UC San Diego
UC San Diego Electronic Theses and Dissertations

Title
Computational and Symbolic Models for Secure Computation

Permalink
https://escholarship.org/uc/item/4tq4r387

Author
Li, Bai Yu

Publication Date
2020

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

UNIVERSITY OF CALIFORNIA SAN DIEGO

Computational and Symbolic Models for Secure Computation

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy

in

Computer Science

by

Bai Yu Li

Committee in charge:

Professor Daniele Micciancio, Chair
Professor Mihir Bellare
Professor Russell Impagliazzo
Professor Farinaz Koushanfar
Professor Deian Stefan

2020

Copyright

Bai Yu Li, 2020

All rights reserved.

The Dissertation of Bai Yu Li is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

Chair

University of California San Diego

2020

DEDICATION

To my wife and my parents, I could not have done it without you.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I am very fortunate to have Daniele Micciancio as my advisor, who continuously encourages and supports me. It was Daniele who showed me the fascinating world of cryptography, guided me to work on many very interesting problems, and shared countless great and insightful ideas with me through the years. I had many eureka moments during our discussions, inspired by Daniele's concise and sharp thoughts on the subject, and I have learned a great deal on how to do research, especially on how to tackle unfamiliar problems and how to communicate and present my ideas.

I thank all of my thesis committee members, Mihir Bellare, Russell Impagliazzo, Farinaz Koushanfar, and Deian Stefan.

I am grateful to have collaborated with many outstanding researchers, Rosario Cammarota, Nicholas Genise, Craig Gentry, Shai Halevi, Siam Hussain, Miran Kim, Farinaz Koushanfar, Mahmoud Maarouf, Daniele Micciancio, and Yongsoo Song. I have gained invaluable experiences from working with my coauthors.

I also thank Wei Dai, Shai Halevi, Kim Laine, Yuriy Polyakov, Victor Shoup, and Yongsoo Song, for our recent discussions on homomorphic approximate encryption schemes.

I thank all my peer students in the UCSD cryptography and theory groups, including (but not limited to) Anant Dhayal, Nicholas Genise, Max Hopkins, Kawa Hosseini, Sankeerth Rao, Mark Schultz, Olivia Simpson, Jessica Sorrell, and Michael Walter. We had a lot of fun during our shared journey at UCSD cryptography and theory groups.

I thank OptumSoft for their financial support of my Ph.D study. Moreover, I thank the Department of Computer Science and Engineering and Graduate Studies for their kind understanding of my special situation.

Mostly importantly, I would like to acknowledge my family and friends. Especially I thank my wife Ziyi Chen, and my parents. I would not be able to chase my dream without your cherished love and encouragement.

Chapter 3 contains a reprint of materials as it appears in our submission to Eurocrypt 2021, Baiyu Li and Daniele Micciancio, *On the Security of Homomorphic Encryption on Approximate Numbers*. The dissertation author was a primary investigator and author of this paper.

Chapter 4 contains a reprint of materials as it appears in Proceedings of the 31st Computer Security Foundations Symposium 2018, Baiyu Li and Daniele Micciancio, *Symbolic security of garbled circuits*. The dissertation author was a primary investigator and author of this paper.

Chapter 5 contains a reprint of materials as it appears in Proceedings of 21st Public-Key Cryptography 2018, Baiyu Li and Daniele Micciancio, *Equational Security Proofs of Oblivious Transfer Protocols*. The dissertation author was a primary investigator and author of this paper.

VITA

2010	Bachelor of Mathematics, University of Waterloo, Waterloo, Canada
2012	Master of Mathematics, University of Waterloo, Waterloo, Canada
2020	Doctor of Philosophy, University of California San Diego

PUBLICATIONS

- Baiyu Li, Daniele Micciancio. *On the Security of Homomorphic Encryption on Approximate Numbers*. In submission to EUROCRYPT 2021.
- Siam Hussain, Baiyu Li, Mahmoud Maarouf, Farinaz Koushanfar, Rosario Cammarota. *TinyGarble2: Smart, Efficient, and Scalable Yao's Garble Circuit.* In submission to DAC 2021.
- Nicholas Genise, Baiyu Li. *Gadget-Based iNTRU Lattice Trapdoors.* Indocrypt 2020.
- Miran Kim, Yongsoo Song, Baiyu Li, Daniele Micciancio. *Semi-parallel Logistic Regression for GWAS on Encrypted Data.* BMC Med Genomics 13, 99 (2020).
- Nicholas Genise, Craig Gentry, Shai Halevi, Baiyu Li, Daniele Micciancio. *Homomorphic Encryption for Finite Automata.* ASIACRYPT 2019.
- Baiyu Li, Daniele Micciancio. *Symbolic Security of Garbled Circuits.* CSF 2018.
- Baiyu Li, Daniele Micciancio. *Equational Security Proofs of Oblivious Transfer Protocols.* PKC 2018.
- Baiyu Li, Daniele Micciancio. *Compactness vs Collusion Resistance in Functional Encryption.* TCC 2016-B.

ABSTRACT OF THE DISSERTATION

Computational and Symbolic Models for Secure Computation

by

Bai Yu Li

Doctor of Philosophy in Computer Science

University of California San Diego, 2020

Professor Daniele Micciancio, Chair

Formalization and modeling are important topics in cryptography. They are crucial for precisely defining cryptographic problems and for rigorously deriving security analysis. In this dissertation, we study some computational and symbolic models in secure computation, namely:

- Passive computational security model for homomorphic approximate encryptions. In this part, we study homomorphic encryption schemes, as they can be used to construct secure computation protocols. We identify the shortcomings of the classical passive security model *indistinguishability under chosen plaintext attacks* when applied to approximate encryption, and we propose and study new security definitions in a model that captures the special properties of approximate encryption. As a concrete example showing the gap between these two models, we present a passive key recovery attack against the popular CKKS homomorphic approximate encryption scheme (Cheon et. al., Asiacrypt 2017) that is IND-CPA secure. We implement our attack against several open-source libraries, and we briefly discuss countermeasures to our attack.

- Symbolic security model for garbled circuits. In this part, we present the first computationally sound symbolic security analysis of Yao's garbled circuits. We extend the symbolic framework of Micciancio (Eurocrypt 2010), and we prove Yao's garbled circuit scheme is (simulation-based) secure in a purely syntactic fashion, which implies concrete computational security statements by the computational soundness theorem of the symbolic model. We also implement our symbolic analysis in Haskell that can prove the security of garbling large circuits within several seconds.
- Equational security model of oblivious transfer (OT) protocols. In this part, we analyze some concrete OT protocols in the equational security framework (Micciancio and Tessaro, ITCS 2013). Our analysis uncovers subtle timing issues in these protocols that are partly caused by shortcomings in the typical OT definitions. We show that the OT length extension protocol can be proved secure using a revised OT definition that adds an acknowledgment bit to the sender output. Our analysis on the "simplest" OT protocol of Chou and Orlandi (Latincrypt 2015) shows that this protocol cannot be proven secure in the equational framework, even using the revised OT definition.

Chapter 1

Introduction

Modern cryptography is a study of robust and secure systems that can provide crucial functionalities and yet can withstand unintended uses or malicious attacks. Due to its critical nature, cryptography must be built on a solid foundation. To do this, we must first define cryptographic problems properly, in accurate mathematical languages. Such mathematical modeling is perhaps the central theme in cryptography, as Avi Wigderson summarized in his recent talk [69] that, "Computer Science is a modeling science, and Cryptography may be the most prolific." As cryptographic concepts can sometimes be counterintuitive, and cryptographic constructions can sometimes be quite complex and tricky to get done right, good cryptographic modeling is essential and can greatly benefit building good solutions to the cryptographic problems that we study.

Generally, we want to achieve two goals with secure computation: correctness and security. Correctness is straightforward to define such that the output of a secure computation protocol should be the same as if the computation is carried out in clear. Security is relatively harder to define: we must use a model to formally capture the capabilities of all feasible attackers. Such model can be categorized into two flavors: either considering attackers as arbitrary algorithms of a certain computational complexity class, or considering attackers as abstract machines that can only perform certain actions on cryptographic constructs. The formal is the *computational model*, and the latter is usually called the *symbolic model*. They come with their own advantages and limitations. The computational model is rooted from computational complexity theories, and it can usually provide security guarantees in the form that, if security is broken, then a hard computational problem can be efficiently solved; so security is tied to inefficiency of solving the hard problems. The symbolic model is built on a certain logic system that abstract cryptographic primitives as perfect objects or blackboxes, and it provides a way to formally derive a rigorous argument that all such actions (including any combinations of them) are prevented; so the security is tied to the impossbility of mounting conceivable attacks. Comparing these two models, the computational model provides stronger security guarantees but security analysis in this model is usually complicated such that informal arguments are commonly used, and on the other hand, the symbolic model uses abstractions to provide higher level analysis such that fully specified security arguments can be formally given. We remark that both models have their own advantages, and one should consider the strategy of using them to maximize security guarantees without compromising clarity and understandability.

We study computational and symbolic models for some primitives that are the core components of secure computation protocols. One of such primitives is fully homomorphic encryption (FHE), which enables arbitrary computation on encrypted data (without decrypting them). Since the seminal work of Gentry [30], FHE has become a prominent and fruitful research topic: it has been shown that the security of some FHE schemes can be based on hardness assumptions about standard lattice problems [10, 12–14], and the efficiency and practicality of FHE schemes have been greatly improved [11,31,67]. One of the efforts to making FHE practical for more real world usages is the CKKS homomorphic encryption scheme on approximate numbers [19–22, 46], which exploits the fact that data and their computation in many real world applications are approximate, so the encryption noises that are inherent to lattice-based encryptions can be treated as approximation errors in computation. In fact, it is observed in [22] that such approximation errors are typically comparable to the floating-point arithmetic errors that can already be tolerated by applications. Not eliminating encryption noises substantially boosts the efficiency of the CKKS scheme, making it practical for many secure machine learning applications [3, 8, 9, 23, 26, 27, 36, 61]. Yet, the CKKS scheme is very different from normal FHE schemes, namely, it does not guarantee *correctness* of the decrypted data. Due to this fact, the widely used passive security model for encryption schemes, *indistinguishability under chosen plaintext attack* (IND-CPA), does not capture the special property of approximate encryption. Although the CKKS scheme has been proven to be secure in the IND-CPA model, it is still subject to a passive key recovery attack. Such gap demonstrates the importance of precisely defining security using a model that captures all the properties of a cryptographic system.

Good modeling not only means precise definitions, but also requires suitable formalization frameworks. Modern cryptographic systems heavily rely on hard computational problems and randomness, and their analysis typically uses various reductions and probabilistic arguments to show that the claimed security property holds due to the lack of efficient solutions to the hard problems. For more and more cryptographic constructions to accomplish sophisticated goals, their security analysis are long and complicated such that they could easily exceed the usual length limit of 30 pages in a typical conference. Definitions themselves are sometimes also quite complex to address various aspects of a system, especially with cryptographic protocols that deal with interactive computation. In order to save space and convey the main results, informal specifications and arguments are often used, and sometimes certain steps in the analysis are intuitively stated without formal justification. The lack of formality has drawn considerable attention in recent years, calling for applying formal methods to identify potentially inaccurate security statements in a mathematical sense. To make formal analysis more economical, it is desirable to design concise yet expressive formalization frameworks. Such frameworks should have a sufficient level of abstraction to provide rigorous and easy to understand mathematical formulations and at the same time to reduce the complexity of security analysis. Moreover, they can be the basis of computer aided security analysis.

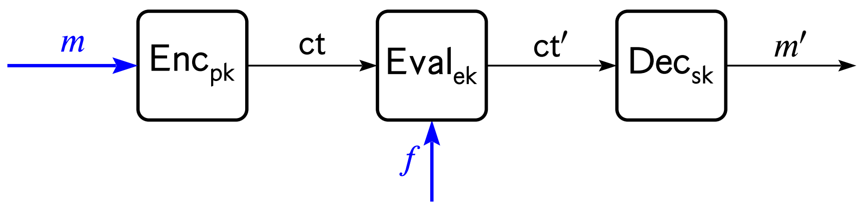

m
$\mathsf{Enc}_{\mathsf{pk}}$
ct
$\mathsf{Eval}_{\mathsf{ek}}$
ct'
$\mathsf{Dec}_{\mathsf{sk}}$
m'
f

Figure 1.1. A passive attacker against a homomorphic encryption scheme may choose/know the plaintext m and the homomorphic computation f (thick blue interfaces), and it can read from black interfaces to learn the ciphertexts $\mathsf{ct}, \mathsf{ct}'$ and the decryption results m'. It is not allowed to temper or inject ciphertexts.

1.1 Our Results

In this thesis, we study several computational and symbolic models for existing primitives that are major building blocks in secure computation, in order to improve our understandings about them.

Passive security model for homomorphic approximate encryption schemes. As we mentioned earlier, the CKKS homomorphic approximate encryption scheme does not satisfy the correctness requirement for encryption schemes, so its security in the passive attack model should be carefully considered. Recall that, a passive attacker can influence honest users on their choices of input to a cryptographic system, she can observe the output of the system given to the users, but she cannot actively tamper or use the system in any unprescribed ways. Figure 1.1 contains an illustration of the capabilities of such an attacker. For proper encryption scheme that meet correctness requirement, $\mathsf{IND\text{-}CPA}$ security is the standard model for passive attacks, in which an adversary is given either a sequence of ciphertexts or the public key, and she is challenged on a ciphertext to distinguish whether it encrypts the first or the second message in a pair chosen by her. Decrypted messages are not considered in such model because they are assumed to be known by the adversary due to the correctness of the encryption scheme. For the CKKS scheme, although it has been proven to be secure in the $\mathsf{IND\text{-}CPA}$ model, such assumption is not adequate, and hence we should consider its security in a different model.

In Chapter 3, we propose a passive security model for (homomorphic) approximate encryption schemes, called $\mathsf{IND\text{-}CPA}^+$, as a natural extension of $\mathsf{IND\text{-}CPA}$. In the $\mathsf{IND\text{-}CPA}^+$ security definitions, an adversary can 1) influence honest users on the choices of plaintext messages and homomorphic computations, 2) observe the resulting ciphertexts, and 3) observe decryption results of well-formed, honestly generated ciphertexts. We stress that a passive attack model should still allow the attacker to access plaintext messages decrypted from honestly generated ciphertexts: as illustrated in Figure 1.1, such an attacker is only attempting to gain information by passively observing a user's behaviors *through the output interfaces* of a system built from the homomorphic encryption scheme, without actively tampering or injecting adversarily chosen ciphertexts. So this definition should be distinguished from the active security notions such as *indistinguishability under chosen ciphertext attacks* (or IND-CCA/IND-CCA2). In particular, as proper encryption schemes are special cases of approximate encryption schemes with no approximation error, $\mathsf{IND\text{-}CPA}^+$ security also applies to proper encryption schemes, and we show that $\mathsf{IND\text{-}CPA}^+$ is equivalent to $\mathsf{IND\text{-}CPA}$ security in this case. However, these two notions are quite different in general, as we show that CKKS is subject to complete key recovery under passive attacks, hence breaking the $\mathsf{IND\text{-}CPA}^+$ security.

In addition, we also propose a simulation-based model, called $\mathsf{SIM\text{-}CPA}^+$ security, as a natural extension of the simulation-based semantic security, or $\mathsf{SIM\text{-}CPA}$ security, for proper encryption schemes. Unlike proper encryption schemes for which $\mathsf{IND\text{-}CPA}$ is equivalent to $\mathsf{SIM\text{-}CPA}$ security, we currently do not know if $\mathsf{IND\text{-}CPA}^+$ implies $\mathsf{SIM\text{-}CPA}^+$ security. Moreover, we propose variants of $\mathsf{IND\text{-}CPA}^+$ security, by either imposing restrictions or making extensions: we parameterize $\mathsf{IND\text{-}CPA}^+$ security by the maximal numbers of ciphertexts and decryptions accessible to the adversary; we consider different query models that release ciphertexts and decryption results in different orders, in particular, we consider non-adaptive and fully adaptive models; we also consider circuit privacy and functional decryption as extensions. For some of these variants, we show separation results, which are unique and do not exist for the $\mathsf{IND\text{-}CPA}$ model.

We report on the implementation and experimental results of our attack against several open-source libraries that implement the CKKS scheme. Our attack can successfully recover the secret key for most of the parameter settings in libraries HEAAN, SEAL, HElib, and PALISADE, on homomorphically computed ciphertexts from very common computations in secure machine learning applications. We prove that if the CKKS scheme is modified by adding a large random noise during decryption, then the resulting scheme is secure in the $\mathsf{IND\text{-}CPA}^{+}$ model.

Symbolic security model and proof for garbled circuits. Next, we move on to study secure computation protocols. The problem of secure computation between distrust parties is perhaps one of the most complex problems solved in cryptography. Among many solutions, Yao's garbled circuits [72, 73] (and its many variants) is the first, and in many situations to minimize the computational complexity, are still the most powerful and preferred primitives that enable secure function evaluation. Moreover, garbled circuits are the basic building blocks of many cryptographic systems. Garbled circuit schemes provide a randomized encoding of a given circuit C and its input x, and its security requires that the garbled encodings $(\tilde{C}, \tilde{x})$ reveal nothing except the output of the computation $C(x)$. Analyzing security of garbled circuits is not easy, partly due to the complexity of the problem of secure function evaluation in general. The first formal security analysis of a basic form of garbled circuit schemes [48] appears approximately 30 years after Yao proposed the protocol [72,73], and analysis with improved security bounds in stronger models [6,39,43], come years later. From the technical perspective, the major source of complexity is perhaps due to the generality of secure computation problem: the details of the garbled circuit construction depends on the specific computation that the parties want to perform, so the hybrids in computational security proofs must be designed to put together many different cases, resulting in a complex probabilistic analysis overall.

In Chapter 4, we analyze the security of garbled circuits in the computational sound symbolic framework of Micciancio [58] (with minor and natural extensions). The symbolic analysis of cryptographic constructions with computational soundness was initially proposed by

Abadi and Rogaway [1], in the following paradigm:

- Setting up a *symbolic model* for expressing the cryptographic computations, by providing a simple language to describe (and analyze) cryptographic protocols without all the details and complications of concrete (complexity based) computational models.
- Proving a general *computational soundness* theorem, translating certain symbolic properties in this abstract model of computation into computational security of the concrete systems instantiated from the symbolic descriptions with computational cryptographic primitives satisfying standard (computational) notions of security.
- Prove that the symbolic decryption of the protocol satisfies certain purely syntactic security properties, i.e., secure in a purely symbolic/syntactical way within the abstract model.
- Conclude, via the computational soundness theorem, that the standard implementation of the protocol (using a concrete, computationally secure instantiations of the cryptographic primitives) satisfies the computational security properties.

As a general purpose framework, the computational soundness theorem is proved only once for all protocols that can be specified in this framework. So, for a specific protocol, its security analysis is simplified to first establish a symbolic specification, and then to prove the symbolic security properties. This approach abstracts away the complexity of probabilistic arguments in computational security proofs, resulting in a simpler and cleaner security analysis that could provide a concise and succinct "core security lemma" showing why the construction is secure. Comparing to other formal cryptographic frameworks, the symbolic security proofs can typically be done in a pen-and-paper fashion, and they are easier to understand for cryptographers who are usually not familiar with concepts in formal methods and programming languages, lowering the barriers for them to apply formal methods in cryptography.

We begin with the symbolic language of [57, 58] that allows to use (arbitrarily nested) encryption and pseudorandom generators, and we extend it to include also randomly chosen bits and random permutations. Adversary's view is expressed by *patterns* of symbolic expressions, which is an extension of the symbolic language by including constructs specifying (computationally) hidden information. The patterns are derived from symbolic expressions co-inductively, by computing the greatest fixed point of a function that deduces information that are necessarily hidden from an adversary if the symbolic system is instantiated using (computationally) secure primitives. We show that the soundness of this extended framework follows easily from the soundness theorem of [58].

Then, we formally specify in this symbolic model a generic variant of Yao's garbled circuit that uses the point-and-permute technique [4]. Along the way, we propose a modular, algebraic way to specify arbitrary circuits in our symbolic model, similar to ideas used in modern high level programming languages such as Hughes' arrows [40, 41]. Our circuit specification is inductive such that an arbitrary circuit can be built from smaller ones using combinators, starting from the basic case of single gates. The benefit of using this algebraic circuit representation is not only a succinct formal definition of circuits, but it also enables using *structured induction* in the associated symbolic security analysis, resulting in a well organized proof.

We consider simulation-based security for garbled circuits, and we describe a symbolic simulator. Using the symbolic language and the algebraic circuit representation, we are able to give a detailed, formal proof showing that the output of Yao's garbling procedure and the output of the simulator are symbolically equivalent, i.e., they map to equivalent symbolic patterns. According to the computational soundness theorem, we can automatically conclude that the garbled circuits variant we analyzed is computationally secure. Note that the garbled circuit scheme we considered is generic, and efficient constructions that use a fixed key cipher [5], free-XOR [47], and half-gate [75] techniques can be adapted with small modifications to the framework and the analysis.

We implemented our symbolic framework as well as the garbled circuit scheme and its security analysis in Haskell. Our programs are short, can generate random circuits and perform symbolic security analysis on them very efficiently.

Equational security model for oblivious transfer protocols. We continue on the study of secure computation protocols, and in Chapter 5, we study oblivious transfer protocols in the equational security framework [59], which offers a formal mathematical model for specifying and analyzing cryptographic protocols in a composable way. Oblivious transfer (OT), in its most commonly used 1-out-of-2 formulation [28], is a two party protocol involving a sender transmitting two messages m_0, m_1 and a receiver obtaining only one of them m_b, in such a way that the sender does not learn which message $b \in \{0, 1\}$ was delivered and the receiver does not learn anything about the other message m_{1-b}. OT is a classic example of secure computation [28, 63], and an important (in fact, complete) building block for secure function evaluation protocols [25, 33, 42, 44, 49, 74] together with garbled circuits.

We examine a generic OT length extension protocol and a very efficient OT protocol [24] in the random oracle model. Our analysis starts with the naive definition of an OT functionality, often used in literature (including in the UC framework [16]) to describe an ideal OT, that the sender has two inputs m_0, m_1 and no output, and the receiver has one input b and one output m_b. Using such a definition, we are able to show that the OT length extension protocol, often considered as a folklore result in cryptography, does not satisfy the security requirement for OT. What missing here is a critical timing information that must be given to the sender to indicate the receipt of the receiver's choice bit b. However, with a simple modification to the OT definitions such that the sender outputs an *acknowledgment a* when the receiver's input b is specified, the OT length extension protocol is indeed provably secure using a simple equational proof.

For the efficient OT protocol, the so-called "simplest OT protocol" of [24], we give a formal specification in the equational framework. We show that, if the naive OT definition is used, then the protocol is insecure against both corrupted sender and corrupted receiver. For the case of corrupted sender, the failure of simulation is due to the lack of the timing information about the receipt of receiver's choice bit b. So, we then modify the protocol using the revised OT definition. This time, we show that the protocol is still insecure against a corrupted receiver, and the problem is that in a real protocol execution the receiver can delay its random oracle query until after seeing the sender's ciphertexts, but in the ideal protocol execution, if the simulator has to output the ciphertexts before seeing the receiver's random oracle query, then it must be able to guess an external random bit correctly before seeing any inputs, which is impossible to achieve with high probability.

With these case studies, we exemplify the use of equational framework in analyzing secure computation protocols. In particular, we are able to specify important timing or dependency relations among different communication channels and computation results, leading to a simple revision of the ideal OT definition. The insecurity results we obtained should not be interpreted as the evidence of concrete attacks to the protocols; rather, they show that some details (e.g., the timing information, which is sometimes categorized into control signals in other frameworks) that are usually omitted in the security analysis may be crucial for a complete security proof. As we mentioned earlier, due to the complexity of maintaining composable security with the traditional models of interactive computation (such as interactive Turing machines), giving a full, detailed formal specification in composable security frameworks in such models is typically considered too complicated to do, yet it can be crucial to obtain a precise mathematical proof of security. Our findings show the usefulness of the highly abstract, algebraic model in the equational framework for formulating and investigating secure computation protocols.

Chapter 2

Preliminaries

In this chapter, we present some common notations and definitions for later chapters. As the topics of this thesis span over several different research areas of cryptography, definitions that are only used by a specific chapter will be introduced as they are needed.

Notations. For a positive integer n, we write $[n] = \{1, \dots, n\}$. We use the notation $\mathbf{a} = (a_0, \dots, a_{n-1})$ for column vectors, and $\mathbf{a}^t = [a_0, \dots, a_{n-1}]$ for rows. Vector entries are indexed starting from 0, and denoted by a_i or $\mathbf{a}[i]$. The dot product between two vectors (with entries in a ring) is written $\langle \mathbf{a}, \mathbf{b} \rangle$ or $\mathbf{a}^t \cdot \mathbf{b}$. Scalar functions $f(\mathbf{a}) = (f(a_0), \dots, f(a_{n-1}))$ are applied to vectors componentwise.

For any finite set A, we write $x \leftarrow A$ for the operation of selecting x uniformly at random from A. More generally, if χ is a probability distribution over A, $x \leftarrow \chi$ selects x according to χ, and we write $\chi(a) = \Pr_{x \leftarrow \chi}\{x = a\}$ for any $a \in A$.

Standard Cryptographic Definitions. In all our definitions, we denote the security parameter by κ. A function f in κ is *negligible* if $f(\kappa) = \kappa^{-\omega(1)}$. We use $\mathsf{negl}(\kappa)$ to denote an arbitrary negligible function in κ. A function g in κ is *overwhelming* if $1 - g(\kappa)$ is negligible.

Let χ_1 and χ_2 be distributions over a finite set A. The *statistical distance* between χ_1 and χ_2 is $\mathsf{SD}(\chi_1, \chi_2) = \frac{1}{2} \sum_{a \in A} |\chi_1(a) - \chi_2(a)|$.

Chapter 3

Passive Security Model of Homomorphic Approximate Encryption

In this chapter we present passive security definitions for (homomorphic) approximate encryption schemes, and we study their relationship with exact encryption schemes. In particular, we show that the CKKS scheme [22], a natural homomorphic approximate encryption scheme that is secure in the $\mathsf{IND\text{-}CPA}$ model under the hardness assumptions of standard lattice problems, is subject to a passive key-recovery attack, hence showing it is not secure in the passive security model for approximate encryption schemes.

3.1 Security Definitions for Exact (Homomorphic) Encryption Schemes

We first recall the formal definitions for exact (homomorphic) encryption schemes: their syntax, correctness, and the passive security definitions. These definitions serve as the basis for our security model of approximate encryption schemes.

A *public-key encryption scheme* with a message space $\mathcal{M}$ is a tuple $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ consisting of three algorithms:

- a randomized key generation algorithm KeyGen that takes the security parameter 1^κ and outputs a secret key sk and a public key pk,
- a randomized encryption algorithm Enc that takes pk and a message $m \in \mathcal{M}$ and outputs a

ciphertext ct, and

- a deterministic decryption algorithm Dec that takes sk and a ciphertext ct and outputs a message m' or a special symbol $\perp$ indicating decryption failure.

We usually parameterize Enc with pk and write $\mathsf{Enc}_{\mathsf{pk}}(\cdot)$ to denote the function $\mathsf{Enc}(\mathsf{pk}, \cdot)$, and similarly we write $\mathsf{Dec}_{\mathsf{sk}}(\cdot)$ for the function $\mathsf{Dec}(\mathsf{sk}, \cdot)$. A public-key encryption scheme is *correct* if for all $m \in \mathcal{M}$ and keys $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{KeyGen}(1^\kappa)$, $\Pr\{\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = m\} = 1 - \mathsf{negl}(\kappa)$, where the probability is over the randomness of Enc.

A public-key *homomorphic* encryption scheme is a public-key encryption scheme with an additional, possibly randomized, *(homomorphic) evaluation* algorithm Eval, and such that KeyGen outputs an additional *evaluation key* ek besides sk and pk. The algorithm Eval takes ek, a circuit $g : \mathcal{M}^l \to \mathcal{M}$ for some $l \geq 1$, and a sequence of l ciphertexts ct_i, and it outputs a ciphertext ct'. The *correctness* of a homomorphic encryption scheme requires that, for all keys $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek})$ in the support of $\mathsf{KeyGen}(1^\kappa)$, for all circuits $g : \mathcal{M}^l \to \mathcal{M}$ and for all $m_i \in \mathcal{M}$, $1 \leq i \leq l$, it holds that

$$\Pr\left\{\begin{array}{l} \mathsf{ct}_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_i) \text{ for } 1 \leq i \leq l, \\ \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Eval}_{\mathsf{ek}}(g, (\mathsf{ct}_i)_{i=1}^l)) = g((m_i)_{i=1}^l) \end{array}\right\} = 1 - \mathsf{negl}(\kappa),$$

where the probability is over the randomness of Enc and Eval. We also require that the complexity of Dec is independent (or a slow growing function) of the size of the circuit g.

In terms of security, we recall the standard security notion of *indistinguishability under chosen plaintext attack*, or $\mathsf{IND\text{-}CPA}$, for public-key (homomorphic) encryption schemes.

Definition 1 ($\mathsf{IND\text{-}CPA}$ Security). *Let* $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a homomorphic encryption scheme. We define an experiment* $\mathsf{Expr}_b^{\mathrm{cpa}}[\mathcal{A}]$ *parameterized by a bit* $b \in \{0, 1\}$ *and an efficient* *adversary $\mathcal{A}$:*

$$\begin{aligned}
\mathsf{Expr}_b^{\mathrm{cpa}}[\mathcal{A}](1^\kappa):\quad & (\mathsf{sk},\mathsf{pk},\mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)\\
& (x_0, x_1) \leftarrow \mathcal{A}(1^\kappa, \mathsf{pk}, \mathsf{ek})\\
& \mathsf{ct} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(x_b)\\
& b' \leftarrow \mathcal{A}(\mathsf{ct})\\
& \mathsf{return}(b')
\end{aligned}$$

We say that the scheme is IND-CPA secure *if for any efficient adversary $\mathcal{A}$, it holds that*

$$\mathsf{Adv}^{\mathrm{cpa}}[\mathcal{A}](\kappa) = |\Pr\{\mathsf{Expr}_0^{\mathrm{cpa}}[\mathcal{A}](1^\kappa) = 1\} - \Pr\{\mathsf{Expr}_1^{\mathrm{cpa}}[\mathcal{A}](1^\kappa) = 1\}| = \mathsf{negl}(\kappa).$$

3.2 Security Definitions for Homomorphic Approximate Encryption Schemes

In this section we present general definitions that accurately capture passive attacks against a (possibly approximate, homomorphic) encryption scheme. We recall that in a passive attack the adversary may control which messages get encrypted, what homomorphic computations are performed on them, and may observe all ciphertexts produced in the process, as well as the decrypted result of the computations.

We first present an indistinguishability-based definition (similar in spirit to the standard IND-CPA notion described in Definition 1). A simulation-based notion is presented in Section 3.2.2. Then, we explore restricted and extended variants of these basic definitions.

3.2.1 Indistinguishability-Based Definition

Our first definition is indistinguishability-based: the adversary chooses a number of pairs of plaintext messages, and its goal is to determine whether the ciphertexts it receives are encryptions of the first or the second plaintext in the pairs. In contrast to Definition 1, our new definition allows an adversary to make *multiple* challenge queries (m_0, m_1), rather than a single one. Our adversary can also issue homomorphic evaluation and decryption queries. We now give the formal definition. For simplicity, and as common in homomorphic encryption schemes, we assume all messages belong to a fixed message space $\mathcal{M}$. In particular, all messages have (or can be padded to) the same length. We refer to our definition as $\mathsf{IND\text{-}CPA}^+$, as it includes $\mathsf{IND\text{-}CPA}$ (see Definition 1) as a special case, where the adversary makes only one encryption query, and no homomorphic evaluation or decryption queries.

Definition 2 ($\mathsf{IND\text{-}CPA}^+$ Security)**.** *Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be a public-key homomorphic (possibly approximate) encryption scheme with plaintext space $\mathcal{M}$ and ciphertext space C. We define an experiment $\mathsf{Expr}_b^{\mathrm{indcpa}^+}[\mathcal{A}]$, parameterized by a bit $b \in \{0, 1\}$ and involving an efficient adversary $\mathcal{A}$ that is given access to the following oracles, sharing a common state $S \in (\mathcal{M} \times \mathcal{M} \times C)^*$ consisting of a sequence of message-message-ciphertext triplets:*

- *An encryption oracle $\mathsf{E}_{\mathsf{pk}}(m_0, m_1)$ that, given a pair of plaintext messages m_0, m_1, computes $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_b)$, extends the state*

  $$S := [S; (m_0, m_1, c)]$$

  with one more triplet, and returns the ciphertext c to the adversary.

- *An evaluation oracle $\mathsf{H}_{\mathsf{ek}}(g, J)$ that, given a function $g : \mathcal{M}^k \to \mathcal{M}$ and a sequence of indices $J = (j_1, \ldots, j_k) \in \{1, \ldots, |S|\}^k$, computes $c \leftarrow \mathsf{Eval}_{\mathsf{pk}}(g, S[j_1].c, \ldots, S[j_k].c)$, extends the state*

  $$S := [S; (g(S[j_1].m_0, \ldots, S[j_k].m_0),\ g(S[j_1].m_1, \ldots, S[j_k].m_1),\ c)]$$

  with one more triplet, and returns the ciphertext c to the adversary. Here and below $|S|$

denotes the number of triplets in the sequence S, and $S[j].m_0$, $S[j].m_1$ and $S[j].c$ denote the three components of the jth element of S.

- *A decryption oracle $\mathsf{D}_{\mathsf{sk}}(j)$ that, given an index $j \leq |S|$, checks that $S[j].m_0 = S[j].m_1$, and, if so, returns $\mathsf{Dec}_{\mathsf{sk}}(S[j].c)$ to the adversary. (If the check fails, a special error symbol $\perp$ is returned.)*

The experiment is defined as

$$\begin{aligned}
\mathsf{Expr}_b^{\text{indcpa}^+}[\mathcal{A}](1^\kappa): \quad & (\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa) \\
& S := [\,] \\
& b' \leftarrow \mathcal{A}^{\mathsf{E}_{\mathsf{pk}}, \mathsf{H}_{\mathsf{ek}}, \mathsf{D}_{\mathsf{sk}}}(1^\kappa, \mathsf{pk}, \mathsf{ek}) \\
& \mathsf{return}(b')
\end{aligned}$$

The advantage of adversary $\mathcal{A}$ against the IND-CPA$^+$ security of the scheme is

$$\mathsf{Adv}^{\text{indcpa}^+}[\mathcal{A}](\kappa) = |\Pr\{\mathsf{Expr}_0^{\text{indcpa}^+}[\mathcal{A}](1^\kappa) = 1\} - \Pr\{\mathsf{Expr}_1^{\text{indcpa}^+}[\mathcal{A}](1^\kappa) = 1\}|,$$

where the probability is over the randomness of $\mathcal{A}$ and the experiment. The scheme $\mathcal{E}$ is IND-CPA$^+$-secure *if for any efficient (probabilistic polynomial time) $\mathcal{A}$, the $\mathsf{Adv}^{\text{indcpa}^+}[\mathcal{A}]$ is negligible in κ.*

As a standard convention, if at any point in an experiment the adversary makes an invalid query (e.g., a circuit g not supported by the scheme, or indices out of range), the oracle simply returns an error symbol $\perp$.

We remark that, while the adversary in Definition 2 is given access to a decryption oracle, this should not be confused with indistinguishability under a *chosen ciphertext attack* (IND-CCA), which models active adversaries with the capability of tampering with (or injecting) arbitrary ciphertexts. Definition 2 only allows for decryption queries on valid ciphertexts that have been honestly computed using the correct encryption and homomorphic evaluation algorithms. Furthermore, the requirement that $S[j].m_0 = S[j].m_1$ is to eliminate trivial attacks where the adversary can distinguish between two computations that lead to different results when computed on exact values.

Exact encryption schemes can be seen as a special case of approximate encryption, with the added correctness requirement. So, Definition 2 can be applied to exact as well as approximate encryption schemes. As a sanity check, we compare our new definition with the traditional formulation of $\mathsf{IND\text{-}CPA}$ security (Definition 1) modeling passive attacks against exact encryption schemes. Perhaps not surprisingly, for the case of exact encryption schemes, our new security definition coincides with the standard notion of $\mathsf{IND\text{-}CPA}$ security.

Lemma 1. *Any exact homomorphic encryption scheme $\mathcal{E}$ is $\mathsf{IND\text{-}CPA}$ secure if and only if it is $\mathsf{IND\text{-}CPA}^+$ secure.*

Proof. It is easy to see that $\mathsf{IND\text{-}CPA}^+$ security implies $\mathsf{IND\text{-}CPA}$ security, as an adversary making only one E query but no other queries in the $\mathsf{IND\text{-}CPA}^+$ experiment is also an $\mathsf{IND\text{-}CPA}$ adversary. So we consider the reverse direction.

Assume $\mathcal{E}$ is $\mathsf{IND\text{-}CPA}$ secure. Let $\mathcal{A}$ be any adversary breaking the $\mathsf{IND\text{-}CPA}^+$ security of $\mathcal{E}$, and assume $\mathcal{A}$ makes at most l queries in total to E and H. We build adversaries $\mathcal{B}^{(i)}$, for $0 \leq i < l$, to break the $\mathsf{IND\text{-}CPA}$ security of $\mathcal{E}$.

$\mathcal{B}^{(i)}$ takes input 1^κ, pk, ek, and it then runs $\mathcal{A}(1^\kappa, \mathsf{pk}, \mathsf{ek})$. It maintains a state $S \in (\mathcal{M} \times \mathcal{M} \times C)^*$ just like $\mathsf{Expr}^{\mathrm{indcpa}^+}$, and it answers oracle queries made by $\mathcal{A}$ as follows:

- For each query (m_0, m_1) to E, if $|S| < i$, then let $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_1)$; if $|S| > i$, then let $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_0)$; and if $|S| = i$, $\mathcal{B}^{(i)}$ sends (m_0, m_1) to $\mathsf{Expr}_b^{\mathrm{cpa}}$ and receives c. The state S is extended by one more triplet (m_0, m_1, c), and c is returned to $\mathcal{A}$.

- For each query (g, J) to H, where $g : \mathcal{M}^k \to \mathcal{M}$ and $J = (j_1, \ldots, j_k)$, let $c \leftarrow$

$\mathsf{Eval}_{\mathsf{ek}}(g, S[j_1].c, \dots, S[j_k].c)$, extend S by one more triplet

$$(g(S[j_1].m_0,\ \dots, S[j_k].m_0),\ g(S[j_1].m_1, \dots, S[j_k].m_1), c),$$

and return c to $\mathcal{A}$.

- For each query j to D, if $j \le |S|$ and $S[j].m_0 = S[j].m_1$, then return $S[j].m_0$ to $\mathcal{A}$; otherwise return an error symbol $\perp$.

Finally, when $\mathcal{A}$ halts with a bit b', $\mathcal{B}^{(i)}$ output this bit.

Since $\mathcal{B}^{(i)}$ does not depend on the secret key sk to answer the D queries, it is a valid adversary in the $\mathsf{IND\text{-}CPA}$ experiment. Now, let $\mathcal{H}^{(i)} = \mathsf{Expr}_0^{\mathrm{cpa}}[\mathcal{B}^{(i)}]$ for $0 \le i < l$, and let $\mathcal{H}^{(l)} = \mathsf{Expr}_1^{\mathrm{cpa}}[\mathcal{B}^{(l-1)}]$. For $1 \le i < l$, note that $\mathcal{H}^{(i)}$ is exactly the same distribution as $\mathsf{Expr}_1^{\mathrm{cpa}}[\mathcal{B}^{(i-1)}]$. Furthermore, by the correctness of exact homomorphic encryption schemes, the D responses from $\mathcal{B}^{(i)}$ to $\mathcal{A}$ are indistinguishable from those in the $\mathsf{IND\text{-}CPA}^{+}$ experiment; so $\mathcal{H}^{(0)}$ and $\mathsf{Expr}_0^{\mathrm{indcpa}^+}[\mathcal{A}]$ are indistinguishable, and the same hold true for $\mathcal{H}^{(l)}$ and $\mathsf{Expr}_1^{\mathrm{indcpa}^+}[\mathcal{A}]$. So $\mathsf{Adv}^{\mathrm{indcpa}^+}[\mathcal{A}] \le \sum_{0 \le i < l} \mathsf{Adv}^{\mathrm{cpa}}[\mathcal{B}^{(i)}] + \mathsf{negl}(\kappa)$, which is negligible since $\mathcal{E}$ is $\mathsf{IND\text{-}CPA}$ secure. □

Notice that the above lemma makes essential use of the correctness of exact encryption schemes, and the proof does not extend to approximate encryption schemes. In fact, for approximate encryption schemes, the result of decryption is not a simple function of the encrypted messages (and the computations performed on them), and may potentially depend (in an indirect, unspecified way) on the scheme's secret key and encryption randomness. So the information provided by decryption queries is not easily computed by the adversary on its own, and, at least in principle, $\mathsf{IND\text{-}CPA}^{+}$ may be a stronger security notion than $\mathsf{IND\text{-}CPA}$ when applied to approximate encryption schemes. We will make this intuition clear in the following sections, proving formal separation results, and providing concrete attacks to actual approximate encryption schemes.

Also note that the above definition does not guarantee circuit privacy in the homomorphically evaluated ciphertexts, as the circuit to be evaluated in a query to oracle H does not depend on the bit b of the $\mathsf{IND\text{-}CPA}^+$ experiment. In a later section we extend our definition with circuit privacy. Here we focus on the basic definition (without circuit privacy) which is the most common in cryptography.

3.2.2 Simulation-Based Definition

As standard in cryptography, secure encryption schemes are expected to hide all partial information about plaintext messages. In the indistinguishability notion of security, this is captured by the task of distinguishing between any two (adversarially chosen) messages. Simulation-based security (also known as semantic security) offers a more direct way to express that ciphertexts provide no useful information at all. We consider standalone simulation-based security, following the "real vs ideal worlds" paradigm: the real world is implemented using the cryptographic scheme to process real input, and the ideal world consists of a *simulator* which is given a minimal amount of information and should produce an output essentially equivalent to a real attack.

We propose the following simulation-based security definition for homomorphic approximate encryption schemes. For simplicity, we consider a plaintext space with fixed message length $\mathcal{M} = \{0, 1\}^l$. The definition is easily extended to variable-length message spaces.

Definition 3 ($\mathsf{SIM\text{-}CPA}^+$ Security)**.** *Let* $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ *be a public-key homomorphic (possibly approximate) encryption scheme with plaintext space* $\mathcal{M} = \{0, 1\}^l$*. Security is defined with respect to an adversary* $\mathcal{A}$ *that is given a public/evalution key* $(\mathsf{pk}, \mathsf{ek})$ *and has access to three (stateful) oracles:*

- *An encryption oracle* $\mathsf{E}(m)$ *that, given a plaintext messages* m*, returns a ciphertext* c*.*
- *An evaluation oracle* $\mathsf{H}(g, J)$ *that, given a function* $g : \mathcal{M}^k \to \mathcal{M}$ *for some* $k \geq 1$ *and a sequence of indices* $J = (j_1, \ldots, j_k) \in \{1, \ldots, |S|\}^k$*, returns a ciphertext* c*.*

- *A decryption oracle* $\mathsf{D}(j)$ *that, given an index* j *returns a plaintext message* m*.*

Oracle queries are answered in two different ways, defining a "real" and an "ideal" experiment. The real experiment maintains a state consisting of a sequence $T \in (\mathcal{M} \times \mathcal{C})^*$ *of message-ciphertext pairs, and the ideal experiment maintains a sequence of messages* $T \in \mathcal{M}^*$ *as its state. The indexes* J *and* j *in the evaluation and decryption queries are required to be in the range* $\{1, \dots, |T|\}$*, where* $|T|$ *is the current length of* T*.*

The real world experiment Real *begins by initializing* $T := [\,]$ *to the empty sequence, and sampling a tuple of keys* $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ *using the scheme's key generation algorithm. Then, the keys* $(\mathsf{pk}, \mathsf{ek})$ *are given to* $\mathcal{A}$*, which is run answering its oracle queries as follows:*

- $\mathsf{E}(m)$*: compute* $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m)$*, extend the state* $T := [T; (m, c)]$ *with one more pair, and return c to* $\mathcal{A}$*.*

- $\mathsf{H}(g, J)$*: compute* $c \leftarrow \mathsf{Eval}_{\mathsf{ek}}(g, (T[j_1].c, \dots, T[j_k].c))$*, extend the state*

$$T := [T; (g(T[j_1].m, \dots, T[j_k].m), c)]$$

  with one more pair, and return c to $\mathcal{A}$*.*

- $\mathsf{D}(j)$*: compute* $m' = \mathsf{Dec}_{\mathsf{sk}}(T[j].c)$ *and return it to* $\mathcal{A}$*.*

The ideal world experiment Ideal *answers the adversary's queries using an efficient (stateful) simulator* $\mathcal{S}$ *(see Fig. 3.1 for an illustration), which maintains its own state, in addition (and without access) to* T*. The ideal experiment begins by initializing* $T := [\,]$ *to the empty sequence, and starting the simulator* $\mathcal{S}$ *which produces a pair of keys* $(\mathsf{pk}, \mathsf{ek})$ *that are given to the adversary* $\mathcal{A}$*. Then, it answers* $\mathcal{A}$*'s oracle queries as follows:*

- $\mathsf{E}(m)$*: send the message* `E` *to the simulator,* $\mathcal{S}$*, which replies with a ciphertext c. The state* $T := [T; m]$ *is extended with one more message m, and the ciphertext c is returned to the adversary.*

- $\mathsf{H}(g, J)$*: send the message* $(\mathtt{H}, g, J)$ *to the simulator* $\mathcal{S}$*, which replies with a ciphertext* c*. The state*

$$T := [T; g(T[j_1], \dots, T[j_k])]$$

is extended with one more message $g(T[j_1], \dots, T[j_k])$*, and* c *is returned to* $\mathcal{A}$*.*

- $\mathsf{D}(j)$*: send* $(\mathtt{D}, j, T[j])$ *to the simulator* $\mathcal{S}$*. The simulator is expected to reply with a message* m' *(possibly different from* $T[j]$*) which is returned to the adversary.*

As usual, in both experiments, whenever $\mathcal{A}$ *makes an invalid query, the oracle returns an error symbol* $\perp$*. The experiments terminate when* $\mathcal{A}$ *halts with an output bit* b*. This bit is the final output of the experiment, and it is denoted by* $\mathsf{Real}[\mathcal{A}](1^\kappa)$ *or* $\mathsf{Ideal}[\mathcal{S}, \mathcal{A}](1^\kappa)$*. The advantage of adversary* $\mathcal{A}$ *in breaking SIM-CPA*$^+$ *security is*

$$\mathsf{Adv}^{\mathrm{simcpa}^+}[\mathcal{A}](\kappa) = |\Pr\{\mathsf{Ideal}[\mathcal{S}, \mathcal{A}](1^\kappa) = 1\} - \Pr\{\mathsf{Real}[\mathcal{A}](1^\kappa) = 1\}|.$$

We say that $\mathcal{E}$ *is* SIM-CPA$^+$-secure *if there exists an efficient (probabilistic polynomial time) simulator* $\mathcal{S}$ *such that, for all efficient* $\mathcal{A}$ *the advantage* $\mathsf{Adv}^{\mathrm{simcpa}^+}[\mathcal{A}]$ *is negligible in* κ*.*

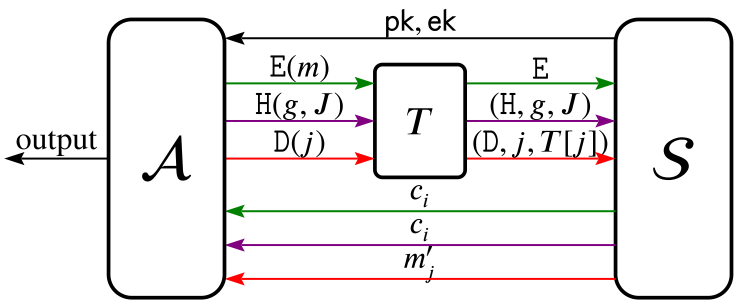


Figure 3.1. The ideal world experiment Ideal that involves a simulator $\mathcal{S}$ and an adversary $\mathcal{A}$. The box between $\mathcal{A}$ and $\mathcal{S}$ indicates that oracle queries from $\mathcal{A}$ are processed (with the help of the state T) by the experiment before sending to $\mathcal{S}$.

In the ideal world experiment, the input to the simulator describes information that is not necessarily protected by the scheme $\mathcal{E}$: the number of plaintexts to be encrypted, the homomorphic computation to be performed, and the exact computation results (which can be derived from the input plaintexts). The simulator's task, given these minimal information, is to simulate any attack that can be mounted by a real world adversary. As we mentioned, our definition makes an assumption that all plaintext messages, including plaintext computation results corresponding to homomorphic evaluations, are of the same bit length l. The definition can be extended to variable length messages by giving the length information $|m|$ to the simulator.

Relations with $\mathsf{IND\text{-}CPA}^+$-security. For exact homomorphic encryption schemes, it is well known [34, 54] that the simulation-based semantic security is equivalent to the indistinguishability-based $\mathsf{IND\text{-}CPA}$ security. So naturally we want to extend such relationship to homomorphic approximate encryption schemes. The following implication result is easy to check.

Lemma 2. *For any homomorphic approximate encryption scheme $\mathcal{E}$, if $\mathcal{E}$ is $\mathsf{SIM\text{-}CPA}^+$-secure, then it is $\mathsf{IND\text{-}CPA}^+$-secure. Moreover, the reduction between the two adversaries preserves the number, type and order of queries.*

Proof. Assume $\mathcal{E}$ is $\mathsf{SIM\text{-}CPA}^+$-secure, and fix an $\mathsf{IND\text{-}CPA}^+$ adversary $\mathcal{A}$. We build two $\mathsf{SIM\text{-}CPA}^+$ adversaries $\mathcal{B}_0$ and $\mathcal{B}_1$: For $b \in \{0, 1\}$, $\mathcal{B}_b$ receives the public keys $(\mathsf{pk}, \mathsf{ek})$, maintains a state $M \in (\mathcal{M} \times \mathcal{M})^*$, runs $\mathcal{A}(1^\kappa, \mathsf{pk}, \mathsf{ek})$ and handles its oracle queries as follows:

- For each $\mathsf{E}(m_0, m_1)$ query, $\mathcal{B}_b$ stores this message pair $M := [M;\ (m_0, m_1)]$, queries its oracle $\mathsf{E}(m^b)$, and then it returns the oracle response c to $\mathcal{A}$.
- For each $\mathsf{H}(g, J)$ query, where $J = (j_1, \dots, j_k)$, $\mathcal{B}_b$ extends its state by a new message pair $M := [M;\ (g(M[j_1].m_0, \dots, M[j_k].m_0),\ g(M[j_1].m_1, \dots, M[j_k].m_1))]$, queries its oracle $\mathsf{H}(g, J)$, and it then returns the oracle response c to $\mathcal{A}$.
- For each $\mathsf{D}(j)$ query, if $M[j].m_0 = M[j].m_1$, $\mathcal{B}_b$ queries its oracle $\mathsf{D}(j)$ and returns the oracle response m' to $\mathcal{A}$; otherwise $\mathcal{B}_b$ returns the error symbol $\perp$ to $\mathcal{A}$.

By $\mathsf{SIM\text{-}CPA}^+$-security, there exists a simulator $\mathcal{S}$ such that $\mathsf{Real}[\mathcal{B}_0] \approx_c \mathsf{Ideal}[\mathcal{S}, \mathcal{B}_0]$ and $\mathsf{Real}[\mathcal{B}_1] \approx_c \mathsf{Ideal}[\mathcal{S}, \mathcal{B}_1]$. Note that $\mathsf{Real}[\mathcal{B}_b]$ and $\mathsf{Expr}_b^{\text{indcpa}^+}[\mathcal{A}]$ are exactly the same distribution for both $b \in \{0, 1\}$. Also note that, in both $\mathsf{Ideal}[\mathcal{S}, \mathcal{B}_0]$ and $\mathsf{Ideal}[\mathcal{S}, \mathcal{B}_1]$, if $M[j].m_0 = M[j].m_1$ for a decryption query $\mathsf{D}(j)$ from $\mathcal{A}$, then the input given to $\mathcal{S}$ in these two ideal world experiments are exactly the same; so $\mathsf{Ideal}[\mathcal{S}, \mathcal{B}_0] = \mathsf{Ideal}[\mathcal{S}, \mathcal{B}_1]$. Therefore the scheme $\mathcal{E}$ is $\mathsf{IND\text{-}CPA}^+$-secure, and our reduction preserves the number, type, and order of queries. $\square$

As exact homomorphic encryption schemes are special cases of homomorphic approximate encryption schemes, we compare $\mathsf{IND\text{-}CPA}$ with $\mathsf{SIM\text{-}CPA}^+$ security. The following lemma shows that (together with Lemma 1), for exact encryptions, $\mathsf{SIM\text{-}CPA}^+$ is also equivalent to $\mathsf{IND\text{-}CPA}$.

Lemma 3. *Any exact homomorphic encryption scheme $\mathcal{E}$ is $\mathsf{IND\text{-}CPA}^+$ secure if and only if it is $\mathsf{SIM\text{-}CPA}^+$ secure.*

Proof. We first show that $\mathsf{IND\text{-}CPA}^+$ security implies $\mathsf{SIM\text{-}CPA}^+$. To do so, we build a simulator $\mathcal{S}$. On start up, $\mathcal{S}$ samples keys $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ honestly using the key generation algorithm, and outputs $(\mathsf{pk}, \mathsf{ek})$. The simulator initializes a state $C := [\,]$ that is a sequence of ciphertexts. Then, it handles oracle queries from an adversary as follows:

- For each $\mathsf{E}(m)$ query, $\mathcal{S}$ receives a message $\mathtt{E}$, computes $c \leftarrow \mathsf{Enc}_{\mathsf{pk}}(0)$, extends its state $C := [C;\ c]$ by one more ciphertext, and returns c.
- For each $\mathsf{H}(g, J)$ query, $\mathcal{S}$ receives $(\mathtt{H}, g, J)$, computes $c \leftarrow \mathsf{Eval}_{\mathsf{ek}}(g, C[j_1], \ldots, C[j_k])$ where $J = (j_1, \ldots, j_k)$, extends its state as $C := [C;\ c]$, and returns c.
- For each $\mathsf{D}(j)$ query, $\mathcal{S}$ receives $(\mathtt{D}, j, m)$, and it returns m.

Now, fix any $\mathsf{SIM\text{-}CPA}^+$ adversary $\mathcal{A}$, and we build an $\mathsf{IND\text{-}CPA}^+$ adversary $\mathcal{B}$: On input $(1^\kappa, \mathsf{pk}, \mathsf{ek})$, $\mathcal{B}$ runs $\mathcal{A}(1^\kappa)$ and sends $(\mathsf{pk}, \mathsf{ek})$ to $\mathcal{A}$. It also maintains a sequence M of plaintexts as its state. For oracle queries from $\mathcal{A}$, $\mathcal{B}$ does the following:

- For each $\mathsf{E}(m)$ query, $\mathcal{B}$ extends $M := [M;\ m]$ by one more message, sends $(m, 0)$ to the oracle E_{pk} and receives c, and then it returns c to $\mathcal{A}$.

- For each $\mathsf{H}(g, J = (j_1, \ldots, j_k))$ query, $\mathcal{B}$ extends $M := [M;\ g(M[j_1], \ldots, M[j_k])]$ by one more message, sends (g, J) to the oracle H_{ek} and receives c, and then it returns c to $\mathcal{A}$.

- For each $\mathsf{D}(j)$ query, $\mathcal{B}$ returns $M[j]$ to $\mathcal{A}$.

By the correctness of exact homomorphic encryption schemes, we see that $\mathsf{Expr}_0^{\mathrm{indcpa}^+}[\mathcal{B}]$ is indistinguishable from the real world experiment $\mathsf{Real}[\mathcal{A}]$. On the other hand, the ideal world experiment $\mathsf{Ideal}[\mathcal{S}, \mathcal{A}]$ is exactly the same as $\mathsf{Expr}_1^{\mathrm{indcpa}^+}[\mathcal{B}]$. Since $\mathcal{E}$ is $\mathsf{IND\text{-}CPA}^+$ secure, it is also $\mathsf{SIM\text{-}CPA}^+$ secure.

For the reverse direction, we can directly apply Lemma 2. $\square$

For homomorphic approximate encryption schemes in general, we do not know if $\mathsf{IND\text{-}CPA}^+$ security implies $\mathsf{SIM\text{-}CPA}^+$ security, and we leave it as an interesting open question.

3.2.3 Restricted Security Notions and Separations Between Them

We have observed that, for exact encryption schemes, $\{\mathsf{IND\text{-}CPA}^+, \mathsf{SIM\text{-}CPA}^+\}$ security is equivalent to the traditional $\mathsf{IND\text{-}CPA}$ security. (See Lemma 1.) We now show that $\{\mathsf{IND\text{-}CPA}^+, \mathsf{SIM\text{-}CPA}^+\}$ is strictly stronger than $\mathsf{IND\text{-}CPA}$, i.e., there are approximate encryption schemes that are provably $\mathsf{IND\text{-}CPA}$ secure (under standard complexity assumptions) but are not $\{\mathsf{IND\text{-}CPA}^+, \mathsf{SIM\text{-}CPA}^+\}$ secure. In order to get a more refined understanding of the gap between these notions, we introduce a natural parameterization of $\mathsf{IND\text{-}CPA}^+$ and $\mathsf{SIM\text{-}CPA}^+$ security, that smoothly interpolates between $\mathsf{IND\text{-}CPA}$ and $\{\mathsf{IND\text{-}CPA}^+, \mathsf{SIM\text{-}CPA}^+\}$. Then, we define a number of restricted notions of security, and show separations between them, showing that there is an infinite chain of (strictly) increasingly stronger definitions, ranging from $\mathsf{IND\text{-}CPA}$ all the way to $\{\mathsf{IND\text{-}CPA}^+, \mathsf{SIM\text{-}CPA}^+\}$.

Restricting the numbers of queries. We parameterize the definition by imposing a bound on the number of queries that may be asked by the adversary.

Definition 4 ((q, ℓ)-$\mathsf{IND\text{-}CPA}^+$ Security)**.** *For any two functions $q(\kappa)$ and $\ell(\kappa)$ of the security parameter κ, we say that a homomorphic encryption scheme is (q, ℓ)-$\mathsf{IND\text{-}CPA}^+$ secure if it satisfies Definition 2 for all adversaries $\mathcal{A}$ that make at most $\ell(\kappa)$ queries to oracles* E, H*, and at most $q(\kappa)$ queries to oracle* D.

We combined the encryption (E) and evaluation (H) queries into a single bound $\ell(\kappa)$ for simplicity, and because both type of queries produce ciphertexts. The definition is easily extended to more general formulations, but we will be primarily interested in the bound q on the number of decryption queries, which are the distinguishing feature of approximate encryption schemes. When ℓ is an arbitrary polynomial, and only the number of decryption queries $q(\kappa)$ is restricted, we say that a scheme is q-$\mathsf{IND\text{-}CPA}^+$ secure.

Now, we can think of $\mathsf{IND\text{-}CPA}$ security as a special case of (q, ℓ)-$\mathsf{IND\text{-}CPA}^+$, for $q = 0$ and $\ell = 1$, as the only query to E/H must be an encryption query. (Oracle E must be called at least once before one can use H to homomorphically evaluate a function on a ciphertext.) So, bounding the number of queries allows to smoothly transition from the traditional $\mathsf{IND\text{-}CPA}$ definition (i.e., $(0, 1)$-$\mathsf{IND\text{-}CPA}^+$ security), to our $\mathsf{IND\text{-}CPA}^+$ (i.e., $(\mathsf{poly}, \mathsf{poly})$-$\mathsf{IND\text{-}CPA}^+$ security).

Similar to $\mathsf{IND\text{-}CPA}^+$ security, we can also consider parameterizations of the $\mathsf{SIM\text{-}CPA}^+$ security using bounds on the numbers of queries that an adversary is allowed to ask. We say that a homomorphic encryption scheme is *(q, ℓ)-$\mathsf{SIM\text{-}CPA}^+$ secure* if it satisfies Definition 3 for all adversaries $\mathcal{A}$ that make at most $\ell(\kappa)$ queries to oracles E, H, and at most $q(\kappa)$ queries to oracle D. When ℓ is an arbitrary polynomial, and only the number of decryption queries $q(\kappa)$ is restricted, then we say that a scheme is q-$\mathsf{SIM\text{-}CPA}^+$ secure.

Naturally, for proper (exact) encryption schemes, all these definitions are equivalent, and it is only in the approximate encryption setting that the definitions can be separated.

In the following proposition we show that there exist some scheme that is secure for up to some fixed number q of decryption queries but insecure for just $q + 1$ decryption queries. We remark that the encryption scheme described in the proof is presented for the sole purpose of separating the two definitions. More natural examples that separate $\mathsf{IND\text{-}CPA}$ and $\mathsf{IND\text{-}CPA}^+$ will be described in Section 3.3, where we present attacks to approximate encryption schemes from the literature.

Proposition 1. *Assume there exist a pseudorandom function and an $\mathsf{IND\text{-}CPA}$-secure exact homomorphic encryption scheme. Then, for any fixed $q \geq 2$, there exists a homomorphic approximate encryption scheme that is (q, ℓ)-$\mathsf{SIM\text{-}CPA}^+$-secure but not $(q + 1, \ell)$-$\mathsf{IND\text{-}CPA}^+$-secure.*

Proof. Let $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$ be an exact homomorphic encryption scheme that is $\mathsf{IND\text{-}CPA}$ secure. Let $\mathsf{PRF} : \{0, 1\}^\kappa \times X \to Y$ be a secure pseudorandom function, and without loss of generality, we can assume that $X = Y = \{0, 1\}^k$, where $k = |\mathsf{sk}|$ is the length of the secret key of $\mathcal{E}$. Let $0 < \beta < 1$ be some small positive number, which will be the upper bound on decryption errors, *i.e.*, the approximation upper bound in $\mathcal{E}'$. Let (π, π^{-1}) be an encoding scheme from $\{0, 1\}^k$ to $[0, \beta)$.

Build a homomorphic approximate encryption scheme $\mathcal{E}' = (\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Dec}', \mathsf{Eval}')$:

- The key generation algorithm $\mathsf{KeyGen}'(1^\kappa)$ samples $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and a PRF key $K \leftarrow \{0, 1\}^\kappa$. Then it outputs $(\mathsf{sk}', \mathsf{pk}, \mathsf{ek})$, where $\mathsf{sk}' = (\mathsf{sk}, K)$.

- The encryption algorithm Enc' and the evaluation algorithm Eval' are identical to Enc and Eval, respectively.

- The decryption algorithm $\mathsf{Dec}'_{(\mathsf{sk},K)}(c)$ first decrypts the ciphertext c to $m = \mathsf{Dec}_{\mathsf{sk}}(c)$. It outputs $m+\pi(\mathsf{PRF}_K(m \mod (q+1)))$ if $m \pmod{(q+1)} \not\equiv 0$, and it outputs $m+\pi(\mathsf{sk} \oplus r)$ for $r = \bigoplus_{i=1}^{q} \mathsf{PRF}_K(i)$ otherwise.

When an adversary $\mathcal{A}$ makes at most q decryption queries, the resulting decryption errors $\{\mathsf{Dec}'_{\mathsf{sk}'}(c_i) - m_i \mid c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_i)\}$ are computationally indistinguishable from random strings due to pseudorandomness of PRF. So one can show that the scheme $\mathcal{E}'$ is (q, ℓ)-SIM-CPA$^+$-secure using a reduction to the IND-CPA security of $\mathcal{E}$. However, if an adversary can make $q+1$ decryption queries, then it can completely recover the secret key sk using the decryption errors as secret shares of sk. So $\mathcal{E}'$ is not $(q+1, \ell)$-IND-CPA$^+$-secure. □

Restricting the query ordering. In the definition of IND-CPA$^+$ security, we did not state any restriction on the relative order of queries made by the adversary. In particular, queries can be made in many rounds, and a later query can depend on the responses from earlier queries. Such notion is called *security with adaptively chosen queries*, or simply *adaptive* security.

There are several other natural query orderings that can be imposed on the adversary, and enforced by an application. For example, it is often the case that input are encrypted and collected in advance, before any homomorphic evaluation or decryption operation takes place. As an extreme situation, one can consider a fully non-adaptive setting, where the adversary specifies all its queries in advance after seeing the public/evaluation key. We call this the *(fully) non-adaptive* model. Non-adaptive security is much easier to formulate, and we fully spell out its definition now.

Definition 5 (Non-Adaptive (q, ℓ)-IND-CPA$^+$ Security). *Let $\mathcal{E}$ be a homomorphic (possibly approximate) encryption scheme $\mathcal{E} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$. Let q and ℓ be two polynomial bounds in κ. We say that $\mathcal{E}$ is* non-adaptively (q, ℓ)-IND-CPA$^+$-secure *if for all efficient adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ consisting of two steps such that*

$$(\{m_0^{(i)}\}_{i=1}^{k}, \{m_1^{(i)}\}_{i=1}^{k}, \{(g_i, J_i)\}_{i=k+1}^{\ell}, \{j_i\}_{i=1}^{q}, \mathsf{st}) \leftarrow \mathcal{A}_0(1^\kappa, \mathsf{pk}, \mathsf{ek}),$$

where $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$, $m_0^{(i)} = g_i(m_0^{(J_i)})$, $m_1^{(i)} = g_i(m_1^{(J_i)})$ *for $i = k+1, \ldots, \ell$, and all g_i are valid circuits with indices $J_i \in \{1, \ldots, \ell\}^*$, the following two distributions are* *indistinguishable to* $\mathcal{A}_1(1^\kappa, \mathsf{st})$*:*

$$\{\ \{c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_0^{(i)})\}_{i=1}^{k},\ \{c_i \leftarrow \mathsf{Eval}_{\mathsf{ek}}(g_i, c_{(J_i)})\}_{i=k+1}^{\ell},\ \{\mathsf{Dec}_{\mathsf{sk}}(c_i) \mid m_0^{j_i} = m_1^{j_i}\}_{i=1}^{q}\ \},$$

and

$$\{\ \{c_i \leftarrow \mathsf{Enc}_{\mathsf{pk}}(m_1^{(i)})\}_{i=1}^{k},\ \{c_i \leftarrow \mathsf{Eval}_{\mathsf{ek}}(g_i, c_{(J_i)})\}_{i=k+1}^{\ell},\ \{\mathsf{Dec}_{\mathsf{sk}}(c_i) \mid m_0^{j_i} = m_1^{j_i}\}_{i=1}^{q}\ \},$$

where the probability is over the randomness of $\mathcal{A}$ *and in* Enc *and* Eval.

We can also define a non-adaptive variant of SIM-CPA$^+$ security in a way similar to Definition 5 such that all queries must be specified by the adversary at once, after seeing the public-key. Note that, the implication result of Lemma 2 applies to any query model, *i.e.*, if a scheme is SIM-CPA$^+$-secure in some query model, then it is IND-CPA$^+$-secure in the same query model.

Typically the same security notion is weaker in the non-adaptive model than in the adaptive model, as some attacks are only feasible in the latter model. We show that this is also the case for homomorphic approximate encryption schemes. As before, the encryption scheme described in the following proof is not intended to be used. It is just a theoretical construction, provided simply for the purpose of showing that a scheme may satisfy one definition but not the other.

Proposition 2. *Assume there exist an IND-CPA-secure exact homomorphic encryption scheme and a secure pseudorandom permutation. Then there exists a homomorphic approximate encryption scheme that is non-adaptively SIM-CPA$^+$-secure, but it is not adaptively* $(2, 2)$*-IND-CPA$^+$-secure.*

Proof. Let $\mathcal{E}$ be an IND-CPA secure exact HE scheme, and let $H : \{0,1\}^\kappa \times X \to X$ be a pseudorandom permutation for some set X that contains the secret key space of $\mathcal{E}$. We first define another pseudorandom permutation $F : \{0,1\}^\kappa \times X \to X$:

$$\forall x \in X.\ F_K(x) = H_K^{-1}(H_K(x) \oplus 1).$$

Notice that $F_K(F_K(x)) = x$ for all $x \in X$. Let (π, π^{-1}) be an encoding scheme from X to $[0, \beta)$ for some small $\beta < 1$.

We now build a homomorphic approximate encryption scheme $\mathcal{E}'$.

- $\mathsf{KeyGen}'(1^\kappa) = (\mathsf{sk}', \mathsf{pk}, \mathsf{ek})$: Sample $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathcal{E}.\mathsf{KeyGen}(1^\kappa)$, and also sample $K \leftarrow \{0,1\}^\kappa$ for the pseudorandom permutation F. Then set $\mathsf{sk}' = (\mathsf{sk}, K)$, and return $(\mathsf{sk}', \mathsf{pk}, \mathsf{ek})$.
- $\mathsf{Enc}'_{\mathsf{pk}}(\cdot)$ and $\mathsf{Eval}'_{\mathsf{ek}}(\cdot,\cdot)$ are exactly the same as $\mathcal{E}.\mathsf{Enc}_{\mathsf{pk}}$ and $\mathcal{E}.\mathsf{Eval}_{\mathsf{ek}}$.
- $\mathsf{Dec}'_{\mathsf{sk},K}(c) = m + \pi(r)$, where $m = \mathcal{E}.\mathsf{Dec}_{\mathsf{sk}}(c)$, $r = F_K(\mathsf{sk})$ if $m = 0$, and $r = F_K(m)$ otherwise.

One can check that F is a pseudorandom permutation against non-adaptive adversaries, *i.e.*, those adversaries who submit their queries all at once. So in the non-adaptive model, the real world approximate decryption result $m + \pi(r)$ obtained from decryption queries can be simulated knowing just m and the bound β, without using the secret key sk. Since $\mathcal{E}$ is $\mathsf{IND\text{-}CPA}$ secure, we see that $\mathcal{E}'$ is non-adaptively $\mathsf{SIM\text{-}CPA}^+$-secure.

But, the noises in decryption results are no longer pseudorandom in the adaptive model. In fact, an adaptive adversary $\mathcal{A}$ against $\mathsf{IND\text{-}CPA}^+$ security experiment can first query the encryption oracle on 0, and then ask to decrypt the corresponding ciphertext to get $e = \pi(F_K(\mathsf{sk}))$. Next, $\mathcal{A}$ asks to encrypt $\pi^{-1}(e) = F_K(\mathsf{sk})$ and then asks to decrypt its ciphertext. At this point $\mathcal{A}$ gets the decryption result $\pi^{-1}(e) + \pi(\mathsf{sk})$, and $\mathcal{A}$ can fully recover sk. So $\mathcal{E}'$ is not adaptively $(2,2)$-$\mathsf{IND\text{-}CPA}^+$-secure. $\square$

3.2.4 Extensions to Circuit Privacy and Functional Decryption Queries

In this section we consider extensions of our definitions that may be interesting in applications. The extensions capture settings where the honest users want to hide the computation performed on the ciphertexts, or can enforce the secure postprocessing of decrypted messages, before any information about them is provided to the intended application. Circuit privacy has already been considered in the standard (exact) FHE setting. Postprocessing with functional decryption queries is a new issue, specific to the setting of approximate encryption schemes. We only provide definitions here, leaving further study of these notions to future work.

Circuit privacy. Homomorphic encryption schemes that hide the computaton performed on the encrypted inputs are called *circuit private*, because the computation is often represented as a circuit. Most homomorphic encryption schemes (in their basic form) are not circuit private. Accordingly, we did not include any circuit privacy requirement to any of our definitions. However, all definitions are easily extended to achieve that property as follows.

For $\mathsf{SIM\text{-}CPA}^+$ security, one changes the information that the oracle H gives to $\mathcal{S}$ when replying to evaluation queries: instead of sending $(\mathtt{H}, g, J)$ to the simulator, H sends $\mathtt{E}$ to $\mathcal{S}$. This informs the simulator to produce something, without knowing the computation (g, J), that should be indistinguishable from the ciphertext produced by homomorphic evaluation $\mathsf{Eval}_{\mathsf{ek}}(g, T[j_1].c, \dots, T[j_k].c)$ in the real world experiment, where $J = \{j_1, \dots, j_k\}$.

For $\mathsf{IND\text{-}CPA}^+$ security, homomorphic evaluation queries specify not one g, but two circuits g_0, g_1, possibly with different index sets J_0, J_1.[1] Then the state S is extended with a tuple (m_0, m_1, c) where $m_0 = g_0(S[j_{0,1}].m_0, \dots, S[j_{0,k_0}].m_0)$, $m_1 = g_1(S[j_{1,1}].m_1, \dots, S[j_{1,k_1}].m_1)$, $c \leftarrow \mathsf{Eval}_{\mathsf{ek}}(g_b, S[j_{b,1}].m_b, \dots, S[j_{b,k_b}].m_b)$, $J_0 = \{j_{0,1}, \dots, j_{0,k_0}\}$, $J_1 = \{j_{1,1}, \dots, j_{1,k_1}\}$, and the ciphertext c is returned to the adversary.

[1]One may consider weaker definitions, with only one J, which reveal the "topology" of the circuit, but not the value of its "gates".

Functional Decryption Extension. We also consider the possiblity that the homomorphic encryption scheme is used in a controlled environment where the result of decryption (of a homomorphic computation) is securely post-processed using some function f, which reveals only some information about the (approximate) result of the homomorphic computation. This function f should be thought of as part of the decryption algorithm (or library implementing the approximate homomorphic encryption scheme), as it is essential for security that the adversay does not get to see the result of decryption m, but only $f(m)$. By restricting the choice of f to some class of allowed functions $\mathcal{L}$, one can limit the amount of information that the adversary can extract from the output of a computation.

Formally, we can extend our definitions of (q, ℓ)-$\{\mathsf{IND\text{-}CPA}^+, \mathsf{SIM\text{-}CPA}^+\}$ to include another parameter, an class $\mathcal{L}$ of efficiently computable *post-processing* functions, and we expand the decryption queries to include a function $f \in \mathcal{L}$. We call such queries the *functional decryption queries*, with the following specification:

- In $\mathsf{IND\text{-}CPA}^+$ definition, the decryption oracle D accepts queries of the form (j, f), where $j \le |S|$ is an index and $f \in \mathcal{L}$ is a post-processing function. If $f(S[j].m_0) = f(S[j].m_1)$, then the oracle D returns $f(\mathsf{Dec}_{\mathsf{sk}}(S[j].c))$. Otherwise, the D returns the error symbol $\perp$.
- In $\mathsf{SIM\text{-}CPA}^+$ definition, functional decryption queries have the form $\mathsf{D}(j, f)$. For each functional decryption query, the simulator is given $(\mathtt{D}, j, f, f(T[j]))$.

These extended security definitions are called $(\mathcal{L}, q, \ell)$-$\{\mathsf{IND\text{-}CPA}^+, \mathsf{SIM\text{-}CPA}^+\}$)-security. Our earlier definitions are the special case where $\mathcal{L} = \{id\}$ only contains the identity function $id(x) = x$, and the adversary can see the full result of decryption.

For $\mathsf{IND\text{-}CPA}^+$ security, the requirement that $f(S[j].m_0) = f(S[j].m_1)$ is to eliminate trivial attacks where the adversary can distinguish between two computations that lead to different results when computed on exact values. For $\mathsf{SIM\text{-}CPA}^+$ security, we let the simulator see the post-processing function and the post-processed result derived from the *exact* values, which captures the maximal information could be gained by an attacker if the scheme were to be secure.

When considering circuit privacy for IND-CPA$^+$-security, we may also expand functional decryption queries by replacing f with a pair of possibly different post-processing functions f_0, f_1. Still, we require that $f_0(S[j].m_0) = f_1(S[j].m_1)$ to eliminate the trivial attack.

3.3 Case Study: The CKKS Homomorphic Encryption Scheme on Approximate Numbers

In this section we examine the CKKS homomorphic approximate encryption scheme [22]. We introduce relevant background and briefly present the definition of the CKKS scheme, and then we show a key-recovery attack in the IND-CPA$^+$ model. We will also discuss potential countermeasures to such attack.

3.3.1 Lattice Cryptography Background

We first present necessary background in lattice cryptography and the CKKS scheme.

Lattices and Rings. A lattice is a (typically full rank) discrete subgroup of $\mathbb{R}^n$. Lattices $L \subset \mathbb{R}^n$ can be represented by a basis, i.e., a matrix $\mathbf{B} \in \mathbb{R}^{n \times k}$ with linearly independent columns such that $L = \mathbf{B}\mathbb{Z}^k$. The length of the shortest nonzero vector in a lattice L is denoted by $\lambda(L)$. The Shortest Vector Problem, given a lattice L, asks to find a lattice vector of length $\lambda(L)$. The Approximate SVP relaxes this condition to finding a nonzero lattice vector of length at most $\gamma \cdot \lambda(L)$, where the approximation factor $\gamma \geq 1$ may be a function of the dimension n or other lattice parameters.

We write $\mathbb{Z}, \mathbb{Q}, \mathbb{R}, \mathbb{C}$ for the sets of integer, rational, real and complex numbers. For any positive $q > 0$, we write $\mathbb{R}_q = \mathbb{R}/(q\mathbb{Z})$ for the set of reals modulo q (as a quotient of additive groups), uniquely represented as values in the centered interval $[-q/2, q/2)$. Similarly, for any positive integer $q > 0$, we write $\mathbb{Z}_q = \mathbb{Z}/(q\mathbb{Z})$ for the ring of integers modulo q, uniquely represented as values in $[-q/2, q/2) \cap \mathbb{Z} = \left\{-\left\lceil \frac{q-1}{2} \right\rceil, \ldots, \left\lfloor \frac{q-1}{2} \right\rfloor\right\}$.

Let $N = 2^k$ be a power of 2, $\zeta_{2N} = e^{\pi\imath/N}$ the principal $(2N)$th complex root of unity. We write $\mathcal{K}^{(2N)} = \mathbb{Q}[X]/(X^N+1)$ for the cyclotomic field of order $2N$, and $\mathcal{O}^{(2N)} = \mathbb{Z}[X]/(X^N+1)$ for its ring of integers. The primitive roots of unity ζ_{2N}^{2j+1}, for $j = 0, \ldots, N-1$, are precisely the roots of the cyclotomic polynomial $X^N + 1$. We omit the index $2N$ and simply write $\mathcal{K}, \mathcal{O}$ and ζ when the value of N is clear from the context. Elements of $\mathcal{K}$ (and $\mathcal{O}$) are uniquely represented as polynomials $a(X) = a_0 + a_1 \cdot X + \ldots + a_{N-1} \cdot X^{N-1}$ of degree less than N, and identified with their vectors of coefficients $\mathbf{a} = (a_0, \ldots, a_{N-1}) \in \mathbb{Q}^N$ (and $\mathbb{Z}^N$). For any positive integer $q > 0$, we write $\mathcal{K}_q = \mathcal{K}/(q\mathcal{K}) \equiv \mathbb{Q}_q^N$ for the set of vectors/polynomials with entries/coefficients reduced modulo q. Similarly for $\mathcal{O} \equiv \mathbb{Z}^N$ and $\mathcal{O}_q \equiv \mathbb{Z}_q^N$.

LWE and Homomorphic Encryption. The *(Ring) Learning With Errors (LWE)* distribution $\mathsf{RLWE}_s(N, q, \chi)$ with *secret* $s \in \mathcal{O}^{(2N)}$ and *error distribution* χ (over $\mathcal{O}^{(2N)}$), produces pairs $(a, b) \in \mathcal{O}_q^{(2N)}$ where $a \leftarrow \mathcal{O}_q^{(2N)}$ is chosen uniformly at random, and $b = s \cdot a + e$ for $e \leftarrow \chi$. The *(decisional) Ring LWE* assumption over $\mathcal{O}^{(2N)}$ with error distribution χ and secret distribution χ' and m samples, states that when $s \leftarrow \chi'$, the product distribution $\mathsf{RLWE}_s(N, q, \chi)^m$ is pseudorandom, i.e., it is computationally indistinguishable from the uniform distribution over $(\mathcal{O}_q \times \mathcal{O}_q)^m$.

For appropriate choices of χ, χ' and q, the Ring LWE problem is known to be computationally hard, based on (by now) standard assumptions on the worst-case complexity of computing approximately shortest vectors in ideal lattices. Theoretical work supports setting the error distribution χ to a discrete gaussian of standard deviation $O(\sqrt{N})$, and setting the secret distribution χ' to either the uniform distribution over $\mathcal{O}_q$, or the same distribution as the errors χ. For the sake of efficiency, the Ring LWE problem is often employed by homomorphic encryption schemes also for narrower secret and error distributions, that lack the same theoretical justifications, but for which no efficient attack is known, e.g., distributions over vectors with binary $\{0, 1\}$ or ternary $\{-1, 0, 1\}$ coefficients.

The *raw* (Ring) LWE encryption scheme works as follows:

- The key generation algorithm picks $s \leftarrow \chi'$, $e \leftarrow \chi$, $a \leftarrow \mathcal{O}_q$, and outputs secret key

$\mathsf{sk} = (-s, 1) \in \mathcal{O}_q^2$ and public key $\mathsf{pk} = (a, b) \in \mathcal{O}_q^2$ where $b = s \cdot a + e$ follows the LWE distribution.

- The encryption algorithm, $\mathsf{Enc}_{\mathsf{pk}}(m)$ picks random $u \leftarrow \{0, 1\}^N$ and $\mathbf{e} = (e_0, e_1) \leftarrow \chi^2$, and outputs $\mathsf{ct} = u \cdot \mathsf{pk} + \mathbf{e} + (0, m) \in \mathcal{O}_q^2$
- The *raw* decryption algorithm $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$ outputs $\langle \mathsf{sk}, \mathsf{ct} \rangle \bmod q$.

The secret and public keys satisfy the property that $\langle \mathsf{sk}, \mathsf{pk} \rangle = e$ equals the short error vector chosen during key generation. We qualified this scheme and the decryption algorithm as "raw" because applying the encryption algorithm, and subsequently decrypting the result (with a matching pair of public and secret keys) does not recover the original message, but only a value close to it. In fact, for any $(\mathsf{sk}, \mathsf{pk})$ produced by the key generation algorithm, we have

$$\mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m)) = u \cdot \langle \mathsf{sk}, \mathsf{pk} \rangle + \langle \mathsf{sk}, \mathbf{e} \rangle + m = m + (ue - se_0 + e_1) \pmod{q}$$

where the perturbation $\tilde{e} = (ue - se_0 + e_1)$ is small because it is a combination of short vectors u, e, s, e_0, e_1. (The size of these vectors is best quantified with respect to the message encoding used by the application, and it is discussed below.) In order to obtain a proper encryption scheme that meets the correctness requirement, the message m must be preprocessed, by encoding it with an appropriate error correcting code, which allows to recover from the error $\tilde{e}$. For example, if m has binary entries, one can multiply m by a scaling factor $\lfloor q/2 \rceil$, and then round (each coefficient of) the output of the raw decryption algorithm to the closest multiple of $\lfloor q/2 \rceil$. For the sake of improving the efficiency of homomorphic computations, the CKKS encryption scheme [22] gets away without applying error correction, and directly using the raw decryption algorithm to produce "approximate" decryptions of the ciphertexts. So, in the following we focus on the "raw" LWE scheme, and postpone the discussion of error correction to later.

By linearity of Enc, LWE encryption directly supports (bounded) addition of ciphertexts: if $\mathsf{ct}_0 = (a_0, b_0)$ and $\mathsf{ct}_1 = (a_1, b_1)$ are encryptions of m_0 and m_1 with noise e_0 and e_1 respectively, then the vector sum

$$\mathsf{ct}_0 + \mathsf{ct}_1 = (a_0 + a_1, b_0 + b_1) \bmod q$$

is an encryption of $m_0 + m_1$ with noise $e_0 + e_1$.

There are several ways to perform homomorphic multiplication on LWE ciphertexts. As in [22], here we focus on the "tensoring" technique of [12] implemented using the "raising the modulus" multiplication method of [32]. This multiplication method uses an appropriate multiple pq of the ciphertext modulus q, and requires an "evaluation key", produced during key generation, which is computed and used as follows:

- $\mathsf{ek} = (a, b) \in \mathcal{O}_{pq}^2$ where $a \leftarrow \mathcal{O}_{pq}$, $e \leftarrow \chi_e$ and $b = as + e + ps^2 \pmod{pq}$.
- Using ek, the product of two ciphertexts $\mathsf{ct}_0 = (a_0, b_0), \mathsf{ct}_1 = (a_1, b_1)$ is computed as

$$\mathsf{ct}_0 \times \mathsf{ct}_1 = (a_0 b_1 + a_1 b_0, b_0 b_1) + \lfloor (a_0 a_1 \bmod q) \cdot \mathsf{ek}/p \rceil .$$

In order to approximately evaluate deep arithmetic circuits, the CKKS scheme combines these addition and multiplication procedures with a *rescaling operation* RS, implemented using the *key switching* technique of [12]. Rescaling requires the use of a sequence of moduli q_l, which for simplicity we assume to be of the form $q_l = q_0 \cdot p^l$ for some base p, e.g., $p = 2$. Ciphertexts may live at different levels, with level l ciphertexts encrypted using modulus q_l. The key generation algorithm takes as auxiliary input the highest number of desired levels L, and produces public and evaluation keys with respect to the largest modulus q_L. CKKS directly supports addition and multiplication only between ciphertexts at the same level. Rescaling is used to map ciphertexts $\mathsf{ct} \in \mathcal{O}_{q_{l+l'}}^2$ to a lower level l with the operation

$$\mathsf{RS}_{l'}(\mathsf{ct}) = \left\lfloor \mathsf{ct}/p^{l'} \right\rceil \in \mathcal{O}_{q_l}^2$$

where the division and rounding are performed componentwise.

The CKKS message encoding. The CKKS scheme considers a vectors of complex numbers (or Gaussian integers) $\tilde{\mathbf{a}}$ as the set of evaluation points $\tilde{a}_j = a(x_j)$ of a real (in fact, integer) polynomial $a(X) \in \mathbb{Z}[X]$. This allows to perform pointwise addition and multiplication of vectors (SIMD style) by means of addition and multiplication of polynomials as $(a(X) \circ b(X))(x_j) = a(x_j) \circ b(x_j)$ for any x_j, where $\circ \in \{+, \times\}$. The evaluation points are chosen among the primitive $(2N)$th roots of unity ζ^{2j+1}, so that the cyclotomic polynomial $X^N + 1$ evaluates to zero at all those points, and reduction modulo $X^N + 1$ does not affect the value of $a(x_j)$. This allows to operate on the polynomials modulo $X^N + 1$, i.e., as elements of the cyclotomic ring $\mathcal{O}$. Since $a(X)$ has real coefficients and primitive roots come in complex conjugate pairs $\zeta^{2j+1}, \zeta^{2(N-j)-1}$, the value of $a(X)$ can be freely chosen only for half of the roots, with the value of $a(\zeta^{2(N-j)-1})$ uniquely determined as the complex conjugate of $a(\zeta^{2j+1})$. So, $a(X)$ is used to represent a vector $\tilde{\mathbf{a}}$ of $N/2$ complex values. Setting the evaluation points to $x_j = \zeta^{4j+1}$ (for $j = 0, \ldots, N/2 - 1$), and using the fact that these points are primitive roots of unity, interpolation and evaluation can be efficiently computed (in $O(N \log N)$ time) using the Fast Fourier Transform.

Let $\varphi \colon \mathcal{O} \to \mathbb{C}^{N/2}$ be the transformation mapping $a(X) \in \mathcal{O} \equiv \mathbb{Z}^N$ to $\varphi(a) = \tilde{\mathbf{a}} = (a(\zeta^{4j+1}))_{j=0}^{N/2-1} \in \mathbb{C}^{N/2}$, and its extension $\varphi \colon \mathcal{S} \to \mathbb{C}^{N/2}$ to arbitrary real polynomials, where $\mathcal{S} = \mathbb{R}[X]/(X^N + 1) \equiv \mathbb{R}^N$. We can identify any polynomial $a \in \mathcal{S}$ by its coefficient vector $(a_0, a_1, \ldots, a_{N-1})$, and we set $\|a\|_2 = \|(a_0, a_1, \ldots, a_{N-1})\|_2$. Similarly we can define $\|a\|_1$ and $\|a\|_\infty$ as the corresponding norms on the coefficient vector. So the transformation $\varphi \colon \mathcal{S} \to \mathbb{C}^{N/2}$ is a scaled isometry, satisfying $\|\varphi(a)\|_2 = \sqrt{N}\|a\|_2$ and $\|\varphi(a)\|_\infty \leq \|a\|_1$. In what follows, we assume, as a message space, the set of complex vectors $\tilde{\mathbf{a}} \in \varphi(\mathcal{O}) \subset \mathbb{C}^{N/2}$ which are the evaluation of polynomials $a(X) \in \mathcal{O}$ with integer coefficients much smaller than the ciphertext modulus q. Arbitrary vectors $\mathbf{z} \in \mathbb{C}^{N/2}$ can be encrypted (approximately) by taking the inverse transform φ^{-1} on a scaled vector $\Delta \cdot \mathbf{z}$, for some scaling factor $\Delta \in \mathbb{R}$, such that $\|\varphi^{-1}(\Delta \cdot \mathbf{z})\| \ll q$ and rounding $\varphi^{-1}(\Delta \cdot \mathbf{z})$ to a nearby point of the form $\varphi(a)$ for some $a(X) \in \mathcal{O}$.

The complete message encoding and decoding functions in CKKS are defined as

- $\mathsf{Encode}(\mathbf{z} \in \mathbb{C}^{N/2}; \Delta) = \lfloor \Delta \cdot \varphi^{-1}(\mathbf{z}) \rceil \in \mathcal{O}$.
- $\mathsf{Decode}(a \in \mathcal{O}; \Delta) = \varphi(\Delta^{-1} \cdot a) \in \mathbb{C}^{N/2}$.

Once encoded, the scaling factor Δ is usually implicitly tied to a plaintext polynomial, so we sometimes omit it when its value is clear from the context.

Since these encoding and decoding operations can be performed without any knowledge of the secret or public keys, sometimes we assume they are performed at the outset, at the application level, before invoking the encryption or decryption algorithms. More specifically, we may assume messages $\varphi(\Delta^{-1} \cdot m) \in \mathbb{C}^{N/2}$ are provided to the encryption algorithm by specifying the integer polynomial $m \in \mathcal{O}$, and the decryption algorithm returns a message $\tilde{\mathbf{m}}' = \mathsf{Decode}(m'; \Delta)$ represented as the underlying polynomial $m' \in \mathcal{O}$ that is an approximation of m. All this is only for the sake of theoretical analysis, and all concrete implementations (of the scheme and our attacks to it) include encoding and decoding procedures as part of the encryption and decryption algorithms. Message encoding can be quite relevant to quantify the amount of noise in a ciphertext. We say that a ciphertext ct *approximately encrypts* message $\tilde{\mathbf{m}}$ with scaling factor Δ and noise $\tilde{\mathbf{e}}$ if $\mathsf{Decode}(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}); \Delta) = \tilde{\mathbf{m}} + \tilde{\mathbf{e}}$.

3.3.2 Key-Recovery Attack to CKKS in $\mathsf{IND\text{-}CPA}^+$ Model

In this section we describe a key recovery attack against the CKKS scheme, including both theoretical and practical analysis. Based on such attack, we can conclude that the CKKS scheme is not $\mathsf{IND\text{-}CPA}^+$ secure.

3.3.3 Theoretical Outline

The technical idea behind the attack is easily explained by exemplifying it on a symmetric key version of LWE encryption. (Breaking the CKKS scheme involves additional complications due to the details of the encoding/decoding functions discussed below.) We recall that in a passive attack (against a symmetric key encryption scheme $E_{\mathbf{s}}(m)$), the adversary can observe the encryption $E_{\mathbf{s}}(m)$ of any message m of its choice. In LWE encryption, the key is a random vector $\mathbf{s} \in \mathbb{Z}_q^n$, and a (possibly encoded) message $m \in \mathbb{Z}_q$ is encrypted as $E_{\mathbf{s}}(m) = (\mathbf{a}, b)$ where $\mathbf{a} \in \mathbb{Z}_q^n$ is chosen at random, and $b = \langle \mathbf{s}, \mathbf{a} \rangle + m + e \pmod q$ for a small random integer perturbation $e \in \mathbb{Z}$. If the encryption scheme works on "approximate numbers", $(m + e)$ is treated as an approximation of m, and the decryption algorithm outputs $D_{\mathbf{s}}(\mathbf{a}, b) = b - \langle \mathbf{s}, \mathbf{a} \rangle = m + e$.

Our most basic attack involves an adversary that asks for an encryption of $m = 0$, so to obtain a ciphertext $\mathsf{ct} = (\mathbf{a}, b)$ where $b = \langle \mathbf{s}, \mathbf{a} \rangle + e \pmod q$. The adversary then asks to compute the identity function $id(x) = x$ on it. (This is the same as performing no computation at all.) Finally, it asks for an approximate decryption of the result, and computes

$$c = b - \mathsf{Dec}_{\mathbf{s}}(\mathsf{ct}) = (\langle \mathbf{s}, \mathbf{a} \rangle + e) - (m + e) = \langle \mathbf{s}, \mathbf{a} \rangle \pmod q. \tag{3.1}$$

This provides a linear equation $\langle \mathbf{s}, \mathbf{a} \rangle = c \pmod q$ in the secret key. Collecting n such linear equations and solving the resulting system (e.g., by gaussian elimination) recovers the secret key $\mathbf{s}$.

It is easy to see that there is nothing special about the message 0, or the fact that no computation is performed: as long as the adversary knows the ciphertext ct (possibly the result of a homomorphic computation) and gets to see the approximate decryption of ct, the same attack goes through. However, the actual scheme described in [22] and subsequent papers, and their open source implementations include several modifications of the above scheme, introduced to make the scheme more useful in practice, but which also make the attack less straightforward. We briefly describe each of these modifications, and how the attack is adapted. In the most general case, our attack requires not just the solution of a linear system of equations, but the use of lattice reduction for the (polynomial time solution) of a lattice approximation problem.

Public Key. First, CKKS is a public key encryption scheme, where, as standard in lattice based encryption, the public key can be seen as a collection of encryptions of 0 values.

This makes no difference in the attack, as the ciphertexts still have the same structure with respect to the secret key, and the (approximate) decryption algorithm is unmodified. Switching to a public key system has the only effect of producing larger noise vectors e in ciphertexts.

Ring Lattices. In order to achieve practical performance, all instantiations of the CKKS scheme make use of cyclic/ideal lattices [55] and the Ring LWE problem [52,53]. Specifically, the vectors $\mathbf{a}, \mathbf{s}$ are interpreted as (coefficients of) polynomials a, s in the power-of-two cyclotomic rings $\mathcal{O}^{(2N)}$ popularized by the SWIFFT hash function [50, 51, 62] and widely used in the implementation of lattice cryptography since then. In a sense, switching to ideal lattices makes the attack only more efficient: the linear equation $\langle \mathbf{s}, \mathbf{a} \rangle = \mathbf{c} \pmod q$ becomes an equation $a \cdot s = c \in \mathcal{O}_q^{(2N)}$ in the cyclotomic ring modulo q, which can be solved (even using a single ciphertext) by computing the (ring) inverse of a, and recovering s as

$$s' = a^{-1} \cdot c \in \mathcal{O}_q. \tag{3.2}$$

A little difficulty arises due to the choice of q. The first implementation of CKKS, the HEAAN library [37] sets q to a power of 2 to simplify the treatment of floating point numbers. Subsequent instantiations of CKKS also use a prime (or square-free) q of the form $h \cdot 2^n + 1$ together with the Number Theoretic Transform for very fast ring operations [51]. For a (sufficiently large) prime q, the probability of a random element a being invertible is very close to 1, but this is not the case when q is a power of two. If a is not invertible, we can still recover partial information about the secret key s, and completely recover s by using multiple ciphertexts.

Euclidean Embedding. In order to conveniently apply the CKKS scheme on practical problems, the input message space is set to $\mathbb{C}^{N/2}$ for some N that is a power of 2, the set of vectors with complex entries, or, more precisely, their floating point approximations. A message $\mathbf{z} \in \mathbb{C}^k$, for some integer $1 \leq k \leq N/2$, can be considered as a vector in $\mathbb{C}^{N/2}$ (by padding it with 0 entries), and it is then encoded to

$$m = \mathsf{Encode}(\mathbf{z}; \Delta) = \left\lfloor \Delta \cdot \varphi^{-1}(\mathbf{z}) \right\rceil \in \mathbb{Z}^N \equiv \mathcal{O},$$

where Δ is some precision factor. The "decode" operation $\mathsf{Decode} : \mathcal{O} \to \mathbb{C}^k$ sends an integer polynomial m to

$$\mathsf{Decode}(m; \Delta) = \varphi(\Delta^{-1} \cdot m) \in \mathbb{C}^k,$$

where the entries corresponding to the 0-paddings are dropped. Decode is an approximate inverse of Encode as $\mathbf{z}' = \mathsf{Decode}(\mathsf{Encode}(\mathbf{z}; \Delta); \Delta)$ is close (but not exactly equal) to $\mathbf{z}$.

This is slightly more problematic for our attack, because a passive adversary only gets to see the result of final decryption $\mathbf{z}' \in \mathbb{C}^k$, rather than the ring element $m' = a \cdot s + b \in \mathcal{O}$ that is required by our attack, in addition to the ciphertext $\mathsf{ct} = (a, b)$. Moreover, given the approximate nature of the encoding/decoding process, $\mathsf{Decode}(m')$ is not even the exact (mathematical) transformation $\varphi(\Delta^{-1} \cdot m')$, but only the result of an approximate floating point computation. We address this by setting $k = N/2$ (so, at least the vector $\mathsf{Decode}(m')$ has the right dimension over $\mathbb{C}$), and re-encoding the message output by the decryption algorithm to obtain $\mathsf{Encode}(\mathsf{Decode}(m'))$.

At this point, depending on the concrete choice of parameters of the scheme, we may have $\mathsf{Encode}(\mathsf{Decode}(m')) = m'$, in which case we can carry out the above attack by setting up a system of linear equations or computing inverses in the cyclotomic ring. We summarize this case in the following theorem.

Theorem 1 (Linear Key-Recovery Attack against CKKS)**.** *Fix a particular instantiation of the CKKS scheme under the Ring-LWE assumption of dimension N and modulus q, and fix a key tuple* $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*. Given $k = O(N)$ ciphertext ct_i for $1 \leq i \leq k$, that are the either encryptions under* pk *or homomorphic evaluations under* ek*, and given their approximate decryption results $\mathbf{z}'_i = \mathsf{Decode}(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_i); \Delta)$ with a scaling factor Δ, if $\mathsf{Encode}(\mathbf{z}'_i; \Delta) = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_i)$ for* *all* $1 \leq i \leq k$*, then we can efficiently recover the secret key* sk *with high probability.*

Moreover, if the ciphertext modulus q *is a prime or a product of distinct primes, then the above holds for all* $k \geq 1$.

3.3.4 Analysis of Encoding/Decoding Errors

To see for what concrete parameters the linear attack can be applied, we take a closer look at the error introduced by the encoding and decoding computation. In practice, since N is a power of 2, the classical Cooley-Tukey FFT algorithm is used to implement the transformation φ and its inverse φ^{-1}, and the computation is done using floating-point arithmetic that could cause round-off errors.

Fix a ciphertext ct, and let $m' = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}) \in \mathcal{O}$ be its approximate decryption (before decoding) with a scaling factor Δ. Let $\hat{\mathbf{z}}' = \mathsf{Decode}(m'; \Delta)$ be the computed value of $\mathbf{z}' = \varphi(\Delta^{-1} \cdot m')$. To carry out the attack, we compute the encoding of $\hat{\mathbf{z}}'$ with the scaling factor Δ: first we apply inverse FFT to compute $\mathbf{u} = \Delta \cdot \varphi^{-1}(\hat{\mathbf{z}}')$, and then we round its computed value $\hat{\mathbf{u}}$ to $m'' = \lfloor \hat{\mathbf{u}} \rceil \in \mathcal{O}$. Let $\varepsilon = \hat{\mathbf{u}} - \mathbf{m}'$ be the *encoding error*, where $\mathbf{m}'$ is the coefficient vector of m'. We see that $\mathsf{Encode}(\mathsf{Decode}(m'; \Delta); \Delta) = m'$ if and only if $\|\varepsilon\|_\infty = \|\hat{\mathbf{u}} - \mathbf{m}'\|_\infty < \frac{1}{2}$.

Assume the relative error in computing the Cooley-Tukey FFT in dimension N is at most μ in l_2 norm. Then $\|\hat{\mathbf{z}}' - \mathbf{z}'\|_2 \leq \mu \cdot \frac{\sqrt{N}}{\Delta}\|\mathbf{m}'\|_2$, $\|\hat{\mathbf{u}} - \mathbf{u}\|_2 \leq \mu(1+\mu) \cdot \|\mathbf{m}'\|_2$, and $\|\mathbf{u} - \mathbf{m}'\|_2 \leq \mu \cdot \|\mathbf{m}'\|_2$. It follows that

$$\|\varepsilon\|_\infty = \|\hat{\mathbf{u}} - \mathbf{m}'\|_\infty \leq \|\hat{\mathbf{u}} - \mathbf{m}'\|_2 \leq (2\mu + \mu^2)\|\mathbf{m}'\|_2.$$

In [15], Brisebarre et. al. presented tight bounds on the relative error μ in applying the Cooley-Tukey FFT algorithm on IEEE-754 floating-point numbers. According to their estimate, $\mu \approx 53 \cdot 2^{-53}$ for $N = 2^{16}$ and double-precision floating-point numbers. So, in such setting, we expect to see $\mathsf{Encode}(\mathsf{Decode}(m'; \Delta); \Delta) \neq m'$, i.e., $\|\varepsilon\|_\infty > \frac{1}{2}$, when $\|m'\|_2 > 2^{45}$. (As we will see in the next section, our experimental results using existing CKKS implementations suggest this is a very conservative estimation.) The rescaling operation can be used to reduce the size of the approximate plaintext m', which is already used to maximize the capacity of homomorphic computation in CKKS.

Lattice attack. In case $\mathsf{Encode}(\mathsf{Decode}(m')) \approx m'$ is only an approximation of what we want for the linear key recovery attack, it is still possible to recover sk by solving a (polynomial time) lattice approximation problem.

Theorem 2 (Lattice Attack against CKKS)**.** *Fix a particular instantiation of the CKKS scheme under the Ring-LWE assumption of dimension N and modulus q, and fix a key tuple* $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$*. Given a ciphertext* $\mathsf{ct} \in \mathcal{O}_q^2$ *with a scaling factor Δ, and given an approximate decryption* $\mathbf{z}' = \mathsf{Decode}(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}); \Delta)$ *of* ct*, if the encoding error* $\varepsilon = \Delta \cdot \varphi^{-1}(\mathbf{z}') - \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$ *satisfies $\|\varepsilon\|_2 \le 2^{-\frac{N}{2}} \cdot q\sqrt{N}$, then the secret key* sk *can be efficiently recovered.*

Proof (sketch). Let $\mathsf{ct} = (a, b)$ for some $a, b \in \mathcal{O}_q$. We consider to the following approximate CVP instance. Let $A = \phi(a) \in \mathbb{Z}^{N \times N}$ be the negacyclic matrix representation of a. Consider the following matrix

$$B = \begin{pmatrix} A & qI_N \\ \mathbf{1}^t & \mathbf{0}^t \end{pmatrix} \in \mathbb{Z}^{(N+1)\times(2N)},$$

where $\mathbf{1}^t = [1, \ldots, 1]$ is a N-dimensional row vector of all 1 entries. Let $\mathcal{L} = \mathcal{L}(B)$ be the integer lattice generated by B, let $\mathbf{u} = \Delta \cdot \varphi^{-1}(\mathbf{z}') \in \mathbb{R}^N$, and let $\mathbf{t} = (\mathbf{u} - \mathbf{b}, 0)^t \in \mathbb{Z}^{N+1}$, where $\mathbf{b}$ is the coefficient vector of b. Our CVP instance asks to find $\mathbf{v} \in \mathcal{L}$ such that $\|\mathbf{v} - \mathbf{t}\|_2 \le \delta$ for some $\delta > 0$.

To set the parameter δ, notice that $\mathbf{v}_0 = (m' - b, \langle \mathbf{1}, \mathbf{s} \rangle)$ is a lattice point, and $\|\mathbf{v}_0 - \mathbf{t}\|_2^2 = \|\mathbf{u} - \mathbf{m}'\|_2^2 \le \|\varepsilon\|_2^2 + \langle \mathbf{1}, \mathbf{s} \rangle^2$. On the other hand, if $m'' - b = A\mathbf{r} + q\mathbf{w}$ for some $\mathbf{r}, \mathbf{w} \in \mathbb{Z}^N$, then $\mathbf{v}_1 = (m'' - b, \langle \mathbf{1}, \mathbf{r} \rangle) \in \mathcal{L}$ is also a lattice point. The distance from $\mathbf{v}_1$ to $\mathbf{t}$ is $|\langle \mathbf{1}, \mathbf{r} \rangle|$. Note that $\mathbf{r} = A^{-1}(m' - b) + A^{-1} \lfloor \varepsilon \rceil \pmod q = \mathbf{s} + A^{-1} \lfloor \varepsilon \rceil \pmod q$. So $|\langle \mathbf{1}, \mathbf{r} \rangle| \ge |\langle \mathbf{1}, \mathbf{s} \rangle| + |\left\langle \mathbf{1}, A^{-1}\varepsilon \right\rangle|$. We can assume that $\lfloor \varepsilon \rceil$ is independent of $m' - b$, so $A^{-1} \lfloor \varepsilon \rceil$

(mod q) is close to uniform, and so it holds with high probability that $|\left\langle \mathbf{1}, A^{-1}\varepsilon \right\rangle| \leq 2\sqrt{3}\cdot q\sqrt{N}$. When $\|\varepsilon\|_2 \leq 2^{-\frac{N}{2}} \cdot q\sqrt{N}$, we can set $\delta = 2\sqrt{3} \cdot q\sqrt{N}$ and obtain $m' - b$ with high probability by solving such CVP instance in polynomial time. Then, we can mount the linear attack as in Theorem 1. □

3.3.5 Experimental Results

The basic idea of our linear attack is so simple that it requires no validation. However, as described in the previous section, a concrete instantiation of the CKKS scheme may include a number of details that make the attack more difficult in practice. Given the simplicity of our attack, we also considered the possibility that the implementations of CKKS may not correspond too closely to the theoretical scheme described in the papers, and included some additional countermeasures to defend against the attack.

To put our linear attack to a definitive test, we implemented it against publicly available libraries HEAAN [37], PALISADE [60], SEAL [66], and HElib [38] that implement the CKKS scheme, and we ran our attack over some homomorphic computations that are commonly used in real world privacy-preserving machine-learning applications. Our experimental results against the libraries are summarized in Tables 3.1 and 3.2. For most of the parameter settings, our attack can successfully and quite efficiently recover the secret key, showing it is widely applicable to these CKKS implementations. In the following, we discuss our experiment and the relevant implementation details of these libraries, and we briefly analyze the results. We also consider RNS-HEAAN [64], an alternative implementation similar to HEAAN that includes RNS (residue number system) optimizations, obtaining similar results.

We did not implement the lattice based attack. The main difficulty in running the lattice attack in our experiment is that it requires lattice reduction in very large dimension, beyond what is currently supported by state of the art lattice reduction libraries. However, the theoretical running time of the attack is polynomial, and the corresponding parameter settings should still be considered insecure. In the following, we refer to our linear attack as *the* attack.

Algorithm 1: The pseudocode outlining our key recovery experiments.

```
Input: Lattice parameters $(N, \log q)$, initial scaling factor $\Delta_0$, plaintext bound $B$,
       and circuit $g$.
1 Sample $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(N, \log q, \Delta_0)$, where $(1, s) = \mathsf{sk}$
2 Sample $\mathbf{z} \leftarrow \mathbb{C}^{N/2}$ such that $|z_i| \leq B$ for all $1 \leq i \leq N/2$
3 Encrypt $\mathsf{ct}_{\mathrm{input}} \leftarrow \mathsf{Enc}_{\mathsf{pk}}(\mathsf{Encode}(\mathbf{z}; \Delta_0))$
4 Evaluate $\mathsf{ct}_{\mathrm{output}} \leftarrow \mathsf{Eval}_{\mathsf{ek}}(g, \mathsf{ct}_{\mathrm{input}})$
5 Decrypt $\mathbf{z}' \leftarrow \mathsf{Decode}(\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_{\mathrm{output}}); \Delta)$, where $\Delta$ is the scaling factor in $\mathsf{ct}_{\mathrm{output}}$
6 Encode $m'' \leftarrow \mathsf{Encode}(\mathbf{z}'; \Delta)$
7 Compute $s' \leftarrow a^{-1} \cdot (m'' - b) \in \mathcal{O}_q$, where $(b, a) = \mathsf{ct}_{\mathrm{output}}$
8 return $s' = s$
```

3.3.6 Implementation of Our Attack and Experiments

A pseudocode outline of our experiment programs is presented in Algorithm 1. Such programs model the situations where an attacker can influence an honest user to perform certain homomorphic computations and can obtain both the final ciphertexts and the decrypted approximate numbers. A successful run indicates that the target CKKS implementation is not $\mathsf{IND\text{-}CPA}^{+}$-secure.

For concrete homomorphic computations, we choose to compute the variance of a wide range of input data to exemplify how our attack may be affected by large underlying plaintexts in extreme cases. We also compute the logistic function $(1 + e^{-x})^{-1}$ and exponential functions e^x using their Maclaurin series up to a degree d, to check whether our attack may be affected by the bigger noises and the possibly adjusted scaling factors due to multi-level homomorphic computations. We remark that all these homomorphic computations are very common in applications of the CKKS scheme.

In our programs, we use the data structures and public APIs provided by each library to carry out the key recovery computation[2]. Note that an attacker is free to use any method, not necessarily these public interfaces, to carry out the attack.

[2]The source code of our attack implementations are available at https://github.com/ucsd-crypto/CKKSKeyRecovery.

Table 3.1. The results of applying our attack on homomorphically computed variance of $N/2 = 2^{15}$ random complex numbers of magnitude $1 \le B \le 2^9$. We carried out the attack against all main open source implementations of CKKS, obtaining similar results. Numbers are packed into all slots, and are encoded using various initial scaling factors Δ_0. For each parameter combination (Δ_0, B), we ran our programs 100 times against each library. A "✓" indicates that, for *all* these libraries with the particular parameters, the attack *always* succeeded to recover sk. A few cells where a number is shown, correspond to extreme parameters where some runs failed to recover sk, and the number is the maximum (over all libraries) of the average l_∞ norms of the encoding error ε. These settings are still subject to attacks based on lattice reduction, see Sections 3.3.4 and 3.3.8 for details.

Attack applied to HEAAN, PALISADE, SEAL, HElib

	B	1	2	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
Variance	$\log \Delta_0 = 30$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	$\log \Delta_0 = 40$	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	$\log \Delta_0 = 50$	✓	✓	✓	✓	✓	✓	1.21	5.41	20.7	80.2

3.3.7 Details on Different Implementations of CKKS

We considered the latest versions of all these libraries: HEAAN version 2.1 [37], PALISADE version 1.10.4 [60], SEAL version 3.5 [66], and HElib version 1.1.0 [38] and RNS-HEAAN [64]. All these libraries implement the transformation φ and its inverse using the classical Cooley-Tukey FFT algorithm on double-precision floating-point numbers. Still, they contain several distinct implementation details relevant to our attack.

Multi-precision integers vs. double-CRT representation. All versions of HEAAN (version 1.0 as in [22], version 1.1 as in [20], and the most recent version 2.1) use multi-precision integers to represent key materials and ciphertexts. Consequently, HEAAN achieves very good accuracy in approximate decryption, but at the same time it rarely introduces any encoding error, resulting in a great success rate in our key recovery experiment.

To improve efficiency, the residual number system, aka double-CRT representation, is adopted to the CKKS scheme in [21], and it is implemented in RNS-HEAAN. Other libraries also implement the RNS variant of CKKS, with some different details:

Table 3.2. The results of applying our attack to homomorphically computed logistic and exponential functions on random real numbers of magnitude $B \in \{1, 2, 8\}$ packed into full $N/2 = 2^{15}$ slots, evaluated using their Maclaurin series of degree $d \in \{5, 10\}$. For each parameter setting, we ran our experimental program 100 times for each library, and here "✓" indicates sk was recovered in all these runs against a particular library. A few cells where a number is shown, correspond to extreme parameters when some runs failed to recover sk, and the number is the average l_∞ norm of the encoding error ε in these runs. For HElib, "n/a" indicates the parameters are not supported by the library.

Attack applied to HEAAN, PALISADE, SEAL, HElib

			HEAAN		PALISADE		SEAL		HElib	
	Δ_0	B	$d = 5$	$d = 10$	$d = 5$	$d = 10$	$d = 5$	$d = 10$	$d = 5$	$d = 10$
Logistic	2^{30}	1	✓	✓	✓	✓	✓	✓	✓	✓
	2^{40}	1	✓	✓	✓	✓	✓	✓	3.1	6.7
	2^{50}	1	✓	✓	✓	✓	✓	✓	8.2	8.2
Exponential	2^{30}	1	✓	✓	✓	✓	✓	✓	✓	✓
		2	✓	✓	✓	✓	✓	✓	n/a	n/a
		8	✓	✓	✓	✓	✓	✓	n/a	n/a
	2^{40}	1	✓	✓	✓	✓	✓	✓	1.9	8.2
		2	✓	✓	✓	✓	✓	✓	n/a	n/a
		8	✓	✓	✓	✓	✓	✓	n/a	n/a
	2^{50}	1	✓	✓	✓	✓	✓	✓	8.1	8.2
		2	✓	✓	✓	✓	✓	✓	n/a	n/a
		8	7.6	15.2	8.1	18.2	2.2	4.3	n/a	n/a

- During decryption, RNS-HEAAN uses only the first RNS tower of ciphertexts; so it expects the scaled plaintext to be much smaller than the 60-bit prime modulus in the first tower. Other libraries convert the double-CRT format to multi-precision integers before applying the canonical embedding; so they support a larger plaintext space and are more accurate.

- During rescaling, RNS-HEAAN uses a power-of-2 rescaling factor, while the other libraries' rescaling factors are the primes or close to primes in the moduli chain. In particular, PALISADE optimizes the rescaling factors to reduce the errors and precision loss in many homomorphic operations [45].

As observed in our experiment, among the RNS implementations of CKKS, our attack was more successful against the libraries using more accurate element representations and scaling factors.

PALISADE. In addition, PALISADE uses extended precision floating-point arithmetic in Decode, which has 64-bit precision on X86 CPUs. This further improves the accuracy of approximate decryption, but perhaps unintentionally making our attack more successful by a tiny margin (comparing to other libraries).

HElib. Unlike other libraries, HElib adjusts the scaling factor used in Encode and many homomorphic operations according to the estimated noise size and the magnitude of the plaintext. It expects the input numbers to have magnitude at most 1 for optimal precisions. So our experiment with HElib chooses random input only within the unit circle.

RNS-HEAAN. Looking back to RNS-HEAAN, its implementation of Decode introduces a small round-off error in a conversion from `uint64_t` to `double`. As a result, such (seemingly unexpected) implementation choice may lead to reduced precision (by only a few bits), but it also results in more failed runs in our experiment. Still, when our attack fails, the encoding errors are quite small, and so RNS-HEAAN is still subject to the lattice reduction attack. We tried to "fix" this by more carefully converting between number systems, and we immediately see a much better success rate for our attack.

3.3.8 Experiment Results

We set up all libraries with the highest supported lattice dimension $N = 2^{16}$, which also corresponds to the highest security level. By the analysis in Section 3.3.4 (and also observed in our experiment), the larger the dimension is, the higher the chance an encoding error may show up (leading to failed attack runs). On the other hand, since the claimed security decreases with larger values of the modulus q, we set it to around 350 bit, which is a secure, yet realistic value for FHE schemes. According to common evaluation methodologies [2], the associated LWE problem provides a level of security well above 256 bits. (Specifically, in dimension $N = 2^{16}$, it is estimated that 256-bits of security are achieved even for moduli q with over 700 bits.)

In all our experiments, we use the full packing mode with $N/2$ slots. For the variance computation, we generate random input numbers with magnitude $B \leq 2^9$. For the experiment on the logistic and the exponential functions, we set the maximal degree of their Maclaurin series to $d \leq 10$, which provides good approximation for inputs smaller than 1.

Our experiments are executed in a 64-bit Linux environment running on an Intel i7-4790 CPU. The attack is very efficient, especially for the RNS-CKKS implementations, as the key recovery computation can benefit from using NTT and parallelization. Each individual run in our experiment finishes within several seconds to just one minute, with most of the running time taken by the key generation and encryption/homomorphic evaluation operations, rather than the attack itself. For each homomorphic computation task, for each parameter setting, and for each library, we run our attack 100 times to record the success rate and the encoding error ε. The results of the experiments with HEAAN, PALISADE, SEAL, and HElib are presented in Tables 3.1 and 3.2. As shown in these tables, our attack *always* succeeded to recover the secret key in *most* parameter settings against *all* the libraries, especially for typical input sizes and scaling factors. The failed cases in both tables correspond to the extreme parameters where the l_2 norm of the underlying plaintext exceeds 2^{52}, showing better practical performance than the worse case analysis in Section 3.3.4. (There are more failed cases with HElib because its adjusted scaling factors are typically larger and so are the plaintexts.) Comparing the results on the logistic and the exponential functions, we conclude that a deeper level of homomorphic computation has no significant effect on our attack, and the runs in the last row of Table 3.2 failed due to larger plaintext sizes. In particular, the encoding error ε with SEAL is smaller than other libraries because its Decode implementation incurs less round-off errors in scaling by Δ^{-1}.

We did a limited number of experiment with RNS-HEAAN because it has a small plaintext space. Nonetheless, we see a consistent but small encoding error of size $\|\varepsilon\|_\infty \leq 2^7$ in our RNS-HEAAN experiments when $B^2\Delta_0 \approx 2^{50}$.

3.3.9 A Provably Secure Countermeasure

We propose a concrete countermeasure to our attacks. The main idea is to add a pseudorandom polynomial with coefficients from $[-B, B]$ to the approximate decryption result, where the ratio $B/|e|$ must be superpolynomial to guarantee security. Such technique is called *noise flooding* in the literature. To make decryption deterministic, we apply a PRF to generate the randomness needed for sampling the noise.

Assume there exists a pseudorandom function $\mathsf{PRF} : \{0,1\}^\kappa \times \{0,1\}^* \to \{0,1\}^*$. For an integer B such that $0 < B < q_L$, we can convert PRF into a pseudorandom function $\mathsf{PRF}[B] : \{0,1\}^\kappa \to \mathcal{O}^2_{q_L} \to (\mathbb{Z} \cap [-B, B])^N$. In addition to CKKS's parameters, we include an integer $t = t(\kappa)$ and B as public parameters. We modify the CKKS scheme by incorporating PRF and its variants $\mathsf{PRF}[B]$ as follows.

- $\mathsf{KeyGen}'(1^\kappa) = (\mathsf{sk}' = (\mathsf{sk}, K), \mathsf{pk}, \mathsf{ek})$: Sample $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ as in CKKS, and also sample $K \leftarrow \{0,1\}^\kappa$ as a PRF key.

- $\mathsf{Enc}'_{\mathsf{pk}}(m) = \mathsf{Enc}_{\mathsf{pk}}(m)$, $\mathsf{Eval}'_{\mathsf{ek}}(g, \{\mathsf{ct}_i\}_i) = \mathsf{Eval}_{\mathsf{ek}}(g, \{\mathsf{ct}_i\}_i)$: The encryption and homomorphic evaluation algorithms stay the same as in CKKS.

- $\mathsf{Dec}'_{\mathsf{sk}'}(\mathsf{ct}) = m' + \mathsf{PRF}[B]_K(\mathsf{ct})$: First we decrypt ct as in CKKS to get $m' = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct})$. Suppose β is the l_∞ upper bound of the error in m', which is available in the tagged information in ct. Then we apply $\mathsf{PRF}[B]$ to add a pseudorandom polynomial $u = \mathsf{PRF}[B]_K(\mathsf{ct})$ to m', where $\|u\|_\infty \leq B$.

From a high level, the modified scheme is $\mathsf{IND\text{-}CPA}^+$-secure because that the decryption oracle can be simulated knowing only the exact plaintext encrypted in a ciphertext. To prove security, we first recall the definition of pseudorandom functions.

Definition 6 (Pseudorandom Functions)**.** *Let* $\mathsf{PRF} : \{0,1\}^\kappa \times X \to Y$ *be an efficiently computable deterministic function. We say* PRF *is a* pseudorandom function *with domain* X *and range* Y *if* *for any efficient distinguisher $\mathcal{A}$,*

$$\left| \Pr_{K \leftarrow \{0,1\}^\kappa} \{\mathcal{A}^{\mathsf{PRF}_K(\cdot)}(1^\kappa) = 1\} - \Pr_{\mathsf{f}}\{\mathcal{A}^{\mathsf{f}(\cdot)}(1^\kappa) = 1\} \right| \leq \mathsf{negl}(\kappa),$$

where f *is taken uniformly at random from the space of all functions* $X \to Y$.

Theorem 3 (Naive noise flooding)**.** *Let* $(\mathsf{KeyGen}', \mathsf{Enc}', \mathsf{Eval}', \mathsf{Dec}')$ *be the approximate HE scheme defined as above. Assume the CKKS scheme is* $\mathit{IND\text{-}CPA}$ *secure, and assume* $t = \Omega(\kappa)$ *and* $B = 2^k \beta$*. Then the modified approximate HE scheme is* $\mathit{IND\text{-}CPA}^+$ *secure.*

Proof. Fix some polynomials q and ℓ. Since the leakage function class contains only the identity function, we omit the function parameter in decryption oracle queries and write $\mathsf{D}(j)$ instead to denote a query for decrypting the j'th ciphertext. Let $b \in \{0, 1\}$ be a bit. We proceed by using the following hybrids.

- $\mathcal{H}_b^{(0)}$: This hybrid is exactly the same as the experiment $\mathsf{Expr}_b^{\text{app}}$ with the modified scheme. In particular, the keys are sampled as $(\mathsf{sk}, \mathsf{pk}, \mathsf{ek}) \leftarrow \mathsf{KeyGen}(1^\kappa)$ and $K \leftarrow \{0,1\}^\kappa$; an encryption oracle query $\mathsf{E}(m_0, m_1)$ is answered with a ciphertext $\mathsf{ct} \leftarrow \mathsf{Enc}'_{\mathsf{pk}}(m_b)$; if (m_0^j, m_1^j) is the j'th plaintext pair and $m_0^j = m_1^j$, and if ct_j is the corresponding ciphertext with error upper bound β_j, then the decryption oracle query $\mathsf{D}(j)$ is answered with $\mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_j) + \mathsf{PRF}[B]_K(\mathsf{ct}_j)$, *i.e.*, the same as in $\mathsf{Dec}'_{\mathsf{sk}'}(\mathsf{ct}_j)$.

- $\mathcal{H}_b^{(1)}$: This hybrid is the same as $\mathcal{H}_b^{(0)}$, except that a random function $\mathsf{f} : \mathcal{O}_{q_L}^2 \to \mathcal{O}_{q_L}$ is sampled at the beginning, and that for each decryption query $\mathsf{D}(j)$ such that $m_0^j = m_1^j$, we deterministically convert f into a random function $\mathsf{f}[B] : \mathcal{O}_{q_L}^2 \to (\mathbb{Z} \cap [-B, B])^N$, and we send $m'_j + \mathsf{f}[B](\mathsf{ct}_j)$ to the adversary, where $m'_j = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_j)$.

- $\mathcal{H}_b^{(2)}$: This hybrid is the same as $\mathcal{H}_b^{(1)}$, except that for each decryption query $\mathsf{D}(j)$ such that $m_0^j = m_1^j$, we send to the adversary $m_b^j + \mathsf{f}[B](\mathsf{ct}_j)$.

Notice that the PRF key K is not used in $\mathcal{H}_b^{(1)}$, so for any adversary $\mathcal{A}$, the hybrids $\mathcal{H}_b^{(0)}$ and $\mathcal{H}_b^{(1)}$ are indistinguishable due to the security of the pseudorandom function PRF.

To show that the hybrids $\mathcal{H}_b^{(1)}$ and $\mathcal{H}_b^{(2)}$ are indistinguishable, consider each decryption oracle query $\mathsf{D}(j)$ such that $m_0^j = m_1^j$. Let $e_j = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{ct}_j) - m_b^j$ be the encryption noise in the ciphertext ct_j; then we have $\|e_j\|_\infty \leq \beta_j$. Also let $h : \mathcal{O} \to \mathcal{O}$ be the function $h(a) = a + u$ where $u \leftarrow (\mathbb{Z} \cap [-B, B])^N$. For each decryption query $\mathsf{D}(j)$ such that $m_0^j = m_1^j$, an adversary receives $m'_j = m_b^j + h(e_j)$ in $\mathcal{H}_b^{(1)}$, and it receives $m''_j = m_b^j + h(0)$ in $\mathcal{H}_b^{(2)}$. The statistical distance between $h(0)$ and $h(e_j)$ is bounded by

$$\mathsf{SD}(h(0), h(e_j)) \leq \frac{1}{2}(\frac{B}{2\beta_j} + \frac{B}{2\beta_j}) = \frac{1}{2^\kappa}.$$

So the adversary's views $\mathcal{V}_1$ in $\mathcal{H}_b^{(1)}$ and $\mathcal{V}_2$ in $\mathcal{H}_b^{(2)}$ have a statistical distance bounded by

$$\mathsf{SD}(\mathcal{V}_{(1)}, \mathcal{V}_{(2)}) \leq q2^{-\kappa},$$

which is negligible.

Finally, notice that the hybrids $\mathcal{H}_b^{(2)}$ for $b \in \{0, 1\}$ do not use the secret key sk, and the decryption oracle queries are answered with the same distribution in these two hybrids. So they are indistinguishable due to $\mathsf{IND\text{-}CPA}$ security. Therefore the modified scheme is $\mathsf{IND\text{-}CPA}^+$-secure. □

Acknowledgement Chapter 3 contains a reprint of materials as it appears in our submission to Eurocrypt 2021, Baiyu Li and Daniele Micciancio, *On the Security of Homomorphic Encryption on Approximate Numbers*. The dissertation author was a primary investigator and author of this paper.

Chapter 4

Symbolic Security Model of Yao's Garbled Circuits

In this chapter we present our results on symbolic cryptography. Specifically, we extend the previous symbolic framework [57, 58] to analyze complex protocols such as garbled circuits and secret sharing schemes. Along the way, we also present a structured, algebraic notation of circuits.

4.1 Symbolic Cryptography

In this section we introduce basic notations used by symbolic cryptography. We use the bit 0 for the Boolean value *false*, and 1 for *true*. For $n \geq 1$, $\{0,1\}^n$ is the set of all Boolean vectors of length n. We can concatenate two Boolean vectors $x \in \{0,1\}^n$ and $y \in \{0,1\}^m$ to obtain $xy \in \{0,1\}^{n+m}$. For any $x \in \{0,1\}^n$, we can think x as a concatenation of n bits, written as $x = x_1 \cdots x_n$, where $x_1, \ldots, x_n \in \{0,1\}$. For any $x, y \in \{0,1\}$, the NAND function $x \uparrow y = \neg(x \wedge y)$ maps x and y to 0 if and only if both x and y are 1.

Our symbolic cryptographic expressions extend those defined in [56] with random bits and a swap operation, which we need to model garbled circuits. Informally, symbolic expressions are built from random keys and (possibly random) bits, using a symmetric encryption scheme, a (length doubling) pseudorandom generator, a pairing (concatenation) operation, and the (random) permutation of pairs. Just as in computational cryptography it is convenient to group bit-strings according to their length, in symbolic cryptography it is customary to classify expressions according to their *shape*, which captures the expression size in a representation independent way. The set of possible shapes for a symbolic expression is defined by the grammar:

$$\mathbf{Shape} \to \mathbb{B} \mid \mathbb{K} \mid (\!|\mathbf{Shape}, \mathbf{Shape}|\!) \mid \{\!|\mathbf{Shape}|\!\}$$

representing the shapes of bits, keys, pairs (of two sub-expressions of arbitrary shape), and encryptions (of messages of arbitrary shape), respectively. For example $(\!|\mathbb{K}, \{\!|\mathbb{B}|\!\}|\!)$ is the shape of a pair consisting of a key and the encryption of a single bit message. Let $\mathbf{B} = \{\mathsf{B}_i \mid i = 1, 2, \dots\}$ be a set of atomic bit symbols, and $\mathbf{K} = \{\mathsf{K}_i \mid i = 1, 2, \dots\}$ a set of atomic key symbols, representing independent uniformly random bits and independent uniformly random keys, respectively. For any shape $s \in \mathbf{Shape}$, we define a corresponding set of expressions of shape s (denoted $\mathbf{Exp}(s)$) according to the grammar rules:

$$\begin{aligned}
\mathbf{Exp}(\mathbb{B}) &\to 0 \mid 1 \mid \mathsf{B}_i \mid \neg\mathbf{Exp}(\mathbb{B}) \\
\mathbf{Exp}(\mathbb{K}) &\to \mathsf{K}_i \mid \mathsf{G}_0(\mathbf{Exp}(\mathbb{K})) \mid \mathsf{G}_1(\mathbf{Exp}(\mathbb{K})) \\
\mathbf{Exp}(\{\!|s|\!\}) &\to \{\!|\mathbf{Exp}(s)|\!\}_{\mathbf{Exp}(\mathbb{K})} \\
\mathbf{Exp}((\!|s,t|\!)) &\to (\mathbf{Exp}(s), \mathbf{Exp}(t)) \\
\mathbf{Exp}((\!|s,s|\!)) &\to \pi[\mathbf{Exp}(\mathbb{B})](\mathbf{Exp}(s), \mathbf{Exp}(s)).
\end{aligned}$$

where s, t range over **Shape**, B_i ranges over **B**, and K_i ranges over **K**. Most symbols are self explanatory: $\neg b$ represents the logical negation of bit b, $(\mathsf{G}_0(k), \mathsf{G}_1(k))$ represents the output of a length doubling pseudorandom generator on seed k (with $\mathsf{G}_0(k)$ the first half of the output, and $\mathsf{G}_1(k)$ the second half,) $\{\!|e|\!\}_k$ is the encryption of e under key k, (e_0, e_1) is the ordered pair with sub-expressions e_0 and e_1, and for any bit b and expressions e_0, e_1 of the same shape, $\pi[b](e_0, e_1)$ represents the pair (e_0, e_1) with the two components swapped if $b = 1$. For example, $\{\!|\, \mathsf{G}_0(\mathsf{K}_1)|\!\}_{\mathsf{G}_1(\mathsf{K}_1)}$ represents the encryption of the first half $\mathsf{G}_0(\mathsf{K}_1)$ of a pseudorandom string (obtained by applying the pseudorandom generator on seed K_1,) encrypted under the second half of the pseudorandom string, while $\pi[\mathsf{B}_1](\mathsf{G}_0(\mathsf{K}_1), \mathsf{G}_1(\mathsf{K}_1))$ represents a pseudorandom string (output by the pseudorandom generator on seed K_1), with the first and second half of the string permuted (swapped) at random depending on the value of the (random) bit B_1.

Note that we can iteratively apply the pseudorandom generator on a key expression k to obtain expressions such as $\mathsf{G}_{b_1}(\mathsf{G}_{b_2}(\cdots(\mathsf{G}_{b_n}(k))))$ for $n \geq 0$ and $b_1, b_2, \ldots, b_n \in \{0, 1\}$. Such expressions are abbreviated as $\mathsf{G}_{b_1 b_2 \ldots b_n}(k)$. Let ε denote the empty bit-string, and let $\{0, 1\}^*$ denote the set of all bit-strings. For any set $S \subseteq \mathbf{Exp}(\mathbb{K})$, we define the sets

$$\begin{aligned} \mathsf{G}^*(S) &= \{\mathsf{G}_w(k) \mid k \in S, w \in \{0,1\}^*\} \\ \mathsf{G}^+(S) &= \{\mathsf{G}_w(k) \mid k \in S, w \in \{0,1\}^*, w \neq \varepsilon\} \end{aligned}$$

obtained by applying the (first or second half of the) pseudorandom generator zero (resp. one) or more times to a key in S. So, for example, $\mathsf{G}^*(\mathbf{K}) = \mathbf{Exp}(\mathbb{K})$ is the set of all (random or pseudorandom) keys. For convenience, we write $\mathbf{K}^*$ for $\mathsf{G}^*(\mathbf{K})$ and $\mathbf{K}^+$ for $\mathsf{G}^+(\mathbf{K})$. If $S = \{k\}$ is a singleton set, we usually write $\mathsf{G}^+(k)$ and $\mathsf{G}^*(k)$ instead of $\mathsf{G}^+(\{k\})$ and $\mathsf{G}^*(\{k\})$.

Patterns are extensions of expressions that include the construct $\{\!\!\{ s \}\!\!\}_{\mathbf{Exp}(\mathbb{K})}$ to represent the encryption of an unknown expression of shape s. The pattern $\{\!\!\{ s \}\!\!\}_{\mathbf{Exp}(\mathbb{K})}$ has shape $\{\!\!\{ s \}\!\!\}$. Formally, patterns are defined by a grammar with variables $\mathbf{Pat}(s)$ indexed by $s \in \mathbf{Shape}$, and the same set of rules as those given for $\mathbf{Exp}(s)$, with the addition of one more rule

$$\mathbf{Pat}(\{\!\!\{ s \}\!\!\}) \to \{\!\!\{ s \}\!\!\}_{\mathbf{Exp}(\mathbb{K})}.$$

$\mathbf{Pat}(s)$ is the set of all patterns of shape s, and $\mathbf{Pat}$ is the set of all patterns (of any shape). Notice that $\mathbf{Pat}(\mathbb{B}) = \mathbf{Exp}(\mathbb{B})$ and $\mathbf{Pat}(\mathbb{K}) = \mathbf{Exp}(\mathbb{K})$ because only encryption gives raise to nontrivial patterns.

Computational evaluation Let κ be the security parameter for cryptographic primitives in the computational setting. For simplicity, all keys are assumed to have length κ. To instantiate our symbolic framework, we assume the existence of a length-doubling pseudorandom generator $\mathcal{G}$ and an $\mathsf{IND\text{-}CPA}$ secure symmetric encryption scheme $(\mathcal{E}, \mathcal{D})$ with keys of length κ.

Definition 7 (Pseudorandom generator)**.** *A deterministic function $\mathcal{G} : \{0,1\}^\kappa \to \{0,1\}^{2\kappa}$ is a secure length-doubling pseudorandom generator if it can be computed in polynomial time and, for any PPT distinguisher $\mathcal{A}$ we have*

$$\left| \Pr_{s \leftarrow \{0,1\}^{2\kappa}} \{\mathcal{A}(s) = 1\} - \Pr_{r \leftarrow \{0,1\}^{\kappa}} \{\mathcal{A}(\mathsf{G}(r)) = 1\} \right| \leq \mathsf{negl}(\kappa).$$

For any symmetric encryption scheme $(\mathcal{E}, \mathcal{D})$ and $b \in \{0,1\}$, the *left-right encryption oracle* $\mathcal{O}_{\mathcal{E},b}$ first samples a uniformly random key $k \leftarrow \{0,1\}^\kappa$, and then it answers any encryption query of the form (m_0, m_1) with a ciphertext $\mathcal{E}(k, m_b)$, where m_0 and m_1 are of the same length.

Definition 8 ($\mathsf{IND\text{-}CPA}$ secure symmetric encryption scheme)**.** *A pair of PPT algorithms $(\mathcal{E}, \mathcal{D})$ is an* $\mathsf{IND\text{-}CPA}$ secure *symmetric encryption scheme with key length κ if the followings hold:*

- **Correctness:** *For any $k \in \{0,1\}^\kappa$ and $m \in \{0,1\}^*$,* $\Pr\{\mathcal{D}(k, \mathcal{E}(k, m)) = m\} = 1$;
- **Security:** *For any PPT distinguisher $\mathcal{A}$,*

$$\left|\Pr\{\mathcal{A}^{\mathcal{O}_{\mathcal{E},0}}(1^\kappa) = 1\} - \Pr\{\mathcal{A}^{\mathcal{O}_{\mathcal{E},1}}(1^\kappa) = 1\}\right| \leq \mathsf{negl}(\kappa),$$

where the probability is over the random choices of $\mathcal{A}$.

We assume that the size of a cipher-text $\mathcal{E}(k, m)$ is a function of the size of the input m, i.e., if two messages have the same length, then their encryption also have the same length. We do not make any special assumption on the encoding of pairs (e_0, e_1), except that e_0 and e_1 can be recovered from (e_0, e_1), and that the size of (e_0, e_1) depends only on the size of e_0 and the size of e_1. For any $x \in \{0,1\}^\kappa$, let $\mathcal{G}_0(x)$ and $\mathcal{G}_1(x)$ be the first and second halves of the bit-string

$\mathcal{G}(x)$, so that $\mathcal{G}(x) = \mathcal{G}_0(x)\mathcal{G}_1(x)$. Let σ be a function mapping $\mathbf{B}$ to $\{0,1\}$, and $\mathbf{K}$ to $\{0,1\}^\kappa$. We can extend σ to map any symbolic expression to a distribution on bit-strings as follows:

$$\begin{array}{ll}
\sigma(0) = 0, & \sigma(1) = 1, \\
\sigma(\mathsf{G}_0(k)) = \mathcal{G}_0(\sigma(k)), & \sigma(\neg b) = 1 - (\sigma(b)), \\
\sigma(\mathsf{G}_1(k)) = \mathcal{G}_1(\sigma(k)), & \sigma(\{\!\!\{e\}\!\!\}_k) = \mathcal{E}(\sigma(k), \sigma(e)), \\
\sigma((e_0, e_1)) = (\sigma(e_0), \sigma(e_1)), & \sigma(\pi[b](e_0, e_1)) = \begin{cases} (\sigma(e_0), \sigma(e_1)) & \text{if } \sigma(b) = 0 \\ (\sigma(e_1), \sigma(e_0)) & \text{if } \sigma(b) = 1 \end{cases}
\end{array}$$

where $k \in \mathbf{Exp}(\mathbb{K})$, and $b \in \mathbf{Exp}(\mathbb{B})$. The computational evaluation $[\![e]\!]$ of an expression e is defined as the probability distribution obtained by first choosing a uniformly random key and bit assignment σ, and then picking a sample from $\sigma(e)$.[1] It is easy to check (by induction) that any two expressions of the same shape evaluate to bit-strings of the same length.

Lemma 4. *For any shape s, all strings in $[\![\mathbf{Exp}(s)]\!]$ have the same bit-length.*

Using this property, we can associate a bit-length to any shape s as the bit-length $|s|$ of any string in the set $[\![\mathbf{Exp}(s)]\!]$, and extend the evaluation of expressions to evaluation of patterns by defining

$$\sigma(\{\!\!\{s\}\!\!\}_k) = \mathcal{E}(\sigma(k), 0^{|s|}).$$

Independence of pseudorandom keys The following definitions are given in [56] to provide a (computationally sound) treatment of symbolic pseudorandom generators. For any two keys $k_1, k_2 \in \mathbf{K}^*$, if $k_2 \in \mathsf{G}^*(k_1)$ then we say that k_1 *yields* k_2, and denote this as $k_1 \preceq k_2$, meaning that k_2 can be obtained from k_1 by repeated application of the pseudorandom generator. By $k_1 \prec k_2$ we mean that $k_1 \preceq k_2$ and $k_1 \neq k_2$. We say that k_1 and k_2 are *independent* if neither $k_1 \preceq k_2$ nor $k_2 \preceq k_1$. The keys $\{k_1, \ldots, k_n\}$ form an independent set if k_i and k_j are independent

[1]Notice that, even for fixed σ and e, the image $\sigma(e)$ is a probability distribution because it involves the use of a probabilistic encryption scheme $\mathcal{E}$.

for all $i \neq j$. The root of any set of keys S is $\mathbf{Roots}(S) = S \setminus \mathsf{G}^+(S)$. Thus S is independent if and only if $S = \mathbf{Roots}(S)$. We recall the following theorem from [56] which shows that independent symbolic keys correspond to (computational) pseudorandom bit-strings.

Theorem 4 ([56, Theorem 1]). *Let $k_1, \dots, k_n \in \mathbf{K}^*$ be a sequence of symbolic keys. Then for any secure length-doubling pseudorandom generator $\mathcal{G}$, the following two conditions are equivalent:*

1. *The keys $k_1, \dots, k_n$ are symbolically independent (i.e., $k_i \preceq k_j$ if and only if $i = j$).*
2. *The distribution $[\![k_1, \dots, k_n]\!]$ is computationally indistinguishable from $[\![r_1, \dots, r_n]\!]$ where $r_1, \dots, r_n \in \mathbf{K}$ are distinct atomic key symbols.*

Equivalence and Renaming of patterns We consider patterns up to simple operations that do not change the probability distributions associated to them. First, let $\equiv$ be the smallest congruence relation on **Pat** such that

$$\begin{aligned} \neg 0 &\equiv 1 & \qquad \pi[0](e_0, e_1) &\equiv (e_0, e_1) \\ \neg 1 &\equiv 0 & \pi[1](e_0, e_1) &\equiv (e_1, e_0) \\ \neg(\neg b) &\equiv b & \pi[\neg b](e_0, e_1) &\equiv \pi[b](e_1, e_0) \end{aligned}$$

for all $e_0, e_1 \in \mathbf{Pat}(s)$, and $b \in \mathbf{Pat}(\mathbb{B})$. It should be clear from the computational interpretation of $\pi[b]$ and $\neg b$ that for any two equivalent patterns $e_0 \equiv e_1$ and any assignment σ, the probability distributions $\sigma(e_0)$ and $\sigma(e_1)$ are identical. Similarly, we define a *random bit renaming* as a function $\alpha_B : \mathbf{B} \to \{b, \neg b \mid b \in \mathbf{B}\}$ such that its projection $\alpha'_B : \mathbf{B} \to \mathbf{B}$ (defined by the condition $\alpha_B(b) \in \{\alpha'_B(b), \neg\alpha'_B(b)\}$) is a bijection on $\mathbf{B}$. Random bit renamings are extended to patterns $\alpha_B : \mathbf{Pat}(s) \to \mathbf{Pat}(s)$ in the obvious way, and it is easy to check that for any pattern $e \in \mathbf{Pat}(s)$ and assignment σ, the distributions $\sigma(e)$ and $\sigma(\alpha_B(e))$ are identical.

For keys, we consider a form of renaming that may change the distribution associated to an expression or pattern, but in a computationally indistinguishable way. Following [56], we define a *pseudorandom key renaming* as a mapping $\alpha_K : S \to \mathbf{K}^*$ on $S \subseteq \mathbf{K}^*$ that preserves G, i.e.,

$$\mathsf{G}_w(k_1) = k_2 \qquad \Longleftrightarrow \qquad \mathsf{G}_w(\alpha_K(k_1)) = \alpha_K(k_2)$$

for all $w \in \{0,1\}^*$ and $k_1, k_2 \in S$. We restate some useful properties of key renamings proved in [56]:

1. [56, Lemma 1] Any pseudorandom key renaming $\alpha_K : S \to \mathbf{K}^*$ is a bijection from S to $\alpha_K(S)$. Moreover, S is independent if and only if $\alpha_K(S)$ is independent.

2. [56, Lemma 2] Any pseudorandom key renaming α_K with domain S can be uniquely extended to a pseudorandom key renaming $\bar{\alpha}_K$ with domain $\mathsf{G}^*(S)$. In particular, any pseudorandom key renaming can be uniquely specified as an extension $\bar{\alpha}_K$ of a bijection $\alpha_K : A \to B$ between independent sets $A = \mathbf{Roots}(S)$ and $B = \alpha_K(A)$.

3. [56, Lemma 5] For any pseudorandom key renaming $\alpha_K : S \to \mathbf{K}^*$ and set of keys $A \subseteq S$, $\alpha_K(\mathbf{Roots}(A)) = \mathbf{Roots}(\alpha_K(A))$.

Pseudorandom key renamings α_K can also be extended to patterns $\alpha_K : \mathbf{Pat}(s) \to \mathbf{Pat}(s)$ in the obvious way, and while the distributions $\sigma(e)$ and $\sigma(\alpha_K(e))$ may, in general be different, they are always computationally indistinguishable.

The following lemma is an easy consequence of Theorem 4, and, despite the fact that we use a larger class of expressions, the proof is virtually identical to that of [56, Corollary 1]. For completeness, a formal proof can be found in the appendix.

Lemma 5. *For any pattern e and pseudorandom key renaming α_K, the distributions $[\![e]\!]$ and $[\![\alpha_K(e)]\!]$ are computationally indistinguishable.*

We refer to a pair of mappings $\alpha = (\alpha_B, \alpha_K)$ (consisting of a random bit renaming α_B and a pseudorandom key renaming α_K) as a *pseudorandom renaming*, or simply a *renaming*. For

$$\begin{array}{rlrl} \mathbf{p}(b,S) = b & & \mathbf{p}((e_0,e_1),S) = (\mathbf{p}(e_0,S),\mathbf{p}(e_1,S)) \\ \mathbf{p}(k,S) = k & & \mathbf{p}(\pi[b](e,e_0),S) = \pi[b](\mathbf{p}(e,S),\mathbf{p}(e_0,S)) \\ \mathbf{p}(\{\!|s|\!\}_k,S) = \{\!|s|\!\}_k & & \mathbf{p}(\{\!|e|\!\}_k,S) = \begin{cases} \{\!|\mathbf{p}(e,S)|\!\}_k & \text{if } k \in S \\ \{\!|s|\!\}_k & \text{if } k \notin S \end{cases} \end{array}$$

Figure 4.1. The pattern function $\mathbf{p} : \mathbf{Pat} \times \wp(\mathbf{Pat}(\mathbb{K})) \to \mathbf{Pat}$, defined for all $b \in \mathbf{Exp}(\mathbb{B})$, $k \in \mathbf{Exp}(\mathbb{K})$, $e, e_0 \in \mathbf{Exp}(s)$, $e_1 \in \mathbf{Exp}(t)$

any pattern $e \in \mathbf{Pat}(s)$, we write $\alpha(e) = \alpha_K(\alpha_B(e)) = \alpha_B(\alpha_K(e))$ for the result of applying the renamings to the pattern e.[2] Two patterns e_0 and e_1 are *equivalent up to renaming*, denoted as $e_0 \approx e_1$, if there exists a renaming $\alpha = (\alpha_B, \alpha_K)$ such that $e_0 \equiv \alpha(e_1)$. When we want to emphasize the renaming α, we write $e_0 \approx_\alpha e_1$. It follows from the previous statements that patterns that are equivalent up to renaming evaluate to probability distributions that are computationally indistinguishable.

Pattern computation Following [57], the mapping from expressions to patterns is defined by two functions:

- A function $\mathbf{p}(e, S)$ mapping an expression (or pattern) e and set of keys $S \subseteq \mathbf{K}^*$ to the pattern representing the view of e to an adversary that can decrypt under (all and only) the keys in S.
- A function $\mathbf{r}(p)$ mapping a pattern p to a corresponding set of keys, which may be recoverable by an adversary that sees all the parts of p.

The definition of these functions is virtually identical to the one given in [56] for expressions with pseudorandom keys, extended with an additional case for our "controlled swap" expressions. Informally, $\mathbf{p}(e, S)$ replaces all subexpressions of e of the form $\{\!|e'|\!\}_k$ for some $k \notin S$ and $e' \in \mathbf{Pat}(s)$, with the pattern $\{\!|s|\!\}_k$. The formal definition is given in Fig. 4.1.

[2]Notice that the mappings α_B and α_K commute, so they can be applied in any order.

$$
\begin{aligned}
\mathbf{Keys}(b) &= \emptyset & \mathbf{Keys}(k) &= \{k\}\\
\mathbf{Keys}(\{\!\{e\}\!\}_k) &= \{k\} \cup \mathbf{Keys}(e) & \mathbf{Keys}(\{\!\{s\}\!\}_k) &= \{k\}\\
\mathbf{Keys}((e_0, e_1)) &= \mathbf{Keys}(e_0) \cup \mathbf{Keys}(e_1) & &\\
\mathbf{Keys}(\pi[b](e_0, e_1)) &= \mathbf{Keys}(e_0) \cup \mathbf{Keys}(e_1) & &\\
\mathbf{Parts}(b) &= \{b\} & \mathbf{Parts}(k) &= \{k\}\\
\mathbf{Parts}(\{\!\{e\}\!\}_k) &= \{\{\!\{e\}\!\}_k\} \cup \mathbf{Parts}(e) & \mathbf{Parts}(\{\!\{s\}\!\}_k) &= \{\{\!\{s\}\!\}_k\}\\
\mathbf{Parts}((e_0, e_1)) &= \{(e_0, e_1)\} \cup \mathbf{Parts}(e_0) \cup \mathbf{Parts}(e_1) & &\\
\mathbf{Parts}(\pi[b](e_0, e_1)) &= \{\pi[b](e_0, e_1)\} \cup \mathbf{Parts}((e_0, e_1)) & &
\end{aligned}
$$

Figure 4.2. The definition of the keys and parts of a sub-expression. As usual $b \in \mathbf{Exp}(\mathbb{B})$, $k \in \mathbf{Exp}(\mathbb{K})$.

The formal definition of $\mathbf{r}$ is more technical, and uses the auxiliary functions **Keys** and **Parts** describing the keys and parts of an expression given in Fig. 4.2. As a matter of notation, for any two expressions e' and e, we say that e' is a *sub-expression* of e, denoted as $e' \Subset e$, if $e' \in \mathbf{Parts}(e)$. Notice that encryption keys k are not considered sub-expressions of $\{\!\{e\}\!\}_k$, as, even an adversary with unlimited decryption capabilities cannot, in general, recover k from $\{\!\{e\}\!\}_k$. Informally, $\mathbf{r}(e)$ is defined as the set of all keys that can be potentially recovered from $\mathbf{Parts}(e)$. In [56], this is defined using a general framework to model partial information in symbolic security analysis. For simplicity, here we only give the definition specialized to our class of expressions.

Definition 9. *For any $e \in$ **Pat**, we define the key recovery function $\mathbf{r} : \mathbf{Pat} \to \wp(\mathbf{Pat}(\mathbb{K}))$ as follows:*

$$\mathbf{r}(e) = \mathsf{G}^* \left(\{k \in \mathbf{Keys}(e) \mid (k \Subset e) \vee (\exists k' \in \mathbf{Keys}(e).k \prec k')\}\right)$$

Informally, $\mathbf{r}(e)$ contains all keys k from $\mathbf{Keys}(e)$ (and pseudorandom keys that can be derived from k) such that either k appears in e as a sub-expression, or k is related to some other key in $\mathbf{Keys}(e)$. The intuition behind this definition is that the adversary can learn a key k either by reading it directly from the parts of e, or by combining different pieces of partial information about k. We refer the reader to [56] for further discussion and justification of this definition.

One can check by induction that the following commutative properties hold for $\mathbf{p}$ and $\mathbf{r}$: For any pattern $e \in \mathbf{Pat}$, set of keys $S \subseteq \mathbf{K}^*$, and pseudorandom renaming α, we have $\alpha(\mathbf{p}(e, S)) = \mathbf{p}(\alpha(e), \alpha(S))$, and $\alpha(\mathbf{r}(e)) = \mathbf{r}(\alpha(e))$.

Computational soundness We can now return to the framework of [57] to associate computationally sound symbolic patterns to cryptographic expressions. The functions $\mathbf{p}$ and $\mathbf{r}$ are used to define, for any $e \in \mathbf{Pat}$, a key recovery operator

$$\mathcal{F}_e(S) = \mathbf{r}(\mathbf{p}(e, S))$$

mapping any set of keys $S \subseteq \mathsf{G}^*(\mathbf{K})$, to the set of keys potentially recoverable by an adversary that is capable of decrypting under the keys in S. This operator is used in [57] to prove the following general computational soundness result.

Theorem 5 ([57, Theorem 1])**.** *Assume the functions* $\mathbf{p}$, $\mathbf{r}$ *satisfy the following properties:*

1. $\mathbf{p}(e, \mathbf{K}^*) = e$
2. $\mathbf{p}(\mathbf{p}(e, S), T) = \mathbf{p}(e, S \cap T)$ *for all* $S, T \subseteq \mathbf{K}^*$
3. $\mathbf{r}(\mathbf{p}(e, T)) \subseteq \mathbf{r}(e)$ *for all* $T \subseteq \mathbf{K}^*$
4. *The distributions* $[\![e]\!]$ *and* $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ *are computationally indistinguishable.*

Then, the key recovery operator $\mathcal{F}_e$ *has a (unique) greatest fixed point* $\mathrm{Fix}(\mathcal{F}_e) = \cap_{i>0} \mathcal{F}_e^{(i)}(\mathbf{K}^*)$, *and the pattern*

$$\mathbf{Pattern}(e) = \mathbf{p}(e, \mathrm{Fix}(\mathcal{F}_e))$$

is computationally sound, in the sense that $[\![\mathbf{Pattern}(e)]\!]$ *and* $[\![e]\!]$ *are computationally indistinguishable distributions.*

One can check that the functions $\mathbf{p}$ and $\mathbf{r}$ satisfy all the conditions 1 to 3 in Theorem 5. For the last condition, the following lemma shows that $[\![e]\!]$ and $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ are indistinguishable for all patterns e. The proof is omitted due to space constraint. Using the soundness theorem of the general symbolic framework of [57] we can then conclude that our symbolic semantics is computationally sound.

Lemma 6. *For any $e \in \mathbf{Pat}$, the probability distributions $[\![e]\!]$ and $[\![\mathbf{p}(e, \mathbf{r}(e))]\!]$ are computationally indistinguishable.*

Recall that renamings commute with the pattern function $\mathbf{p}$, i.e., for any expression e and for any set of keys $S \subseteq \mathbf{K}^*$, $\mathbf{p}(\alpha(e), \alpha(S)) = \alpha(\mathbf{p}(e, S))$. It follows that $\mathbf{Pattern}(\alpha(e)) = \alpha(\mathbf{Pattern}(e))$, and therefore we can extend the computational soundness theorem to pattern equivalence up to renaming. That is, for any two expressions e_1 and e_2, symbolic equivalence (up to pseudorandom renaming) of their patterns $\mathbf{Pattern}(e_1)$ and $\mathbf{Pattern}(e_2)$ implies that the two probability distributions $[\![e_1]\!]$ and $[\![e_2]\!]$ are computationally indistinguishable.

Theorem 6. *For any two symbolic expressions e_0, e_1, if $\mathbf{Pattern}(e_0) \approx \mathbf{Pattern}(e_1)$, then $[\![e_0]\!]$ and $[\![e_1]\!]$ are computationally indistinguishable.*

4.2 Inductive Circuit Representation

Traditionally, boolean circuits are described by two sets of gates $\{g_i\}_{i=1}^{q}$ and wires $\{w_i\}_{i=1}^{p}$ and a description of how they are connected together. Each wire carries a boolean value, that is either given as part of the input to the circuit, or is computed by a gate. Each gate is associated to a number of input and output wires, and sets the value of the output wires to some fixed function of the values of the input wires. For simplicity, we consider circuits using just two types of gates:

- a NAND gate that on input two boolean values x_0, x_1, computes the output $y = x_0 \uparrow x_1$, and
- a DUP gate, which duplicates the value on its single input wire x to its two output wires $y_0 = y_1 = x$.

The NAND function itself is complete for the set of all boolean functions, and the DUP gate can be used to implement arbitrary fan-out. So any boolean circuit can be converted to this notation. A circuit with n input wires and m output wires computes a boolean function $f : \{0,1\}^n \to \{0,1\}^m$.

This traditional formalization of circuits is completely unstructured, making it inconvenient to use in symbolic constructions and proofs of security. Below we present an alternative way to describe boolean circuits, which is inductive (larger circuits are built from smaller ones), and supports definitions and proofs by structural induction.

We begin by putting some structure on the set of input and output wires of a circuit, by defining the notion of a *wire bundle*. Informally, the shape of a wire bundle is defined by a well parenthesized expression like $(\circ, (\circ, \circ))$. Formally, we can define bundle to be either a single wire (represented by the symbol $\circ$), or an ordered pair (u, v) where u and v are wire bundles. The size of a bundle is simply the number of wires in it, i.e., the number of $\circ$ subexpressions. Each wire $\circ$ carries a bit $b \in \{0,1\}$, and a bundle of n wires naturally carries a bit vector in $\{0,1\}^n$, but the additional bundle structure will give us easier access to individual bits, without having to index them. We remark that the grouping of wires is not associative, i.e., $((u, v), w)$ is different from $(u, (v, w))$.

We define circuits inductively, specifying a number of basic circuits, and some general operations to combine them together. Each circuit takes as input a bundle of wires, and produces as output another bundle. The set of circuits with input shape s and output shape t is denoted by $\mathrm{Circuit}(s, t)$. Circuits, their inputs and outputs, and the functions they compute, are formally specified in the following definition, with the base and inductive cases illustrated in Fig. 4.3 and 4.4.

Definition 10. *A* circuit *is either a basic circuit from* $\{\mathbf{Swap}, \mathbf{Assoc}, \mathbf{Unassoc}, \mathbf{Dup}, \mathbf{NAnd}\}$*, or it is a composite circuit built using operations* $\ggg$ *and* $\mathbf{First}$*. The semantics of basic circuits are:*

- **Swap** *consumes wires* (u, v) *and produces wires* (v, u)*.*
- **Assoc** *consumes wires* $(u, (v, w))$ *and produces wires* $((u, v), w)$*.*

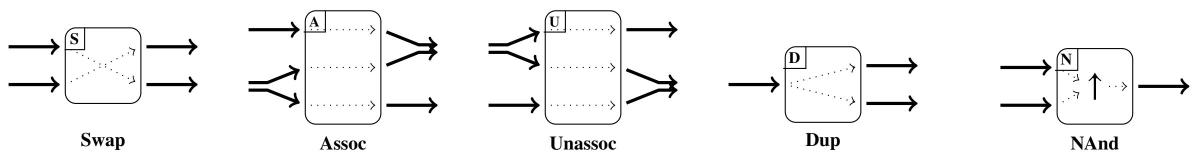


Figure 4.3. The atomic circuits **Swap**, **Assoc**, **Unassoc**, **Dup**, and **NAnd**. The dotted lines indicate how values are transferred from input wires to output wires. For **Swap**, **Assoc**, and **Unassoc**, an arrow may represent a bundle of more than one wires.

- **Unassoc** *consumes wires* $((u, v), w)$ *and produces wires* $(u, (v, w))$.
- **Dup** *consumes a single wire* w *and produces wires* (w, w).
- **NAnd** *consumes wires* (u, v)*, where* u *and* v *are single wires carrying bits* x *and* y*, and its output is a single wire that carries the bit* $x \uparrow y$*.*

For composite circuits, assume C_0 *is a circuit that takes* u *as input wires and produces output wires* w*, and* C_1 *a circuit that takes* w *as input wires and produces output wires* v*. Then*

- $C_0 \ggg C_1$ *is a circuit that takes input* u *and produces output* v*, obtained by first applying* C_0 *on* u *to get an intermediate result* w*, and then applying* C_1 *on* w *to get* v*.*
- $\mathbf{First}(C_0)$ *is a circuit that takes input wires* (u, u') *and produces output wires* (w, u') *for any wires* u'*, where* w *is the output of* C_0 *on input* u*, and* u' *is left unchanged by the circuit.*

To evaluate a circuit, we define the function $\mathbf{Ev}(C, w)$ that takes a circuit $C \in \mathrm{Circuit}(s, t)$ and a wire bundle w of shape s, and return a bundle of shape t according to the above semantics. For simplicity, we usually just write $C(x)$ for the boolean value carried on the wires $u = \mathbf{Ev}(C, w)$ where x is the value carried on w.

We remark that the circuit concatenation operation $\ggg$ is associative, i.e., $(C_0 \ggg C_1) \ggg C_2$ and $C_0 \ggg (C_1 \ggg C_2)$ produce the same circuit. So, we may omit the parentheses when writing a sequence of concatenations $C_0 \ggg C_1 \ggg C_2$.

For a circuit C, we say that C' is a *sub-circuit* of C if one of the following holds:

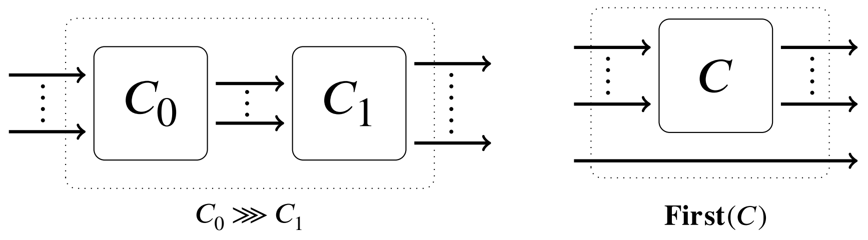


Figure 4.4. Composite circuits $C_0 \ggg C_1$ and $\mathbf{First}(C)$ using operations $\ggg$ and **First** on circuits C_0, C_1, C. Dotted lines draw the boundaries of composite circuits.

- $C' = C$, or
- $C = C_0 \ggg C_1$ and C' is a sub-circuit of C_0 or C_1, or
- $C = \mathbf{First}(C_0)$ and C' is a sub-circuit of C_0.

Example 1. To illustrate our circuit notation, consider the function $f((x, y), z) = (x \wedge y, y \to z)$, where $y \to z \equiv \neg y \vee z$ is the logical implication operation. First we define an operation **Second** on circuits such that $\mathbf{Second}(C)$ is a circuit that takes as input a wire bundle (u, v) and produces as output a bundle (u, w), where v is the input of C and w is the output of C:

$$\mathbf{Second}(C) = \mathbf{Swap} \ggg \mathbf{First}(C) \ggg \mathbf{Swap}$$

Since $x \uparrow x = \neg x$, the circuit $\mathbf{Not} = \mathbf{Dup} \ggg \mathbf{NAnd}$ computes the negation of an input bit, and the circuit $\mathbf{And} = \mathbf{NAnd} \ggg \mathbf{Not} = \mathbf{NAnd} \ggg \mathbf{Dup} \ggg \mathbf{NAnd}$ computes the function $(x, y) \mapsto (x \wedge y)$. Since $y \to z = (\neg y) \vee z = y \uparrow (\neg z)$, the circuit $\mathbf{Imp} = \mathbf{Second}(\mathbf{Not}) \ggg \mathbf{NAnd}$ computes the function $(y, z) \mapsto (y \to z)$. Putting them together, we obtain a circuit

$$\begin{aligned} C = {} & \mathbf{First}(\mathbf{Second}(\mathbf{Dup}) \ggg \mathbf{Assoc}) \ggg \mathbf{Unassoc} \\ & \ggg \mathbf{First}(\mathbf{And}) \ggg \mathbf{Second}(\mathbf{Imp}) \end{aligned}$$

for the function $f((x, y), z) = (x \wedge y, y \to z)$, illustrated graphically in Fig. 4.5. Notice how the first part of the computation consisting of the **Dup**, **Assoc** and **Unassoc** gates is used to route the input wires to the appropriate subcircuit.

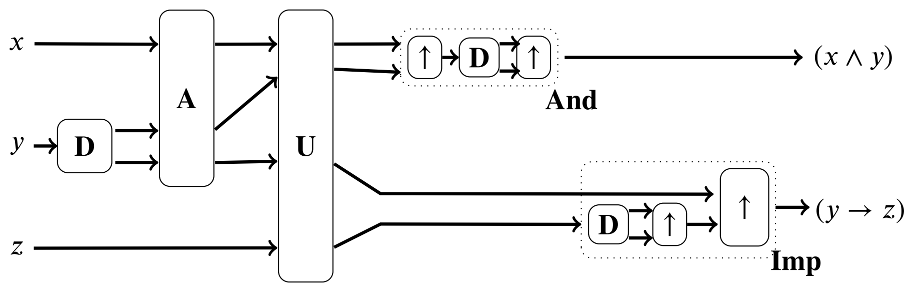


Figure 4.5. The circuit that computes the function $f((x, y), z) = (x \wedge y, y \to z)$.

Remark 1. With our circuit notation, a circuit with q gates and p wires can be represented using a string of size $O(qd \log q)$, where d is the depth of the circuit. We can convert the traditional DAG-like circuit notation to our inductive circuit representation by organizing gates into layers according to their depth. For a layer with q_i gates, the computation of these gates can be described using $q_i \log q_i$ many **First** and **Second** operations together with q_i basic circuits. To rearrange wires after a layer of q_i gates, we can add $O(q_i \log q_i)$ many **Swap**, **Assoc**, and **Unassoc** gates. The entire circuit can be concatenated from layers using $\ggg$ operations. So the size of such representation is $O(qd \log q)$.

4.3 Symbolic Garbled Circuits

We recall the definition of circuit garbling schemes in the computational setting [6,48].

Definition 11 (Syntax)**.** *A* garbling scheme *is a pair of PPT algorithms* $(\mathtt{Garble}, \mathtt{GEval})$[3] *where*

- $\mathtt{Garble}(C, x) = (\widetilde{C}, \tilde{x})$*: The circuit garbling algorithm takes a circuit C and a boolean vector x as input, and it produces a garbled circuit* $\widetilde{C}$ *and a garbled input* $\tilde{x}$*.*

[3]Usually a garbling scheme consists of three algorithms $(\mathtt{GCircuit}, \mathtt{GInput}, \mathtt{GEval})$ such that $\mathtt{GCircuit}(C) = (\widetilde{C}, L)$ produces a garbled circuit $\tilde{C}$ and labels L for the input wires, and $\mathtt{GInput}(L, x) = \tilde{x}$ produces garbled input $\tilde{x}$ using the labels. Such a syntax is useful to define adaptive security. However, we choose a simplified syntax of two algorithms that is sufficient to define selective security and convenient for our analysis.

- $\texttt{GEval}(\widetilde{C}, \tilde{x}) = y$: *The garbled circuit evaluation algorithm takes a garbled circuit* $\widetilde{C}$ *and a garbled input* $\tilde{x}$ *as input, and it produces a boolean vector* y *as output.*

Definition 12 (Correctness and security)**.** *For a garbling scheme* $(\texttt{Garble}, \texttt{GEval})$*, we say that*

- *it is* correct *if* $\texttt{GEval}(\texttt{Garble}(C, x)) = C(x)$ *for all circuits* C *and boolean vectors* x*;*
- *it is* (selectively) secure *if there exists a PPT simulator* $\texttt{Simulate}(\cdot, \cdot)$ *such that for any circuit* C *and input* x*, the distributions* $\texttt{Simulate}(C, C(x))$ *and* $\texttt{Garble}(C, x)$ *are computationally indistinguishable.*

Strictly speaking, a simulator should not gain access to a circuit, and instead, it should take the topology of a circuit as input. To simplify discussion, we use the actual circuit as its topology representation rather than introducing new notations. This can be justified by the facts that 1) there is only one primitive gate in our circuit notation, namely the NAND gate, and 2) our simulator (defined later) does not exploit the function computed by the NAND gate.

Symbolic garbled circuit We consider garbling schemes where the output of all algorithms `Garble`, `GEval`, and `Simulate` are expressions in our symbolic language **Exp**. This will allow us to analyze both the correctness and security properties of the scheme in a purely symbolic manner, without resorting to the power (and complications) of the full computational model of cryptography. The circuit garbling construction described here is essentially the one with the point-and-permute technique as described in [4]. In this section we present `Garble` and `GEval`, and we will define `Simulate` and prove security in the next section.

Let ϵ denote a special symbolic expression whose computational evaluation is the empty string. We slightly change the notation of atomic key symbols by using both subscripts and superscripts to index them: an atomic key is a symbol K_i^j where $i \in \{1, 2, \dots\}$ and $j \in \{0, 1\}$. With this notation, the set of atomic keys is now $\mathbf{K} = \{\mathsf{K}_1^0, \mathsf{K}_1^1, \mathsf{K}_2^0, \mathsf{K}_2^1, \dots\}$. To hide the input of a circuit, the garbling algorithm encodes values carried on wires using *labels* of shape $(\!|\mathbb{B}, (\!|\mathbb{K}, \mathbb{K}|\!)|\!)$, one for each wire. We call a bundle of labels a *label expression*.

Formally, we first define a function `Label` that on input a bundle shape s, outputs a collection of wire labels:

$$\begin{aligned}
&\texttt{Label}(\circ) = (\mathsf{B}_h, (\mathsf{K}^0_h, \mathsf{K}^1_h)) \text{ where} \\
&\qquad h \leftarrow \textbf{new} \\
&\texttt{Label}((s,t)) = (\texttt{Label}(s), \texttt{Label}(t))
\end{aligned}$$

The instruction $h \leftarrow \textbf{new}$ picks a fresh index h (e.g., using a counter), used to define a new symbolic label $(\mathsf{B}_h, (\mathsf{K}^0_h, \mathsf{K}^1_h))$.

A garbled input has two parts: an encoded input expression that is a bundle of shape $(\mathbb{B}, \mathbb{K})$, and an output mask expression that is a bundle of bits. The function `GEnc` encodes a boolean vector using bits and keys in a label expression:

$$\begin{aligned}
&\texttt{GEnc}((\mathsf{B}, (\mathsf{K}^0, \mathsf{K}^1)), 0) = (\mathsf{B}, \mathsf{K}^0) \\
&\texttt{GEnc}((\mathsf{B}, (\mathsf{K}^0, \mathsf{K}^1)), 1) = (\neg\mathsf{B}, \mathsf{K}^1) \\
&\texttt{GEnc}((L_0, L_1), (x_0, x_1)) = (\texttt{GEnc}(L_0, x_0), \texttt{GEnc}(L_1, x_1))
\end{aligned}$$

The output masks are used to decode an encoded expression. It is formed by the bits in a label expression:

$$\begin{aligned}
&\texttt{GMask}((\mathsf{B}, (\mathsf{K}^0, \mathsf{K}^1))) = \mathsf{B} \\
&\texttt{GMask}((L_0, L_1)) = (\texttt{GMask}(L_0), \texttt{GMask}(L_1))
\end{aligned}$$

The core of the garbling algorithm is a recursive function `Gb`, which takes as input a circuit and a label expression for the input wires, and outputs a symbolic expression of the garbled circuit and a label expression for the output wires.

$$\mathtt{Gb} :: \mathrm{Circuit}(s,t) \times \mathbf{Exp} \to \mathbf{Exp} \times \mathbf{Exp}$$

$$\mathtt{Gb}(\mathbf{Swap}, (u, v)) = \epsilon, (v, u)$$

$$\mathtt{Gb}(\mathbf{Assoc}, (u, (v, w))) = \epsilon, ((u, v), w)$$

$$\mathtt{Gb}(\mathbf{Unassoc}, ((u, v), w)) = \epsilon, (u, (v, w))$$

$$\mathtt{Gb}(C_0 \ggg C_1, u) = (\widetilde{C}_0, \widetilde{C}_1), v \text{ where}$$

$$\quad \widetilde{C}_0, w = \mathtt{Gb}(C_0, u)$$

$$\quad \widetilde{C}_1, v = \mathtt{Gb}(C_1, w)$$

$$\mathtt{Gb}(\mathbf{First}(C), (u, w)) = \widetilde{C}, (v, w) \text{ where}$$

$$\quad \widetilde{C}, v = \mathtt{Gb}(C, u)$$

$$\mathtt{Gb}(\mathbf{Dup}, (b, (k^0, k^1))) = \epsilon, w \text{ where}$$

$$\quad w = ((b, \mathsf{G}_0(k^0), \mathsf{G}_0(k^1)), (b, \mathsf{G}_1(k^0), \mathsf{G}_1(k^1)))$$

$$\mathtt{Gb}(\mathbf{NAnd}, ((b_i, (k_i^0, k_i^1)), (b_j, (k_j^0, k_j^1)))) = \widetilde{C}, w \text{ where}$$

$$\quad h \leftarrow \mathbf{new}$$

$$\quad \widetilde{C} = \pi[b_i](\pi[b_j](\{\!\!\{\{\!\!\{(\neg\mathsf{B}_h, \mathsf{K}_h^1)\}\!\!\}_{k_j^0}\}\!\!\}_{k_i^0}, \{\!\!\{\{\!\!\{(\neg\mathsf{B}_h, \mathsf{K}_h^1)\}\!\!\}_{k_j^1}\}\!\!\}_{k_i^0}),$$

$$\qquad \pi[b_j](\{\!\!\{\{\!\!\{(\neg\mathsf{B}_h, \mathsf{K}_h^1)\}\!\!\}_{k_j^0}\}\!\!\}_{k_i^1}, \{\!\!\{\{\!\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\!\}_{k_j^1}\}\!\!\}_{k_i^1}))$$

$$\quad w = (\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$$

The full garbling procedure can be obtained by composing the above functions. On input a circuit C and a boolean vector x, it picks random labels for the input wires using `Label`, calls `Gb` to generate a garbled circuit $\widetilde{C}$ and output labels, and then calls `GEnc` and `GMask` to produce a garbled input $\tilde{x}$. Note that the second parameter of `GEnc` is a bundle of bits rather than a boolean vector. In the definition of `Garble` below we slightly abuse notation and use x to denote a bundle of bits $x_1, \dots, x_n$ of a suitable shape, which can be efficiently constructed from x and s.

```
$\texttt{Garble} :: \text{Circuit}(s,t) \times \{0,1\}^n \to \mathbf{Exp}$
$\texttt{Garble}(C,x) = (\widetilde{C}, \tilde{x})$ where
    $u \leftarrow \texttt{Label}(s)$
    $\widetilde{C}, v = \texttt{Gb}(C,u)$
    $\tilde{x} = (\texttt{GEnc}(u,x), \texttt{GMask}(v))$
```

Next, we consider the garbled circuit evaluation algorithm GEval. The core part of GEval is a recursive function GEv that takes a garbled circuit and an encoded input expression, producing an encoded output expression. Any encoded output is also an encoded input for evaluating subsequent garbled circuits. We include a circuit as another input of GEv, which is used to determine the shapes of output wires. Ideally we can use the circuit's topology instead, but for simplicity we just use the circuit itself and we do not exploit the function computed by a circuit.

```
$\texttt{GEv} :: \text{Circuit}(s,t) \times \mathbf{Exp} \times \mathbf{Exp} \to \mathbf{Exp}$
$\texttt{GEv}(\mathbf{Swap}, \epsilon, (u,v)) = (v,u)$
$\texttt{GEv}(\mathbf{Assoc}, \epsilon, (u,(v,w))) = ((u,v),w)$
$\texttt{GEv}(\mathbf{Unassoc}, \epsilon, (u,(v,w))) = ((u,v),w)$
$\texttt{GEv}(\mathbf{Dup}, \epsilon, (b,k) = ((b, \mathsf{G}_0(k)), (b, \mathsf{G}_1(k)))$
$\texttt{GEv}(\mathbf{NAnd}, \widetilde{C}, ((b'_0,k_0),(b'_1,k_1))) = (b,k)$ where
    $\pi[b_0](r_0,r_1) \quad = \widetilde{C}$
    $\pi[b_1](e_0,e_1) \quad = \text{if } b'_0 \equiv b_0 \text{ then } r_0 \text{ else } r_1$
    $\{\!\!\{ \{\!\!\{ (b,k) \}\!\!\}_{k_1} \}\!\!\}_{k_0} = \text{if } b'_1 \equiv b_1 \text{ then } e_0 \text{ else } e_1$
$\texttt{GEv}(C_0 \ggg C_1, (\widetilde{C}_0, \widetilde{C}_1), u) = \texttt{GEv}(C_1, \widetilde{C}_1, w)$ where
    $w = \texttt{GEv}(C_0, \widetilde{C}_0, u)$
$\texttt{GEv}(\mathbf{First}(C), \widetilde{C}, (u,w)) = (v,w)$ where
    $v = \texttt{GEv}(C, \widetilde{C}, u)$
```

We briefly explain how GEv works. For the basic circuits **Swap**, **Assoc**, **Unassoc**, and **Dup** whose corresponding garbled circuits are ϵ, it simply rearranges the bits and keys in the encoded input to form an encoded output, except for **Dup** where it generates and then splits a pseudo-random key in the encoded output. For **NAnd**, it parses the corresponding garbled circuit as permutations controlled by atomic bits b_0, b_1, and it selects the entry corresponding to the bits b'_0, b'_1. In the above definition, we use pattern matching syntax that is usually found in functional programming languages to parse $\widetilde{C}$ and select the subexpression $\{\!\!\{ \{\!\!\{ (b,k) \}\!\!\}_{k_1} \}\!\!\}_{k_0}$. One can verify that, if (b'_i, k_i) is in the encoded input to **NAnd** for $i \in \{0,1\}$, then $b'_i \in \{b_i, \neg b_i\}$ and the entry selected using bits b'_0, b'_1 are doubly encrypted under keys k_0,k_1. So the expression (b,k) extracted by GEv is well-defined. For the composite circuits $C_0 \ggg C_1$ and $\mathbf{First}(C)$, GEv produces an encoded output expression recursively in a way similar to how **Ev** evaluates these circuits.

Notice that the output of GEv are bit symbols rather than boolean values. The function Decode uses the output masks to decode a garbled output into a boolean vector:

$$\begin{aligned}&\texttt{Decode}((b,k),b') = \text{if } b \equiv b' \text{ then } 0 \text{ else } 1\\&\texttt{Decode}((u_0,u_1),(d_0,d_1)) = (\texttt{Decode}(u_0,d_0),\texttt{Decode}(u_1,d_1))\end{aligned}$$

Finally, the full evaluation algorithm GEval is defined as[4]:

$$\begin{aligned}&\texttt{GEval} :: \mathrm{Circuit}(s,t) \times \mathbf{Exp} \times \mathbf{Exp} \to \{0,1\}^n\\&\texttt{GEval}(C,\widetilde{C},\tilde{x}) = \texttt{Decode}(\texttt{GEv}(C,\widetilde{C},u),d) \text{ where}\\&\quad (u,d) = \tilde{x}\end{aligned}$$

The following theorem shows that our garbling scheme is correct. Briefly speaking, the encoded input expressions contain the sufficient bits and keys to obtain the encoded output from the garbled circuit expression, and the output masks provide information for decoding the encoded output.

Theorem 7. *For any circuit $C \in \mathrm{Circuit}(s,t)$ and any boolean vector x of shape s, we have* $\texttt{GEval}(C,\texttt{Garble}(C,x)) = C(x)$.

[4]Notice that Decode outputs a bundle of bits. Here we slightly abuse notation and assume a boolean vector can be extracted from a bundle of bits.

4.4 Symbolic Simulation of Garbled Circuits

In this section we define a simulator $\texttt{Simulate}(\cdot, \cdot)$, and we present our proof that, for any circuit C and any boolean vector x, the expressions $\texttt{Garble}(C, x)$ and $\texttt{Simulate}(C, C(x))$ are equivalent up to renaming. Together with the computational soundness theorem of our symbolic framework, such proof implies that the garbled circuit scheme of the previous section is computationally secure.

Symbolic simulator Recall that a simulator must output a symbolic expression that represents a garbled circuit and a garbled input, and a garbled input consists of an encoded input and output masks. The simulator has no access to the circuit input values, so it picks the random bit and the first random key from each label to form the encoded input:

$$\texttt{SEnc}((\mathsf{B}, (\mathsf{K}^0, \mathsf{K}^1))) = (\mathsf{B}, \mathsf{K}^0)$$
$$\texttt{SEnc}((L_0, L_1)) = (\texttt{SEnc}(L_0), \texttt{SEnc}(L_1))$$

In order to correctly evaluate the simulated garbled circuit on the simulated garbled input, we adjust the output masks according to the circuit output value. Given a label expression and a boolean vector representing the circuit output value, the function $\texttt{SMask}$ computes the output masks:

$$\texttt{SMask}((\mathsf{B}, (\mathsf{K}^0, \mathsf{K}^1)), 0) = \mathsf{B}$$
$$\texttt{SMask}((\mathsf{B}, (\mathsf{K}^0, \mathsf{K}^1)), 1) = \neg\mathsf{B}$$
$$\texttt{SMask}((L_0, L_1), (y_0, y_1)) = (\texttt{SMask}(L_0, y_0), \texttt{SMask}(L_1, y_1))$$

The core of our simulator is a recursive function $\texttt{Sim}$ that consumes a circuit and a label expression for input wires, and produces a symbolic expression of the simulated garbled circuit and a label expression for output wires:

$$\texttt{Sim} :: \text{Circuit}(s,t) \times \textbf{Exp} \to \textbf{Exp} \times \textbf{Exp}$$

$$\texttt{Sim}(\textbf{Swap}, (u, v)) = \epsilon, (v, u)$$

$$\texttt{Sim}(\textbf{Assoc}, (u, (v, w))) = \epsilon, ((u, v), w)$$

$$\texttt{Sim}(\textbf{Unassoc}, ((u, v), w)) = \epsilon, (u, (v, w))$$

$$\texttt{Sim}(C_0 \ggg C_1, u) = (\widehat{C}_0, \widehat{C}_1), v \text{ where}$$

$$\quad \widehat{C}_0, w = \texttt{Sim}(C_0, u)$$

$$\quad \widehat{C}_1, v = \texttt{Sim}(C_1, w)$$

$$\texttt{Sim}(\textbf{First}(C), (u, w)) = \widehat{C}, (v, w) \text{ where } \widehat{C}, v = \texttt{Sim}(C, u)$$

$$\texttt{Sim}(\textbf{Dup}, (b, (k^0, k^1))) = \epsilon, w \text{ where}$$

$$\quad w = ((b, (\mathsf{G}_0(k^0), \mathsf{G}_0(k^1))), (b, (\mathsf{G}_1(k^0), \mathsf{G}_1(k^1))))$$

$$\texttt{Sim}(\textbf{NAnd}, ((b_i, (k_i^0, k_i^1)), (b_j, (k_j^0, k_j^1)))) = \widehat{C}, w \text{ where}$$

$$\quad h \leftarrow \textbf{new}$$

$$\quad \widehat{C} = \pi[\mathsf{B}_i](\pi[\mathsf{B}_j](\{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^0}\}\!\}_{k_i^0}, \{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^1}\}\!\}_{k_i^0}),$$
$$\qquad\qquad \pi[\mathsf{B}_j](\{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^0}\}\!\}_{k_i^1}, \{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^1}\}\!\}_{k_i^1}))$$

$$\quad w = (\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$$

Notice that, for any circuit C and any label expression u, if $\widetilde{C}, v = \texttt{Gb}(C, u)$ and $\widehat{C}, w = \texttt{Sim}(C, u)$, then the subscript h of any atomic key symbol K_h^i that appears in $(\widetilde{C}, v)$ and $(\widehat{C}, w)$ follows the same ordering.

Our simulator is composed of the above functions. It takes a circuit C and a boolean vector y as input, and it generates a simulated garbled circuit using Sim and a simulated garbled input using SEnc and SMask:

$$\texttt{Simulate} :: \text{Circuit}(s,t) \times \{0,1\}^m \to \textbf{Exp}$$

$$\texttt{Simulate}(C, y) = (\widehat{C}, \hat{x}) \text{ where}$$

$$\quad u \leftarrow \texttt{Label}(s)$$

$$\quad \widehat{C}, v = \texttt{Sim}(C, u)$$

$$\quad \hat{x} = (\texttt{SEnc}(u), \texttt{SMask}(v, y))$$

Symbolic proof of security For this paper we present a pen-and-paper symbolic security proof, which can also be adapted to a machine-checked proof using verification tools. For any bit expression $b \in \mathbf{Pat}(\mathbb{B})$ and any $x \in \{0,1\}$, we introduce the notation $b^{\oplus x}$ to shorten our proofs:

$$b^{\oplus x} = \begin{cases} b & \text{if } x = 0 \\ \neg b & \text{if } x = 1 \end{cases}$$

We say that a label expression w is *strongly independent* if $\mathbf{Keys}(w)$ is a set of independent keys and, if $w = (b, (k^0, k^1))$ is a single label then $k^0 \neq k^1$, and if $w = (u, v)$ where u and v are label expressions, then u and v are both strongly independent and $\mathbf{Keys}(u) \cap \mathbf{Keys}(v) = \emptyset$.

Let us start with some technical lemmas that are helpful to derive our main result. The first lemma can be easily verified by induction on the definition of $\mathtt{Gb}$.

Lemma 7. *For any circuit C and label expression u, if $\widetilde{C}, v = \mathtt{Gb}(C, u)$ and $k \in \mathbf{Keys}(\widetilde{C}) \cap \mathbf{Parts}(\widetilde{C})$, then $k \in \mathbf{K}$ is an atomic key symbol.*

Our next lemma shows that $\mathtt{Gb}$ produces strongly independent output labels from strongly independent input labels. Furthermore, any key in the output label expression is yielded from either a new atomic key introduced in the garbled circuit or a key in the input labels, and it does not yield any other key in the garbled circuit. The formal proof is done using structural induction on circuits, and it is omitted due to space constraint.

Lemma 8. *For any circuit C and any strongly independent label expression u such that $\widetilde{C}, v = \mathtt{Gb}(C, u)$, v is strongly independent, and the following hold for all $k \in \mathbf{Keys}(v)$:*

1. $\mathsf{G}^+(k) \cap \mathbf{Keys}((\widetilde{C}, u)) = \emptyset$;
2. $\exists k' \in \mathbf{Keys}((\widetilde{C}, u)) \cap \mathbf{Parts}((\widetilde{C}, u)). k' \preceq k$.

A quick observation on $\mathtt{Gb}$ is that, for any circuit C, if $k \in \mathbf{Pat}(\mathbb{K})$ appears in $\widetilde{C}$, then either k is in a plaintext message and so $k \in \mathbf{Parts}(\widetilde{C})$, or k is used as an encryption key. The former case has been considered in Lemma 7. The following lemma characterizes the latter case, and it can be proved using structural induction on circuits.

Lemma 9. *For any circuit C and any label expression u such that u is strongly independent and $\widetilde{C}, v = \mathtt{Gb}(C, u)$, if $\{\![e]\!\}_k \in \mathbf{Parts}(\widetilde{C})$ for some expression e and some key $k \in \mathbf{Pat}(\mathbb{K})$, then the following hold:*

1. $\mathsf{G}^{+}(k) \cap \mathbf{Keys}(\widetilde{C}) = \emptyset$*;*

2. $\mathsf{G}^{*}(k) \cap \mathbf{Keys}(v) = \emptyset$*;*

3. $\exists k' \in \mathbf{Keys}((\widetilde{C}, u)) \cap \mathbf{Parts}((\widetilde{C}, u)).k' \preceq k$*.*

For the rest of paper, let us fix a circuit $C \in \mathrm{Circuit}(s, t)$ and a boolean vector $x \in \{0, 1\}^n$, where s is a shape of n wires and t is a shape of m wires. Let $e = (\widetilde{C}, \tilde{x}) = \mathtt{Garble}(C, x)$ be the symbolic expression of the garbled circuit and the garbled input of C on input x. Since $\mathcal{F}_e$ is monotone, the greatest fixed point of $\mathcal{F}_e$ exists and it can be computed in polynomially many steps. Let $S = \mathrm{Fix}(\mathcal{F}_e)$ and $e' = \mathbf{p}(e, S)$. Then $\mathbf{Roots}(S) \subseteq \mathbf{Keys}(e)$ and $S = \mathcal{F}_e(S) = \mathbf{r}(\mathbf{p}(e, S)) = \mathbf{r}(e')$. For any label $(b, (k^0, k^1))$, we say that it satisfies the *label invariant* if

$$b \in \mathbf{B}, \exists z \in \{0, 1\} \text{ such that } k^z \in S, k^{1-z} \notin S, \tag{4.1}$$

and we call z the *actual value* of the label $(b, (k^0, k^1))$.

Lemma 10. *For any sub-circuit C' of C, and for any label expression u, if $\widetilde{C}', v = \mathtt{Gb}(C', u)$ and all labels $(b, (k^0, k^1)) \Subset u$ satisfy the label invariant, then all labels $(\bar{b}, (\bar{k}^0, \bar{k}^1)) \Subset v$ satisfy the label invariant.*

Proof. We use induction on the structure of circuit C'. For the base case, C' is an atomic circuit:

- $C' = \mathbf{Swap}$, $\mathbf{Assoc}$, or $\mathbf{Unassoc}$: Any label $(\bar{b}, (\bar{k}^0, \bar{k}^1)) \Subset v$ is also a sub-expression of u. So the lemma holds.

- $C' = \mathbf{Dup}$: Suppose $u = (b, (k^0, k^1))$ satisfies the label invariant with an actual value z. If $(\bar{b}, (\bar{k}^0, \bar{k}^1)) \Subset v$, then $\bar{b} = b$, $\bar{k}^0 = \mathsf{G}_h(k^0)$, and $\bar{k}^1 = \mathsf{G}_h(k^1)$ for some $h \in \{0, 1\}$. So $\bar{b} \in \mathbf{B}$. Let $\bar{z} = z$. Then $\bar{k}^{\bar{z}} = \mathsf{G}_h(k^z) \in S$. Assume towards a contradiction that $\bar{k}^{1-\bar{z}} \in S$. Then $\mathsf{G}_h(k^{1-z}) = \bar{k}^{1-\bar{z}} \in \mathsf{G}^*(k')$ for some $k' \in \mathbf{Keys}(e')$ where $k' \in \mathbf{Parts}(e')$ or $\exists k'' \in \mathbf{Keys}(e')$ such that $k' \prec k''$. Notice that $e' = \mathbf{p}((\widetilde{C}, \tilde{x}), S) = (\mathbf{p}(\widetilde{C}, S), \mathbf{p}(\tilde{x}, S))$, and $\tilde{x}$ contains only atomic keys. So $k'' \in \mathbf{Keys}(\widetilde{C'}) \subseteq \mathbf{Keys}(\widetilde{C})$. We have two cases:

  - $\mathsf{G}_h(k^{1-z}) \neq k'$: $k^{1-z} \in \mathsf{G}^*(k') \subseteq S$, a contradiction.

  - $\mathsf{G}_h(k^{1-z}) = k'$: Now $k' \notin \mathbf{Parts}(e')$, and thus $\{\!|g'|\!\}_{k'} \in \mathbf{Parts}(e')$ for some pattern g'. So $\{\!|g'|\!\}_{k'} \in \mathbf{Parts}(\mathbf{p}(\widetilde{C}, S))$ and $\{\!|g|\!\}_{k'} \in \mathbf{Parts}(\widetilde{C})$ for some expression g such that $g' = \mathbf{p}(g, S)$. By Lemma 9, $\mathsf{G}^+(k') \cap \mathbf{Keys}(\widetilde{C}) = \emptyset$ and hence $k'' \notin \mathbf{Keys}(\widetilde{C})$, a contradiction.

  Therefore $(\bar{b}, (\bar{k}^0, \bar{k}^1))$ satisfies the label invariant.

- $C' = \mathbf{NAnd}$: The only label in v is $(\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$. Notice that the expressions in $\mathbf{Parts}(e)$ that contain $\mathsf{K}_h^0, \mathsf{K}_h^1$ are the following and their sub-expressions:

$$\{\!|\{\!|(\neg\mathsf{B}_h, \mathsf{K}_h^1)|\!\}_{k_j^0}|\!\}_{k_i^0}, \{\!|\{\!|(\neg\mathsf{B}_h, \mathsf{K}_h^1)|\!\}_{k_j^0}|\!\}_{k_i^1},$$
$$\{\!|\{\!|(\neg\mathsf{B}_h, \mathsf{K}_h^1)|\!\}_{k_j^1}|\!\}_{k_i^0}, \{\!|\{\!|(\mathsf{B}_h, \mathsf{K}_h^0)|\!\}_{k_j^1}|\!\}_{k_i^1},$$

  where $((b_i, (k_i^0, k_i^1)), (b_j, (k_j^0, k_j^1))) = u$. Observe that these four expressions can be generated as

$$\{\!|\{\!|(\mathsf{B}_h^{\oplus(x_i \uparrow x_j)}, \mathsf{K}_h^{x_i \uparrow x_j})|\!\}_{k_j^{x_j}}|\!\}_{k_i^{x_i}} \text{ for } x_i, x_j \in \{0, 1\}.$$

  Let $\bar{z} = z_i \uparrow z_j$. By assumption, we have $k_i^{z_i}, k_j^{z_j} \in S$ and $k_i^{1-z_i}, k_j^{1-z_j} \notin S$, so $k^{\bar{z}} = \mathsf{K}_h^{\bar{z}} \in S$ and $k^{1-\bar{z}} = \mathsf{K}_h^{1-\bar{z}} \notin S$, and Condition 4.1 holds for $(\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$.

Next, consider composite circuits. Assume the lemma holds for all sub-circuits of C'. Then we have these cases:

- $C' = C'_0 \ggg C'_1$: Suppose $\widetilde{C} = (\widetilde{C}'_0, \widetilde{C}'_1)$ where $\widetilde{C}'_0, w = \texttt{Gb}(C'_0, u)$ and $\widetilde{C}'_1, v = \texttt{Gb}(C'_1, w)$. Since C'_0 and C'_1 are both sub-circuits of C', by assumption we see that Condition 4.1 holds for all labels in u and consequently, for all labels in w, and so it holds for all labels in v.

- $C' = \mathbf{First}(C'')$: Suppose $u = (u'', w)$ and $v = (v'', w)$ such that $\widetilde{C}, v'' = \texttt{Gb}(C'', u'')$. For any label $(\bar{b}, (\bar{k}^0, \bar{k}^1)) \Subset v$, it is either a sub-expression of v'' or it is a sub-expression of w. For the former case, since C'' is a sub-circuit of C', Condition 4.1 holds for $(\bar{b}, (\bar{k}^0, \bar{k}^1))$ by induction hypothesis. For the latter case, since $w \Subset u$, Condition 4.1 holds for this label by assumption.

Therefore the lemma holds for any circuit C. □

Let $f = (\widehat{C}, \hat{x}) = \texttt{Simulate}(C, C(x))$ be the symbolic expression of simulated garbled circuit of C on output $C(x)$. Let $T = \text{Fix}(\mathcal{F}_f)$, which satisfies $\mathcal{F}_f(T) = \mathbf{r}(\mathbf{p}(f, T)) = T$. The following lemma shows that, for each key pair k^0, k^1 in f, exactly one of k^0 and k^1 is in T.

Lemma 11. *For any sub-circuit C' of C and any label expression u such that $\widehat{C}', v = \texttt{Sim}(C', u)$, if all labels $(b, (k^0, k^1)) \Subset u$ satisfy the label invariant with actual value 0, then all labels $(\bar{b}, (\bar{k}^0, \bar{k}^1)) \Subset v$ satisfy the label invariant with actual value 0.*

Proof. We can directly apply the proof of Lemma 10 except for the base case when $C' = \mathbf{NAnd}$:

- $C' = \mathbf{NAnd}$: The label in v is $(\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$. The expressions in $\mathbf{Parts}(f)$ that contain $\mathsf{K}_h^0, \mathsf{K}_h^1$ are the following and their sub-expressions:

$$\{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^0}\}\!\}_{k_i^0}, \{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^0}\}\!\}_{k_i^1},$$
$$\{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^1}\}\!\}_{k_i^0}, \{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^1}\}\!\}_{k_i^1},$$

where $((b_i, (k_i^0, k_i^1)), (b_j, (k_j^0, k_j^1))) = u$. Let $\bar{z} = 0$. By assumption, $k_i^0, k_j^0 \in T$ and $k_i^1, k_j^1 \notin T$. So $k^{\bar{z}} = \mathsf{K}_h^0 \in T$ and $k^{1-\bar{z}} = \mathsf{K}_h^1 \notin T$, and Condition 4.1 holds for $(\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$ with actual value 0.

For the rest of the cases, the proof of Lemma 10 applies with actual value 0. $\square$

Now we are ready to prove our main result that the patterns of the real garbled circuit and the simulated garbled circuit are equivalent up to renaming.

Theorem 8. *For any circuit $C \in \mathrm{Circuit}(s, t)$ and any boolean vector $x \in \{0, 1\}^n$, where s is a shape of n wires,* $\mathbf{Pattern}(\mathtt{Garble}(C, x)) \approx \mathbf{Pattern}(\mathtt{Simulate}(C, C(x)))$.

Proof. Let $u = ((\mathsf{B}_1, (\mathsf{K}_1^0, \mathsf{K}_1^1)), \dots, (\mathsf{B}_n, (\mathsf{K}_n^0, \mathsf{K}_n^1)))$ be the label expression in $\mathtt{Garble}$. Let $\widetilde{C}, v = \mathtt{Gb}(C, u)$. One can check that, for any sub-circuit C' of C, if $\widetilde{C'}, v' = \mathtt{Gb}(C', u')$ and $\widehat{C'}, w' = \mathtt{Sim}(C', u')$ for any label expression u' of an appropriate shape, then $v' = w'$. Since $\mathtt{Sim}$ is also applied on C and u in $\mathtt{Simulate}$, we can write $\widehat{C}, v = \mathtt{Sim}(C, u)$.

Let $e = (\widetilde{C}, \tilde{x}) = \mathtt{Garble}(C, x)$, $f = (\widehat{C}, \hat{x}) = \mathtt{Simulate}(C, C(x))$, $S = \mathrm{Fix}(\mathcal{F}_e)$, and $T = \mathrm{Fix}(\mathcal{F}_f)$. We can write $\widetilde{C} = (\widetilde{C}_1, \dots, \widetilde{C}_q)$ and $\widehat{C} = (\widehat{C}_1, \dots, \widehat{C}_q)$, where $\widetilde{C}_i, v_i = \mathtt{Gb}(C_i, u_i)$ and $\widehat{C}_i, v_i = \mathtt{Sim}(C_i, u_i)$ for some atomic sub-circuit C_i of C and some label expression u_i. To show $\mathbf{Pattern}(e) = \mathbf{p}(e, S) \approx \mathbf{p}(f, T) = \mathbf{Pattern}(f)$, we first show $(\mathbf{p}(\widetilde{C}_1, S), \dots, \mathbf{p}(\widetilde{C}_q, S)) \approx (\mathbf{p}(\widehat{C}_1, T), \dots, \mathbf{p}(\widehat{C}_q, T))$ with respect to a pseudorandom renaming $\alpha = (\alpha_B, \alpha_K)$, and then we show $\mathbf{p}(\tilde{x}, S) \approx_\alpha \mathbf{p}(\hat{x}, T)$.

For the first part, let α_B be the random bit renaming $\alpha_B(\mathsf{B}_i) = \mathsf{B}_i^{\oplus z_i}$ for all $\mathsf{B}_i \in \mathbf{B}$, where z_i is the actual value of the label that contains B_i. Let α_K be the bijection on $\mathbf{K}$ such that $\alpha_K(\mathsf{K}_i^{z_i}) = \mathsf{K}_i^0$ and $\alpha_K(\mathsf{K}_i^{1-z_i}) = \mathsf{K}_i^1$ for each $\mathsf{K}_i^0, \mathsf{K}_i^1$. We claim that, for any sub-circuit C' of C and for any label expression u', if $\widetilde{C'}, v' = \mathtt{Gb}(C', u')$ and $\widehat{C'}, v' = \mathtt{Sim}(C', u')$, then Condition 4.1 holds for all labels in v' and $\mathbf{p}(\widetilde{C'}, S) \approx_\alpha \mathbf{p}(\widehat{C'}, T)$.

Proof of claim: Notice that all labels in u satisfy Condition 4.1. By Lemma 10, all labels in v' also satisfy Condition 4.1. We use induction on the structure of C' to show $\mathbf{p}(\widetilde{C'}, S) \approx_\alpha \mathbf{p}(\widehat{C'}, T)$. For the base case, C' is an atomic circuit:

- $C' = $ **Swap**, **Assoc**, **Unassoc**, or **Dup**: Both $\widetilde{C'}$ and $\widehat{C'}$ are the empty garbled circuit ϵ, so $\mathbf{p}(\widetilde{C'}, S) = \mathbf{p}(\widehat{C'}, T)$.

- $C' = \textbf{NAnd}$: Suppose $u' = ((b_i, (k_i^0, k_i^1)), (b_j, (k_j^0, k_j^1)))$ and $v' = (\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$. Let z_i, z_j and z_h be the actual values of the labels $(b_i, (k_i^0, k_i^1))$, $(b_j, (k_j^0, k_j^1))$ and $(\mathsf{B}_h, (\mathsf{K}_h^0, \mathsf{K}_h^1))$, respectively. We know from the proof of Lemma 10 that $z_h = z_i \uparrow z_j$. So we can apply α_K and get

$$\begin{aligned}
\widetilde{C'} = {} & \pi[B_i](\pi[B_j](\{\!\{\{\!\{(\mathsf{B}_h^{\oplus(0\uparrow0)}, \mathsf{K}_h^{0\uparrow0})\}\!\}_{k_j^0}\}\!\}_{k_i^0},\\
& \qquad \{\!\{\{\!\{(\mathsf{B}_h^{\oplus(0\uparrow1)}, \mathsf{K}_h^{0\uparrow1})\}\!\}_{k_j^1}\}\!\}_{k_i^0}),\\
& \quad \pi[B_j](\{\!\{\{\!\{(\mathsf{B}_h^{\oplus(1\uparrow0)}, \mathsf{K}_h^{1\uparrow0})\}\!\}_{k_j^0}\}\!\}_{k_i^1},\\
& \qquad \{\!\{\{\!\{(\mathsf{B}_h^{\oplus(1\uparrow1)}, \mathsf{K}_h^{1\uparrow1})\}\!\}_{k_j^1}\}\!\}_{k_i^1}))\\
\approx_\alpha {} & \pi[B_i^{\oplus z_i}](\pi[B_j^{\oplus z_j}](\{\!\{\{\!\{(\mathsf{B}_h^{\oplus(0\uparrow0)\oplus z_h}, \mathsf{K}_h^{(0\uparrow0)\oplus z_h})\}\!\}_{k_j^0}\}\!\}_{k_i^0},\\
& \qquad \{\!\{\{\!\{(\mathsf{B}_h^{\oplus(0\uparrow1)\oplus z_h}, \mathsf{K}_h^{(0\uparrow1)\oplus z_h})\}\!\}_{k_j^1}\}\!\}_{k_i^0}),\\
& \quad \pi[B_j^{\oplus z_j}](\{\!\{\{\!\{(\mathsf{B}_h^{\oplus(1\uparrow0)\oplus z_h}, \mathsf{K}_h^{(1\uparrow0)\oplus z_h})\}\!\}_{k_j^0}\}\!\}_{k_i^1},\\
& \qquad \{\!\{\{\!\{(\mathsf{B}_h^{\oplus(1\uparrow1)\oplus z_h}, \mathsf{K}_h^{(1\uparrow1)\oplus z_h})\}\!\}_{k_j^1}\}\!\}_{k_i^1}))\\
\equiv {} & \pi[B_i](\pi[B_j](\{\!\{\{\!\{(\mathsf{B}_h^{\oplus\mu(z_i,z_j)}, \mathsf{K}_h^{\oplus\mu(z_i,z_j)})\}\!\}_{k_j^0}\}\!\}_{k_i^0},\\
& \qquad \{\!\{\{\!\{(\mathsf{B}_h^{\oplus\mu(z_i,1-z_j)}, \mathsf{K}_h^{\oplus\mu(z_i,1-z_j)})\}\!\}_{k_j^1}\}\!\}_{k_i^0}),\\
& \quad \pi[B_j](\{\!\{\{\!\{(\mathsf{B}_h^{\oplus\mu(1-z_i,z_j)}, \mathsf{K}_h^{\oplus\mu(1-z_i,z_j)})\}\!\}_{k_j^0}\}\!\}_{k_i^1},\\
& \qquad \{\!\{\{\!\{(\mathsf{B}_h^{\oplus\mu(1-z_i,1-z_j)}, \mathsf{K}_h^{\oplus\mu(1-z_i,1-z_j)})\}\!\}_{k_j^1}\}\!\}_{k_i^1})),
\end{aligned}$$

where $\mu(d_i, d_j) = (d_i \uparrow d_j) \oplus z_h$ for $d_i, d_j \in \{0, 1\}$. In particular, $\mu(z_i, z_j) = 0$. By Condition 4.1, $k_i^{z_i}, k_j^{z_j}, \mathsf{K}_h^{z_h} \in S$, $k_i^{1-z_i}, k_j^{1-z_j}, \mathsf{K}_h^{1-z_h} \notin S$, and $b_i^{\oplus z_i}, b_j^{\oplus z_j} \in \textbf{Parts}(\mathbf{p}(e, S))$. So $k_i^0, k_j^0 \in \alpha(S)$, $k_i^1, k_j^1 \notin \alpha(S)$, and the pattern $\alpha(\mathbf{p}(\widetilde{C'}, S)) = \mathbf{p}(\alpha(\widetilde{C'}), \alpha(S))$ is equivalent to

$$\begin{aligned}
& \pi[B_i](\pi[B_j](\{\!\{\{\!\{(\mathsf{B}_h, \mathsf{K}_h^0)\}\!\}_{k_j^0}\}\!\}_{k_i^0}, \{\!\{\{\!\{(\!|\mathbb{B}, \mathbb{K}|\!)\}\!\}_{k_j^1}\}\!\}_{k_i^0}),\\
& \quad \pi[B_j](\{\!\{\{\!\{(\!|\mathbb{B}, \mathbb{K}|\!)\}\!\}\}\!\}_{k_i^1}, \{\!\{\{\!\{(\!|\mathbb{B}, \mathbb{K}|\!)\}\!\}\}\!\}_{k_i^1}))
\end{aligned}$$

On the other hand, by Lemma 11, $k_i^0, k_j^0, \mathsf{K}_h^0 \in T$ and $k_i^1, k_j^1, \mathsf{K}_h^1 \notin T$. So the pattern $\mathbf{p}(\widehat{C'}, S)$ of $\widehat{C'}$ is

$$\pi[B_i](\pi[B_j](\{\!\{ \{\!\{ (\mathsf{B}_h, \mathsf{K}_h^0) \}\!\}_{k_j^0} \}\!\}_{k_i^0}, \{\!\{ \{\!\{ (\!| \mathbb{B}, \mathbb{K} |\!) \}\!\}_{k_j^1} \}\!\}_{k_i^0},$$
$$\pi[B_j](\{\!\{ \{\!\{ (\!| \mathbb{B}, \mathbb{K} |\!) \}\!\} \}\!\}_{k_j^1}), \{\!\{ \{\!\{ (\!| \mathbb{B}, \mathbb{K} |\!) \}\!\} \}\!\}_{k_j^1}))$$

Thus $\mathbf{p}(\widetilde{C'}, S) \approx_\alpha \mathbf{p}(\widehat{C'}, T)$.

For the induction step, assuming the claim holds for sub-circuits C_0' and C_1' of C, it is easy to check that the claim also holds for the cases $C' = \mathbf{First}(C_0')$ and $C' = C_0' \ggg C_1'$. Therefore our claim follows.

For the second part, let $y = C(x)$. Then for any $i \in [m]$, y_i is the actual value of the corresponding output wire. Since $\tilde{x} = ((\mathsf{K}_1^{x_1}, \mathsf{B}_1^{\oplus x_1}), \dots, (\mathsf{K}_n^{x_n}, \mathsf{B}_n^{\oplus x_n}), (b_1, \dots, b_m))$, we can calculate

$$\alpha(\tilde{x}) = ((\mathsf{K}_1^0, \mathsf{B}_1), \dots, (\mathsf{K}_n^0, \mathsf{B}_n), (b_1^{\oplus y_i}, \dots, b_m^{\oplus y_m})) = \hat{x}.$$

So $\tilde{x} \approx_\alpha \hat{x}$, and thus $\mathbf{p}(\tilde{x}, S) \approx_\alpha \mathbf{p}(\hat{x}, T)$.

Therefore the theorem holds. $\square$

As a corollary of Theorem 6 and 8, we can now conclude that our garbled circuit scheme is computationally secure.

Corollary 1. *For any circuit $C \in \mathrm{Circuit}(s, t)$ and any $x \in \{0, 1\}^n$ where s is a shape of n wires, the probability distributions $[\![\mathtt{Garble}(C, x)]\!]$ and $[\![\mathtt{Simulate}(C, C(x))]\!]$ are computationally indistinguishable.*

4.5 Implementation

As a proof of concept, we have implemented our symbolic framework as well as the garbling scheme and the simulator in Haskell. [5] In our implementation, we added a normalization

[5]The source code can be found at https://github.com/b5li/SymGC.

operation `norm` on patterns, such that:

$$\begin{array}{ll} \texttt{norm}\,(\texttt{Not}\,(\texttt{Bit False})) = \texttt{Bit True} & \texttt{norm}\,(\pi\,(\texttt{Bit False})\;p\;q) = \texttt{Pair}\,(\texttt{norm}\,q)\,(\texttt{norm}\,p) \\ \texttt{norm}\,(\texttt{Not}\,(\texttt{Bit True})) = \texttt{Bit False} & \texttt{norm}\,(\pi\,(\texttt{Bit True})\;p\;q) = \texttt{Pair}\,(\texttt{norm}\,p)\,(\texttt{norm}\,q) \\ \texttt{norm}\,(\texttt{Not}\,(\texttt{Not}\,e)) = \texttt{norm}\,e & \texttt{norm}\,(\pi\,(\texttt{Not}\,b)\;p\;q) = \texttt{norm}\,(\pi\;\;b\;q\;p) \end{array}$$

The equivalence relation $\equiv$ on patternsis checked using syntactic equality on normalized patterns. Random bit renaming and pseudo-random key renaming are implemented using maps on normalized bit and key patterns. Thus we can check equivalence up to renaming by first applying renaming maps to normalized patterns and then checking for equivalence.

To build symbolic expressions of the real and the simulated garbled circuits, the pseudo-code definitions of the garbling scheme and the simulator in Sections 4.3 and 4.4 were directly translated into Haskell code. The bit and key renamings α_B and α_K were constructed recursively as in the proof of Lemma 8.

So far, given a circuit and a boolean vector of an appropriate shape, our programs are able to produce symbolic expressions of the real and the simulated garbled circuits, compute their patterns, and check if these patterns are equivalent up to renaming. The whole implementation consists of about 500 lines of Haskell code, and its performance is fairly good: For example, with a randomly generated circuit that contains about 10000 NAND subcircuits and a 112-dimension boolean vector, the entire process of generating the real and the simulated garbled circuits, computing their patterns, and checking for symbolic equivalence runs in about 1.3 second on a Linux desktop with an Intel I7-4790 CPU running at 3.60GHz. Notice that the number of NAND subcircuits and the dimension of the input vector together determine the number of atomic keys in the garbled circuit expression, which affects how fast the greatest fixed point of the recoverable key set can be reached. Further optimization is possible, for example, we could expand our circuit notation by adding AND and XOR as basic circuits. As a reference, an AES encryption circuit usually consists of about 5k AND and 20k XOR gates, which can be implemented using

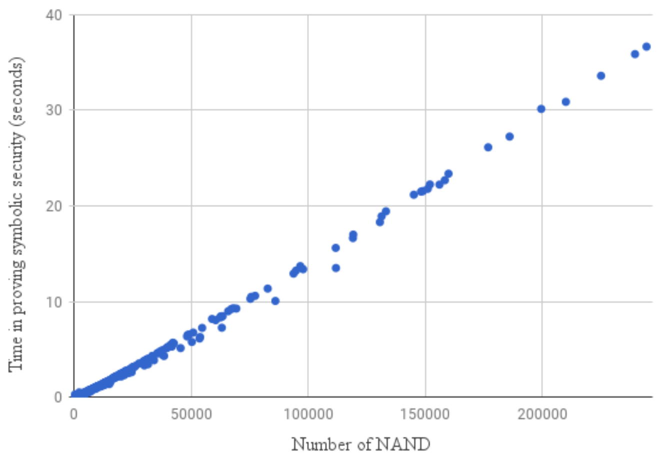


Figure 4.6. Running times of proving symbolic security of the garbling scheme using our implementation. Experiments were run on a Linux desktop with an Intel I7-4790 CPU running at 3.60GHz. Each point corresponds to a randomly generated test case, where the circuit may contain up to 250k **NAnd** subcircuits and the input vector may have up to 128 components. For each test case we measure the total time spent on generating the real and the simulated garbled circuit expressions, computing their patterns, and then checking for symbolic equivalence on patterns. The horizontal axis measures the number of **NAnd** subcircuits in a circuit, and the vertical axis measures the time in seconds.

about 90k **NAnd** inductively.

We conducted automated tests using the QuickCheck test framework to perform symbolic security analysis on randomly generated circuits and boolean vectors, and the performance results are shown in Fig. 4.6.

We remark that our automated tests run on a circuit-by-circuit basis, that is, given a circuit and a boolean vector, the test ensures that the resulting garbled circuit is computationally secure. In fact, our program can check that, for any cryptographic system that is built using primitives in our symbolic framework, an instance for a given input is computationally secure. It is also interesting to translate our proofs into a machine-checked flavor using verification tools, but such work is out of the scope of the current paper, and we would like to explore it in the future.

Acknowledgement Chapter 4 contains a reprint of materials as it appears in Proceedings of the 31st Computer Security Foundations Symposium 2018, Baiyu Li and Daniele Micciancio, *Symbolic security of garbled circuits*. The dissertation author was a primary investigator and author of this paper.

Chapter 5

Equational Security Model of Oblivious Transfer Protocols

In this chapter we present our results on formalizing oblivious transfer protocols in the equational security framework and their security analysis. Our analysis shows that the equational framework provides a concise and clean approach to regiously analyze secure computation protocols.

5.1 Equational Security Framework

In this section we review the equational framework of [59], and define the notation used in this chapter. For completeness, we will first recall some background on the (standard) theory that gives a precise meaning to systems of equations as used in [59] and in this chapter. This material is important to give a solid mathematical foundation to the equational framework, but is not essential to follow the rest of the chapter, and the reader may want to skip directly to the following paragraph describing our computational models and notational conventions.

Domain theoretical background. The mathematical foundation of the equational framework is provided by domain theory. Here we give just enough background to describe the systems studied in this chapter, and refer the reader to [35, 65, 68] for a detailed treatment. Recall that a partially ordered set (or poset) is a set X equipped with a reflexive, transitive and antisymmetric relation $\leq$. All posets in this chapter are complete partial orders (CPOs), i.e., any (possibly empty) chain $x_1 < x_2 < \ldots$ has a least upper bound $\sup_i x_i$ in X. The Cartesian product $X \times Y$ of two CPOs is also a CPO with the component-wise partial order $(x_1, y_1) \leq (x_2, y_2) \iff x_1 \leq x_2 \wedge y_1 \leq y_2$. These posets are endowed with the *Scott topology*, where a subset $C \subseteq X$ is *closed* if for all $x \in C$, $y \leq x$ implies $y \in C$, and any chain in C has a least upper bound in C. A set is *open* if its complement is closed. The standard topological definition of *continuous function* still applies here, and continuous functions (with respect to the Scott topology) are exactly the functions that preserve limits $f(\sup_i x_i) = \sup_i f(x_i)$. The set of all continuous functions from CPOs X to Y is denoted by $[X \to Y]$. Any (Scott) continuous function is necessarily *monotone*, i.e., for all $x, y \in X$, if $x \leq y$ then $f(x) \leq f(y)$. All CPOs X have a minimal element $\bot = \sup \emptyset$, called the *bottom*, which satisfies $\bot \leq x$ for all $x \in X$.

For any set A, we can always construct a *flat* CPO $A_\bot = A \cup \{\bot\}$ by including a unique bottom element $\bot$. The partial order in $A_\bot$ consists of $\bot \leq x$ for all $x \in A$. It should be easy to see that all nonempty closed sets in $A_\bot$ contain $\bot$, and open sets in $A_\bot$ are exactly the subsets of A and the whole $A_\bot$. Functions $f : A \to B$ between sets can be lifted to strict functions $f : A_\bot \to B_\bot$ between the corresponding flat CPOs by setting $f(\bot) = \bot$. The bottom element usually designates the situation where no (real) input or output is given yet.

For any CPO X, every continuous functions $f : X \to X$ admits a *least fixed point*, denoted as $\mathrm{fix}(f)$, which is the minimal $x \in X$ such that $f(x) = x$. The least fixed point can be obtained by taking the limit of the sequence $\bot, f(\bot), f^2(\bot), \ldots$. A system of mutually recursive equations can be solved via least fixed point computation. Such a solution describes the final outputs of interactive computations between nodes in a network. By Bekič's theorem [71], the least fixed point of such a system can be computed one component at a time: For example, the system $(x, y) = (f(x, y), g(x, y))$ can be solved by computing first $\hat{x} = \mathrm{fix}(\lambda x.f(x, \mathrm{fix}(\lambda y.g(x, y))))$ and then $\hat{y} = \mathrm{fix}(\lambda y.g(\hat{x}, y))$, and the least fixed point of the system is $(\hat{x}, \hat{y})$.

We can also model probabilistic behaviors in equational framework. A *probability* *distribution* on a CPO X is a function $p\colon X \to [0,1]$ such that[1] $p(A) + p(B) = p(A \cup B)$ for all disjoint $A, B \subseteq X$ and $p(X) = 1$. As usual, we say that a probability p is *negligible* if for all $x \in X$, $p(x) < n^{-c}$ for any constant $c > 1$, where n is a *security parameter*.[2] Similarly, p is *overwhelming* if $1 - p$ is negligible. If X is a CPO, then the *set of probability distributions over* X, denoted by $D(X)$, is also a CPO, where for any two distributions $p \leq q$ (in $D(X)$) if and only if $p(A) \leq q(A)$ for any open subset $A \subseteq X$. *Probabilistic functions* are just (continuous) functions between sets of distributions with respect to this ordering relation.

Computational model. We recall that the execution model of [59] consists of a network, with nodes representing computational units, and (directed) edges modeling communication channels. (See below for details.) Each channel is associated with a partially ordered set of channel "histories" or "behaviors", representing all possible messages or sequences of messages that may be transmitted on the channel over time. The partial order represents temporal evolution, so for any two histories $h_1 \leq h_2$ means that h_2 is a possible extension (or future) of h_1. The standard example is that of finite sequences $M^* = \{(m_1, \dots, m_k) : k \geq 0, \forall i.m_i \in M\}$ of messages from a ground set M, ordered according to the prefix partial order. By combining the set M^∞ of infinite sequences of messages from M, we get a CPO M^ω. Another common example, modeling a channel capable of delivering only a single message, is the flat partial order $M_\perp$, consisting of all messages in M and a special bottom element $\perp$ denoting the fact that no message has been transmitted yet. Different incoming and outgoing channels (incident to a single node) are combined taking Cartesian products, so that each node can be thought as having just one input and one output. The computational units at the nodes are modeled as functions $\mathsf{F}\colon X \to Y$ from the incoming channels to the outgoing channels, satisfying the natural monotonicity requirement that for any $h_1 \leq h_2$ in X, we have $\mathsf{F}(h_1) \leq \mathsf{F}(h_2)$ in Y. Informally,

[1]In general we should consider the Borel algebra on X when defining probability distributions on X. Here we simply use X instead since we work on finite sets and discrete probabilities.

[2]In the asymptotic setting, cryptographic protocols are parameterized by a security parameter n. For notational simplicity, we consider this security parameter n as fixed throughout the chapter.

monotonicity captures the intuition that once a party transmits a message, it cannot go back in time and take it back. A probabilistic computational unit can be modeled as a function of type $X \to \mathcal{D}(Y)$, where $\mathcal{D}$ is the probability monad. We may also consider units with limited computational power in the monadic approach, which is an important extension to the equational framework. However, as all the protocols considered in this chapter run in constant time, for simplicity we do not formalize computational cost (e.g. running time, space, etc) in our analysis.

Computation units can be connected to a communication network N to form a system, where N is also a monotone function. Such a system is again a monotone function mapping external input channels to external output channels of all the units, and it is modeled as a composition of functions describing all the units and the network. Syntactically, function compositions can be simplified by substitution and variable elimination, and, when recursive definition is involved, by using fixed point operations. In general, we use the notation $(\mathsf{F}|\mathsf{G})$ to denote the system composed by functions F and G, where the composition operator "$|$" is associative. The main advantage of the equational framework is that it has a mathematically clean and well defined semantics, where functions can be completely described by mathematical equations (specifying the relation between the input and the output of the units), and composition simply combines equations together. The equational approach also provides a simple and precise way to reason about relations between systems. For example, equivalent components (in the sense of having equivalent equations) can be replaced by each other, and when considering probabilistic behaviors, if a component is indistinguishable from another component, then they can be used interchangeably with negligible impact on the behavior of the entire system.

Security. The definition of security in the equational framework follows the well-accepted simulation-based security paradigm. In this chapter we consider only OT protocols, which are two-party protocols between a sender program and a receiver program. An ideal functionality F is a function from $X = X_0 \times X_1$ to $Y = Y_0 \times Y_1$, where X_i (Y_i) is the external input (output) of party $\mathsf{P_i}$. An environment is a function $\mathsf{Env} : Y^\omega \to X^\omega \times \{\top\}_\perp$ such that it takes as input the output history (as a sequence of evolving messages) of a system, and it produces a sequence of evolving inputs to the system and a decision bit t. Here a sequence of messages $x_0 x_1 \dots$ over X is *evolving* if $x_i \leq_X x_{i+1}$ for all i, where $x_i \in X$ and $\leq_X$ is the partial order of X. An experiment between an environment Env and a system S, is executed as follows: Env generates an evolving sequence of input $x_0 x_1 \dots$ to S such that S outputs $y_i = \mathsf{S}(x_i)$ for each x_i, Env takes as input the sequence $y_0 y_1 \dots$, and it eventually produces an external decision bit t. We write $\mathsf{Env}[\mathsf{S}]$ for the output (distribution) t of this experiment. When all parties are honest, the real system is a composition of the network N and two parties P_0 and P_1, denoted as $(\mathsf{P}_0|\mathsf{P}_1|\mathsf{N})$, and it must be equivalent to the ideal functionality F. When a party P_i is *corrupted*, the real system is composed by the remaining honest party and the network, and the ideal system is composed by F and a monotone simulator Sim. We say that a protocol is secure against the corruption of P_i if there exists a simulator Sim as a computation unit such that the systems $(\mathsf{N}|\mathsf{P}_{(1-\mathsf{i})})$ and $(\mathsf{Sim}|\mathsf{F})$ are indistinguishable by any environment that produces a decision bit in polynomial time in the output length of the system and the security parameter.

A distinctive feature of the equational framework is the ability to specify fully asynchronous systems. An environment might not provide a complete input to a system at once, that is, the input to certain channels might be $\perp$. So we must consider such asynchronous environments when analyzing the security of a protocol.

It is an very interesting and important open question to compare the equational framework (with the full extension of computational security) with the UC model and its variants (for example, the simplified models of [17, 70].)

Notations. Now we briefly mention our notational conventions. In this chapter we mainly use flat CPOs, i.e., partially ordered sets $\mathbb{X}$ with a bottom element $\perp \in \mathbb{X}$ such that $x_1 \leq x_2$ iff $x_1 = \perp$ or $x_1 = x_2$. These are used to model simple communication channels that can transmit a single message from $\mathbb{X} \setminus \{\perp\}$, with $\perp$ representing the state of the channel before the transmission of the message. For any CPO $\mathbb{X}$, we write $\mathbb{X}^{\times 2} = \{(x, y) : x, y \in \mathbb{X}, x \neq \perp, y \neq \perp\}_\perp$

for the CPO of *strict* pairs over $\mathbb{X}$ and $\bot$. The elements of a pair $z \in \mathbb{X}^{\times 2}$ are denoted $z[0]$ and $z[1]$, with $z[i] = \bot$ when $z = \bot$ or $i = \bot$. The operation of combining two elements into a strict pair is written $\langle x, y \rangle$. Notice that $\langle x, \bot \rangle = \langle \bot, y \rangle = \bot$, and therefore $\langle x, \bot \rangle[0] = \langle \bot, y \rangle[1] = \bot$ even when $x, y \neq \bot$. For any set A, we write $x \leftarrow A_\bot$ for the operation of selecting an element $x \neq \bot$ uniformly at random from A.

It is easily verified that for any pairs $z, \langle x_0, x_1 \rangle, \langle y_0, y_1 \rangle$, strict function f and strict binary operation $\odot$,

$$z = \langle z[0], z[1] \rangle \tag{5.1}$$

$$f(\langle x_0, x_1 \rangle[i]) = \langle f(x_0), f(x_1) \rangle[i] \tag{5.2}$$

$$\langle x_0, x_1 \rangle[i] \odot \langle y_0, y_1 \rangle[i] = \langle x_0 \odot y_0, x_1 \odot y_1 \rangle[i] \tag{5.3}$$

The followings are common CPOs and operations:

- The CPO $\mathbb{T} = \{\top\}_\bot$, representing *signals*, i.e., messages with no information content.
- The CPO $\mathbb{B} = \{0, 1\}_\bot$ of single bit messages, often used to select an element from a pair.
- The CPO $\mathbb{M}_n = \{0, 1\}^n_\bot$ of bit-strings of length n.
- $x!y = \langle x, y \rangle[1]$, the operation of *guarding* an expression y by some other expression x. Notice that $x!y = y$, except when $x = \bot$, and can be used to "delay" the transmission of y until after x is received.
- $x! = x!\top$, testing that $x > \bot$.

As an example, using the notation introduced so far, we can describe the ideal (1-out-of-2) OT functionality by the equations in Fig. 5.1. (Notice that this functionality is parameterized by a message space $\mathbb{M}$.) The first line specifies the names of the functionality (OT), input channels (m_2, b) and output channel(s) m. This is followed by a specification of the type of each channel:

the input interface includes a message pair $m_2 = \langle m_0, m_1 \rangle \in \mathbb{M}^{\times 2}$ from a sender and a selection bit $b \in \mathbb{B}$ from a receiver. The output interface is a single message $m \in \mathbb{M}$ sent to the receiver while the sender does not get any information from the functionality. The last line $m = m_2[b]$ is an equation specifying the value of the output channel(s) as a function of the input channels. The functionality is illustrated by a diagram showing the names of the function and the input/output channels.

$$\begin{array}{rcl} \mathsf{OT}_{\mathbb{M}}(m_2, b) & = & m \\ m_2 & : & \mathbb{M}^{\times 2} \\ b & : & \mathbb{B} \\ m & : & \mathbb{M} \\ m & = & m_2[b] \end{array}$$

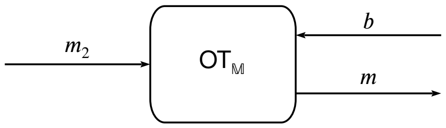


Figure 5.1. A naive OT functionality: the receiver gets the selected message $m = m_2[b]$, and the sender does not get anything at all.

In the rest of this chapter, equational variables usually belong to unique domains (e.g., $m_2 : \mathbb{M}_n^{\times 2}$.) So from now on, we will omit such type specifications when defining functions using equations, and we will follow the convention listed in Table 5.1 for naming variables.

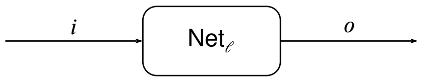
Table 5.1. Frequently used variables and their domains.

Variable name	Domain	Variable name	Domain
m	$\mathbb{M}_n$	m'	$\mathbb{M}_\ell$
m_2	$\mathbb{M}_n^{\times 2}$	m'_2	$\mathbb{M}_\ell^{\times 2}$
c_0,c_1	$\mathbb{M}_\ell$	c_2	$\mathbb{M}_n^{\times 2}$
a,a'	$\mathbb{T}$	b,b'	$\mathbb{B}$
i,o	$\mathbb{M}_n$	i_2,o_2	$\mathbb{M}_\ell^{\times 2}$
k	$\mathbb{K}_n$	k_2	$\mathbb{K}_n^{\times 2}$
q	$(G^2 \times G)_\perp$	q_2	$(G^2 \times G)_\perp^{\times 2}$
X,Y	$G_\perp$		

5.2 Formalizations of Oblivious Transfer Extension

As an abbreviation, when the message space $\mathbb{M} = \{0,1\}^n_\perp$ is the set of all bitstrings of length n, we write OT_n instead of $\mathsf{OT}_{\mathbb{M}}$. Consider the following OT *length extension* problem: given an OT_n channel for messages of some (sufficiently large) length n, build an OT functionality OT_ℓ for messages of length $\ell > n$. The goal is to implement OT_ℓ making a single use of the basic OT_n functionality, possibly with the help of an auxiliary (unidirectional, one-time) communication channel for the transmission of messages from the sender to the receiver. For simplicity,[3] we model the communication channel as a functionality Net_ℓ that copies its input of length ℓ to the output of the same length:

$$\begin{aligned} \mathsf{Net}_\ell(i) &= o \\ o &= i \end{aligned}$$



The OT length extension protocol is specified by a pair of Sender and $\mathsf{Receiver}$ programs, which are interconnected (using the OT_n and $\mathsf{Net}_{2\ell}$ functionalities) as shown in Fig. 5.2. Notice how the external input/output interface of the system corresponding to a real execution of the protocol in Fig. 5.2 is the same as that of the ideal functionality $\mathsf{OT}_\ell(m'_2, b') = m'$ the protocol is trying to implement.

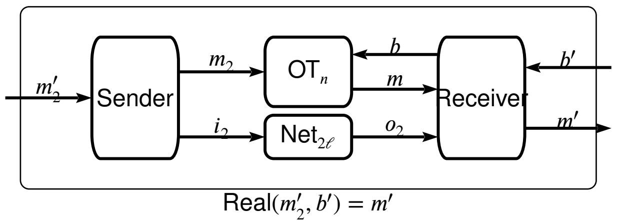


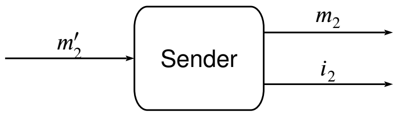
Figure 5.2. A real execution of a candidate OT length extension protocol. The protocol consists of a Sender and a $\mathsf{Receiver}$ programs that communicate using OT_n and $\mathsf{Net}_{2\ell}$ functionalities.

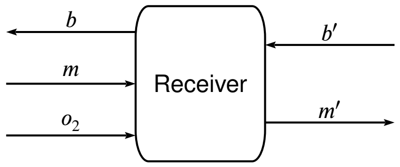
A natural approach to design an OT length extension protocol is to make use of a

[3]This corresponds to a perfectly secure communication channel. More complex/realistic communication channels are discussed at the end of this section.

pseudorandom generator $\mathcal{G} : \mathbb{M}_n \to \mathbb{M}_\ell$ that stretches a short random seed of length n into a long pseudorandom string of length ℓ. Using such pseudorandom generator, one may define candidate Sender and Receiver programs as follows:

$$\begin{aligned}
&\mathsf{Sender}(m_2') = (m_2, i_2)\\
&\quad m_2 \leftarrow \mathbb{M}_n^{\times 2}\\
&\quad i_2[0] = m_2'[0] \oplus \mathcal{G}(m_2[0])\\
&\quad i_2[1] = m_2'[1] \oplus \mathcal{G}(m_2[1])
\end{aligned}
\qquad
\begin{aligned}
&\mathsf{Receiver}(m, o_2, b') = (b, m')\\
&\quad b = b'\\
&\quad m' = o_2[b'] \oplus \mathcal{G}(m)
\end{aligned}$$





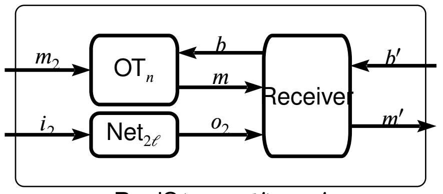
In words, these programs work as follows:

- The sender picks a pair m_2 of two random seeds, and passes (one of) them to the receiver using the OT_n functionality. It then stretches the two seeds using the pseudorandom generator $\mathcal{G}$, and uses the generator's output as a one-time pad to "mask" the actual messages before they are transmitted to the receiver over the communication channel $\mathsf{Net}_{2\ell}$.
- The receiver selects one of the two seeds from the OT_n functionality, expands it using the pseudorandom generator, and uses the result to "unmask" the corresponding message from $\mathsf{Net}_{2\ell}$.

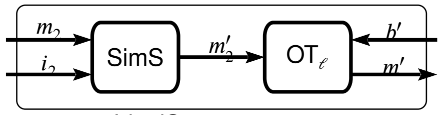
It is easy to show that the protocol is correct, in the sense that combining the equations of OT_n, $\mathsf{Net}_{2\ell}$, Sender and Receiver as shown in Fig. 5.2 results in a system $\mathsf{Real}(m_2', b') = m'$ that is perfectly equivalent to the defining equation $m' = m_2'[b']$ of the ideal functionality OT_ℓ. Intuitively, the protocol also seems secure because only one of the two seeds can be recovered by the receiver, and the unselected message is protected by an unpredictable pseudorandom pad.

But security of cryptographic protocols is a notoriously tricky business, and deserves a closer look.

We first consider the security of the protocol when the sender is corrupted. The attack scenario corresponds to the real system obtained by removing the Sender program from the protocol execution in Fig. 5.2. Following the simulation paradigm, security requires exhibiting an efficient simulator program SimS (interacting, as a sender, with the ideal functionality OT_ℓ) such that the following real and ideal systems are computationally indistinguishable:



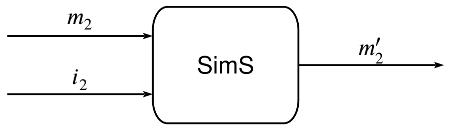
$\mathsf{RealS}(m_2, i_2, b') = m'$



$\mathsf{IdealS}(m_2, i_2, b') = m'$

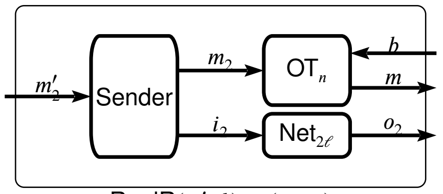
Security is easily proved by defining the following simulator:

$$\mathsf{SimS}(m_2, i_2) = m_2'$$
$$m_2'[0] = i_2[0] \oplus \mathcal{G}(m_2[0])$$
$$m_2'[1] = i_2[1] \oplus \mathcal{G}(m_2[1])$$

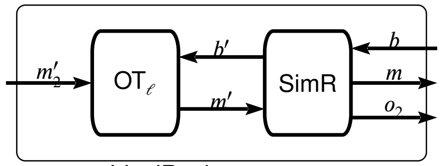


We observe that RealS and IdealS are perfectly equivalent because they both simplify to $m' = i_2[b'] \oplus \mathcal{G}(m_2[b'])$. So, the protocol is perfectly secure against corrupted senders.

We now turn to analyzing security against a corrupted receiver. This time we need to come up with a simulator SimR such that the following real and ideal executions are equivalent:



$\mathsf{RealR}(m_2', b) = (m, o_2)$



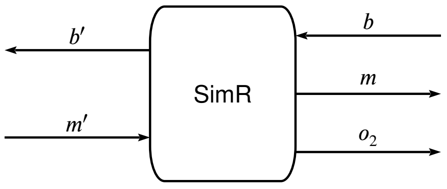
$\mathsf{IdealR}(m_2', b) = (m, o_2)$

Of course, this time we can only aim at proving computational security, i.e., coming up with a simulator such that RealR and IdealR are computationally indistinguishable. We begin by writing down explicitly the equations that define the real system execution. Combining the equations for Sender, OT_n and $\mathsf{Net}_{2\ell}$, we obtain the following system:

$$\begin{aligned}
&\mathsf{RealR}(m_2', b) = (m, o_2)\\
&\quad m_2 \quad \leftarrow \mathbb{M}_n^{\times 2}\\
&\quad o_2[0] \ = m_2'[0] \oplus \mathcal{G}(m_2[0])\\
&\quad o_2[1] \ = m_2'[1] \oplus \mathcal{G}(m_2[1])\\
&\quad m \quad\ \ = m_2[b]
\end{aligned}$$

So, the simulator may proceed by picking m_0, m_1 at random on its own, and set $m = m_2[b]$ just as in the real execution. However, the simulator cannot compute o_2 as in RealR because it does not know m_2'. This is addressed by using the same message m' twice, counting on the pseudorandom masking to hide this deviation from a real protocol execution. Formally, the simulator SimR is defined as follows:

$$\begin{aligned}
&\mathsf{SimR}(m', b) = (b', m, o_2)\\
&\quad b' \quad\ = b\\
&\quad m_2 \quad \leftarrow \mathbb{M}_n^{\times 2}\\
&\quad m \quad\ \ = m_2[b]\\
&\quad o_2[0] = m' \oplus \mathcal{G}(m_2[0])\\
&\quad o_2[1] = m' \oplus \mathcal{G}(m_2[1])
\end{aligned}$$



Combining SimR with OT_ℓ results in the ideal system:

$$\begin{aligned}
&\mathsf{IdealR}(m_2', b) = (m, o_2)\\
&\quad m_2 \quad \leftarrow \mathbb{M}_n^{\times 2}\\
&\quad o_2[0] \ = m_2'[b] \oplus \mathcal{G}(m_2[0])\\
&\quad o_2[1] \ = m_2'[b] \oplus \mathcal{G}(m_2[1])\\
&\quad m \quad\ \ = m_2[b]
\end{aligned}$$

As expected, the two systems IdealR, RealR are indistinguishable for both $b = 0$ and $b = 1$. For example, $\mathsf{RealR}(m_2', 0)$ and $\mathsf{IdealR}(m_2', 0)$ are equivalent because they are both computationally indistinguishable from the process that chooses $m \leftarrow \mathbb{M}_n$ and $c \leftarrow \mathbb{M}_\ell$ at random and sets $o_2 = \langle m_2'[0] \oplus \mathcal{G}(m), c\rangle$. The case when $b = 1$ is similar. At this point it would be very tempting to conclude that RealR and IdealR are equivalent, but they are not: they can be easily distinguished by an environment that sets $m_2' \neq \perp$ and $b = \perp$. In fact, $\mathsf{IdealR}(m_2', \perp) = (\perp, \perp)$, but $\mathsf{RealR}(m_2', \perp) = (\perp, o_2)$, where $o_2 \neq \perp$. So, IdealR and RealR are not equivalent, and the simulator SimR is not valid.

Insecurity in general. By generalizing the above idea, we can show that, for any simulator SimR there is an environment Env that can distinguish the two systems RealR and IdealR with nonnegligible probability. We build Env that works in two stages:

$$\begin{aligned}
&\mathsf{Env}_0(m, o_2) = (b, m_2', t) \text{ where}\\
&\quad b = \perp,\ m_2' \leftarrow \mathbb{M}_n^{\times 2},\ t = (o_2 > \perp)\\
&\mathsf{Env}_1(m, o_2) = (b, m_2', t) \text{ where}\\
&\quad b \leftarrow \{0,1\},\ m_2' \leftarrow \mathbb{M}_n^{\times 2},\ t = (\mathcal{G}(m) + o_2[b] = m_2'[b])
\end{aligned}$$

Notice that the output of the ideal system $\mathsf{IdealR}(m_2', b) = (m, o_2)$ is $(b', m, o_2) \leftarrow \mathsf{SimR}(m_2'[b'], b)$, where b' is an internal channel. Since b' ranges over a flat CPO, and $m_2'[\perp] = \perp$, the value of b' resulting from a least fixed point computation is given by $(b', \_, \_) = \mathsf{SimR}(\perp, b)$. In particular, b' may depend only on the external input b. We denote using $\mathsf{SimR}(b)^{b'}$ the random variable b' computed on input b.

Let $p = \Pr\{\mathsf{SimR}(\perp)^{b'} = \perp\}$ and $q = \Pr\{\mathsf{SimR}(\perp,\perp)^{o_2} = \perp\}$. We have $\Pr\{\mathsf{Env}_i[\mathsf{RealR}] = \top\} = 1$ for all $i \in \{1, 2\}$. For the ideal system, we have

$$\begin{aligned}
\Pr\{\mathsf{Env}_0[\mathsf{IdealR}] = \top\} = &\Pr\{\mathsf{SimR}(\perp,\perp)^{o_2} > \perp\} \cdot p\\
&+ \Pr\{\mathsf{SimR}(\perp, m_2'[b'])^{o_2} > \perp\} \cdot (1-p)\\
= &(1-q)p + \Pr\{\mathsf{SimR}(\perp, m_2'[b'])^{o_2} > \perp\} \cdot (1-p).
\end{aligned}$$

Since $\Pr\{\mathsf{Env}_0[\mathsf{RealR}] = \top\} = 1$, $\Pr\{\mathsf{Env}_0[\mathsf{IdealR}] = \top\}$ must be overwhelming; and since $\Pr\{\mathsf{SimR}(\bot,\bot)^{o_2} > \bot\} \leq \Pr\{\mathsf{SimR}(\bot, m'_2[b'])^{o_2} > \bot\}$, p must be negligible. Finally, notice that

$$\begin{aligned}\Pr\{\mathsf{Env}_1[\mathsf{IdealR}] = \top\} = &\Pr\{\mathcal{G}(m) + o_2[b] = m'_2[b] \mid \mathsf{SimR}(\bot)^{b'} = \bot\} \cdot p \\ &+ \Pr\{\mathcal{G}(m) + o_2[b] = m'_2[b] \mid \mathsf{SimR}(\bot)^{b'} > \bot\} \cdot (1-p).\end{aligned}$$

If $\mathsf{SimR}(\bot)^{b'} > \bot$, then $\Pr\{b' = b\} = \frac{1}{2}$ and so

$$\Pr\{\mathcal{G}(m) + o_2[b] = m'_2[b] \mid \mathsf{SimR}(\bot)^{b'} > \bot\} = \frac{1}{2}(1 + \frac{1}{2^\ell}).$$

This implies that $\Pr\{\mathsf{Env}_2[\mathsf{IdealR}] = \top\} = \frac{1}{2} + \epsilon$ for some negligible $\epsilon > 0$, and so Env can distinguish the two systems.

The discrepancy between the two systems as shown above highlights a subtle timing bug in the protocol: in order to carry out the simulation, the transmission of i_2 should be delayed until after the receiver has selected her bit b. However, this information is not available to the sender, and fixing the protocol requires revising the definition of OT, as we will do in the next section.

Other communication channels. We conclude this section with a discussion of other possible communication channels and weaker OT variants that leak some information to the environment. For example, one may replace the perfectly secure communication channel $\mathsf{Net}_{\mathbb{M}}$ with an authenticated channel $\mathsf{AuthNet}_{\mathbb{M}}(i, e_i) = (o, e_o)$ that also takes an input $e_i : \mathbb{T}$ and provides an output $e_o : \mathbb{M}$ to the environment. The environment output $e_o = i$ is used to leak the transmitted message as well as the timing information about when the message is transmitted. The environment input e_i is used to allow the environment to delay the transmission of the message $o = e_i!i$ to the receiver.

Similarly, one may consider the OT variants that leak the input timing information $e_o = (m_2!\top, b!\top)$ to the environment, and allow the environment to delay the OT output $m = e_i!m_2[b]$.

This idea is similar to the "message header" in the UC models proposed in [17, 70].

We remark that none of these modifications affect the analysis presented in this section. In particular, considering a perfectly secure communication channel Net only makes our insecurity result stronger. Also, leaking the signal $b!\top$ to the environment does not solve the timing bug in the protocol: in order to fix the bug, the sender needs to delay the transmission of i_2 until $b > \bot$. So, it is not enough to provide this information to the environment. The timing signal $b!\top$ needs to be provided as an input to the honest sender.

5.2.1 Revised Definitions

We have seen that the "standard" OT definition is inadequate even to model and analyze a simple OT length-extension protocol. In Fig. 5.3 we provide a revised definition of oblivious transfer that includes an acknowledgment informing the sender of when the receiver has provided her selection bit.

$$\begin{aligned} \mathsf{OT}'_{\mathbb{M}}(m_2, b) &= (a, m) \\ m &= m_2[b] \\ a &= (b > \bot) \end{aligned}$$

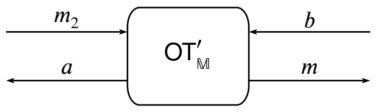
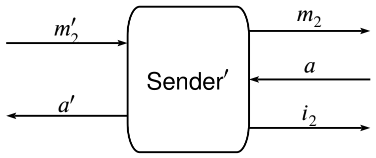


Figure 5.3. A revised OT functionality.

We use this revised definition to build and analyze a secure OT length-extension protocol, similar to the one described in the previous section. The OT length extension uses the same Receiver program as defined in the previous subsection, but modifies Sender by using the signal a to delay the transmission of the message i_2. The new Sender$'$ also forwards the signal a to the environment to match the new OT$'$ definition:

$$\begin{aligned} \mathsf{Sender}'(m'_2, a) &= (a', m_2, i_2) \\ (m_2, i'_2) &\leftarrow \mathsf{Sender}(m'_2) \\ a' &= a \\ i_2 &= a!i'_2 \end{aligned}$$



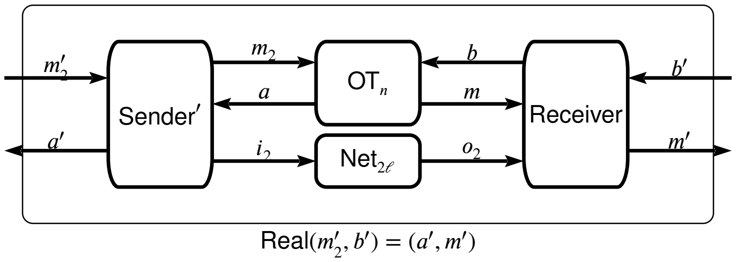
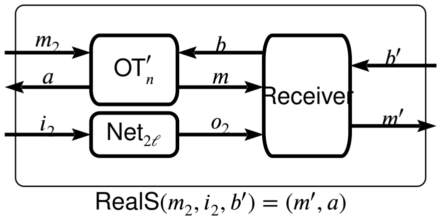
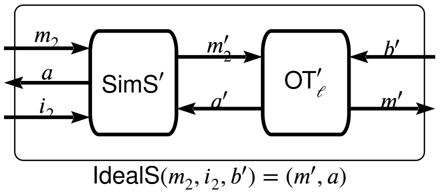


Figure 5.4. A normal execution of the OT Length Extension protocol.

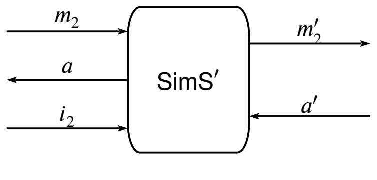
The Sender and Receiver programs are interconnected using OT'_n and $\mathsf{Net}_{2\ell}$ as shown in Fig. 5.4. As in the previous section, it is easy to check that the protocol is correct, i.e., combining and simplifying all the equations from the real system in Fig. 5.4 produces a set of equations identical to the revised definition of the ideal functionality $\mathsf{OT}'(m'_2, b') = (a', m')$. Security when the sender is corrupted is also similar to before. The real and ideal systems in this case are given by





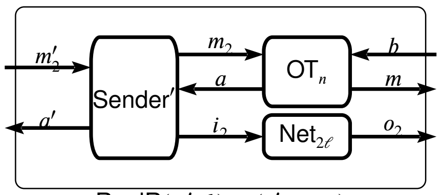
We see that this time SimS′ has an additional input a' and output a. We adapt the simulator from the previous section simply by adding an equation that forwards the a' signal from OT' to the external environment:

$$\begin{aligned} \mathsf{SimS}'(m_2, i_2, a') &= (a, m'_2) \\ m'_2 &= \mathsf{SimS}(m_2, i_2) \\ a &= a' \end{aligned}$$

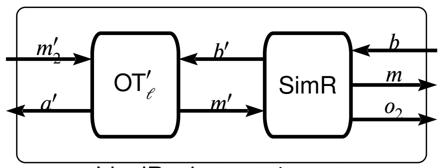


RealS(m_2, i_2, b') and Ideal(m_2, i_2, b') are equivalent because they both output $m' = o_2[b'] \oplus \mathcal{G}(m_2[b'])$ and $a = (b' > \bot)$. So, the protocol is still perfectly secure against corrupted senders according to the revised OT' definition.

We now go back to the analysis of security against corrupted receivers. The real and ideal systems are:



$\mathsf{RealR}(m'_2, b) = (a', m, o_2)$



$\mathsf{IdealR}(m'_2, b) = (a', m, o_2)$

No change to the simulator are required: we use exactly the same "candidate" simulator SimR as defined in the previous subsection. Combining and simplifying the equations, gives the following real and ideal systems:

$$
\begin{array}{ll}
\mathsf{RealR}(m'_2, b) = (a', m, o_2) & \qquad \mathsf{IdealR}(m'_2, b) = (a', m, o_2) \\
\quad m_2 \ \leftarrow \mathbb{M}_n^{\times 2} & \qquad\quad m_2 \ \leftarrow \mathbb{M}_n^{\times 2} \\
\quad c_0 \ = m'_2[0] \oplus \mathcal{G}(m_2[0]) & \qquad\quad c_0 \ = m'_2[b] \oplus \mathcal{G}(m_2[0]) \\
\quad c_1 \ = m'_2[1] \oplus \mathcal{G}(m_2[1]) & \qquad\quad c_1 \ = m'_2[b] \oplus \mathcal{G}(m_2[1]) \\
\quad o_2 \ = b!\langle c_0, c_1 \rangle & \qquad\quad o_2 \ = \langle c_0, c_1 \rangle \\
\quad m \ = m_2[b] & \qquad\quad m \ = m_2[b] \\
\quad a' \ = (b > \perp) & \qquad\quad a' \ = (b > \perp)
\end{array}
$$

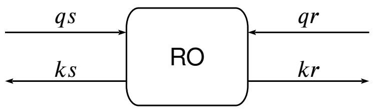
Now, when $b = \perp$, we have $\mathsf{RealR}(m'_2, \perp) = \mathsf{IdealR}(m'_2, \perp) = (\perp, \perp, \perp)$. So, no adversary can distinguish the two systems by not setting b. On the other hand, when $b \neq \perp$, RealR and IdealR are identical to the real and ideal systems from the previous section, augmented with the auxiliary output $a' = (b > \perp) = \top$. As we already observed in the previous subsection, these two distributions are computationally indistinguishable, proving that the length extension protocol is secure against corrupted receivers.

5.3 Formalizations of the Simplest OT Protocol

In this section we consider the OT protocol proposed by Chou and Orlandi in [24]. In the original paper, this is described as a protocol to execute l instances of 1-out-of-m OT, in parallel, i.e., the sender provides an l-dimensional vector of m-tuples of messages, and the receiver (non-adaptively) selects one message from each tuple. For simplicity, we consider the most basic case where $l = 1$ and $m = 2$, i.e., a single OT execution of a basic OT protocol as defined in the previous sections. This is without loss of generality because our results are ultimately negative. So, fixing $l = 1$ and $m = 2$ only makes our results stronger. Our goal is to show that this protocol is not provably secure in the equational framework according to a fully asynchronous simulation-based security definition. In order to formally analyze security, we begin by giving a mathematical description of the protocol and model of [24] using the equational framework.

The Random Oracle model The protocol of [24] is designed and analyzed in the random oracle model [7]. So, both parties have access to an ideal functionality RO implementing a random function with appropriately chosen domain Q and range K. Queries from the sender and receiver are answered consistently, and, in general, RO can receive multiple (adaptively chosen) queries from both parties. Formally, the random oracle is modeled by the following functionality, where $f^*(x_1, x_2, \ldots,) = (f(x_1), f(x_2), \ldots)$ is the standard extension of f to sequences:

$$
\begin{aligned}
&\mathsf{RO}_{Q,K}(qs, qr) = (ks, kr)\\
&\quad qs, qr \;:\; Q^*\\
&\quad ks, kr \;:\; K^*\\
&\quad f \quad\; \leftarrow [Q \to K]\\
&\quad ks \quad = f^*(qs)\\
&\quad kr \quad = f^*(qr)
\end{aligned}
$$



The random oracle starts by picking a function $f : Q \to K$ uniformly at random, and then it uses f to answer any sequence of queries $qs, qr \in Q^*$ from each party. We give separate channels to access RO to the sender (qs) and receiver (qr) to model the fact that random oracle queries are implemented as local computations, and each party is not aware of if/when other players access the oracle. The Sender and $\mathsf{Receiver}$ programs from the protocol of [24] only make a small number of queries (two and one respectively.) Moreover, the two sender queries are chosen simultaneously, non-adaptively. So, for simplicity, we restrict $\mathsf{RO}(q_2, q) = (k_2, k)$ to an oracle that receives just a pair of queries $q_2 = \langle q_0, q_1 \rangle \in Q_\perp^{\times 2}$ from the sender and one query $q \in Q_\perp$ from the receiver. We remark that in order to prove security, one should consider an arbitrary (still polynomial) number of (sequential, adaptively chosen) queries to model the adversary/environment ability to compute the RO function locally an arbitrary number of times.[4] However, since our results are negative, fixing the number of queries only makes our result stronger: we show that the protocol is not provably secure even against the restricted class of adversaries that make only this very limited number of random oracle queries.

It has been observed, for example in [18], that a protocol analyzed stand-alone in the traditional random oracle model might lose its security when composed with other instances of protocols in the same random oracle model: either each instance uses an independent random oracle such that the real composed system cannot assume a single hash function, or the composed system suffers from transferability attack. A modified notion called *global random oracle* was proposed in [18] to allow a composed system achieving UC security when all protocols can access a single global random oracle. With respect to this issue, the OT protocol of [24] cannot be claimed UC secure and it should be re-defined in the global random oracle model or an equivalent notion. However, such issue is independent of the negative result we are going to present. Since our motivation is to illustrate the use of equational framework, for simplicity, we still consider the traditional random oracle model as used in [24].

The protocol In order to facilitate a comparison with the original paper, we use as far as possible the same notation as [24]. Let $G = \langle B \rangle$ be a group generated by an element B of prime

[4]This can be modeled by letting qs and qr range over the set of sequences of queries Q^*, partially ordered according to the prefix ordering relation.

order p. Following [24], we use additive group notation, so that the group elements are written as xB for $x = 0, \dots, p-1$.[5] In [24] it is assumed that group elements have unique, canonical representations (which allows for equality testing), and group membership can be efficiently checked. Here, for simplicity, we assume that all messages representing group elements are syntactically valid, i.e., whenever a program expects a group element from G as input, it will always receive the valid representation of a such a group element (or $\perp$ if the no message has been sent), even when this value is adversarially chosen. This is easily enforced by testing for group membership, and mapping invalid strings to some standard element, e.g., the group generator B.

The protocol uses a random oracle $\mathsf{RO}(q_2, q) = (k_2, k)$ for functions with domain $Q = G^2 \times G$ and range $K = \{0,1\}^n$, which receives two (parallel) queries $q_2 = \langle q_0, q_1 \rangle \in Q_\perp^{\times 2}$ from the sender and one query $q \in Q_\perp$ from the receiver.

The protocol also uses a symmetric encryption scheme $(\mathtt{E}, \mathtt{D})$, with the same message space $\mathbb{M}_n$ as the OT functionality, and key and ciphertext space $\mathbb{K}_n = \{0,1\}_\perp^n$ equal to the range of the random oracle. In addition, the scheme is assumed to satisfy the following properties:

1. Non-committing: There exist PPT $\mathcal{S}_1$,$\mathcal{S}_2$ such that, for all $m \in \mathbb{M}_n$, the following distributions are identical:[6]

$$\{(e,k) \quad : \quad k \leftarrow K, e \leftarrow \mathtt{E}(k,m)\}$$
$$\{(e,k) \quad : \quad e \leftarrow \mathcal{S}_1, k \leftarrow \mathcal{S}_2(e,m)\}$$

2. Robustness: Let S be a set of keys chosen independently and uniformly at random from $\mathbb{K}_n$. For any PPT algorithms $\mathcal{A}$, if $e \leftarrow \mathcal{A}(S)$, then the set $V_{S,e} = \{k \in S \mid \mathtt{D}(k,e) \neq \perp\}$ of keys under which e can be successfully decrypted has size at most 1 with overwhelming

[5]Chou and Orlandi use additive notation to match their efficient implementation based on elliptical curve groups. Here we are not concerned with any specific implementation, but retain the additive notation to match [24] and facilitate the comparison with the original protocol description.

[6]In fact, computational indistinguishability is enough, but it is easy to achieve perfect security.

probability (over the choice of S and the randomness of $\mathcal{A}$.)

A simple encryption scheme satisfying these property is given by $\mathtt{E}(m, k) = (m, 0^n) \oplus k$, i.e., padding the message with a string of zeros for redundancy, and masking the result with a one-time pad.

The protocol of [24] can be described by the equations in Fig. 5.5, and its execution is depicted in Fig. 5.6. We briefly explain the normal protocol execution: Sender first samples a random group element X and sends it to Receiver; once it receives Y from Receiver, it submits a pair of queries q_2 to RO; and once it receives random keys k_2 from RO, it encrypts messages m_2 under the keys k_2, and it sends the ciphertext pair c_2 to Receiver. On the other hand, Receiver first samples a random group element yB, and upon receiving X from Sender it computes $Y = bX + yB$ and sends it to Sender; it then submits a query q to RO, and once the random key k and the ciphertexts c_2 are all received, it decrypts $c_2[b]$ using k to get the desired message m.

$$
\begin{array}{ll}
\mathsf{Sender}(m_2, k_2, Y) = (q_2, X, c_2) & \qquad \mathsf{Receiver}(k, X, c_2, b) = (q, Y, m) \\
\quad x \quad\;\; \leftarrow \mathbb{Z}_p^* & \qquad\quad y \leftarrow \mathbb{Z}_p^* \\
\quad X \quad\;\, = xB & \qquad\quad Y = bX + yB \\
\quad q_2[0] = ((X, Y), xY) & \qquad\quad q \;= ((X, Y), yX) \\
\quad q_2[1] = ((X, Y), xY - xX) & \qquad\quad m = \mathtt{D}(k, c_2[b]) \\
\quad c_2[0] \leftarrow \mathtt{E}(k_2[0], m_2[0]) & \\
\quad c_2[1] \leftarrow \mathtt{E}(k_2[1], m_2[1]) &
\end{array}
$$

Figure 5.5. The OT protocol of Chau and Orlandi.

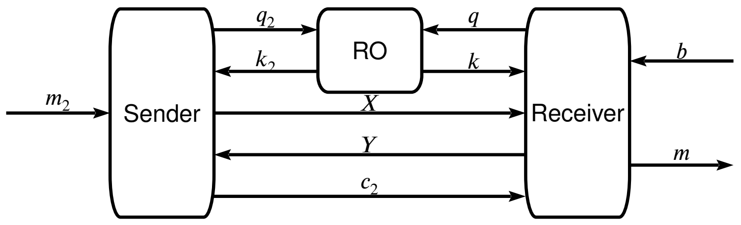
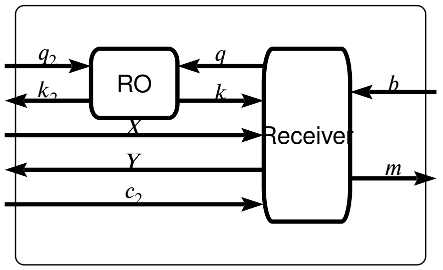


Figure 5.6. A normal execution of the OT protocol of Chou and Orlandi.

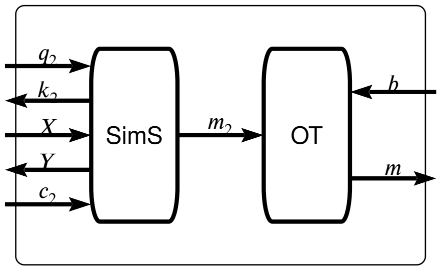
In the following subsections, we show that this protocol is insecure, both according to the classic OT definition given in Fig. 5.1, and according to our revised OT′ definition of Fig. 5.3 that includes the signal $a = (b > \bot)$ to the sender. Specifically, first, in Subsections 5.3.1 and 5.3.2 we show that if the definition from Fig. 5.1 is used, then the protocol is insecure against corrupted senders and corrupted receivers. The sender insecurity is for reasons very similar to those leading to the failure simulation in Section 5.2. Unlike the case of OT length extension, when considering the revised OT′ definition and modifying the sender program accordingly, we show in Subsection 5.3.3 that the modified protocol is still insecure against corrupted senders and corrupted receivers.

5.3.1 Corrupted sender

We begin our analysis of the OT protocol with respect to the standard OT functionality, and we first consider the case when the sender is corrupted. The corresponding real and ideal systems are shown in the following diagrams:



RealS$(q_2, X, c_2, b) = (k_2, Y, m)$



IdealS$(q_2, X, c_2, b) = (k_2, Y, m)$

For the protocol to be secure, the two systems should be computationally indistinguishable (for some simulator program SimS.) Just like the case of OT length extension, there exists an environment that can distinguish the two systems. We now describe an environment Env that works in two stages Env_0 and Env_1, and show that for any SimS, at least one of Env_0 and Env_1 distinguishes the real and ideal systems with nonnegligible advantage. We recall that a distinguishing environment connects to all input and output channels of the system, and produces one external output $t \in \{\bot, \top\}$. The distinguishing advantage of Env_i is given by

$$\mathtt{Adv}[\mathsf{Env}_i] = \left|\Pr\{\mathsf{Env}_i[\mathsf{RealS}] = \top\} - \Pr\{\mathsf{Env}_i[\mathsf{IdealS}] = \top\}\right|.$$

The two stages of the distinguisher work as follows:

- $\mathsf{Env}_0(k_2, Y, m) = (q_2, X, c_2, b, t)$ sets $q_2 = \bot$, $X = B$, $c_2 = \bot$ and $b = \bot$, and outputs $t = (Y > \bot)$.
- $\mathsf{Env}_1(k_2, Y, m) = (q_2, X, c_2, b, t)$ sets $q_2 = \bot$, $X = B$, $c_2 = \bot$ and $b = 0$, and outputs $t = (Y > \bot)$.

Notice that the only difference between these two stages is in the value of b. Using the equations for the $\mathsf{Receiver}$, we see that in the real system $Y > \bot$ if and only if $b > \bot$. In particular, we have $\Pr\{\mathsf{Env}_0[\mathsf{RealS}] = \top\} = 0$ and $\Pr\{\mathsf{Env}_1[\mathsf{RealS}] = \top\} = 1$. On the other hand, we have

$$\Pr\{\mathsf{Env}_0[\mathsf{IdealS}] = \top\} = \Pr\{\mathsf{Env}_1[\mathsf{IdealS}] = \top\} \tag{5.4}$$

because when interacting with IdealS, the output value t is independent of b. So, if we let p be the probability in (5.4), the two stages of Env have advantage $\mathtt{Adv}[\mathsf{Env}_0] = p$ and $\mathtt{Adv}[\mathsf{Env}_1] = 1 - p$. It follows that either Env_0 or Env_1 has distinguishing advantage at least $1/2$.

Intuitively, this environment can distinguish the real and the ideal systems because a corrupted sender (interacting with the real system RealS), learns when the receiver sets $b > \bot$ by observing the incoming message $Y > \bot$, but in the ideal system this timing information is not passed to the simulator.

5.3.2 Corrupted receiver

We have seen that when using the standard OT definition, the protocol is not secure against corrupted senders. Now we turn to analyzing the protocol against corrupted receivers

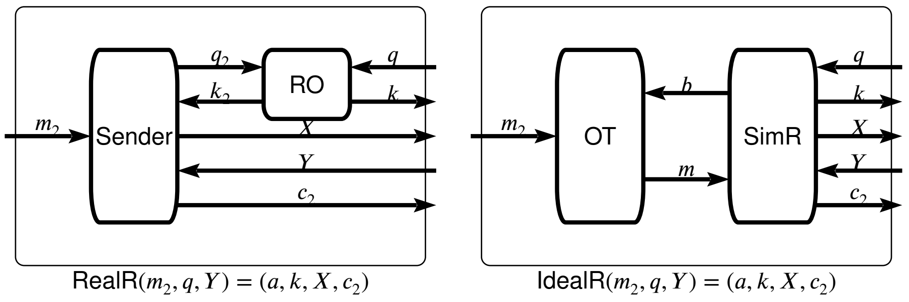

m_2
Sender
q_2
k_2
RO
q
k
X
Y
c_2
$\mathsf{RealR}(m_2, q, Y) = (a, k, X, c_2)$
m_2
OT
b
m
SimR
q
k
X
Y
c_2
$\mathsf{IdealR}(m_2, q, Y) = (a, k, X, c_2)$

Figure 5.7. The real and ideal systems when receiver is corrupted.

with respect to the standard OT definition. The real and ideal system in this case are shown in Fig. 5.7.

Security requires that the real and the ideal systems are indistinguishable for some simulator program SimR. Unfortunately, as we are about to show, no such simulator exists.

Proposition 3. *For the OT protocol in Fig. 5.5, when the receiver is corrupted, for any receiver simulator* SimR*, there is an environment that distinguishes the two systems with nonnegligible probability.*

Proof. We build an environment that works in three stages, denoted by Env_i for $i \in \{0, 1, 2\}$:

$$\mathsf{Env}_0(k, X, c_2) = (m_2, q, Y, t) \text{ where}$$

$$d \leftarrow \{0, 1\},\ y \leftarrow \mathbb{Z}_p^*,\ m_2 = \bot,\ Y = dX + yB,\ q = \bot,\ t = (c_2 = \bot)$$

$$\mathsf{Env}_1(k, X, c_2) = (m_2, q, Y, t) \text{ where}$$

$$d \leftarrow \{0, 1\},\ y \leftarrow \mathbb{Z}_p^*,\ m_2 \leftarrow \mathbb{M}_n^{\times 2},\ Y = dX + yB,\ q = \bot,\ t = (c_2 > \bot)$$

$$\mathsf{Env}_2(k, X, c_2) = (m_2, q, Y, t) \text{ where}$$

$$d \leftarrow \{0, 1\},\ y \leftarrow \mathbb{Z}_p^*,\ m_2 \leftarrow \mathbb{M}_n^{\times 2},\ Y = dX + yB,\ q = ((X, Y), yX),$$

$$t = (\mathbb{D}(k, c_2[d]) = m_2[d])$$

Assume there exists a receiver simulator SimR. With the real system, Env_i outputs $t = \top$ with probability 1 for all $i \in \{0, 1, 2\}$. So $\Pr\{\mathsf{Env}_i[(\mathsf{OT}|\mathsf{SimR})] = \top\}$ must be overwhelming for all $i \in \{0, 1, 2\}$.

Notice that in the ideal system both b and m are internal channels such that $m = m_2[b]$, and we can simplify the output of the ideal system as $(k, X, c_2) \leftarrow \mathsf{SimR}(m_2[b], q, Y)$. For $i = 0, 1, 2$, let u_i denote the (random variable of) the input to SimR when working with Env_i, and let $\mathsf{SimR}(u_i)^b$ denote the (random variable of) the value of b given input u_i. The external input channels to SimR are q and Y, and their values are $\perp$ in both Env_0 and Env_1. If SimR sets $b = \perp$ when $q = \perp$ and $Y = \perp$, then it cannot tell the difference between Env_0 and Env_1, and thus at least one of Env_0 and Env_1 has a nonnegligible distinguishing advantage. So $\Pr\{\mathsf{SimR}(u_0)^b > \perp\}$ must be overwhelming. Since SimR is a monotone function, $\Pr\{\mathsf{SimR}(u_i)^b > \perp\}$ is also overwhelming for $i \in \{1, 2\}$. In particular, let $\epsilon = \frac{1}{2}\Pr\{\mathsf{SimR}(u_1)^b = \perp\}$, then ϵ is negligible.

Now consider Env_1, which sets $q = \perp$ and samples Y from the distribution $\{dX + yB \mid y \leftarrow \mathbb{Z}_p^*\} \equiv \{yB \mid y \leftarrow \mathbb{Z}_p^*\}$. So q and Y are independent of d, and thus $\Pr\{\mathsf{SimR}(u_1)^b = d\} = \Pr\{\mathsf{SimR}(u_1)^b = 1 - d\} = \frac{1}{2} - \epsilon$.

Finally, when working with Env_2 we have

$$\begin{aligned}
&\Pr\{\mathsf{Env}_2[(\mathsf{OT}|\mathsf{SimR})] = \top\} = \Pr\{\mathtt{D}(k, c_2[d]) = m_2[d]\} \\
=&\Pr\{\mathtt{D}(k, c_2[d]) = m_2[d] \mid \mathsf{SimR}(u_2)^b = d\}\Pr\{\mathsf{SimR}(u_2)^b = d\} \\
&+ \Pr\{\mathtt{D}(k, c_2[d]) = m_2[d] \mid \mathsf{SimR}(u_2)^b = 1 - d\}\Pr\{\mathsf{SimR}(u_2)^b = 1 - d\} \\
&+ \Pr\{\mathtt{D}(k, c_2[d]) = m_2[d] \mid \mathsf{SimR}(u_2)^b = \perp\}\Pr\{\mathsf{SimR}(u_2)^b = \perp\}
\end{aligned}$$

Since SimR is monotone, $\frac{1}{2} - \epsilon = \Pr\{\mathsf{SimR}(u_1)^b = 1 - d\} \leq \Pr\{\mathsf{SimR}(u_2)^b = 1 - d\}$, and thus $\Pr\{\mathsf{SimR}(u_2)^b = d\} \leq \frac{1}{2} + \epsilon$. On the other hand, when $\mathsf{SimR}(u_2)^b = 1 - d$, it holds that $\mathsf{SimR}(u_i)^b \in \{1 - d\}_\perp$ for $i \in \{0, 1\}$ and thus SimR has no access to $m_2[d]$, and since $m_2[d]$ is independently sampled from $\mathbb{M}_n$, SimR cannot guess it correctly with probability more than $\frac{1}{2^n}$. So we can bound the probability

$$\Pr\{\mathsf{Env}_2[(\mathsf{OT}|\mathsf{SimR})] = \top\} \leq \frac{1}{2} + \epsilon + \frac{1}{2^n} + 2\epsilon,$$

which is close to $\frac{1}{2}$. Therefore the environment can distinguish the real and the ideal systems with nonnegligible probability. □

5.3.3 Revised OT definition

The timing issue with a corrupted sender is similar to the one for OT length extension that is fixed by adding an acknowledgment signal. So it is natural to ask if the insecurity problems can be resolved by modifying the protocol according to the revised functionality OT'. Clearly, changing the definition requires also modifying the sender program to output a signal a in order to match OT'. Since the sender receives only one message (Y) from the receiver, there is only one sensible way to modify the protocol to produce this additional output: setting $a = (Y > \bot)$. Formally, we consider the following modified sender program:

$$\begin{aligned}
&\mathsf{Sender}'(m_2, k_2, Y) = (a, q_2, X, c_2)\\
&\quad (q_2, X, c_2) \leftarrow \mathsf{Sender}(m_2, k_2, Y)\\
&\quad a \qquad\qquad = (Y > \bot)
\end{aligned}$$

It is easy to verify that a real protocol execution $(\mathsf{Sender}' \mid \mathsf{RO} \mid \mathsf{Receiver}) : (m_2, b) \mapsto (a, m)$ is equivalent to the ideal functionality $\mathsf{OT}' : (m_2, b) \mapsto (a, m)$.

For security, we start with the case when the receiver is corrupted. The real and ideal systems are depicted in Fig. 5.8. Notice that the additional bit a is not provided to the simulator but is instead given to the environment. So any receiver simulator SimR that connects to OT' to form the ideal system in the revised OT definition has the same interface as a receiver simulator in the standard OT definition. Thus we obtain the same result as in Proposition 3 that the modified protocol is insecure against corrupted receivers.

When the sender is corrupted, the sender simulator is now provided with an additional bit $a = (b > \bot)$, as shown in Fig. 5.9. This small modification is the key to prove security for the OT length extension protocol, so one might speculate, as we did in the previous version of this paper, that security could also hold for the current protocol in the case of sender corruption. On the contrary, this modification is not enough. As we are exploring the useability of the equational

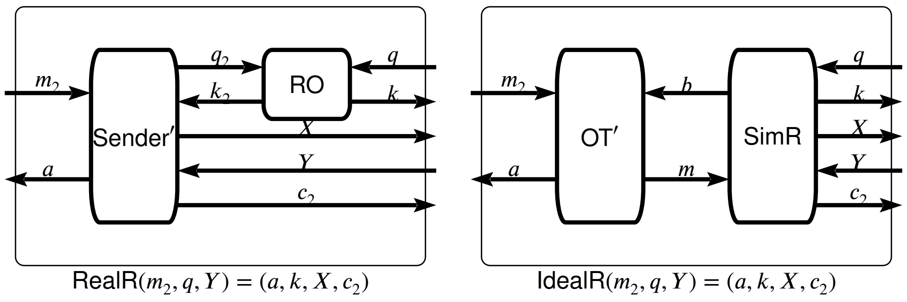


Figure 5.8. The real and ideal systems when receiver is corrupted, under revised OT definition.

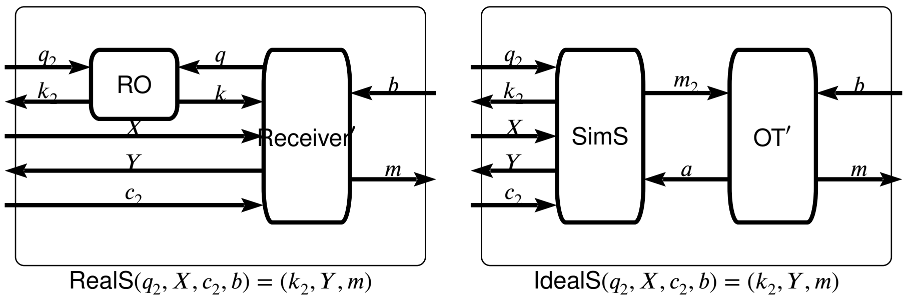


Figure 5.9. The real and ideal systems when sender is corrupted, under revised OT definition.

framework, we show in the following why the natural simulation strategy that takes advantage of the signal a fails at proving security.

The speculated simulator is shown below. As we are presenting negative results, we limit the power of a corrupted sender such that it can send at most one pair of RO queries q_2 and it obtains at most one pair of keys k_2.

```
$\mathsf{SimS}(q_2, X, a, c_2) = (k_2, Y, m_2)$
    $f$         $\leftarrow [(G^2 \times G) \rightarrow K]$
    $k_2$       $= f^*(q_2)$
    $y$         $\leftarrow \mathbb{Z}_p^*$
    $Y$         $= X!a!yB$
    $m_2[0]$    $=$ if $(\exists i. q_2[i] = ((X, Y), \ldots))$ then $\mathtt{D}(k_2[i], c_2[0])$
    $m_2[1]$    $=$ if $(\exists i. q_2[i] = ((X, Y), \ldots))$ then $\mathtt{D}(k_2[i], c_2[1])$
```

Let us derive an equation for m. In the real system RealS, the message m satisfies the equation

$$m = \mathtt{D}(f((X, bX + yB), yX), c_2[b]), \tag{5.5}$$

where y is sampled uniformly at random from $\mathbb{Z}_p^*$ by the honest receiver. In the ideal system $\mathsf{IdealS} = (\mathsf{SimS}|\mathsf{OT}')$, notice that $a = (b > \bot)$, and so

$$m = \mathtt{D}(f((X, X!b!yB), W), c_2[b]), \tag{5.6}$$

where y is sampled uniformly at random from $\mathbb{Z}_p^*$ by the simulator and W is some element of G chosen by the environment. In both equations (5.5) and (5.6), $c_2[b]$ is an input to the system given by the environment. By a careful examination, we can see that the value of m as computed in these two equations could be different if the environment sets W to be distinct from yX. We follow this idea to construct the following environment:

```
$\mathsf{Env}(k_2, Y, m) = (q_2, X, c_2, b, t)$ where
    $x \leftarrow \mathbb{Z}_p^*$, $X = xB$, $w \leftarrow \mathbb{Z}_p^*$, $W = wB$, $b \leftarrow \{0, 1\}$,
    For $i \in \{0, 1\}$:
        $q_2[i] = ((X, Y), W)$, $c_2[i] \leftarrow \mathtt{E}(k_2[i], 0)$,
    $t = (m > \bot)$
```

In the real system, Env outputs $t = \top$ only in two cases: either the key $k = f((X, Y), yX)$ obtained by the receiver is same as the key $k_2[b] = f((X, Y), W)$ used by Env to encrypt $m_2[b]$ in the ciphertext $c_2[b]$, where f is a random function sampled by RO, or the decryption succeeds when $k \neq k_2[b]$. For a sufficiently large key space $\mathbb{K}_n$, since $yX = yxB$ and $W = wB$ are independently sampled and uniformly distributed, the probability ϵ that $k = k_2[b]$ is negligible. Since $(\mathtt{E}, \mathtt{D})$ is a robust encryption scheme, when $k \neq k_2[b]$ the decryption can succeed with only a negligible probability δ. So Env outputs $t = \top$ with a negligible probability $\epsilon + (1 - \epsilon)\delta$. But in the ideal system, the decryption always succeeds and thus we get $m = 0 > \bot$, which implies that Env outputs $t = \top$ with probability 1. Therefore Env has a nonnegligible distinguishing advantage.

We remark that, if the above simulator SimS has access to a DDH oracle O that answers on input (X, Y, W) whether $W = yxB$ for $X = xB$ and $Y = yB$, then we can modify the equations for m_2 in SimS to prove sender security with respect to the revised OT definition:

$$m_2[0] = \texttt{if } (\exists i. q_2[i] = ((X, Y), W) \texttt{ and } \mathsf{O}(X, Y, W) = \top) \texttt{ then } \mathtt{D}(k_2[i], c_2[0])$$
$$m_2[1] = \texttt{if } (\exists i. q_2[i] = ((X, Y), W) \texttt{ and } \mathsf{O}(X, Y, W) = \top) \texttt{ then } \mathtt{D}(k_2[i], c_2[1])$$

That is, if a RO query contains a triple of group elements satisfying the DDH condition, then SimS uses the corresponding key to decrypt both $c_2[0]$ and $c_2[1]$ and assigns the resulting plaintext to $m_2[0]$ and $m_2[1]$, respectively. As already noted by Genç, Iovino, and Rial [29], sender security holds with certain gap-DH groups in which the CDH problem is hard but the DDH problem is easy to solve.

Acknowledgement Chapter 5 contains a reprint of materials as it appears in Proceedings of 21st Public-Key Cryptography 2018, Baiyu Li and Daniele Micciancio, *Equational Security Proofs of Oblivious Transfer Protocols*. The dissertation author was a primary investigator and author of this paper.

Bibliography

[1] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *J. Cryptology*, 20(3):395, 2007.

[2] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. Homomorphic encryption security standard. Technical report, HomomorphicEncryption.org, Toronto, Canada, November 2018. https://homomorphicencryption.org/standard/.

[3] Ahmad Al Badawi, Luong Hoang, Chan Fook Mun, Kim Laine, and Khin Mi Mi Aung. Privft: Private and fast text classification with homomorphic encryption. *CoRR*, abs/1908.06972, 2019.

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

[5] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy*, pages 478–492. IEEE Computer Society, 2013.

[6] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 134–153, 2012.

[7] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[8] Flávio Bergamaschi, Shai Halevi, Tzipora T. Halevi, and Hamish Hunt. Homomorphic training of 30,000 logistic regression models. In *ACNS 2019*, volume 11464 of *LNCS*, pages 592–611. Springer, 2019.

[9] Fabian Boemer, Yixing Lao, Rosario Cammarota, and Casimir Wierzynski. ngraph-he: a graph compiler for deep learning on homomorphically encrypted data. In *CF 2019*, pages 3–13. ACM, 2019.

[10] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical GapSVP. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 868–886. Springer, 2012.

[11] Zvika Brakerski, Craig Gentry, and Shai Halevi. Packed ciphertexts in lwe-based homomorphic encryption. In *Public Key Cryptography*, volume 7778 of *Lecture Notes in Computer Science*, pages 1–13. Springer, 2013.

[12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Trans. Comput. Theory*, 6(3):13:1–13:36, 2014.

[13] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from Ring-LWE and security for key dependent messages. In *CRYPTO 2011*, volume 6841 of *LNCS*, pages 505–524. Springer, 2011.

[14] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. *SIAM J. Comput.*, 43(2):831–871, 2014.

[15] Nicolas Brisebarre, Mioara Joldes, Jean-Michel Muller, Ana-Maria Nanes, and Joris Picot. Error analysis of some operations involved in the cooley-tukey fast fourier transform. *ACM Trans. Math. Softw.*, 46(2):11:1–11:27, 2020.

[16] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. In *Foundations of Computer Science, 2001. Proceedings. 42nd IEEE Symposium on*, pages 136–145, Oct 2001.

[17] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 3–22, 2015.

[18] Ran Canetti, Abhishek Jain, and Alessandra Scafuro. Practical UC security with a global random oracle. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 597–608, New York, NY, USA, 2014. ACM.

[19] Hao Chen, Ilaria Chillotti, and Yongsoo Song. Improved bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2019, Part II*, volume 11477 of *LNCS*, pages 34–54. Springer, 2019.

[20] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. Bootstrapping for approximate homomorphic encryption. In *EUROCRYPT 2018, Part I*, volume 10820 of *LNCS*, pages 360–384. Springer, 2018.

[21] Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *SAC 2018*, volume 11349 of *LNCS*, pages 347–368. Springer, 2018.

[22] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yong Soo Song. Homomorphic encryption for arithmetic of approximate numbers. In *ASIACRYPT 2017, Part I*, volume 10624 of *LNCS*, pages 409–437. Springer, 2017.

[23] Jung Hee Cheon, Duhyeong Kim, and Jai Hyun Park. Towards a practical clustering analysis over encrypted data. *IACR Cryptology ePrint Archive*, 2019:465, 2019.

[24] Tung Chou and Claudio Orlandi. The simplest protocol for oblivious transfer. In Kristin Lauter and Francisco Rodríguez-Henríquez, editors, *Progress in Cryptology – LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings*, pages 40–58. Springer International Publishing, 2015.

[25] Claude Crépeau, Jeroen Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *Advances in Cryptology — CRYPT0' 95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings*, pages 110–123, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[26] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: compiler and runtime for homomorphic evaluation of tensor programs. *CoRR*, abs/1810.00845, 2018.

[27] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: an optimizing compiler for fully-homomorphic neural-network inferencing. In *PLDI 2019*, pages 142–156. ACM, 2019.

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

[29] Ziya Alper Genç, Vincenzo Iovino, and Alfredo Rial. "The simplest protocol for oblivious transfer" revisited. *IACR Cryptology ePrint Archive*, 2017:370, 2017.

[30] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC 2009*, pages 169–178. ACM, 2009.

[31] Craig Gentry, Shai Halevi, and Nigel P. Smart. Fully homomorphic encryption with polylog overhead. In *EUROCRYPT*, volume 7237 of *Lecture Notes in Computer Science*, pages 465–482. Springer, 2012.

[32] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO 2012*, volume 7417 of *LNCS*, pages 850–867. Springer, 2012.

[33] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[34] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[35] C. A. Gunter and D. S. Scott. Handbook of theoretical computer science (vol. b). chapter Semantic Domains, pages 633–674. MIT Press, Cambridge, MA, USA, 1990.

[36] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. In *AAAI 2019*, pages 9466–9471. AAAI Press, 2019.

[37] HEAAN (release 2.1). https://github.com/snucrypto/HEAAN, 2018. SNUCRYPTO.

[38] HElib (release 1.1.0). https://github.com/homenc/HElib, 2020. IBM.

[39] Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, pages 149–178, 2016.

[40] John Hughes. Generalising monads to arrows. *Sci. Comput. Program.*, 37(1-3):67–111, 2000.

[41] John Hughes. Programming with arrows. In *Advanced Functional Programming, 5th International School, AFP 2004, Tartu, Estonia, August 14-21, 2004, Revised Lectures*, pages 73–129, 2004.

[42] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer – efficiently. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2008. Proceedings*, pages 572–591, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[43] Zahra Jafargholi and Daniel Wichs. Adaptive security of yao's garbled circuits. In *Theory of Cryptography - 14th International Conference, TCC 2016-B, Beijing, China, October 31 - November 3, 2016, Proceedings, Part I*, pages 433–458, 2016.

[44] Joe Kilian. Founding crytpography on oblivious transfer. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 20–31, New York, NY, USA, 1988. ACM.

[45] Andrey Kim, Antonis Papadimitriou, and Yuriy Polyakov. Approximate homomorphic encryption with reduced approximation error. Cryptology ePrint Archive, Report 2020/1118, 2020. https://eprint.iacr.org/2020/1118.

[46] Duhyeong Kim and Yongsoo Song. Approximate homomorphic encryption over the conjugate-invariant ring. In *ICISC 2018*, volume 11396 of *LNCS*, pages 85–102. Springer, 2018.

[47] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *ICALP (2)*, volume 5126 of *Lecture Notes in Computer Science*, pages 486–498. Springer, 2008.

[48] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[49] Yehuda Lindell and Benny Pinkas. Secure two-party computation via cut-and-choose oblivious transfer. *Journal of Cryptology*, 25(4):680–722, 2011.

[50] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 144–155. Springer, 2006.

[51] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In *FSE 2008*, volume 5086 of *LNCS*, pages 54–72. Springer, 2008.

[52] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6):43:1–43:35, 2013.

[53] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for Ring-LWE cryptography. In *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 35–54. Springer, 2013.

[54] Silvio Micali, Charles Rackoff, and Bob Sloan. The notion of security for probabilistic cryptosystems. *SIAM J. Comput.*, 17(2):412–426, 1988.

[55] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007.

[56] Daniele Micciancio. Symbolic encryption with pseudorandom keys. Cryptology ePrint Archive, Report 2009/249, 2009. https://eprint.iacr.org/2009/249.

[57] Daniele Micciancio. Computational soundness, co-induction, and encryption cycles. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 362–380, 2010.

[58] Daniele Micciancio. Symbolic encryption with pseudorandom keys. In *EUROCRYPT (3)*, volume 11478 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2019.

[59] Daniele Micciancio and Stefano Tessaro. An equational approach to secure multi-party computation. In *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*, ITCS '13, pages 355–372, New York, NY, USA, 2013. ACM.

[60] PALISADE lattice cryptography library (release 1.10.4). https://gitlab.com/palisade/, 2020. PALISADE Project.

[61] Saerom Park, Jaewook Lee, Jung Hee Cheon, Juhee Lee, Jaeyun Kim, and Junyoung Byun. Security-preserving support vector machine with fully homomorphic encryption. In *SafeAI@AAAI 2019*, volume 2301 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2019.

[62] Chris Peikert and Alon Rosen. Efficient collision-resistant hashing from worst-case assumptions on cyclic lattices. In *TCC 2006*, volume 3876 of *LNCS*, pages 145–166. Springer, 2006.

[63] Michael O. Rabin. How to exchange secrets with oblivious transfer, 1981. Technical Report, TR-81 edn. Aiken Computation Lab, Harvard University.

[64] RNS-HEAAN. https://github.com/KyoohyungHan/FullRNS-HEAAN, 2018. SNU-CRYPTO.

[65] Dana S. Scott. Domains for denotational semantics. In *Proceedings of the 9th Colloquium on Automata, Languages and Programming*, pages 577–613, London, UK, UK, 1982. Springer-Verlag.

[66] Microsoft SEAL (release 3.5). https://github.com/Microsoft/SEAL, April 2020. Microsoft Research, Redmond, WA.

[67] Nigel P. Smart and Frederik Vercauteren. Fully homomorphic SIMD operations. *Des. Codes Cryptogr.*, 71(1):57–81, 2014.

[68] Viggo Stoltenberg-Hansen, Ingrid Lindström, and Edward R. Griffor. *Mathematical Theory of Domains*. Cambridge University Press, New York, NY, USA, 1994.

[69] Avi Wigderson. On the impact of cryptographic thinking on tcs and beyond, November 2020. Workshop: Matches made in heaven: Cryptography and Theoretical Computer Science.

[70] Douglas Wikström. Simplified universal composability framework. In *Theory of Cryptography - TCC 2016-A*, volume 9562 of *LNCS*, pages 566–595. Springer, 2016.

[71] Glynn Winskel. *The Formal Semantics of Programming Languages: An Introduction*. MIT Press, Cambridge, MA, USA, 1993.

[72] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164, 1982.

[73] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167, 1986.

[74] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *Foundations of Computer Science, Proceedings of FOCS'86*, pages 162–167. IEEE Computer Society, 1986.

[75] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *EUROCRYPT (2)*, volume 9057 of *Lecture Notes in Computer Science*, pages 220–250. Springer, 2015.