of them as output (i.e., v_z). As exemplified Figure 4.2, we can construct an n-to-1 C_{mux} using a balanced tree with size $(n{-}1)$ and depth $\lceil \log_2 n \rceil$, where v_z is copied from its leaf node all the way to the root node. Obviously (see Section 3.3.2 for similar discussions), $T(C_{mux})$ only depends on n and reveals nothing about z. We note that the multiplexer circuit introduced here differs from the one in logic design in that the selection is hardwired into C_{mux} instead of being an input.

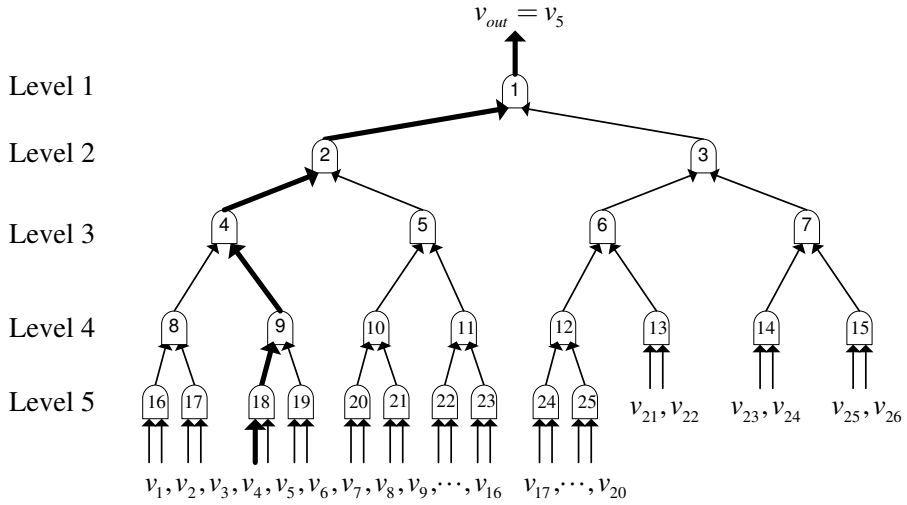

Figure 4.2: An example of multiplexer sub-circuit C_{mux} with n=26 and selection decision z=5.

4.3.3 The CTO Algorithm

We start with an overview of the CTO algorithm $\mathcal{O}$. On input of circuit C, $\mathcal{O}$ obfuscates it gate by gate (i.e. from node n+1 to n+s). As depicted in Figure. 4.3, $\mathcal{O}$ maintains up to d permutation subcircuits C_π^1, $\cdots$, C_π^d and they are updated before each gate is obfuscated. For each gate $g_{n+j}(v_{a_j}, v_{b_j})$, we obfuscate it by replacing it with $g'_{n+j}(v'_{a_j}, v'_{b_j})$ where g'_{n+j} has the same function as g_{n+j} does, and v'_{a_j} and v'_{b_j} are the outputs of two multiplexer circuits $C_{mux}^{a_j}$ and $C_{mux}^{b_j}$ respectively. If we denote by $c_{i,k}$ the k-th output of C_π^i, then the set of inputs of $C_{mux}^{a_j}$ and that of $C_{mux}^{b_j}$ are

$$\{c_{i,k}, \text{ where } 1 \leq i \leq d, \ t_i = 1 \ ^4 \text{ and } k = ((2j-2) \ mod \ 2^i) + 1\} \qquad (4.2)$$

[4] t_i=1 denotes that C_π^i is switched on to provide inputs to multiplexer circuits; t_i=0 denotes that C_π^i does not provide input to any multiplexer circuit.

and

$$\{c_{i,k},\ \text{where } 1 \leq i \leq d,\ t_i = 1 \text{ and } k = ((2j-2)\ mod\ 2^i) + 2\} \tag{4.3}$$

respectively, where $(2j-2) mod\ 2^i$ means that $(2j-2)$ wraps around after it reaches 2^i-1, namely, only the first 2^i outputs are used by multiplexer circuits. Multiplexer subcircuits $C_{mux}^{a_j}$ and $C_{mux}^{b_j}$ are non-reusable, namely, $\mathcal{O}$ generates a distinctive pair of $(C_{mux}^{a_j},\ C_{mux}^{b_j})$ for each gate $g_{n+j}(v_{a_j},\ v_{b_j})$. In contrast, the d permutation subcircuits C_π^1 through C_π^d will last for a while, more precisely, each C_π^i refers to a $3{\cdot}2^{i-1}$-to-$3{\cdot}2^{i-1}$ permutation subcircuit while t_i=1, the reference is nullified when t_i is changed to 0, and C_π^i will point to a new $3{\cdot}2^{i-1}$-to-$3{\cdot}2^{i-1}$ subcircuit when t_i is reset to 1

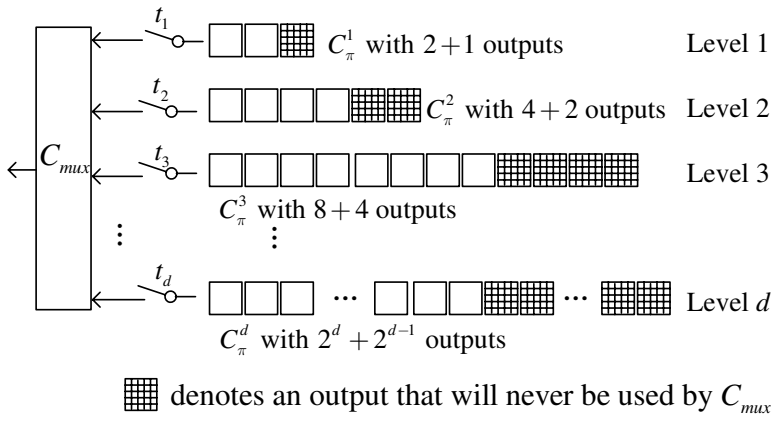

Figure 4.3: An obfuscation network that consists of a d-input multiplexer C_{mux} and d permutation subcircuits $C_\pi^1, \cdots, C_\pi^d$, where each C_π^i $(1{\leq}i{\leq}d)$ has $2^i{+}2^{i-1}$ outputs and one of the first 2^i outputs of each C_π^i contributes to an input of C_{mux} provided that switch t_i is on.

Now we introduce the update of $C_\pi^1, \cdots, C_\pi^d$ as well as d. Initially, $d = \lceil \log_2(n) \rceil$

(i.e. $2^{d-1}<n\leq 2^d$) and $\mathcal{O}$ produces an n-input-$3\cdot 2^{d-1}$-output permutation subcircuit C_π^d that takes as inputs v_1 through v_n (the n inputs of C). Since only level d is occupied, $\mathcal{O}$ sets $t_d t_{d-1}\cdots t_1$ (which also serves as a counter with t_1 the least significant bit) to $10\cdots 0$. Then, $\mathcal{O}$ starts to obfuscate each gate sequentially by replacing $g_{n+j}(v_{a_j}, v_{b_j})$ with $g'_{n+j}(v'_{a_j},v'_{b_j})$. After each gate g_{n+j} is obfuscated, the d permutation subcircuits are updated as follows:

- If $t_1 = 0$, then $\mathcal{O}$ generates a 3-input-3-output C_π^1 taking as inputs v'_{a_j}, v'_{b_j} and $v'_{n+j}=g'_{n+j}(v'_{a_j},v'_{b_j})$, increments the counter $t_d t_{d-1}\cdots t_1$ by 1 (i.e. set t_1 to 1).
- If $t_1=1$ and there is a b such that $1<b<d$ and $t_b t_{b-1}\cdots t_1=01\cdots 1$, then $\mathcal{O}$ generates a $3\cdot 2^{b-1}$-input-$3\cdot 2^{b-1}$-output C_π^b taking as inputs v'_{a_j}, v'_{b_j}, v'_{n+j} and all the outputs of C_π^1 through C_π^{b-1}, increments the counter $t_d t_{d-1}\cdots t_1$ by 1 (i.e. set t_b to 1 and set $t_{b-1}\cdots t_1$ to $0\cdots 0$).
- Otherwise, it holds that $t_d\cdots t_1=1\cdots 1$, $\mathcal{O}$ generates a $3\cdot 2^d$-input-$3\cdot 2^d$-output C_π^{d+1} taking as inputs v'_{a_j}, v'_{b_j}, v'_{n+j} and all the outputs of C_π^1 through C_π^d, increments the counter by 1 (i.e. d is incremented by 1 and $t_d t_{d-1}\cdots t_1$ is set to $10\cdots 0$).

$\mathcal{O}$ moves on to the next gate (i.e. increments j by 1) and repeats the above operations until the last gate g_{n+s} is obfuscated. We can see that each of the first 2^i outputs of C_π^i is used (by multiplexer subcircuits) exactly once (i.e. $t_i t_{i-1}\cdots t_1$ counts from $10\cdots 0$ to $11\cdots 1$) before C_π^i is nullified (i.e. t_i is set to 0) and its outputs are re-

shuffled (together with those in lower levels) and copied to a higher level subcircuit. The counter $t_d t_{d-1} \cdots t_1$ is incremented by 1 after each gate obfuscation.

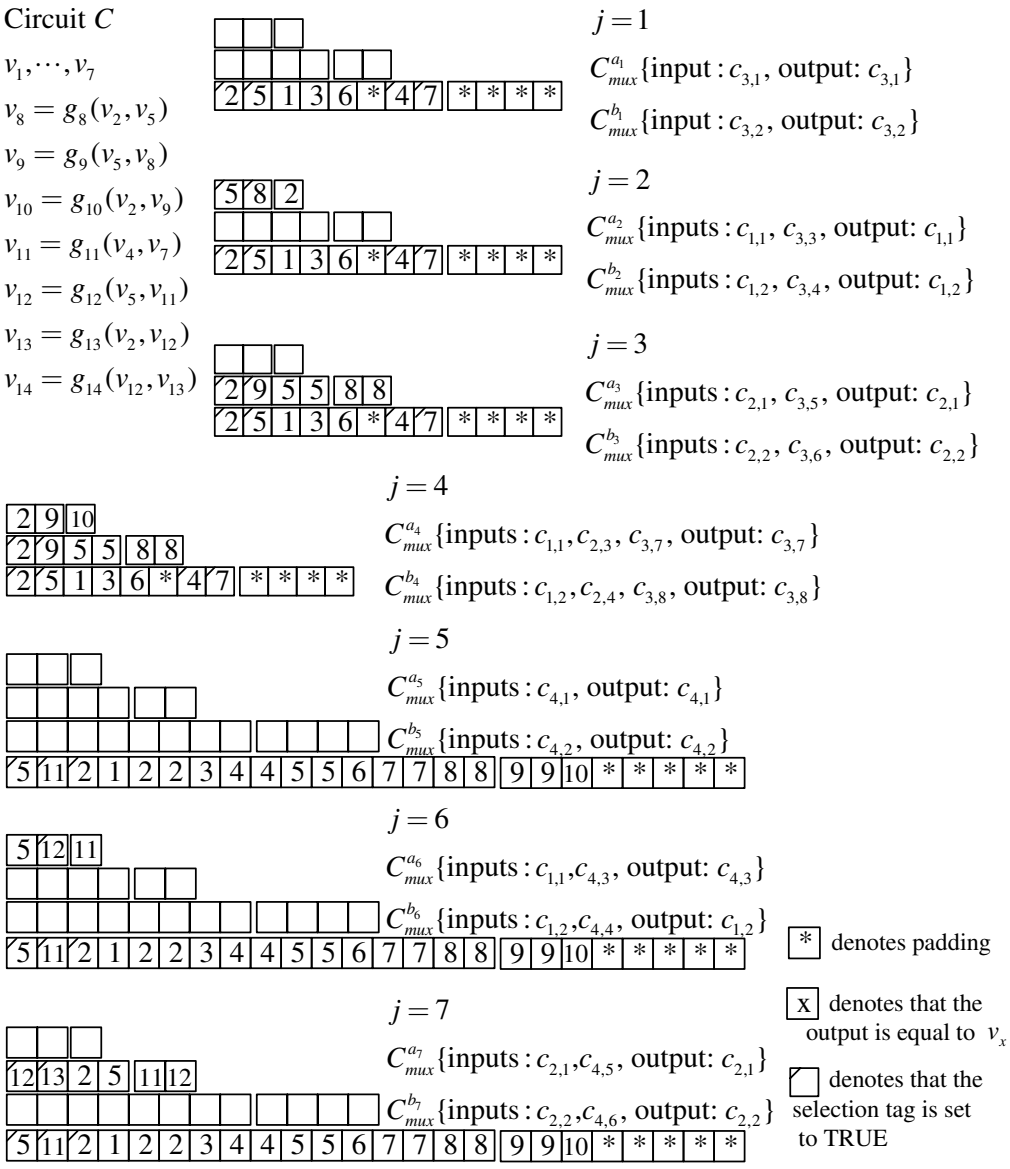

Figure 4.4: An example of how to determine permutation function π and selection z in each step.

We have described how to generate the circuit topology of obfuscated circuit. Obviously, $T(C')$ only depends on $(n, s, 1)$ since the topologies of all the intermedi-

ately generated C_{mux}'s and C_π's are fully determined by their input lengths. It only remains to show how to determine the function of the gates in C', which fall into two categories : g'_{n+j} that corresponds to g_{n+j} and the gates in C_{mux} and C_π. As we have discussed, g'_{n+j} is assigned the same function as g_{n+j}, C_π is determined by π and C_{mux} depends on n and z. Thus, it only remains to explain how to determine the permutation π and the selection z for each C_π and C_{mux} such that for all $j \in \{1, \cdots, s\}$: $v'_{a_j}=v_{a_j}$ and $v'_{b_j}=v_{b_j}$. We illustrate how to determine π and z using a simple example in Figure 4.4. We suppose that the current (to be obfuscated next) gate number is j and that C^i_π is to permute a set of inputs that correspond to v_{L_1}, v_{L_2}, $\cdots$ $v_{L_{3\cdot 2^{i-1}}}$, where each v_k denotes the result of node k in C. The cardinality of the set is less than $3{\cdot}2^{i-1}$ since there may be padded constants and overlapping results. We list out the node numbers of inputs to the next 2^{i-1} gates ($g_{n+j}(v_{a_j},v_{b_j})$ through $g_{n+j+2^{i-1}-1}(v_{a_{j+2^{i-1}-1}},v_{b_{j+2^{i-1}-1}})$):

$$a_j,\ b_j,\ a_{j+1},\ b_{j+1},\ \cdots,\ a_{j+2^{i-1}-1},\ b_{j+2^{i-1}-1} \tag{4.4}$$

and construct a bijection π_1: $A \rightarrow \pi_1(A)$, where $A = \{L_1, \cdots, L_{3\cdot 2^{i-1}}\} \cap \{a_j, b_j, \cdots, a_{j+2^{i-1}-1}, b_{j+2^{i-1}-1}\}$ and $\pi_1(x)$ is the position number of the leftmost x that appears in list (4.4) (x may appear more than once). We also define an arbitrary bijection π_2:

$$\{1, \cdots, 3 \cdot 2^{i-1}\} - A \rightarrow \{1, \cdots, 3 \cdot 2^{i-1}\} - \pi_1(A)\ . \tag{4.5}$$

Finally, we get permutation $\pi=\pi_1\cup\pi_2$ and mark the selection tags of the outputs (of C^i_π) whose positions are given by $\pi_1(A)$ as TRUE. The selection z of $C^{a_j}_{mux}$ (resp., $C^{b_j}_{mux}$) is determined by choosing a $c_{i,k}$ from set (4.2) (resp., set (4.3)) whose selection tag is TRUE and letting $z = i$. Figure 4.4 is an example of how a small circuit is obfuscated step by step. Index j starts from the first gate (j=1) to the last one (j=7) and for each gate, all the switched-on permutation circuits are displayed using grids, and the inputs to the two multiplexer circuits are also indicated. For example, when j=1, A=$\{1, \cdots, 7\}\cap\{2,5,5,8,2,9,4,7\}=\{2,4,5,7\}$ and hence $\pi(2)$, $\pi(4)$, $\pi(5)$ and $\pi(7)$ is set to 1, 7, 2 and 8 respectively. It follows that the 1st, 2nd, 7th and 8th outputs of C^3_π are all tagged as TRUE. Thus, the outputs of $C^{a_1}_{mux}$ and $C^{b_1}_{mux}$ are $c_{3,1}$ and $c_{3,2}$ respectively and the selections of $C^{a_1}_{mux}$ and $C^{b_1}_{mux}$ are both 3.

Correctness. Now we discuss that each $C^{a_j}_{mux}$ (resp., $C^{b_j}_{mux}$) can always ensure its output v'_{a_j}=v_{a_j} (resp., v'_{b_j}=v_{b_j}) by identifying the tagged (as TRUE) input out of its input candidates given in Eq. 4.2 (resp., Eq. 4.3). During each round, the counter $t_d t_{d-1}\cdots t_1$ counts from $10\cdots0$ (i.e. at initialization or when d is incremented by 1) to $11\cdots1$, and $\mathcal{O}$ will obfuscate 2^{d-1} gates of C (unless there are less than 2^{d-1} gates left) before entering the next round. The input nodes of the 2^{d-1} gates are respectively listed as

$$a_j,\ b_j,\ a_{j+1},\ b_{j+1},\ \cdots,\ a_{j+2^{d-1}-1},\ b_{j+2^{d-1}-1}\ , \tag{4.6}$$

which we call the list of level d. Without loss of generality, we consider an arbitrary

a_w in the above list.

- If v_{a_w} has at least a copy in the outputs of C_π^d (i.e. $a_w < j$).
  - If a_w appears only once in the list or it is the leftmost one among those values that appear in the list and are equal to a_w, then $\pi(a_w)$ is equal to its position number and it is hit (tagged as TRUE and selected by $C_{mux}^{a_w}$) in the d-th level (selection decision z equals to d).
  - Otherwise, there is at least a a_u (or b_u) that is equal to a_w and $u<w$. Again without loss of generality, we assume that it is a_u. Thus, after a_u is hit on level d and gate $g_{n+u}(v_{a_u},v_{b_u})$ is obfuscated, v_{a_u} (v_{a_u}=v_{a_w}) will be copied and permutated to a lower level i'. If on level i', a_w is the first among those node numbers (in the corresponding list of level i') whose values are equal to a_w, it is hit on level i'. Otherwise, the operation is repeated recursively until all $g_{n+u}(a_u,b_u)$'s satisfying $u<w$ and (a_u or b_u)=a_w are obfuscated.
- Otherwise, it holds that $a_w \geq j$ and gate g_{a_w} has not been obfuscated yet. We wait until g_{a_w} is obfuscated and v_{a_w} is copied to a lower level i'. Analogously, if in the corresponding list of level i' there is no such a_u or b_u that is equal to a_w with $u<w$, then a_w will be hit on level i', otherwise, the recursion continues until a_w is hit.

4.3.4 Security and Overhead

Theorem 4.3 (Size-efficient CTO)**.** *Let C be the original circuit with input length n, size s and depth d, let $\mathcal{O}$ be the CTO algorithm introduced in in Section 4.3.3, and let C' be the resulting circuit by invoking $\mathcal{O}$ on C, then it holds that*

1. *For any valid input x, $C'(x)$=$C(x)$.*
2. *$T(C')$ discloses nothing more than $(n, s, 1)$ in an information-theoretic sense, namely, there is an efficient algorithm such that for any $(n, s, 1)$ it can produce the corresponding $T(C')$.*
3. *The size and depth of C' are $s{\cdot}O(\log^3 s)$ and $s{\cdot}O(\text{loglog}s)$ respectively.*

Proof. We have already shown in Section 4.3.3 that $\mathcal{O}$ ensures v'_{a_j}=v_{a_j}, v'_{b_j}=v_{b_j}, and $g'_{n+j}(v'_{a_j},v'_{b_j}) = g_{n+j}(v_{a_j},v_{b_j})$ for all $1{\leq}j{\leq}s$, so it holds that $C'(x)$=$C(x)$. We assume an algorithm SIM that on input $(n, s, 1)$ it constructs an arbitrary n-input-s-gate-1-output circuit $C1$, obfuscates $C1$ using $\mathcal{O}$ to get $C1'$ and produces $T(C1')$ as output, where $T(C1') = T(C')$ because, as we have discussed in Section 4.3.3, the topology of the obfuscated circuit is generated by $\mathcal{O}$ in a manner independent of the gate functions of the original circuit. Therefore, $T(C')$ discloses only $(n, s, 1)$.

Now we discuss the overhead factor. The obfuscation process can be considered as working in rounds, where d is initialized to $\lceil\log_2(n)\rceil$ and each round starts with $t_d t_{d-1}\cdots t_1 = 10\cdots0$ and ends when it reaches $11\cdots1$ and d is incremented by 1.

During each round, $\mathcal{O}$ obfuscates 2^{d-1} gates (unless it reaches the last gate) and generates 1 instance of C_π^d, 2^0 instance of C_π^{d-1}, 2^1 instances of C_π^{d-2}, $\cdots$, 2^{d-1-i} instances of C_π^i, $\cdots$, 2^{d-2} instances of C_π^1 and $2\begin{pmatrix} d-1 \\ 0 \end{pmatrix}$ instances of 1-input C_{mux}, $2\begin{pmatrix} d-1 \\ 1 \end{pmatrix}$ instances of 2-input C_{mux}, $\cdots$, $2\begin{pmatrix} d-1 \\ d-1 \end{pmatrix}$ instances of d-input C_{mux} and g'_{n+j}, $\cdots$, $g'_{n+j+2^{d-1}-1}$. Thus, the total number of gates generated during this round is

$$\begin{aligned} & O(3 \cdot 2^{d-1} \cdot (\log_2 3 \cdot 2^{d-1})^2) + O(1) \sum_{i=1}^{d-1} 2^{d-1-i} \cdot 3 \cdot 2^{i-1} \cdot (\log_2(3 \cdot 2^{i-1}))^2 \\ & +2 \sum_{k=0}^{d-1} k \begin{pmatrix} d-1 \\ k \end{pmatrix} + 2^{d-1} \\ & = O(d^2 2^d) + O(2^d) \sum_{i=1}^{d-1} i^2 + (d-1)2^{d-1} + 2^{d-1} \\ & = O(d^3 2^d) \end{aligned} \tag{4.7}$$

and the circuit depth accumulated in this round is

$$\begin{aligned}
&O((\log_2 3 \cdot 2^{d-1})^2) + \sum_{i=1}^{d-1} 2^{d-1-i} O((\log_2 3 \cdot 2^{i-1})^2) + \sum_{k=1}^{d} \begin{pmatrix} d-1 \\ k-1 \end{pmatrix} \log_2 k + 2^{d-1} \\
&= O(d^2) + O(1) \sum_{i=1}^{d-1} i^2 2^{d-i} + O(1) \sum_{k=1}^{d} \begin{pmatrix} d-1 \\ k-1 \end{pmatrix} \log_2 k + 2^{d-1} \\
&= O(2^d) + O(2^d \log_2 d) \\
&= O(2^d \log_2 d)
\end{aligned} \tag{4.8}$$

Suppose that there are R rounds (i.e. $d \in \{\lceil \log_2(n) \rceil, \cdots, \lceil \log_2(n) \rceil + \mathrm{R}-1\}$), then it holds that

$$\sum_{d=\lceil \log_2 n \rceil}^{\lceil \log_2 n \rceil + R - 1} 2^{d-1} \leq s \tag{4.9}$$

namely, these R rounds are sufficient to obfuscate s gates. It follows that $R = O(1) \cdot \log_2(\frac{s}{n})$ and the number of gates in C' is

$$\sum_{d=\lceil \log_2 n \rceil}^{\lceil \log_2 n \rceil + R - 1} O(d^3 2^d) < (\lceil \log_2 n \rceil + R)^3 \sum_{d=\lceil \log_2 n \rceil}^{\lceil \log_2 n \rceil + R - 1} O(2^d) \leq O(\log_2^3 s) \cdot s \tag{4.10}$$

and the depth of C' is

$$\sum_{d=\lceil \log_2 n \rceil}^{\lceil \log_2 n \rceil + R - 1} O(2^d \log_2 d) < \log_2(\log_2 n + R) \sum_{d=\lceil \log_2 n \rceil}^{\lceil \log_2 n \rceil + R - 1} O(2^d) = O(\log_2 \log_2 s) \cdot s \tag{4.11}$$

□

Note that the depth of C' is independent of that of C since $T(C')$ is uniform for all C's that have the same input length, output length and circuit size.

4.4 Summary of the Chapter

We have presented two efficient CTO algorithms and the overhead factors of the depth-efficient CTO and the size-efficient CTO are shown in Table 4.1, where the big-oh notation hides only small constant factors. The depth-efficient CTO is unconditionally better than the size-efficient CTO in terms of depth overhead. In contrast, with regard to size overhead, the size-efficient CTO is superior to the depth-efficient CTO for circuits whose depth d is of order $O(\log^2 s)$ or above. A disadvantage of the size-efficient CTO is that the depth of its obfuscated circuit is $O(\text{loglog}s)s$, approximately the same order as $O(s)$, and hence the resulting circuit loses the advantage of parallel computation, namely, the parallel computation time of C' is of roughly the order as the sequential computation time of C.

Table 4.1: Comparisons between the depth-efficient CTO and the size-efficient CTO, where s, d, s' and d' are respectively the size and depth of the original circuit and those of the obfuscated circuit.

	depth-efficient CTO	size-efficient CTO
size overhead factor (s'/s)	$O(d{\cdot}\log s)$	$O(\log^3 s)$
depth overhead factor (d'/d)	$O(\log s)$	$O(\text{loglog}s)s/d$

Now we give more concrete comparisons using small-depth circuits as an example. Small-depth circuits are a common form of circuits and are defined as circuits with

n inputs, size $O(n^{k_1})$ and depth $O(\log^{k_2} n)$ (k_1 and k_2 are positive integers). In this case, the size overhead factors for depth-efficient CTO and size-efficient CTO are $O(1)\cdot\log^{k_2+1} n$ and $O(1)\cdot\log^3 n$ respectively, and the depth overhead factors for depth-efficient CTO and size-efficient CTO are $O(1)\cdot\log n$ and $O(1)\cdot n^{k_1}\log\log n$ / $\log^{k_2} n$ respectively. Figure 4.5 and Figure 4.6 visualize the above comparisons,

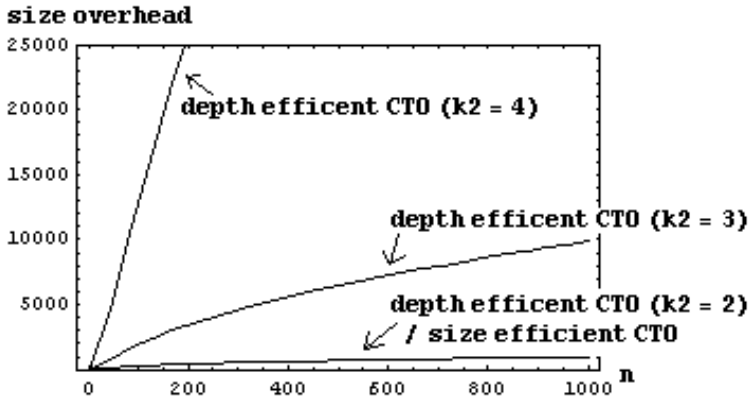

Figure 4.5: The size overheads of size-efficient CTO and depth-efficient CTO (k_2=2, 3, 4) for small-depth circuits.

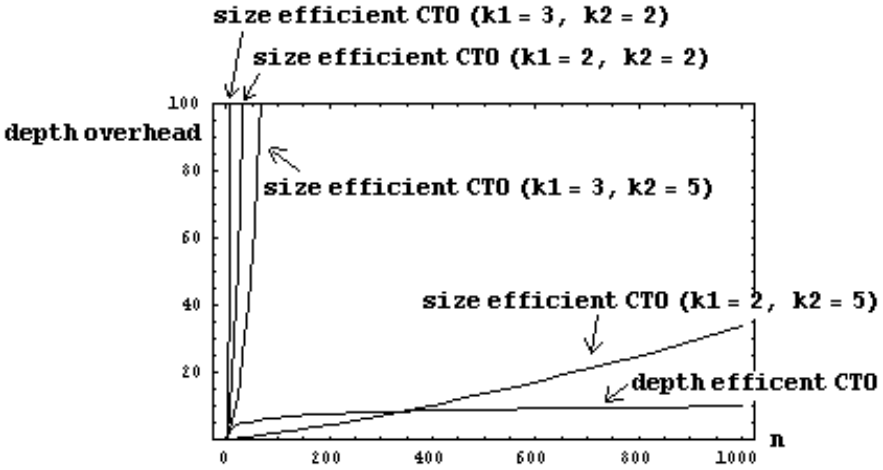

Figure 4.6: The depth overheads of size-efficient CTO (k_1=2, 3 k_2=2, 5) and depth-efficient CTO for small-depth circuits.

where for simplicity $O(1)$ is assumed to be equal to 1.

Updated with the above results, the one-time-use SCC protocol in Chapter 3 (Protocol 3.2) becomes more efficient in reducing unnecessary increase in circuit size and depth. Furthermore, CTO is also indispensable to build reusable SCC protocols which are to be introduced in the next chapter.

Chapter 5

Towards Reusable SCC Protocols for $\mathcal{P}(\{x, C, IR, C(x)\})$

5.1 Building Reusable $\mathcal{P}(\{x, C, IR, C(x)\})$

We reduce the problem of building reusable protocols for $\mathcal{P}(\{x, C, IR, C(x)\})$ to constructing algebraic privacy homomorphism, then provide definition for efficient PH, and discuss its security in the black-box model.

5.1.1 Representation of Gate Functions (Revisited)

The SCC protocols introduced in Chapter 3 are only one-time-use mostly due to the following reasons:

1. Each result v_i ($1 \leq i \leq n+s$) is encrypted deterministically. Thus, if a circuit is computed twice on different inputs, then for each i, Bob can simply compare $E_k(v_i)=W_i^{v_i}$ against $E_k(v_i')=W_i^{v_i'}$ to determine whether $v_i=v_i'$ or not (see the discussion in Section 3.4).

2. The truth table representation does not support secure computation of a gate for multiple times despite that the four items of each truth table are encrypted and permuted. For example, if Bob computes $E_k(g_i)$ twice and he always draws the same item from the table, then he knows that the two inputs of g_i in the first computation are respectively equal to those in the second. Even if he draws different items, he can infer that at least one input carries different values during the two rounds of computation.

We will fix the first problem by introducing semantically secure encryptions, which are by definition probabilistic [1]. As for the second, we need an alternative representation other than truth tables, namely, the Algebraic Normal Form (ANF). That is, any Boolean function can be expressed as an XOR sum of AND products as follows:

$$\begin{aligned}
g(x_1) &= b_0 + b_1 \cdot x_1 \\
g(x_2, x_1) &= b_{00} + b_{01} \cdot x_1 + b_{10} \cdot x_2 + b_{11} \cdot x_2 \cdot x_1 \\
g(x_3, x_2, x_1) &= b_{000} + b_{001} \cdot x_1 + b_{010} \cdot x_2 + b_{011} \cdot x_2 \cdot x_1 \\
&\qquad + b_{100} \cdot x_3 + b_{101} \cdot x_3 \cdot x_1 + b_{110} \cdot x_3 \cdot x_2 + b_{111} \cdot x_3 \cdot x_2 \cdot x_1 \\
&\vdots \\
g(x_n, \cdots, x_1) &= \sum_{a_n \cdots a_1 \in \{0,1\}^n} b_{a_n \cdots a_1} \cdot x_n^{a_n} \cdots x_1^{a_1}
\end{aligned} \tag{5.1}$$

[1] Under a probabilistic encryption, if a plaintext is encrypted twice, then with an overwhelming probability we would get two different ciphertexts, which are called re-enryptions of each other.

where all x_i's and $b_{a_n \cdots a_1}$'s are from {0,1} and "+" and "·" denote XOR and AND respectively. As we assume all gates in a circuit are of fan-in 2, we represent each gate g with $[b_{00}, b_{01}, b_{10}, b_{11}]$ meaning that it computes the function of

$$g(x_2, x_1) = b_{00} + b_{01} \cdot x_1 + b_{10} \cdot x_2 + b_{11} \cdot x_2 \cdot x_1 \tag{5.2}$$

We call this new representation "parameter table" and still use square brackets to enclose the four items. The parameter table representation is equivalent to the truth table representation as they both can represent any Boolean function, and we will show that the former is superior to the latter as it supports repeated computation in its encrypted format.

5.1.2 Reusable $\mathcal{P}(\{x, C, IR, C(x)\})$ Using Algebraic PH

We first construct a reusable SCC protocol for $\mathcal{P}(\{x, C, IR, C(x)\})$ using a special algebraic PH, then show that the special PH is implied by any algebraic PH, and finally we prove the security of the whole protocol.

With CTO, we can obtain an obfuscated circuit whose topology reveals nothing even to unbounded adversaries. Thus, the problem of SCC is reduced to: For each gate g_{n+i}, given its encrypted inputs $E_e(v_{a_i})$ and $E_e(v_{b_i})$, and its encrypted parameter table $[E_e(b_{00}), E_e(b_{01}), E_e(b_{10}), E_e(b_{11})]$, how can Bob obtain $E_e(g_{n+i}(v_{a_i}, v_{b_i}))$ without knowing anything regarding b_{00}, b_{01}, b_{10}, b_{11}, v_{a_i}, v_{b_i} and $g_{n+i}(v_{a_i}, v_{b_i})$? Assume that there is a semantically secure algebraic PH $(G,E,D,\boxplus,\boxtimes)$ (see the

definition in Section 2.2.1) with plaintext space $\mathcal{M}$={0,1}, then the above problem can be easily solved, namely, Bob computes as in Figure 5.1 to get $E_e(g_{n+i}(v_{a_i}, v_{b_i}))$.

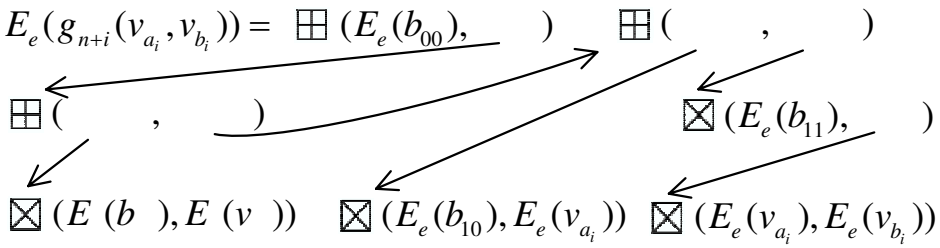

Figure 5.1: Given $E_e(b_{00}), E_e(b_{01}), E_e(b_{10}), E_e(b_{11})$, $E_e(v_{a_i})$ and $E_e(v_{b_i})$, we can obtain $E_e(g_{n+i}(v_{a_i}, v_{b_i}))$ by 3 invocations of "⊞" and 4 invocations of "⊠".

Unfortunately, most PHs are not designed to have exactly a plaintext space of $\mathcal{M}$={0,1} with an exception being the algebraic PH by Sander et al. [22]. Now we show that an algebraic PH with $\mathcal{M}$={0,1} is implied by a more general class of algebraic PHs (Proposition 5.1) or even an arbitrary algebraic PH (Proposition 5.2).

Proposition 5.1. *Assume that (G,E,D,⊞,⊠) is a semantically secure algebraic PH defined on plaintext space $\mathcal{M}'$, and that $\#(\mathcal{M}')$ (the cardinality of $\mathcal{M}'$) is even, and define E' and D' as*

$$\begin{aligned} E'_e(m) &= E_e(m + 2r) \\ D'_d(c) &= D_d(c) \bmod 2 \end{aligned} \tag{5.3}$$

with $m \in \{0,1\}$ and $r \in_U \{0,1,\cdots,\#(\mathcal{M}')/2-1\}$, then the algebraic PH (G,E',D',⊞,⊠) defined on plaintext space $\mathcal{M}$={0,1} is also semantically secure.

Proof sketch. We can check the correctness of the resulting PH as follows:

$$\begin{aligned}
&D'_d(\boxplus(E'_e(m_1), E'_e(m_2))) \\
&= D'_d(\boxplus(E_e(m_1 + 2r_1), E_e(m_2 + 2r_2))) \\
&= D_d(E_e(m_1 + 2r_1 + m_2 + 2r_2)) \bmod 2 \\
&= m_1 + m_2 \bmod 2 \\
&D'_d(\boxtimes(E'_e(m_1), E'_e(m_2))) \\
&= D'_d(\boxtimes(E_e(m_1 + 2r_1), E_e(m_2 + 2r_2))) \\
&= D_d(E_e(m_1 m_2 + 2(m_2 r_1 + m_1 r_2 + 2r_1 r_2))) \bmod 2 \\
&= m_1 m_2 \bmod 2
\end{aligned} \tag{5.4}$$

By the definition of semantic security, ciphertext $E'_e\ (m) = E_e\ (m{+}2r)$ reveals nothing substantial (to computationally bounded adversaries) regarding $m{+}2r$, and hence the least significant bit of $m{+}2r$, namely m. Therefore, the resulting PH is also semantically secure. □

Proposition 5.2. *A semantically secure algebraic PH implies a semantically secure algebraic PH with plaintext space $\mathcal{M}$={0,1}.*

Proof sketch. Suppose that $(G,E,D,\boxplus,\boxtimes)$ is a semantically secure PH with plaintext space $\mathcal{M}'$, then by overriding $\boxplus$ with $\boxplus'$ and restricting the plaintext space to $\mathcal{M}$={0,1}, we obtain a new algebraic PH $(G,E,D,\boxplus',\boxtimes)$, where $\boxplus'(E_e(m_1),E_e(m_2))=$

E_e $((1-m_1m_2\ )(m_1+m_2))$ is computed with the following steps:

$$\begin{aligned}
&E_e(m_1+m_2) = \boxplus(E_e(m_1), E_e(m_2)) \\
&E_e(m_1m_2) = \boxtimes(E_e(m_1), E_e(m_2)) \\
&E_e(-m_1m_2) = \boxtimes(E_e(-1), E_e(m_1m_2)) \\
&E_e(-m_1m_2(m_1+m_2)) = \boxtimes(E_e(-m_1m_2), E_e(m_1+m_2)) \\
&E_e((1-m_1m_2)(m_1+m_2)) = \boxplus(E_e(m_1+m_2), E_e(-m_1m_2(m_1+m_2))) \ .
\end{aligned} \tag{5.5}$$

Note that -1 is the additive inverse of multiplicative identity (i.e. 1) and an instance of $E_e(-1)$ should be revealed to the party doing the above computation, but it does not weaken the security of the resulting PH because in the notion of semantic security an adversary is allowed to have access to the encryption oracle. Therefore, $(G,E,D,\boxplus',\boxtimes)$ with $\mathcal{M}=\{0,1\}$ is still semantically secure. The resulting encryption scheme is additively and multiplicatively homomorphic under $\boxplus'$ and $\boxtimes$ as for $m_1\in\{0,1\}$ and $m_2\in\{0,1\}$ it holds that

m_1	m_2	$\boxplus'(E_e(m_1),E_e(m_2))$	$\boxtimes(E_e(m_1),E_e(m_2))$
0	0	$E_e(0)$	$E_e(0)$
0	1	$E_e(1)$	$E_e(0)$
1	0	$E_e(1)$	$E_e(0)$
1	1	$E_e(0)$	$E_e(1)$

and the conclusion follows. □

Protocol 5.1 (The reusable SCC protocol against semi-honest Bob)**.** *Assume that*

there is an algebraic PH and Alice wants Bob to compute circuit C on inputs $x^{(1)}$, $x^{(2)}$, $\cdots$, $x^{(L)}$, then they execute the following protocol:

1. *Alice transforms C into C' (by means of CTO), modifies the algebraic PH (using the technique in the proof of Proposition 5.2) so that the plaintext space is $\{0,1\}$, randomly selects a pair of encryption decryption keys (e, d), and sends to Bob the encrypted format of C', namely*

$$(n, s', 1), (a'_1, b'_1, E_e(g'_{n+1})), (a'_2, b'_2, E_e(g'_{n+2})), \cdots, (a'_{s'}, b'_{s'}, E_e(g'_{n+s'})) \quad (5.6)$$

   where each g'_i $(n+1 \leq i \leq n+s')$ is represented using parameter table $[b^i_{00}, b^i_{01}, b^i_{10}, b^i_{11}]$ and $E_e(g'_i) = [E_e(b^i_{00}), E_e(b^i_{01}), E_e(b^i_{10}), E_e(b^i_{11})]$.

2. *For each input $x^{(j)} = v^{(j)}_1 \cdots v^{(j)}_n$ $(1 \leq j \leq L)$, Alice sends to Bob $E_e(v^{(j)}_1) \cdots E_e(v^{(j)}_n)$, Bob computes as in Figure 5.1 gate by gate to obtain $E_e(C'(x^{(j)}))$, which is sent to Alice and decrypted to get $C(x^{(j)}) = C'(x^{(j)})$.*

Now we prove that 2-round reusable SCC protocols are implied by algebraic PH, namely, Protocol 5.1 is secure if the algebraic PH is semantically secure.

Theorem 5.1 (correctness and privacy in the semi-honest model)**.** *Let C be the circuit to be computed and $x^{(1)}$, $x^{(2)}$, $\cdots$, $x^{(L)}$ be its inputs, and assume that Bob is semi-honest and computationally bounded, and that algebraic PH $(G,E,D,\boxplus,\boxtimes)$ is semantically secure, then the reusable SCC protocol (Protocol 5.1) correctly and privately (without revealing anything substantial to Bob) computes C on the L inputs.*

Proof. By the correctness of algebraic PH and CTO, semi-honest Bob correctly computes each gate and hence the whole encrypted circuit. By the definition of CTO (Definition 3.2), there is a polynomial-time algorithm such that given trivial information (e.g. $(n,s,1)$ or $(n,s,d,1)$) it efficiently produces $T(C')$, namely

$$(n, s', 1), (a'_1, b'_1), (a'_2, b'_2), \cdots, (a'_{s'}, b'_{s'}) \ . \tag{5.7}$$

In other words, $T(C')$ reveals nothing substantial even to unbounded adversaries. Since gates are encoded with parameter tables and are encrypted together with those inputs $x^{(1)}$, $\cdots$, $\cdots$, $x^{(L)}$, those ciphertexts reveal nothing substantial to computationally bounded Bob by the definition of semantic security. Therefore, the conclusion holds. □

It is much easier to prove that the converse direction is also true, namely, semantically secure algebraic PH is implied by secure 2-round reusable SCC protocols.

Theorem 5.2. *If there exists a 2-round SCC protocol* Π *that can correctly and privately (without revealing anything substantial) compute any circuit* C *on as many inputs as Alice wants against computationally bounded Bob, then semantically secure algebraic PH exists for any plaintext space.*

Proof sketch. A 2-round reusable SCC protocol already implies an encryption scheme ($G1$, $E1$, $D1$) with which Alice hides her private inputs and circuits from Bob and $\{0,1\}$ must be a subset of the plaintext space of ($G1$, $E1$, $D1$). Suppose that we

are to construct an algebraic PH ($G2$, $E2$, $D2$, $\boxplus$, $\boxtimes$) with plaintext space $\mathcal{M}$, then let $G2$ be the same as $G1$, represent each element a of $\mathcal{M}$ with binary format so that $E2$ is done by invoking $E1$ on a bit by bit, and define $D2$ accordingly. Alice compiles two circuits C_+ and $C_\times$ which do addition and multiplication on $\mathcal{M}$ respectively and publish their encrypted formats as $\boxplus$ and $\boxtimes$. Therefore, with $\boxplus$ and $\boxtimes$, anybody (Bob) can do addition and multiplication on ciphertexts but only the one who knows the decryption key (Alice) can decrypt them. As Bob learns nothing substantial from the ciphertexts, ($G2$, $E2$, $D2$, $\boxplus$, $\boxtimes$) is semantically secure.

□

5.1.3 On the Efficiency and Security of Algebraic PH

Before discussing the security of algebraic PH, it is necessary to focus on only efficient ones. For example, the algebraic PH introduced in [22] is unconditionally secure, but it is not efficient as it has a homomorphic operation that inputs two ciphertexts c_1 and c_2 and produces c_3 with $|c_3|=|c_1|+|c_2|$. If Alice and Bob execute Protocol 5.1 using the PH, then Alice takes no advantage of Bob because she has to decrypt each final ciphertext $E_k(C(x^{(j)}))$, whose length is an exponential function in the circuit depth and so is the amount of time needed for decryption. Note that the time it takes to decrypt a ciphertext is at least proportional to its length.

Carried to the extreme, the inefficiency can be illustrated with a trivial PH: Suppose that we have a semantically secure encryption scheme (G, E, D), then for

any operation "$\star$" on plaintexts, the corresponding homomorphic function, denoted by "$\bigstar$", simply takes c_1 and c_2 as inputs and write the output as $c_1 \bigstar c_2$ (no further actions are taken), and decryption is defined as $D_d(c_1 \bigstar c_2) = D_d(c_1) \star D_d(c_2)$. It is not hard to prove that the extended PH (G, E, D, $\bigstar$) is also semantically secure, but it makes the protocol trivial since Alice has to spend some extra time on encryption and decryption before performing the necessary computation (computing the circuit in plaintext).

Definition 5.1 (Strongly and weakly efficient PH)**.** *A PH is strongly (resp., weakly) efficient if for any of its homomorphic function* $\bigstar$*, and for any ciphertext-pair* (c_1, c_2)*, the length of the resulting ciphertext* $\bigstar(c_1, c_2)$ *is not larger than* $max(|c_1|, |c_2|)$ *(resp.,* $max(|c_1|, |c_2|)+Z$*), where* Z *is a positive constant and function max returns the larger value of the two candidates.*

With the strongly efficient algebraic PH, computing the circuit in ciphertext incurs no increase on the length of the final output ciphertext, and with the weak counterpart the increase is bounded by $Z \cdot d'$ with d' being the circuit depth. It turns out to be relatively easy to construct an efficient PH that supports only one homomorphic function, and most existing additive or multiplicative (but not both) PHs are strongly efficient in the sense of Definition 5.1. In contrast, most algebraic PHs are not efficient with an exception being the one introduced in [23], which unfortunately supports only one multiplication on ciphertexts and hence is not truly multiplicative.

How secure an efficient algebraic PH can be remains an open problem, but its upper bound is quite clear, namely, it is at most semantically secure under non-adaptive chosen-ciphertext attacks (IND-CCA1 secure): First, it cannot be non-malleably secure, namely, given any plaintext-ciphertext pair $(a, E_e(a))$ with $a \neq 0$ (resp., $a \neq 1$), an adversary is able to increment (resp., multiply) by a the semantics of any ciphertext c by replacing c with c', where $c' \leftarrow \boxplus(c, E_e(a))$ (resp., $c' \leftarrow \boxtimes(c, E_e(a))$). Second, Dolev et al. [65] showed that under adaptive chosen-ciphertext attacks (CCA2) semantic security and non-malleable security are equivalent. Since we have already shown that PH is not non-malleably secure (under CCA2), it cannot be IND-CCA2 secure either.

Next we assume an IND-CCA1 secure encryption scheme (G, E, D) having oracle access to $\boxplus$ and $\boxtimes$ and prove that the resulting PH, $(G, E, D, \boxplus, \boxtimes)$, in the black-box model preserves IND-CCA1 security. In other words, the homomorphic properties take no effect on the security, which is contrary to the negative result of Boneh and Lipton [26] that the homomorphic properties will lead to potential attacks on deterministic algebraic PH.

The IND-CCA1 security for an encryption scheme (G, E, D) can be modeled with the following game [11, Protocol 14.1]:

Game 5.1.

1. *Alice chooses a pair of keys by* $(e,d) \leftarrow G(1^t)$*, where the encryption key* e *is revealed to Bob if* (G,E,D) *is a public-key encryption scheme.*

2. *Bob can have oracle access to E_e and D_d until he is satisfied.*

3. *Based on the information achieved, Bob selects two distinct messages m_0 and m_1 of the same length and sends them to Alice.*

4. *Alice tosses a fair coin $b \in_U \{0,1\}$ and selects a value c^* from the distribution of $E_e(m_b)$.*

5. *After receiving c^*, Bob is only allowed to have oracle access to E_e. Finally, he guesses b by answering either 0 or 1.*

If Bob has no strategy to win Game 5.1 better than random guessing, then (G,E,D) is IND-CCA1 secure. The rigorous definition for IND-CCA1 is given by Goldreich as follows [7, Definition 5.4.14]:

Definition 5.2 (IND-CCA1 security)**.** *For public-key schemes: A public-key encryption scheme, (G,E,D), is said to be* ***IND-CCA1 secure*** *if for every pair of probabilistic polynomial oracle machines, A_1 and A_2, for every positive polynomial p, and all sufficiently large t and $z \in \{0,1\}^{poly(t)}$ it holds that*

$$|p_{t,z}^{(0)} - p_{t,z}^{(1)}| < \frac{1}{p(t)} \tag{5.8}$$

where

$$p_{t,z}^{(i)} \stackrel{def}{=} \Pr\left[\begin{array}{c} v = 0, where \\ ((x_0, x_1), \sigma) \leftarrow A_1^{E_e, D_d}(e, z) \\ c^* \leftarrow E_e(x_i) \\ v \leftarrow A_2^{E_e}(\sigma, c^*) \end{array}\right] \tag{5.9}$$

(e,d) $\leftarrow G(1^t)$, $|x_0|=|x_1|=t$ *and the probability is taken over the internal coin tosses of* G, E_e, A_1 *and* A_2.

For private-key schemes: The definition is identical except that A_1 *gets the security parameter* 1^t *instead of the encryption key* e.

In the above definition, Bob is decoupled into a pair of oracle machines ($A_1^{E_e,D_d}$, $A_2^{E_e}$). That is, A_1 has oracle access to both E_e and D_d (see step 2 in Game 5.1) while A_2 is restricted to E_e (step 5). In the joint work of A_1 and A_2, σ denotes the state information A_1 passes to A_2. Since we use non-uniform formulations z is a non-uniform auxiliary input of A_1 (A_2's counterpart is included in σ). Finally, A_2 outputs v as its guess.

The formal definition of IND-CCA1 security for (G, E, D, $\boxplus$, $\boxtimes$) is almost same as Definition 5.2 except that Bob can have oracle access to $\boxplus$ and $\boxtimes$ at all times, namely, ($A_1^{E_e,D_d}$,$A_2^{E_e}$) is replaced by ($A_1^{E_e,D_d,\boxplus,\boxtimes}$,$A_2^{E_e,\boxplus,\boxtimes}$).

Theorem 5.3 (IND-CCA1 secure PH in the black-box model)**.** *Let (G, E, D) be an IND-CCA1 secure encryption scheme, let* $\boxplus$ *and* $\boxtimes$ *be as in Equation (2.7) and Equation (2.8) respectively and assume that only oracle access is allowed to* $\boxplus$ *and*

⊠, *then the PH (G, E, D, ⊞, ⊠) in the black-box model is also IND-CCA1 secure.*

Proof. As discussed, we only need to prove that the condition of Definition 5.2 still holds if we replace $(A_1^{E_e,D_d}, A_2^{E_e})$ by $(A_1^{E_e,D_d,\boxplus,\boxtimes}, A_2^{E_e,\boxplus,\boxtimes})$. The equivalence of $A_1^{E_e,D_d}$ and $A_1^{E_e,D_d,\boxplus,\boxtimes}$ is quite straightforward as both ⊞ and ⊠ are implied by E_e and D_d. In other words, we can efficiently emulate ⊞ (resp., ⊠) by decrypting the input ciphertexts, computing the sum (resp., product) of the corresponding plaintexts and encrypting the result. The remaining difficulty is the reduction from $A_2^{E_e,\boxplus,\boxtimes}$ to $A_2^{E_e}$. Obviously, the input-output behaviors of ⊞ and ⊠ cannot be emulated using only E_e, but we can implement a function $\mathcal{O}^{+,\times}$ using E_e such that its output distribution is computationally indistinguishable to those of ⊞ and ⊠. For example, we can let $\mathcal{O}^{+,\times}$ be as follows:

```
O^{+,×} (E_e(x_0), E_e(x_1)){
    return E_e(0);
}
```

while the corresponding outputs of ⊞ and ⊠ are $E_e(x_0+x_1)$ and $E_e(x_0\times x_1)$ respectively. By the definition of semantic security, $A_2^{E_e}$ cannot distinguish between the distribution of $E_e(0)$ and that of $E_e(x_0+x_1)$ or $E_e(x_0\times x_1)$, namely, whatever can be efficiently computed from the oracle access to ⊞ and ⊠ can also efficiently computed from scratch. Thus, replacing (⊞,⊠) by $\mathcal{O}^{+,\times}$ has no effect on A_2's decision, and $A_2^{E_e,\boxplus,\boxtimes}$ and $A_2^{E_e,\mathcal{O}^{+,\times}}$ compute almost the same function (with a negligible difference). Therefore, it suffices that $A_2^{E_e,\boxplus,\boxtimes}$ can be reduced to $A_2^{E_e,\mathcal{O}^{+,\times}}$, and then to

$A_2^{E_e}$ in a computational sense and the conclusion immediately follows. □

5.1.4 Extending the Protocol to the Malicious Model

Protocol 5.1 only works in the semi-honest model as we cannot prevent Bob always returning to Alice a flipped (recall that no PH is non-malleably secure) result $E_e(C(x^{(j)})\oplus 1)$ instead of $E_e(C(x^{(j)}))$, but if Alice has a prior knowledge, $(x^{(0)}, C(x^{(0)}))$, regarding C, then she can defeat malicious Bob by making use of semantic security as follows:

Protocol 5.2 (The reusable SCC protocol against malicious Bob)**.** *Assume that there is an algebraic PH and Alice wants Bob to compute circuit C on inputs $x^{(1)}$, $x^{(2)}$, $\cdots$, $x^{(L)}$, and that Alice knows a pair of input-output $(x^{(0)}, C(x^{(0)}))$, then they execute the following protocol:*

1. *As in Protocol 5.1, Alice transforms C into C', adjusts the plaintext space of the algebraic PH to $\{0,1\}$, randomly selects a pair of encryption decryption keys (e, d), and sends to Bob the encrypted format of C'.*

2. *For each input $x^{(j)}=v_1^{(j)}\cdots v_n^{(j)}$ $(1\leq j\leq L)$, instead of sending to Bob the encrypted $x^{(j)}$, Alice sends in sequence of m challenges, where for each challenge Alice tosses a fair coin $b\in_U\{0,1\}$, and if $b=0$, then let the challenge be a re-encryption of $E_e(x^{(j)})=E_e(v_1^{(j)})\cdots E_e(v_n^{(j)})$, otherwise $(b=1)$ let the challenge be a re-encryption of $E_e(x^{(0)})$.*

3. *On each challenge, Bob computes the encrypted circuit (as described in Protocol 5.1) to get the corresponding encrypted output and sends it back to Alice.*

4. *Alice checks if all challenges that correspond to $E_e(x^{(0)})$ are replied with $E_e($ $C'(x^{(0)})$ $)$ and the rest replies (corresponding to $E_e(x^{(j)})$) are consistent (decrypted to the same result). If so, then we get $C(x^{(j)})$=$C'(x^{(j)})$, otherwise, Alice detects Bob's cheating.*

Theorem 5.4 (correctness and privacy in the malicious model)**.** *Let C be the circuit to be computed and $x^{(1)}$, $x^{(2)}$, $\cdots$, $x^{(L)}$ be its inputs, and assume that Bob is computationally bounded, and that the algebraic PH is semantically secure, then Protocol 5.2 correctly and privately (without revealing anything substantial to Bob) computes C on the L inputs in the semi-honest model, and if Bob cheats Alice during the computation of $x^{(j)}$, Alice will detect it with a probability of $1-2^{-m}$.*

Proof. Protocol 5.2 is extended from Protocol 5.1, and hence the correctness and privacy in the semi-honest model can be proved analogously. Bob can cheat Alice successfully if and only if Bob flips all the encrypted results of $E_e(C'(x^{(j)}))$ and honestly returns the rest (those corresponding to $E_e(C'(x^{(0)}))$). By the definition of semantic security, Bob learns nothing substantial from the ciphertexts and hence he cannot decide which result(s) to flip better than random guessing. There are

$$\binom{m}{0} + \binom{m}{1} + \cdots + \binom{m}{m} = 2^m \tag{5.10}$$

possible choices for Bob to flip the m results. Therefore, Bob wins with a probability of 2^{-m}. □

5.1.5 Remarks

We have presented 2-round reusable SCC protocols for $\mathcal{P}(\{x,\ C,\ IR,\ C(x)\})$ and proved their equivalence to algebraic PH. Alice encrypts the circuit C' only once and lets Bob compute it for as many inputs as she want and thus Alice can make uses of Bob's computing power. For example, in the semi-honest model, suppose that an encryption operation costs A steps of computation and a decryption operation takes B steps, then Alice spends $4A{\cdot}s'$ steps on encrypting C', $A{\cdot}n$ steps on encrypting an input and B steps on decrypting an output. The average number of steps for computing each input is

$$\frac{4A \cdot s' + L(A \cdot n + B)}{L} = \frac{4A \cdot s'}{L} + A \cdot n + B \ , \tag{5.11}$$

which converges to $A{\cdot}n + B$ as L increases. Thus, the more rounds of computation Bob performs, the more advantages Alice gains from the protocol.

The results of this section (Section 5.1) can be summarized as follows:

1. It is well-known that algebraic PH defined on plaintext space {0,1} allows Bob to compute Boolean circuits using ciphertexts, but protocols were usually carried out (e.g. [66, 22]) assuming that Bob knows all the gate functions.

We hide the gate functions from Bob by representing them with parameter tables that can be encrypted and then computed repeatedly without revealing anything substantial.

2. We show how to construct an algebraic PH with plaintext space {0,1} from an arbitrary algebraic PH, and hence the former is implied by the latter.

3. With the above-listed results and those of CTO, the problem of constructing 2-round reusable SCC protocols is reduced to finding algebraic PH and vice versa. Although PH cannot be non-malleably secure, we propose a reusable SCC protocol (Protocol 5.2) that defeats malicious Bob.

5.2 Hardware Implementation of Reusable $\mathcal{P}(\{x, C, IR, C(x)\})$

Instead of using privacy homomorphism, we introduce how to implement $\mathcal{P}(\{x, C, IR, C(x)\})$ with hardware to combat probing attacks.

5.2.1 Securing Circuits against Probing Attacks

In the past few decades, significant progress has been made in building most cryptographic problems on complexity-theoretic foundations. However, most analyses are conducted only on algorithm level and an indiscreet implementation may render a provably secure algorithm fragile. This is because the implementation of an

algorithm may not be a black-box, namely, it may reveal some private information regarding a computation (e.g., a private key). For example, the power consumption or the time it takes for some operation might vary for different internal operands [67, 68], which may be exploited by attackers to break a sound algorithm (e.g. RSA) without necessarily solving the underlying intractable problem (e.g. integer factorization). Another threat is the probing attack (a form of side channel attacks), where an attacker can place metal needles on wires (of a circuit) chosen at his choice and read the corresponding results during the computation [69, 70]. Several countermeasures [71, 72, 73] against side channel attacks haven been proposed without mathematical justification, some of them already broken [74, 75].

The reusable $\mathcal{P}(\{x,\ C,\ IR,\ C(x)\})$ corresponds to a practical problem, which is more difficult than the one in Section 4.1 as the adversary is better equipped this time. That is, computationally bounded Alice is about to implement a private function f on a circuit against computationally bounded Bob, who is capable of observing the circuit topology (i.e. knowing $T(C)$) as well as conducting a probing attack (i.e. knowing x, IR, $C(x)$). Then, how could f be implemented such that computationally bounded Bob learns nothing substantial from the repeated computation of circuit C (on different inputs)?

Note that Bob knows x, $T(C)$, IR and $C(x)$ and it only remains to recover the functions of the gates, which is an easy task (by observing the input-output behavior of each gate during the repeated computation) given x, IR and $C(x)$. We

will solve the problem by means of stateful gates, namely, gates augmented with private memories to make their functions change statefully. As a result, the input-output behaviors of gates reveal nothing substantial to Bob.

5.2.2 Modified Self-shrinking Generators with Constant Output Rates

Before presenting the solution, we introduce a simple and efficient PRG candidate, namely, self-shrinking generators (SSGs). An SSG operates by applying shrinking rules to a single LFSR (introduced in Section 2.2.2): Consider the sequence of an LFSR, $(s_0, s_1, s_2, \cdots)$, as a sequence of bit pairs

$$(s_0, s_1), (s_2, s_3), \cdots \tag{5.12}$$

and for each pair (s_{2i},s_{2i+1}), s_{2i+1} is sequentially produced as the output of the SSG iff s_{2i}=1 (otherwise discarded). As the self-shrinking generator comprises only one LFSR, the security parameter here is the state length (of the LFSR) t.

The drawback of the original SSG is that its output rate is determined by control bits s_{2i}'s and thus not constant. A straightforward method to solve the problem is to buffer the outputs with a memory. To save memory, we use an alternative approach:

For each $2t$-bit subsequence of the LFSR

$$s_{2tj}, s_{2tj+1}, s_{2tj+2}, s_{2tj+3}, \cdots, s_{2tj+2t-2}, s_{2tj+2t-1}, \tag{5.13}$$

output only the first $s_{2tj+2q+1}$ $(0 \leq q < t)$ whose control bit $s_{2tj+2q}=1$, namely, the j-th output of the generator

$$\mathcal{G}[j] = \mathcal{L}[2tj + 2q + 1] = s_{2tj+2q+1} \tag{5.14}$$

and q satisfies

$$s_{2tj}s_{2tj+2}\cdots s_{2tj+2q-2}s_{2tj+2q} = 00\cdots01 \ , \tag{5.15}$$

where $\mathcal{G}[j]$ and $\mathcal{L}[k]$ denote respectively the j-th output of SSG $\mathcal{G}$ and the k-th output of LFSR $\mathcal{L}$. Note that such a $q \in \{0, 1, \cdots, t-1\}$ always exists as for any j, $s_{2tj}s_{2tj+2}\cdots s_{2tj+2t-2}$ is not all zero (by the property of the LFSR-sequence, see e.g. [34]). Thus, the output rate of the modified SSG is regularly clocked (i.e., 1 bit output for every $2t$-bit-LFSR-sequence) by slowing down the output rate $t/2$ times compared with the original SSG. The output of the modified SSG can be viewed as a subsequence of that of the original SSG, so the security remains the same (i.e., a subsequence of a pseudorandom sequence is also pseudorandom). Hereafter, we mean by SSG the modified version instead of the original one.

5.2.3 Overview of the Protocol

As shown in Figure 5.2, we augment a stateless gate with a t-bit private memory and some XOR taps to produce a stateful gate (s-gate hereafter) such that (1) the state is updated using XOR taps in a way analogous to an LFSR (see Section 2.2.2); (2) the function of the s-gate changes statefully by XORing the inputs and output of the stateless gate with XOR sums of some memory contents. When physically encapsulated, the s-gate protects itself from external attacks (any tampering by force would only destroy the memory contents due to the multi-layer structure of the present-day gate implementation technique). Therefore, we assume that the s-gate is a black-box and adversaries can only learn its input-output behavior by probing its incoming and outgoing wires. We first introduce the protocol and then detail how to implement the s-gates with minimal number of memory cells and XOR taps.

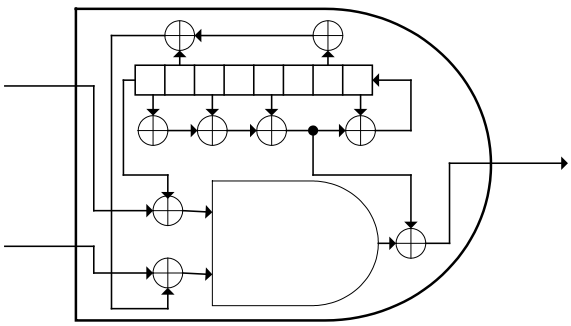

Figure 5.2: A stateful gate internally consists of a stateless gate, a private memory and some XOR taps on the memory, where the function of the s-gate depends on that of the stateless gate and the current state.

Protocol 5.3. *Assume that Alice is to implement function f (with minimized circuit*

C) to repeatedly compute inputs $x^{(1)}$, $x^{(2)}$, $\cdots$, $x^{(L)}$ *against Bob, then they do as follows:*

1. *(Alice's task): Alice transforms C into C′ (by means of CTO), and for each* $1 \leq i \leq n+s'$, *she independently, uniformly and randomly chooses a t-bit non-zero key* $k^{(i)}$ *from*

$$\mathcal{K} = \{0,1\}^t - \{\overbrace{0\cdots0}^{t}\} \tag{5.16}$$

and implement a stateful circuit C″ by replacing each stateless gate g'_{n+i} *(*$1 \leq i \leq s'$*) in C′ with the stateful counterpart, whose function at state j is*

$$g''_{n+i}(v''_{a'_i}, v''_{b'_i})[j] = g'_{n+i}(v''_{a'_i} \oplus \mathcal{G}^{(a'_i)}[j], v''_{b'_i} \oplus \mathcal{G}^{(b'_i)}[j]) \oplus \mathcal{G}^{(n+i)}[j] \tag{5.17}$$

where $\mathcal{G}^{(i)}$ *[j] denotes the j-th output of SSG* $\mathcal{G}^{(i)}$ *with initial state* $k^{(i)}$.

2. *(Computation of the circuit): For each input* $x^{(j)} = v'^{(j)}_1 \cdots v'^{(j)}_n$ *(*$1 \leq j \leq L$*), encrypt it by XORing each* $v'^{(j)}_i$ *(*$1 \leq i \leq n$*) with* $\mathcal{G}^{(i)}$ *[j], and feed the encrypted input to the stateful circuit at state j. After computation, XOR the circuit output with* $\mathcal{G}^{(n+s')}$ *[j] to get* $C(x^{(j)}) = C'(x^{(j)}) = v'^{(j)}_{n+s'}$.

3. *(Bob's attack): During the L rounds of computation, Bob is able to know the topology of the stateful circuit and read (by conducting a probing attack) its inputs, intermediate results and outputs.*

Theorem 5.5 (correctness and privacy)**.** *Let f be the function to be computed and*

$x^{(1)}$, $x^{(2)}$, $\cdots$, $x^{(L)}$ be its inputs, assume that computationally bounded Bob knows the topology of the implemented circuit as well as its inputs, intermediate results and outputs, and assume that SSG $\mathcal{G}$ is a PRG, then Protocol 5.3 correctly and privately (without revealing anything substantial to Bob) computes f on the L inputs.

Proof. Both C and C' compute function f, so we only need to show that C'' computes the same function as C' does. C'' and C' share the same topology, and we denote by v''_i and v'_i $(1{\leq}i{\leq}n{+}s')$ the result of node i in C'' and that in C' respectively. We claim that for any same input, $v''_i{=}v'_i{\oplus}\mathcal{G}^{(i)}[j]$ holds for all $1{\leq}i{\leq}n{+}s'$. It is clear that $v''_i{=}v'_i{\oplus}\mathcal{G}^{(i)}[j]$ holds for $1{\leq}i{\leq}n$, and by induction,

$$\begin{aligned} v''_{n+i} &= g'_{n+i}(v''_{a'_i} \oplus G^{(a'_i)}[j], v''_{b'_i} \oplus G^{(b'_i)}[j]) \oplus G^{(n+i)}[j] \\ &= g'_{n+i}(v'_{a'_i}, v'_{b'_i}) \oplus G^{(n+i)}[j] = v'_{n+i} \oplus G^{(n+i)}[j] \end{aligned} \tag{5.18}$$

holds for $1{\leq}i{\leq}s'$. Therefore, C'' outputs $v''_{n+s'}{=}v'_{n+s'}{\oplus}G^{(n+s')}\,[j]$, which is decrypted to get $v'_{n+s'}$, namely, C'' correctly computes the function of C', namely f. As for the privacy, $T(C''){=}T(C')$ reveals nothing substantial to Bob by the definition of CTO, and since $v''_i{=}v'_i{\oplus}\mathcal{G}^{(i)}[j]$ holds for $1{\leq}i{\leq}n{+}s'$ and each $k^{(i)}$ is chosen independent from each other, computationally bounded Bob only get only pseudorandom noise (by the definition of PRG) from v''_i during the repeated computation. □

5.2.4 Implementing an S-gate with a t-bit memory

As introduced in Protocol 5.3, the function of each s-gate depends on the states of 3 SSGs, which are all t bits long and are chosen independently from one another. It seems that we need a $3t$-bit memory to implement the s-gate and more generally, an $(F+1)$-bit memory for an s-gate of fan-in F, but we will show that a t-bit memory suffices for any gate, no matter of its fan-in.

Suppose that we are to implement an s-gate computing the following function

$$g''_{F+1}(v''_1, \cdots, v''_F)[j] = g'_{F+1}(v''_1 \oplus \mathcal{G}^{(1)}[j], v''_2 \oplus \mathcal{G}^{(2)}[j], \cdots, v''_F \oplus \mathcal{G}^{(F)}[j]) \oplus \mathcal{G}^{(F+1)}[j] \tag{5.19}$$

with g'_{F+1} being the function of the corresponding stateless gate, then we need to emulate $\mathcal{G}^{(1)}$, $\mathcal{G}^{(2)}$, $\cdots$, $\mathcal{G}^{(F+1)}$, namely, $F+1$ LFSRs, denoted by $\mathcal{L}^{(1)}$, $\mathcal{L}^{(2)}$, $\cdots$, $\mathcal{L}^{(F+1)}$ respectively. The initial states of the $F+1$ LFSRs are independently chosen, but they all have the same feedback vector. As illustrated in Figure 5.2, we can implement only one LFSR (e.g., $\mathcal{L}^{(1)}$), represent the outputs of the rest LFSRs using the state of the one implemented (e.g., $\mathcal{L}^{(1)}$), determine the output of each $\mathcal{G}^{(i)}$ $(1 \leq i \leq F+1)$ from the corresponding $\mathcal{L}^{(i)}$, XOR $\mathcal{G}^{(i)}[j]$ $(1 \leq i \leq F)$ with v''_i prior to being inputted to the stateless g'_{F+1} and XOR the result with $\mathcal{G}^{(F+1)}[j]$ prior to being produced as an output.

Proposition 5.3. *Suppose that $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(i)}$ are two LFSRs with the same update matrix M and that their j-th states are S_j and $S_j^{(i)}$ respectively, then there exists a*

$d_i \in \{0, \cdots, 2^t - 2\}$ such that $S_j^{(i)}$=$S_j \cdot M^{d_i}$ holds for all $j \geq 0$.

Proof. Due to the periodic property of LFSRs, there exists a $d_i \in \{0, \cdots, 2^t - 2\}$ such that S_{d_i}=$S_0^{(i)}$. According to Equation (2.10) (see Section 2.2.2), it holds that

$$S_{d_i} = (s_{d_i}, \cdots, s_{d_i+t-1}) = S_0 \cdot M^{d_i} \tag{5.20}$$

where M is non-singular since a_0=1 (i.e. an XOR tap over the first memory cell) is a necessary (but not sufficient) condition of having maximized period of LFSR-sequence (cf. [31]). Thus, we have

$$S_0^{(i)} = S_0 \cdot M^{d_i} \tag{5.21}$$

and the conclusion follows if we right multiply both sides of the above equation with M^j. □

Thus, we can learn the j-th state of $\mathcal{L}^{(i)}$ (i.e., $S_j^{(i)}$) from the corresponding state of $\mathcal{L}^{(1)}$ (i.e., S_j) and M^{d_i}. It is not necessary to compute d_i and then raise M to power d_i to get M^{d_i} as it would be inefficient to determine the value of $d_i \in \{0, \cdots, 2^t - 2\}$. We introduce how to compute M^{d_i} in time $O(t^3)$ in Section 5.2.5.

After we get M^{d_i}, the output of $\mathcal{L}^{(i)}$ ($2 \leq i \leq F+1$) can be written as a linear combination of the memory contents of $\mathcal{L}^{(1)}$, namely, if we denote the first column

vector of M^{d_i} by $(b_0^{(i)}, \cdots\ b_{t-1}^{(i)})$, then the first sub-equation of $S_j^{(i)}$=S_{j+d_i}=$S_j \cdot M^{d_i}$ is

$$\mathcal{L}^{(i)}[j] = s_j^{(i)} = s_{j+d_i} = \sum_{p=0}^{t-1} b_p^{(i)} s_{j+p} = b_0^{(i)} s_j \oplus \cdots \oplus b_{t-1}^{(i)} s_{j+t-1} \tag{5.22}$$

where $\mathcal{L}^{(i)}[j]$ is the output of $\mathcal{L}^{(i)}$ at the j-th state and $s_j \cdots s_{j+t-1}$ is the corresponding memory contents of $\mathcal{L}^{(1)}$. Finally, the output of $\mathcal{G}^{(i)}$ is obtained from that of $\mathcal{L}^{(i)}$ (see Equation (5.14) and (5.15)). Thus, we need $w(b_0^{(i)} \cdots b_{t-1}^{(i)})-1$ XOR taps to represent the output of $\mathcal{L}^{(i)}$, where $w(\cdot)$ denotes the hamming weight (the number of 1's) of a binary string.

Therefore, each s-gate only requires a memory of t bits. We proceed to the estimate of the number of XOR taps (within each s-gate), which are used to update the internal state, represent the outputs of $\mathcal{L}^{(2)}$, $\cdots$, $\mathcal{L}^{(F+1)}$ and do encryption/decryption. The total number is

$$\begin{aligned} & w(a_0 \cdots a_{t-1}) - 1 + \sum_{i=2}^{F+1} (w(b_0^{(i)} \cdots b_{t-1}^{(i)}) - 1) + F + 1 \\ & = w(a_0 \cdots a_{t-1}) + \sum_{i=2}^{F+1} w(b_0^{(i)} \cdots b_{t-1}^{(i)}) \end{aligned} \tag{5.23}$$

As the initial state of each $\mathcal{L}^{(i)}$ $(1 \leq i \leq F+1)$ is uniformly and independently chosen from key space $\mathcal{K}$ (see Equation 5.16), $b_0^{(i)} \cdots b_{t-1}^{(i)}$ $(2 \leq i \leq F+1)$ is also uniformly distributed (refer to Proposition 5.4) in $\mathcal{K}$ and hence the expectation value of w

$(b_0^{(i)} \cdots b_{t-1}^{(i)})$ is

$$\begin{aligned} & \tfrac{1}{2^t-1}(1 \cdot \begin{pmatrix} t \\ 1 \end{pmatrix} + 2 \cdot \begin{pmatrix} t \\ 2 \end{pmatrix} + \cdots + t \cdot \begin{pmatrix} t \\ t \end{pmatrix}) \\ & = \tfrac{t}{2^t-1}(\begin{pmatrix} t-1 \\ 0 \end{pmatrix} + \begin{pmatrix} t-1 \\ 1 \end{pmatrix} + \cdots + \begin{pmatrix} t-1 \\ t-1 \end{pmatrix}) \\ & = \tfrac{2^{(t-1)}}{2^t-1} t \approx \tfrac{t}{2} \end{aligned} \tag{5.24}$$

It follows that the expectation value of (5.23) is

$$w(a_0 \cdots a_{t-1}) + \frac{tF}{2} < t + \frac{tF}{2} \ . \tag{5.25}$$

Proposition 5.4 (Distribution of $b_0^{(i)} \cdots b_{t-1}^{(i)}$)**.** *Suppose that $\mathcal{L}^{(1)}$ and $\mathcal{L}^{(i)}$ are two LFSRs with the same update matrix M, denote their j-th states by S_j and $S_j^{(i)}$ respectively with $S_j^{(i)} = S_j \cdot M^{d_i}$ and $0 \leq d_i \leq 2^t - 2$ (see Proposition 5.3), denote the first column vector of M^{d_i} by $b_0^{(i)} \cdots b_{t-1}^{(i)}$, then for S_0 and $S_0^{(i)}$ that are independently and uniformly distributed in*

$$\mathcal{K} = \{0,1\}^t - \{\overbrace{0 \cdots 0}^{t}\} \tag{5.26}$$

the first column vector $b_0^{(i)} \cdots b_{t-1}^{(i)}$ of the resulting M^{d_i} is also uniformly distributed in $\mathcal{K}$.

Proof. It holds that $S_j^{(i)} = S_{j+d_i}$ since they both equal to $S_j \cdot M^{d_i}$, so it holds that

$$s_{j+d_i} = b_0^{(i)} s_j + b_1^{(i)} s_{j+1} + \cdots + b_{t-1}^{(i)} s_{j+t-1} \ . \tag{5.27}$$

Thus, $b_0^{(i)} \cdots b_{t-1}^{(i)}$ cannot be all-zero as it implies $s_{j+d_i}=0$ for all $j \geq 0$, namely, $b_0^{(i)} \cdots b_{t-1}^{(i)} \in \mathcal{K}$. We define a function $\hbar : \{0, 1, \cdots, 2^t - 2\} \rightarrow \mathcal{K}$ as

$$\hbar(d_i) = b_0^{(i)} \cdots b_{t-1}^{(i)} \ . \tag{5.28}$$

For any $x, y \in \{0, 1, \cdots, 2^t - 2\}$ with $x<y$, it must hold that $\hbar(x) \neq \hbar(y)$, otherwise, it follows that $s_{x+j}=s_{y+j}$ for all $j \geq 0$ and hence the period of the LFSR (i.e. $y-x$) is less than 2^t-1. Therefore, $\hbar$ is a bijection and for an arbitrary $z \in \mathcal{K}$, the following equation holds:

$$\begin{aligned}
&\Pr(b_0^{(i)} \cdots b_{t-1}^{(i)} = z) \\
&= \sum_{x \in \mathcal{K}} \Pr(S_0 = x \text{ and } b_0^{(i)} \cdots b_{t-1}^{(i)} = z) \\
&= \sum_{x \in \mathcal{K}} \Pr(S_0 = x \text{ and } S_0^{(i)} = x \cdot M^{\hbar^{-1}(b_0^{(i)} \cdots b_{t-1}^{(i)})}) \\
&= \sum_{x \in \mathcal{K}} \Pr(S_0 = x) \Pr(S_0^{(i)} = x \cdot M^{\hbar^{-1}(b_0^{(i)} \cdots b_{t-1}^{(i)})}) \\
&= \tfrac{1}{\#(\mathcal{K})}
\end{aligned} \tag{5.29}$$

namely, $b_0^{(i)} \cdots b_{t-1}^{(i)}$ is uniformly distributed in $\mathcal{K}$. □

Minimization results. Each s-gate of fan-in F can be constructed with a t-bit

memory and averagely no more than $t(1+F/2)$ XOR taps. We note that $t(1+F/2)$ is an overestimate in the sense that there would be (partially) overlapping XOR taps on an LFSR. For instance, in Figure 5.3, by reusing overlapping XOR taps, the minimum number of taps is 4 instead of

$$w(s_0 s_1 \cdots s_7) - 1 + w(b_0^{(2)} b_1^{(2)} \cdots b_7^{(2)}) - 1 = 6 \ . \tag{5.30}$$

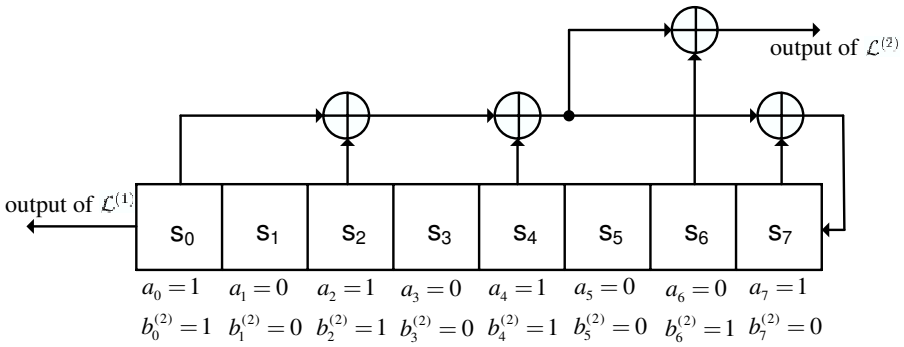

Figure 5.3: Two LFSRs sharing the same memory with feedback vector 10101001 and $b_0^{(2)} b_1^{(2)} \cdots b_7^{(2)}$=10101010.

Since we assume gates of fan-in 2, the s-gates in Protocol 5.3 can all be efficiently implemented with a t-bit memory and no more than $2t$ XOR taps.

5.2.5 Obtaining M^{d_i} from S_0, S_{d_i} and M in time $O(t^3)$

We recall that S_0=$(s_0, \cdots, s_{t-1})$ and $S_0^{(i)}$=S_{d_i}=$(s_{d_i}, \cdots, s_{d_i+t-1})$. First, we compute

$$\begin{aligned} S_{t-1} &= (s_{t-1}, \cdots, s_{2t-2}) = S_0 \cdot M^{t-1} \\ S_{d_i+t-1} &= (s_{d_i+t-1}, \cdots, s_{d_i+2t-1}) = S_{d_i} \cdot M^{t-1} \ . \end{aligned} \tag{5.31}$$

and the following equation holds:

$$\begin{bmatrix} s_{d_i} & s_{d_i+1} & \cdots & s_{d_i+t-1} \\ s_{d_i+1} & s_{d_i+2} & \cdots & s_{d_i+t} \\ \vdots & \vdots & & \vdots \\ s_{d_i+t-1} & s_{d_i+t} & \cdots & s_{d_i+2t-2} \end{bmatrix} = \begin{bmatrix} s_0 & s_1 & \cdots & s_{t-1} \\ s_1 & s_2 & \cdots & s_t \\ \vdots & \vdots & & \vdots \\ s_{t-1} & s_t & \cdots & s_{2t-2} \end{bmatrix} \cdot M^{d_i} \tag{5.32}$$

If we denote the above equation by $A_{d_i}=A_0 \cdot M^{d_i}$, then A_{d_i} and A_0 are both nonsingular, more generally, Proposition 5.5 holds.

Proposition 5.5. *Suppose that $\mathcal{L}^{(1)}$ is a t-bit LFRS with a period of 2^t-1 and update matrix M, denote its j-th state by $S_j=(s_j, \cdots, s_{j+t-1})$ with initial state S_0, then for any $j \geq 0$, matrix*

$$A_j = \begin{bmatrix} s_j & s_{j+1} & \cdots & s_{j+t-1} \\ s_{j+1} & s_{j+2} & \cdots & s_{j+t} \\ \vdots & \vdots & & \vdots \\ s_{j+t-1} & s_{j+t} & \cdots & s_{j+2t-2} \end{bmatrix} \tag{5.33}$$

is nonsingular.

Proof. As $\mathcal{L}^{(1)}$ has a period of 2^t-1, we have $S_j=S_{j+2^t-1}$ and $s_j=s_{j+2^t-1}$. Hence, it follows that $A_j=A_{j+2^t-1}$. Since every non-zero state appears exactly once per

period, there exists a d such that $0 \leq d \leq 2^t - 2$ and

$$S_d = (s_d, \cdots, s_{d+t-2}, s_{d+t-1}) = (0, \cdots, 0, 1) \ . \tag{5.34}$$

A_d is nonsingular since it can be written as

$$A_d = \begin{bmatrix} 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & & 1 & s_{d+t} \\ \vdots & & \iddots & \iddots & \vdots \\ 0 & 1 & s_{d+t} & \cdots & s_{d+2t-3} \\ 1 & s_{d+t} & s_{d+t+1} & \cdots & s_{d+2t-2} \end{bmatrix} . \tag{5.35}$$

Due to the equation

$$A_j = A_{j+2^t-1} = A_d \cdot M^{j+2^t-1-d} \tag{5.36}$$

where both A_d and M are nonsingular, it follows that A_j is also nonsingular. □

Therefore, we can obtain M^{d_i} efficiently by computing $A_0^{-1} \cdot A_{d_i}$, which takes time $O(t^3)$.

5.2.6 Remarks

Protocol 5.3 defeats probing attacks by augmenting each gate with a t-bit memory and $2t$ XOR taps, and thus the size of the resulting stateful circuit is $3t$ times as large as that of the original one. More importantly, due to the structure of the

s-gate (see Figure 5.2), the stateful circuit has a depth that is 3 times as large as that of the original one. We compare the results of Ishai et al. [70, Table 1] with ours in Table 5.1. Since their results are only intended for defending probing attacks, we do not include the overhead incurred by CTO, which is used to hide the information of circuit topology. We also provide a more detailed performance

Table 5.1: A comparison between the results of Ishai et al. and ours, where both approaches transform circuit C of size s and depth d to circuit C' of size s' and depth d', Ishai et al. assume that the adversary can simultaneously probe up to p wires chosen at his will, and they use notation $\hat{O}$ to hide large constants, poly-log factors and polynomials in the security parameter whereas we assume a security parameter of t.

	Privacy type	d'/d	s'/s
Their results in randomized model	perfect	$O(\log p)$	$O(p^2)$
Their results in de-randomized model	computational	$O(\log p)$	$O(p^2)+\hat{O}(p^3)$
Our results	computational	3	$3t$

evaluation in Table 5.2, where security parameter t and the number of probed wires p are assigned with practical values (t=64 and 128, p=10, 100 and 1000). We note that Ishai et al. use $O(p^2)$ to hide small constants and use $\hat{O}(p^3)$ to hide large constants, poly-log factors and polynomials in t, but have not specified the value or range of the small constant, large constant and polynomial. Hence, in Table 5.2 we assume for simplicity $O(\log p)=\log_2 p$, $O(p^2)=p^2$ and $\hat{O}(p^3)=t^2p^3$, which would be an underestimate but the resulting overhead factors are still larger than our results.

In summary, their results are stronger in that they can defeat unbounded adver-

Table 5.2: A perform comparison between Ishai et al's overhead factors and ours.

		p=10	p=100	p=1000
Overheads of Ishai et al's approach in the randomized model	t=64	d'/d=3.32 s'/s=10^2	d'/d=6.64 s'/s=10^4	d'/d=9.97 s'/s=s'/s=10^6
	t=128	d'/d=3.32 s'/s=10^2	d'/d=6.64 s'/s=10^4	d'/d=9.97 s'/s=s'/s=10^6
Overheads of Ishai et al's approach in the derandomized model	t=64	d'/d=3.32 s'/s=4.10×10^6	d'/d=6.64 s'/s=4.10×10^9	d'/d=9.97 s'/s=4.10×10^{12}
	t=128	d'/d=3.32 s'/s=1.64×10^7	d'/d=6.64 s'/s=1.64×10^{10}	d'/d=9.97 s'/s=1.64×10^{13}
Overheads of our approach	t=64	d'/d=3 s'/s=192	d'/d=3 s'/s=192	d'/d=3 s'/s=192
	t=128	d'/d=3 s'/s=384	d'/d=3 s'/s=384	d'/d=3 s'/s=384

saries in the randomized model, but the overhead is quite expensive as it depends on the number of wires being probed. A major advantage of our results is that the computation time (i.e. circuit depth) of C' is increased only by a constant factor. In addition, the circuit size is increased only linearly (in the security parameter) and we do not put any restriction on the number of wires being probed. The drawback of our approach is that s-gates should be carefully implemented such that the private memories are securely protected. Furthermore, all the s-gates should be uniformly clocked and any unsynchronized gate would lead to a false output.

5.3 Summary of the Chapter

In this chapter, we put forward two distinct approaches to achieve reusable SCC protocols for $\mathcal{P}(\{x, C, IR, C(x)\})$. One is by means of encryption schemes, namely,

results are all encrypted and computation is done in ciphertexts. The overhead of this approach is bandwidth expansion: each bit is encrypted into a t-bit ciphertext with t large enough to defeat the potential adversary (Bob). Another weakness is its dependence on efficient algebraic PH the existence of which remains open. The other approach is by tamper-proof hardware. We can always protect a circuit by simply putting it into a safe box, so the challenge is how to secure a circuit with as less amount of tamper-proof hardware as possible. The solution is to protect circuit topology using CTO and to make gate functions stateful such that inputs, intermediate results and outputs are well encrypted. In this case, the overhead is that each gate needs a t-bit tamper-proof memory, where t is the security parameter of the LFSR-based PRG that defeats Bob.

Chapter 6

Conclusions and Future Work

6.1 Summary of Contribution

We discuss the SCC protocols by which Alice makes Bob compute circuits without disclosing to him anything substantial. We show that existing techniques for symmetric two-party computation are not suitable as the encrypted circuits are one-time use and are unable to hide the circuit topology. We propose the notion of CTO, which can aid any SCC protocol in hiding their skeleton and can defeat circuit topology observers in practice. We offer two different CTO methods that are respectively efficient in their overheads of depth and size. We also show how to improve the symmetric two-party computation protocols using CTO. We represent the gate functions with ANF parameter tables, which can be repeatedly and securely computed in their encrypted formats, and hence reduce the problem of computing with encrypted function to computing with encrypted data. Then, based on the results of CTO and ANF parameter tables, we construct a reusable SCC protocol with algebraic PH, where we show that an algebraic PH with plaintext space {0,1}

is implied by any algebraic PH. We extend the SCC protocol to defeat malicious Bob despite that no PH is non-malleably secure. Finally, we provide another reusable SCC protocol that corresponds to a secure implementation of circuits against probing attacks, where we study the properties of the LFSR sequence and reduce the overhead to as low as possible.

6.2 Future Work

Following the results described in this thesis, we could proceed to a similar problem: how can Alice make Bob execute a software program p directly (without converting p into a circuit) such that Bob learns nothing substantial? The significance of this problem is that software programs are sometimes more efficient in that their conditional branchings can prevent unnecessary loops. We could augment the circuit code with some conditional branching instructions and the resulting code is intermediate between Boolean circuits and programs. Then, we would mainly focus on how to execute an encrypted branching instruction and there are some related results [76, 77] under the models of Turing Machine and Random-Access Machine. Once we obtain protocols for securely computing the intermediate code, they can be easily extended to those that securely compute programs.

Author's Publications

Journal Paper

1. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "How to Privately Utilize Untrustworthy Computing Power," *Journal of Information Assurance and Security*, Vol. 1, Issue 2, pp. 149 - 158, Dynamic Publishers.

2. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "A Study on the Security of Privacy Homomorphism," *International Journal of Network Security*, Vol. 6, No. 1, pp. 33 - 39, January 2008 (to appear).

Conference Paper

1. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "Securely Utilizing External Computing Power.," In Proceedings of IEEE Conference on Information Technology Coding and Computing (ITCC 2005), pp. 762 - 767, IEEE Press.

2. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "On the Possibility of Privacy-Preserving Biometric Identification," In Pre-proceedings of Privacy Enhancing Technologies Workshop (PET'05).

3. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "An Improved Secure Two-Party Computation Protocol," In Proceedings of the SKLOIS Conference on Information Security and Cryptology (CISC 2005), LNCS 3822. pp. 221-232.

4. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "Program Obfuscation via Oblivious Circuit Evaluation," In Proceedings of the SKLOIS Conference on Information Security and Cryptology (CISC 2005), pp. 305 - 314, Higher Education Press.

5. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "A Study on the Security of Privacy Homomorphism (extended abstract)," In Proceedings of international conference on information technology : new generations (ITNG 2006). pp.470 - 475. IEEE Press.

6. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "Hiding Circuit Topology from Unbounded Reverse Engineers," In Proceedings of 11th Australasian Conference on Information Security and Privacy (ACISP 2006), LNCS 4058, pp. 171 - 182.

7. Yu Yu and Jussipekka Leiwo and Benjamin Premkumar, "Private Stateful

Circuits Secure against Probing Attacks," In Proceedings of ACM Symposium on Information, Computer and Communications Security (ASIACCS 2007), pp. 63 - 69, ACM Press.

Appendix A

Design of Compilers for Producing Circuits from C-style Code

A.1 Data Types and Data Declaration

The compiler supports three data types: Boolean, signed integer and unsigned integer. Boolean is a 1-bit-long data type mostly used in selection statements. Signed integer and unsigned integer are data types that can be declared to be of arbitrary length. Unsigned integers are internally represented as base 2. Thus, the value of an unsigned integer A_n with the representation $a_n \cdots a_1$ is simply its base 2 value, namely, $\sum_{i=1}^{n} a_i \times 2^{i-1}$. Signed integers are represented using two's complement, that is, the value of a signed integer B_n=$b_n \cdots b_1$ is $-2^{n-1} \times b_n + \sum_{i=1}^{n-1} b_i \times 2^{i-1}$. We can also

declare constants without specifying their data types. For example,

unsigned int (30) A;

bool b;

sigined int (50) C;

const D=15;

is a list of data declarations where A, b, C and D are declared to be a 30-bit-long unsigned integer, a Boolean, a 50-bit-long signed integer and constant 15 respectively.

A.2 Language Syntax

The language acceptable by the compiler is defined using the Backus-Naur Form (BNF) that consists of a set of a production rules. A production rule states that the symbol (i.e. non-terminal) on the left-hand side of the ":" must be replaced by one of the alternatives on the right hand side, where the alternatives are separated by "|"s. For example,

```
symbol :
        alternative1
        | alternative2
        ...
```

With production rules, programmers can write code recognizable by the compiler. For the compiler, the recognition is done by applying the production rules in reverse (i.e. LL(1) grammar): it parses the input source code terminal (basic unit that makes sense to the compiler, e.g., "if", "for" and ";") by terminal, chooses the right rule by looking at only the current terminal on the input and takes the corresponding action. The grammar defined by the compiler can be summarized with the following production rules:

```
statements : statement
        |   statements  statement


statement : variable ':=' expression ';'
        |   RETURN expression ';'
        |   IF '(' bool ')' '{' statements '}'
        |   IF '(' bool ')' '{' statements '}'
            ELSE            '{' statements '}'
        |   FOR variable ':=' expression TO expression
            '{'          statements       '}'
expression : variable
        |   constant
        |   '(' expression ')'
        |   NOT expression
```

```
        |   expression logical_operator expression
        |   expression arithmetic_operator expression
bool    :   TRUE
        |   FALSE
        |   expression compare expression
        |   '(' bool ')'
        |   bool logical_operator bool
```

where the rules are simplified for the sake of demonstration. For example, operators (e.g. $+$, $-$, $\times$, $\div$) are considered to be of the same operator precedence and there is a reduce-shift conflict when parsing "if" and "if-else" statements, but all those problems can be solved by introducing more detailed rules.

A.3 Operations between Expressions

With the corresponding production rule, the compiler reduces an operation between two expressions to a new expression with some actions taken (i.e. generating gates for the new expression) and the new expression will be further referred to by other operations. We first show how the operations between unsigned integer expressions are implemented by the compiler and then reduce the operations between signed integer expressions to the unsigned analogue. We assume that A_m (resp., B_n) is an m-bit-long (resp., n-bit-long) integer expression with binary representation $a_m \cdots a_1$ (resp., $b_n \cdots b_1$). Of course, each label a_i/b_j corresponds to the result of an input-bit,

or a gate (intermediately generated), or even a constant bit.

Logical operators can be either unary (i.e. NOT) or binary (e.g. AND, OR and XOR) and the operands can be Boolean or integers. For uniformity, we treat Boolean as 1-bit-long integer and denote an arbitrary logic operator by "$*$", then the gate generating [1] algorithm can be described using the following pseudo-code:

```
program 1 Logic_Op (A_n, B_n/-, *)
    for i = 1 to n do
        if * = NOT
            c_i ← gate(a_i) = ā_i
        else
            c_i ← gate(a_i,b_i) = a_i*b_i
        end if
    end for
    result = c_n ··· c_1
end program
```

where $c \leftarrow$ gate(a,b) means generating a gate whose inputs are a and b and whose output are labeled by c. Note that labels are reusable, e.g., $a \leftarrow$ gate(a,b) indicates that a gate with inputs a and b is generated and label a is reassigned to the resulting gate. Addition/subtraction between unsigned integers are handled as follows:

[1] In the appendix, for the sake of brevity, all the gate-generating pseudo-code produces gates with fan-in bounded by 3. Since in the mainbody of the thesis we assume fan-in 2 for all gates, we can further eliminate NOT gates (fan-in=1) by absorbing them into their adjacent gates and replace each of those fan-in-3 gates by fan-in-2 gates.

program 2 Unsigned_Add/Sub (A_n, B_n)

$c_1 \leftarrow$ 0/1 ($c_1 \leftarrow$0 in case of ADD and $c_1 \leftarrow$1 otherwise)

for i = 1 to n do

$s_i \leftarrow$ gate(a_i,b_i,c_i) = $a_i \oplus b_i \oplus c_i$ / $a_i \oplus \bar{b}_i \oplus c_i$

$c_{i+1} \leftarrow$ gate(a_i,b_i,c_i) = carry($a_i \oplus b_i \oplus c_i$)/carry($a_i \oplus \bar{b}_i \oplus c_i$)

end for

sum = $c_{n+1} s_n \cdots s_1$ / difference = $\bar{c}_{n+1} s_n \cdots s_1$

end program

where carry$(a \oplus b \oplus c) \stackrel{def}{=} (a \wedge b) \vee (b \wedge c) \vee (a \wedge c)$ and $2n$ Boolean gates are generated. Multiplication can be implemented by invoking the above subroutine, namely,

program 3 Unsigned_Mul (A_m, B_n)

$s_{m+n} \cdots s_1 \leftarrow$ 0

for i = 1 to n do

for j = 1 to m do

$c_j \leftarrow$ gate(a_j,b_i) = $a_j b_i$

end for

$s_{i+m} \cdots s_i \leftarrow$ Add($s_{i+m-1} \cdots s_i$, $c_m \cdots c_1$)

end for

product = $s_{m+n} \cdots s_1$

end program

Thus, multiplication needs at most $3mn$ gates. The gates of rounding division

"DivR", truncating division "DivT" and modular arithmetic "Mod" can be generated with the following subroutine:

program 4 Unsigned_DivR/DivT/Mod(A_m, B_n)

$r_{m+n}\cdots r_1 \leftarrow \overbrace{0\cdots 0}^{n}\ a_m\cdots a_1$

for i = m to 1 do

$\bar{q}_i t_{n+1}\cdots t_1 \leftarrow$ Sub($r_{i+n}\cdots r_i$, $0b_n\cdots b_1$)

for j = 1 to n+1 do

$r_{i+j-1} \leftarrow$ gate(q_i,t_j,r_{i+j-1}) = $(q_i \wedge t_j)\vee(\bar{q}_i \wedge r_{i+j-1})$

end for

end for

if DivR

$\bar{q}_0 t_{n+1}\cdots t_1 \leftarrow$ Sub($r_n\cdots r_1 0$, $0b_n\cdots b_1$)

$q_{m+1}\cdots q_1 \leftarrow$ Add($q_m\cdots q_1$, $\overbrace{0\cdots 0}^{m-1}\ q_0$)

end if

quotient = $q_m\cdots q_1$ / remainder = $r_n\cdots r_1$

end program

We proceed to arithmetic operations on signed operands. The addition and subtraction between signed expressions are similar to their unsigned counterparts as we use two's-complement integer representation. Other operations can be implemented by invoking their signed analogue as follows:

program 5 2's_complement (A_m, b) /*If b=1, 2's-complement A_m;

otherwise, do nothing.*/

$c_1 \leftarrow b$

for i = 1 to m

$c_{i+1} \leftarrow$ gate(a_i,b,c_i) = $b \wedge$carry($\bar{a}_i \oplus c_i$)

$a_i \leftarrow$ gate(a_i,b,c_i) = $(b \wedge (\bar{a}_i \oplus c_i)) \vee (\bar{b} \wedge a_i)$

end for

2's-complement = $a_m \cdots a_1$

end program

program 6 Signed_Mul/DivR/DivT/Mod(A_m, B_n)

$s_a \leftarrow a_m$

$s_b \leftarrow b_n$

$s \leftarrow$ gate(s_a,s_b) = $s_a \oplus s_b$

$A_{m-1} \leftarrow$ 2's_complement($a_{m-1} \cdots a_1$,s_a)

$B_{n-1} \leftarrow$ 2's_complement($b_{n-1} \cdots a_1$,s_b)

$r_o \cdots r_1 \leftarrow$ Unsigned_Mul/DivR/DivT/Mod(A_{m-1}, B_{n-1})

$r_o \cdots r_1 \leftarrow$ 2's_complement($r_o \cdots r_1$, s)

result = $sr_o \cdots r_1$

end program

A.4 Comparisons between Expressions

The compiler will generate a Boolean indicating the result of comparison between expressions. There are six comparison operators as depicted in Table A.1, where (A_n==B_n, A_n!=B_n), (A_n>B_n, A_n<=B_n) and (A_n<B_n, A_n>=B_n) are complementary pairs and A_n>B_n can be viewed as B_n<A_n. Thus, it suffices to show the pseudo-code of A_n==B_n and A_n<B_n as follows:

Table A.1: Comparisons between A_n and B_n and their syntax.

Operation	Syntax	Result
Same	A_n==B_n	1 if A_n=B_n; 0 otherwise
Not same	A_n!=B_n	0 if A_n=B_n; 1 otherwise
Great than	A_n>B_n	1 if A_n>B_n; 0 otherwise
Greater than or equal to	A_n>=B_n	1 if A_n>B_n or A_n=B_n; 0 otherwise
Less than	A_n<B_n	1 if A_n<B_n; 0 otherwise
Less than or equal to	A_n<=B_n	1 if A_n<B_n or A_n=B_n; 0 otherwise

```
program 7 Same(A_n, B_n)
    c_1=1
    for i = 1 to n
        c_{i+1} ← gate(a_i,b_i,c_i) = c_i∧(a_i⊕b_i⊕1)
    end for
    result = c_{n+1}
end program

program 8 Less_Than(A_n, B_n)
```

```
    c_1=1
    for i = 1 to n-1
        c_{i+1} ← gate(a_i,b_i,c_i) = carry(a_i⊕b̄_i⊕c_i)
    end for
    if A_n and B_n are unsigned expressions
        c̄_{n+1} ← gate(a_n,b_n,c_n) = carry(a_n⊕b̄_n⊕c_n)
    else
        c_{n+1} ← gate(a_n,b_n,c_n) = (a_n∧c̄_n)∨(b̄_n∧c̄_n)∨(a_n∧b̄_n∧c_n)
    end if
    result = c_{n+1}
end program
```

A.5 Selection Statements and Value Assignments

The two forms of selection statements supported by our compiler are "IF(<bool>)-<statements>" and "IF(<bool>)-<statements>-ELSE-<statements>", where bool is the label of the Boolean expression in the parentheses. When the selection statements are executed on computers, the control is passed to the statement following "IF" if "bool" is nonzero, otherwise it is passed to the second statement (if ELSE is present). However, we cannot generate gates this way because Boolean circuits are acyclic. We should generate circuits in a data-independent way, which is not hard to achieve since most operations in selection statements can be done identically as

those outside "if" with an exception being the value assigning operation. This is because that the compiler generates new gates to store intermediate results and the values of variables are not updated until value assignments are performed. Based on the fact that the value of a variable is updated only if the "bool" is non-zero, we initialize (at the time of compiling) a stack with only one item "TRUE" (1) on its top. Stack operations are defined as follows:

```
//data[ ] indicates a stack with t items in it
program 9 Init_Stack()
    data[0] ← TRUE
    t ← 1
end program

program 10 push (bool)
    data[t] ← bool
    t ← t+1
end program

program 11 pop ()
    t ← t-1
end program
```

```
program 12 top()
    return data[t-1];
end program
```

where "bool" is a label of the expression passed to subroutine "push". When entering/leaving the scope of "if" or "else" whose expression in the parentheses of "if" is labeled by "b", the compiler does the following actions:

program 13 enter_if/enter_else (b)

    $a \leftarrow$ top()

    $c \leftarrow$ gate(a,b) = $b \wedge a$ / $\bar{b} \wedge a$

    push (c)

end program

program 14 leave_ifleave_else (b)

    pop()

end program

Thus, the pseudo-code of value assignment "A_n:=B_n", whether in selection statements or not, can be uniformly written as:

program 15 assign_value (A_n, B_n)

    $c \leftarrow$ top()

    for i = 1 to n

$a_i \leftarrow$ gate(a_i,b_i,c) = $(c \wedge b_i) \vee (\bar{c} \wedge a_i)$

end for

$A_n = a_n \cdots a_1$

end program

A.6 Iteration Statements

The compiler supports the iteration statement of

FOR <variable> := <expression1> TO <expression2> { <statements> }

where <statements> are repeatedly executed until "<variable>" exceeds the range of the two expressions. However, a Boolean circuit is a directed acyclic graph and thus Boolean gates are not re-excutable. To solve this problem, the compiler treats the iteration statement as a macro and unrolls it during the prepocessing stage to produce:

<variable> := <expression1>

<statements>

<variable> := <variable> + *increment*

<statements>

⋮

<variable> := <expression2>

<statements>

where <expression1> and <expression2> should be constant expressions and increment is 1 if <expression1> is less than <expression2> and is -1 otherwise. The unrolled code is functionally equivalent to the corresponding iteration statement and it can be easily converted to a circuit. Since our compiler requires that the number of iterations is determined at compile time, it does not support statements such as

WHILE (<expression>) { <statements> }

where the <statement> is executed repeatedly as long as <expression> remains true. Nevertheless, we can rephrase it to

FOR i := 1 TO max { IF(<expression>) {<statements> } }

where max is the maximal number of iterations that the "while-statement" cannot exceed. For a polynomial-time program, max is polynomially bounded and the resulting circuit is polynomial-size.

A.7 The "return" Statement

Normally a program or a function will halt after a "return" statement and will return a value (if any) to the the environment that called it, but in our case, "RETURN <expression>;" only indicates that the gates labeled by <expression> carry circuit output. Thus, the compiler will mark the gates that correspond to <expression> as output-gates (as opposite to intermediate-gates) and proceed to parsing the code.

Bibliograpy

[1] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan, "Private information retrieval," in *FOCS*, 1995, pp. 41–50.

[2] Michael J. Freedman, Yuval Ishai, Benny Pinkas, and Omer Reingold, "Keyword search and oblivious pseudorandom functions.," in *Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005)*, 2005, pp. 303–324.

[3] Omer Barkol and Yuval Ishai, "Secure computation of constant-depth circuits with applications to database search problems.," in *CRYPTO*, 2005, pp. 395–411.

[4] Josh D. Cohen and Michael J. Fischer, "A robust and verifiable cryptographically secure election scheme (extended abstract)," in *FOCS*, 1985, pp. 372–382.

[5] Jonathan Katz, Steven Myers, and Rafail Ostrovsky, "Cryptographic counters and applications to electronic voting.," in *EUROCRYPT*, 2001, pp. 78–92.

[6] David Wagner, "Cryptographic protocols for electronic voting.," in *CRYPTO*, 2006, p. 393.

[7] Oded Goldreich, *Foundations of Cryptography: Basic Applications*, vol. 2, Cambridge University Press, May 2004.

[8] Ernest F. Brickell and Yacov Yacobi, "On privacy homomorphisms (extended abstract).," in *EUROCRYPT*, 1987, pp. 117–125.

[9] Ronald Lorin Rivest, Adi Shamir, and Leonard Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 26, no. 1, pp. 96–99, 1983.

[10] Ronald L. Rivest, Leonard M. Adleman, and Michael L. Dertouzos, "On data banks and privacy homomorphisms," *Foundations of Secure Computation*, pp. 169–180, 1978.

[11] Wenbo Mao, *Modern Cryptography: Theory and Practice*, Prentice Hall PTR, 2004.

[12] Shafi Goldwasser and Silvio Micali, "Probabilistic encryption.," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[13] Andrew Chi-Chih Yao, "Theory and applications of trapdoor functions," in *FOCS*, 1982, pp. 80–91.

[14] Manuel Blum and Shafi Goldwasser, "An efficient probabilistic public-key encryption scheme which hides all partial information.," in *CRYPTO*, 1984, pp. 289–302.

[15] Tatsuaki Okamoto and Shigenori Uchiyama, "A new public-key cryptosystem as secure as factoring.," in *EUROCRYPT*, 1998, pp. 308–318.

[16] Pascal Paillier, "Public-key cryptosystems based on composite degree residuosity classes.," in *EUROCRYPT*, 1999, pp. 223–238.

[17] Josep Domingo-Ferrer, "A new privacy homomorphism and applications.," *Inf. Process. Lett.*, vol. 60, no. 5, pp. 277–282, 1996.

[18] Josep Domingo-Ferrer, "A provably secure additive and multiplicative privacy homomorphism.," in *ISC*, 2002, pp. 471–483.

[19] Jung Hee Cheon and Hyun Soo Nam, "A cryptanalysis of the original domingo-ferrer's algebraic privacy homomophism," Cryptology ePrint Archive, Report 2003/221, 2003, `http://eprint.iacr.org/`.

[20] Feng Bao, "Cryptanalysis of a provable secure additive and multiplicative privacy homomorphism," in *Proceedings of the International Workshop on Coding and Cryptography (WCC 2003)*, 2003, pp. 43–50.

[21] David Wagner, "Cryptanalysis of an algebraic privacy homomorphism.," in *ISC*, 2003, pp. 234–239.

[22] Tomas Sander, Adam Young, and Moti Yung, "Non-interactive cryptocomputing for NC^1.," in *FOCS*, 1999, pp. 554–567.

[23] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim, "Evaluating 2-dnf formulas on ciphertexts.," in *Proceedings of the 2nd Theory of Cryptography Conference (TCC 2005)*, 2005, pp. 325–341.

[24] Joan Feigenbaum and Michael Merritt, "Open questions, talk abstracts and summary of discussions.," *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, , no. 2, pp. 1–45, 1991.

[25] Niv Ahituv, Yeheskel Lapid, and Seev Neumann, "Processing encrypted data.," *Commun. ACM*, vol. 30, no. 9, pp. 777–780, 1987.

[26] Dan Boneh and Richard J. Lipton, "Algorithms for black-box fields and their application to cryptography (extended abstract).," in *CRYPTO*, 1996, pp. 283–297.

[27] Gilbert Sandford Vernam, "Cipher printing telegraph systems for secret wire and radio telegraphic communications," *Journal of American Institution of Electronic Engineering*, vol. 55, 1926.

[28] Stephen K. Park and Keith W. Miller, "Random number generators: Good ones are hard to find.," *Commun. ACM*, vol. 31, no. 10, pp. 1192–1201, 1988.

[29] Lenore Blum, Manuel Blum, and Mike Shub, "A simple unpredictable pseudo-random number generator.," *SIAM J. Comput.*, vol. 15, no. 2, pp. 364–383, 1986.

[30] Niels Ferguson and Bruce Schneier, *Practical cryptography*, Wiley, 2003.

[31] Solomon W. Golomb and Soloman Golomb, *Shift Register Sequences*, Aegean Park Press, Laguna Hills, CA, USA, 1981.

[32] Gustavus J. Simmons, *Contemporary Cryptology: The Science of Information Integrity*, IEEE Press, Piscataway, NJ, USA, 1994.

[33] Don Coppersmith, Hugo Krawczyk, and Yishay Mansour, "The shrinking generator.," in *CRYPTO*, 1993, pp. 22–39.

[34] Willi Meier and Othmar Staffelbach, "The self-shrinking generator.," in *EUROCRYPT*, 1994, pp. 205–214.

[35] Jovan Dj. Golic, "Correlation analysis of the shrinking generator.," in *CRYPTO*, 2001, pp. 440–457.

[36] Patrik Ekdahl, Willi Meier, and Thomas Johansson, "Predicting the shrinking generator with fixed connections.," in *EUROCRYPT*, 2003, pp. 330–344.

[37] RSA Security Lab, "What is a linear feedback shift register?," `http://www.rsasecurity.com/rsalabs/node.asp?id=2175`.

[38] Oded Goldreich, *Foundations of Cryptography: Basic Tools*, vol. 1, Cambridge University Press, January 2000.

[39] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang, "On the (im)possibility of obfuscating programs.," in *CRYPTO*, 2001, pp. 1–18.

[40] Ben Lynn, Manoj Prabhakaran, and Amit Sahai, "Positive results and techniques for obfuscation.," in *EUROCRYPT*, 2004, pp. 20–39.

[41] Martín Abadi, Joan Feigenbaum, and Joe Kilian, "On hiding information from an oracle (extended abstract)," in *STOC*, 1987, pp. 195–203.

[42] Tomas Sander and Christian F. Tschudin, "Protecting mobile agents against malicious hosts.," in *Mobile Agents and Security*, 1998, pp. 44–60.

[43] Tomas Sander and Christian F. Tschudin, "Towards mobile cryptography.," in *IEEE Symposium on Security and Privacy*, 1998, pp. 2–15.

[44] Robert J. McEliece, "A public-key cryptosystem based on algebraic coding theory," Deep Space Network Progress Report 42-44, Jet Propulsion Laboratory, California Institute of Technology, 1978.

[45] Sergio Loureiro, *Mobile Code Protection*, Ph.D. thesis, Ecole Nationale Superieure des Telecommunication, 2001.

[46] Andrew Chi-Chih Yao, "How to generate and exchange secrets (extended abstract)," in *FOCS*, 1986, pp. 162–167.

[47] Michael Oser Rabin, "How to exchange secrets by oblivious transfer," Tech. Rep. TR-81, Harvard Aiken Computation Laboratory, 1981.

[48] Shimon Even, Oded Goldreich, and Abraham Lempel, "A randomized protocol for signing contracts," *Commun. ACM*, vol. 28, no. 6, 1985.

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

[50] Phillip Rogaway, *The round complexity of secure protocols*, Ph.D. thesis, Laboratory for Computer Science, MIT, 1991.

[51] Magnus Pettersson, "The Match On Card Technology:White paper," Precise Biometrics AB, `http://www.precisebiometrics.com`.

[52] Timo Grassmann and Andreas Karl, "System-on-Card:TechnoFile3," Infineon Technologies AG.

[53] "ISO/IEC 7816-2 Information Technology - Identification Cards - Integrated Circuit(s) Cards with Contacts," 1999, Part 2: Dimensions and location of the contacts.

[54] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman, *The Design and Analysis of Computer Algorithms.*, Addison-Wesley, 1974.

[55] M. J. Fischer, "Lectures on network complexity," Universität Frankfurt/Main, 1974.

[56] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella, "Fairplay - secure two-party computation system (awarded best student paper).," in *USENIX Security Symposium*, 2004, pp. 287–302.

[57] Donald Beaver, "Correlated pseudorandomness and the complexity of private computations.," in *STOC*, 1996, pp. 479–488.

[58] Moni Naor, Benny Pinkas, and Reuban Sumner, "Privacy preserving auctions and mechanism design.," in *ACM Conference on Electronic Commerce*, 1999, pp. 129–139.

[59] Yehuda Lindell and Benny Pinkas, "A proof of Yao's protocol for secure two-party computation," *Electronic Colloquium on Computational Complexity (ECCC)*, , no. 063, 2004.

[60] Stephen R. Tate and Ke Xu, "On garbled circuits and constant round secure function evaluation," Tech. Rep. 2003-02, CoPS Lab, 2003.

[61] Leslie G. Valiant, "Universal circuits (preliminary report)," in *STOC*, 1976, pp. 196–203.

[62] Andrew Chi-Chih Yao, "Protocols for secure computations (extended abstract)," in *FOCS*, 1982, pp. 160–164.

[63] Miklós Ajtai, János Komlós, and Endre Szemerédi, "An O(n log n) sorting network," in *STOC*, 1983, pp. 1–9.

[64] Kenneth E. Batcher, "Sorting networks and their applications.," in *Proc. AFIPS Spring Joint Computing Conference*, 1968, pp. 307–314.

[65] Danny Dolev, Cynthia Dwork, and Moni Naor, "Non-malleable cryptography," in *STOC*, 1991, pp. 542–552.

[66] Martín Abadi and Joan Feigenbaum, "Secure circuit evaluation: A protocol based on hiding information from an oracle," *Journal of Cryptology*, vol. 2, no. 1, pp. 1–12, 1990.

[67] Paul C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems.," in *CRYPTO*, 1996, pp. 104–113.

[68] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun, "Differential power analysis.," in *CRYPTO*, 1999, pp. 388–397.

[69] R. Anderson and M. Kuhn, "Tamper resistance - a cautionary note.," in *Proc. 2nd Usenix Workshop on Electronic Commerce*, 1996, pp. 1–11.

[70] Yuval Ishai, Amit Sahai, and David Wagner, "Private circuits: Securing hardware against probing attacks.," in *CRYPTO*, 2003, pp. 463–481.

[71] John Daemen and Vincent Rijmen, "Resistance against implementation attacks: A comparative study of the AES proposals.," in *Proc. 2nd Advanced*

Encryption Standard (AES) Candidate Conference,, 1999, `http://csrc.nist.gov/encryption/aes/round1/conf2/aes2conf.htm`.

[72] Thomas S. Messerges, "Securing the AES finalists against power analysis attacks," in *FSE*, 2001, pp. 150–164.

[73] Louis Goubin and Jacques Patarin, "DES and differential power analysis (the "duplication" method)," in *CHES*, 1999, pp. 158–172.

[74] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi, "Towards sound approaches to counteract power-analysis attacks.," in *CRYPTO*, 1999, pp. 398–412.

[75] Jean-Sébastien Coron and Louis Goubin, "On boolean and arithmetic masking against differential power analysis.," in *CHES*, 2000, pp. 231–237.

[76] Nicholas Pippenger and Michael J. Fischer, "Relations among complexity measures," *Journal of the ACM*, vol. 26, no. 2, pp. 361–381, 1979.

[77] Oded Goldreich and Rafail Ostrovsky, "Software protection and simulation on oblivious RAMs," *Journal of the ACM*, vol. 43, no. 3, pp. 431–473, 1996.