Cryptographic Tools for the Cloud

by

Sergey Gorbunov

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Author . . . .
Department of Electrical Engineering and Computer Science
May 18, 2015

Certified by . . . .
Vinod Vaikuntanathan
Assistant Professor
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Chairman, Department Committee on Graduate Thesis

Cryptographic Tools for the Cloud

by
Sergey Gorbunov

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2015, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Classical cryptography is playing a major role in securing the Internet. Banking transactions, medical records, personal and military messages are transmitted securely through the Internet using classical encryption and signature algorithms designed and developed over the last decades. However, today we face new security challenges that arise in cloud settings that cannot be solved effectively by these classical algorithms. In this thesis, we address three major challenges that arise in cloud settings and present new cryptographic algorithms to solve them.

Privacy of data. How can a user efficiently and securely share data with multiple authorized receivers through the cloud? To address this challenge, we present attribute-based and predicate encryption schemes for circuits of any arbitrary polynomial size. Our constructions are secure under the standard learning with errors (LWE) assumption. Previous constructions were limited to Boolean formulas, captured by the complexity class $NC1$.

Privacy of programs. How can a user share a program, which may include some secrets, preserving its functionality and without leaking any information about the secrets? Program obfuscation is a mechanism that allows to scramble a program preserving its input/output functionality while preventing reverse engineering. We describe a new graph-induced multilinear maps from lattices and show how it can be used to construct a candidate general purpose program obfuscator. Our construction uses standard (random) integer lattices. Previous constructions of mutilinear maps relied on hardness of problems in either principal ideal lattices or integers and were subjected to many algebraic attacks.

Integrity of computations. How can a user outsource computations over a large database to the cloud and allow anyone efficiently authenticate the results? To address this, we present a fully homomorphic signature scheme for arbitrary circuits. The scheme allows the cloud server to run arbitrary computation, represented by circuit C, on the signed data x to get $y = C(x)$ and produce a short "proof" σ that can be used by anyone to authenticate the output y. Our scheme is secure under the short integer solution (SIS) problem in standard lattices. Previous constructions of homomorphic signatures were limited to evaluating polynomials of constant degree.

Thesis Supervisor: Vinod Vaikuntanathan, Assistant Professor

Dedication

To my grandfather.

Acknowledgments

I want to thank my advisor Vinod Vaikuntanathan. Vinod helped me tremendously to grow as a cryptographer and a scientist. I thank him for all the guidance, his wisdom, energy, advices and support. His passion and dedication to the field inspired me in many ways. He taught me to always look for the next question to ask, but pick research problems very carefully. I will remain grateful to him forever.

I would like to thank Hoeteck Wee for mentoring me throughout the last four years, his countless lessons, for hosting me at ENS and our collaborations. Hoeteck's persistence towards working on a problem motivated and challenged me on many occasions. I also thank him for helping me figure out life choices.

I want to thank my mentors at MIT: Shafi Goldwasser, Silvio Micali and Ron Rivest. I thank Shafi for her guidance and enormous support. I thank Silvio for his energy, advices and teaching me to always strike for perfection. I thank Ron for welcoming me into his reading group, his teachings and advices.

I would like to thank Charlie Rackoff for introducing me into the field and inspiring me to pursue graduate studies in it. Charlie's wisdom and scientific approach are exceptional. He challenged me and surprised every time with his creativity.

I want to thank IBM Crypto group that I was fortunate to be a part of during Summer'13: Sanjam Garg, Charanjit Jutla, Hugo Krawczyk, Tal Rabin, Mariana Raykova and Daniel Wichs, and especially my mentors Craig Gentry and Shai Halevi. I thank Tal for introducing and welcoming me into this wonderful crypto family. I thank Shai for hosting me, for his technical guidance, countless advices and our collaboration. I thank Daniel for our collaboration and many advices.

Many thanks to Yael Tauman Kalai. Yael's energy, wisdom and creative thinking influenced my development significantly over the last few years. I also thank her for support and somehow always finding time to answer my questions.

I thank Alon Rosen for many helpful technical and non-technical discussions, encouragement, and hosting me at IDC.

I will forever be grateful to my friend Dai Lê. His unconditional support and wisdom were invaluable. From personal to career advices, Dai was always there to listen and help me. He taught me some of the most important aspects of life: to enjoy it, yet remain committed to personal values. I also thank him for introducing me to the great food and the great music of Toronto.

I thank my close friends Yuval Filmus, Rakess Sivanandarajah and Dhinakaran Vinayagamurthy for all the good times and their kindness. I also thank Dhinakaran for our collaboration and many valuable discussions.

I would like to thank many cryptographers around the world for helpful discussions and advices: Benny Applebaum, Shweta Agrawal, Nir Bitansky, Dan Boneh, Zvika Brakerski, Ran Canetti and Yevgeniy Dodis.

I want to thank my other colleagues and friends at the University of Toronto and MIT for many valuable discussions: Maryam Aliakbarpour, Jaiganesh Balasundaram, Itay Berman, Siavosh Benabbas, Trevor Brown, Alon Cohen, Michael Coloumbe, Akshay Degwekar , Lila

Fontes, Kaveh Ghasemloo, Wesley George, Justin Holmgren, David Kordalewski, Joel Oren, Madalina Persu, Sunoo Park, Robert Robere, Adam Sealfon, and Prashant Vasudevan.

I thank administrative staff at MIT – Joanne Talbot Hanley, Debbie Lehto and especially Linda Lynch – for helping me navigate the system.

This thesis would not have been possible without help and support of my family, my parents, Vadim and Olga and my sister, Anna. I will forever be grateful for their unconditional patience and love.

I thank my second half, Ksenia, for bringing together all the reasons for me, inspiration, love, support, patience and being the ultimate source of happiness in my life.

Please forgive me any omissions.

Contents

Chapter 1

Introduction

The Internet has transformed how we communicate, share information and do business. Today almost every task can be performed online via a few touches of computer keys. We constantly generate, update, query and analyze large volumes of data. We store our data in the cloud and our daily lives rely on integrity and availability of this data on the cloud servers at all times. Yet, *we lose control of the data once it leaves our personal devices.*

Emails, photos, documents, and even this thesis are stored somewhere in the cloud. The cloud service provider "knows" everything about us, so an attacker just needs to find one (or few) weakness in their system to gain absolute control of *everyones data*. So the fundamental question is:

How can we secure data in the cloud?

Before we can answer this question, we must understand what we mean by security. The cloud is used for many purposes: storage, data sharing, outsourcing computations, etc. Therefore, we must first fully understand what notion of security is needed for each use case.

Systems to the Rescue? A standard approach taken by practitioners is to isolate cloud servers from adversaries. However, servers have many points of entry (at software and hardware levels) and systems often rely on many assumptions that are not well understood or formulated. But an adversary only needs to find a single broken door to compromise the entire system. As practice shows, adversaries are very successful in finding broken doors, gaining access to the systems and stealing data. We need a better approach where we can reduce the security of an entire system to simple, concise and well-understood assumption.

Cryptography to the Rescue? Classical cryptography has helped us tremendously to secure the Internet. It provides formal security models and concrete intractability assumptions which are simpler to study. If often allows to reduce security of an entire system to a single crypto-system built from strong mathematics. So, why can't we use the same standard cryptographic algorithms to help us secure the data in the cloud? A simple answer is that these algorithms are not flexible enough: They only offer an *all-or-nothing* mechanism to data control. For instance, in the standard encryption, there is only one secret key. The person that holds the secret key has absolute control over the data, and everyone else has no control. Now, suppose you want to securely share a digital photo with a selected group of your friends. Of course, encrypting the photo under everyones keys is very inefficient. Alternatively, you may encrypt the photo under *someone's* key and put all trust into that person, assuming that he/she will only share it with the selected friends. Naturally, humans also fail and social engineering attacks are some of the easiest to exploit. We need new cryptographic tools that naturally enable solutions to the security challenges that arise in the cloud-settings.

1.1 Privacy of Data: Attribute-based and Predicate Encryption

Suppose a client holding large data would like to utilize a cloud server to efficiency share it with many selected recipients. However, if the cloud server turns malicious or gets compromised all of the data its clients store may be leaked.

Can we let a client store an encryption of its data in the cloud and enable selected recipients *to be able to decrypt?*

For example, suppose a movie provider may wish to encrypt a movie with corresponding *public* meta-data attributes (e.g. "comedy", "2011", "English"), store the encryption in the cloud (or broadcast it over physical media) and ensure that only the authorized subscribers are able to decrypt it. Or a client may wish to outsource inspection of network log files or credit card transactions for intrusions and fraud to various investigating agencies. In this case, it should be able to encrypt the files under the corresponding *secret* meta-data attributes, store the encryption in the cloud and ensure that only authorized investigators learn *the files or the meta-data attributes.*

Attribute-based encryption [SW05, GPSW06] is a mechanism which enables fine-grained control of access to encrypted data. In attribute-based encryption, an encryption of a message m is labeled with a *public* attribute vector a (also called the "index"), and secret keys are associated with predicates P. A secret key sk_P decrypts the ciphertext and recovers the message m if and only if a satisfies the predicate, namely if and only if $P(a) = 1$.

Attribute-based encryption captures as a special case previous cryptographic notions such as identity-based encryption (IBE) [Sha84, BF01, Coc01] and fuzzy IBE [SW05]. It has also found applications in scenarios that demand complex policies to control access to encrypted data (such as medical or multimedia data [APG+11, LYZ+13, PPM+14]), as well as in designing cryptographic protocols for verifiably outsourcing computations [PRV12].

In contrast to attribute-based encryption, in predicate encryption [BW07, SBC+07, KSW08] the ciphertexts are associated with descriptive *private attribute* a in addition to messages m, secret keys are associated with a predicate P, and a secret key decrypts the ciphertext to recover m if and only if $P(a) = 1$. The security requirement for predicate encryption enforces privacy of a and the plaintext even amidst multiple secret key queries: an adversary holding secret keys for different query predicates learns nothing about the attribute a and the plaintext m if none of them is individually authorized to decrypt the ciphertext.

In the past few years, there has been significant progress in attribute-based and predicate encryptions in terms of efficiency, security guarantees, and diversifying security assumptions [GPSW06, BW07, SBC$^+$07, KSW08, Wat09, LW10, LOS$^+$10, CHKP12, ABB10a, OT10, AFV11, GMW15]. On the other hand, little progress has been made in terms of supporting larger classes of predicates. The state of the art is Boolean formulas [GPSW06, LOS$^+$10, OT10], which is a subclass of log-space computations. While these existing encryption systems were expressive,

"the central challenge to construct a system that supports creation of keys for any predicate",

in both public and private attribute settings was posed by Boneh, Sahai and Waters [BSW12].

Our Results. We construct attribute-based encryption schemes where keys can be generated for circuits of every a-priori bounded depth, based on the learning with errors (LWE) assumption. In particular, we show:

Theorem 1.1.1 (informal). *Under the LWE assumption, there exists an attribute-based encryption scheme for all circuits, with succinct ciphertexts independent of the size of the circuits.*

In the course of our construction, we present a new framework for constructing attribute-based encryption schemes, based on a primitive that we call "two-to-one recoding" (TOR). Our methodology departs significantly from the current line of work on attribute-based encryption [GPSW06, LOS$^+$10] and instead, builds upon the connection to garbled circuits developed in the context of *bounded* collusions [SS10b, GVW12]. Along the way, we make the first major progress towards the 25-year-old open problem of constructing reusable garbled circuits. In particular, we obtain the first construction of reusable garbled circuits that satisfies authenticity, but not privacy.

We also construct a predicate encryption scheme for circuits of every a-priori bounded depth, based on the learning with errors (LWE) assumption. In particular, we show:

Theorem 1.1.2. *Under the LWE assumption, there exists a predicate encryption scheme for all circuits, with succinct ciphertexts and secret keys independent of the size of the circuit.*

To support circuits of depth d, in both attribute-based and predicate encryptions the parameters of the schemes grow with poly(d), and we require sub-exponential $n^{\Omega(d)}$ hardness of the LWE assumption.

Our result on predicate encryption subsumes all prior works under standard cryptographic assumptions, apart from a few exceptions pertaining to the inner product predicate [BW07, KSW08, OT12a]. These results achieve a slightly stronger security notion for predicate encryption, known as full (or strong) security (please refer to Sections 4.3.1, 4.2.1 for definitions).

In a recent work, Garg et al. [GGH$^+$13b] gave a candidate construction of predicate encryption (in fact, the construct a more general primitive known as functional encryption) for arbitrary circuits. However, the construction relies on "multi-linear maps" [GGH13a, CLT13], for which we have few candidates and which rely on complex intractability assumptions that are presently poorly understood and not extensively studied in the literature and were recently attacked [CLT14, CHL$^+$15, HJ15].

Relations to Functional Encryption. As we explained above, predicate encryption is more powerful notion than attribute-based encryption because it guarantees secrecy of the attribute vector a wrt users that are not authorized to decrypt m. In turn, functional encryption is more powerful notion than predicate encryption. This is because in functional encryption the attribute vector a secret hidden even with respect to users that are authorized to decrypt and learn m. In other words, roughly speaking, functional encryption guarantees that no information about the attribute a other than the result $P(a)$ can be learned. (As usually defined, functional encryption does not even have a separate message m as the input and the user only learns $P(a)$ in clear). In Table 1.1 we summarize the interface for the three notions, their security guarantees and major known constructions from standard assumptions. It remains a fascinating open problem to go beyond predicate encryption and realize functional encryption from standard learning with errors.

Notion	Interface	Security Guarantees	Constructions
ABE	$\mathsf{ct} \leftarrow \mathsf{Enc}(a, m)$	m is secret iff $P(a) = 0$	Formulas: [GPSW06, LOS$^+$10] [OT10, Boy13a, BV14]
		a is always public	Circuits: **This Thesis**,[BGG$^+$14]
PE	$\mathsf{ct} \leftarrow \mathsf{Enc}(a, m)$	a, m are secret iff $P(a) = 0$	Formulas: [AFV11, KSW08, OT12b]
			Circuits: **This Thesis**
FE	$\mathsf{ct} \leftarrow \mathsf{Enc}(a)$	**a** is always secret,	Formulas: [KSW08]
		user learns $P(a)$	Circuits: **Open**

Table 1.1: Distinctions between attribute-based, predicate and functional encryptions (ABE, PE, and FE, resp.), as well as notable constructions from *standard assumptions* such as bilinear or learning-with-errors. In all three notions, the secrey key sk_P is generated with a special key-generation algorithm.

1.2 Privacy of Programs: Program Obfuscation

Suppose a client has a program that it would like to share with many users through the cloud.

How can it let other run the program without revealing secrets in its code?

On the high level, what we want is an obfuscator algorithm that complies a program into another "unintelligible" program that preserves the input/output functionality of the original. The formal study of this question was initiated by Barak et al. [BGI$^+$01], who presented some lower bounds for strong security models (namely, they showed that virtual-black box security is impossible to achieve), but the realization of weaker models (known as indistinguishability obfuscation (IO)) was left a fascinating open problem. In a recent break-through, Garg et al. [GGH$^+$13b] presented the first candidate IO obfuscator. They also construct a new tool, graded multilinear maps, that is used crucially in their construction. On the high level, multilinear maps allow to encode secrets and perform (restricted) homomorphic operations over these encodings. Moreover, at the end of the computation, given an encoding at the "output level" and a special zero-test parameter, it is possible to test if the encoded value encodes zero. As it turns out, this functionality is powerful enough to realize numerous cryptographic applications such as witness encryption, functional encryption, program obfuscation and many more [GGSW13, GGH$^+$13c, GGH$^+$13b, BGG$^+$14, BZ14].

Our Results. We present a new "graph-induced" variant of multilinear maps and provide a new candidate general purpose obfuscator. In this variant, the multilinear map is defined with respect to a directed acyclic graph. Namely, encoded value are associated with paths in the graph, and it is only possible to add encoding relative to the same paths, or to multiply encodings relative to "connected paths" (i.e., one ends where the other begins). Our candidate construction of graph-induced multilinear maps does not rely on ideal lattices or hard-to-factor integers [GGH13a, CLT13]. Rather, *we use standard random lattices* such as those used in LWE-based cryptography. We follow a similar outline to the previous constructions, except our instance generation algorithm takes as input a description of a graph. Furthermore, our zero-tester *does not* include any secrets about the relevant lattices. Rather, in our case the zero-tester is just a random matrix, similar to a *public key* in common LWE-based cryptosystems.

Giving up the algebraic structure of ideal lattices and integers could contribute to a better understanding of the candidate itself, reducing the risk of unforeseen algebraic crypt-analytical attacks. Both of the previous constructions [GGH13a, CLT13] were recently attacked due to their extensive algebraic properties [CLT14, CHL$^+$15, HJ15]. On the flip side, using our construction is sometimes harder to use than previous construction, exactly because we give up some algebraic structure. For that same reason, we were not able so far to reduce any of our new construction to "nice" hardness assumptions.

Connections of Program Obfuscation and Functional Encryption. Recently, Bitansky et al. [BV15] and Ananth et al. [AJ15] showed a fascinating connection between program obfuscation and functional encryption. In particular, they how to construct indistinguishability obfuscation from singe-key compact functional encryption. Intuitively, in single-key compact functional encryption only one secret key is ever issued and the running time of the encryption algorithm must be independent of the function description size or its output length. Unfortunately, in existing single-key constructions of functional encryption, the runtime of the encryption algorithm either depends on the size of the function description size [GVW12], or the output length [GKP$^+$13b]. Also, if one is able to extend our predicate encryption to such notion of functional encryption (even for a single key), then he/she would obtain full-fledged program obfuscator from standard learning with errors assumption.

1.3 Integrity of Computation: Homomorphic Signatures

Suppose a user holds a large database and it wants to outsource computations over it to a cloud server.

How can he/she (and others) authenticate the results of the computations produced by the cloud?

In a homomorphic signature scheme, a user signs some large dataset x using her secret signing key and uploads the signed data to an untrusted remote cloud server. The cloud server can then run some computation $y = f(x)$ over the signed data and homomorphically derive a *short* signature $\sigma_{f,y}$ certifying that y is the correct output of the computation f. Anybody can verify the tuple $(f, y, \sigma_{f,y})$ using Alice's public verification key and become convinced of this fact without having to retrieve the entire underlying data.

Many prior works consider the question of homomorphic signatures and homomorphic message authentication codes (MACs with private verification) for restricted homomorphisms, and almost exclusively for *linear functions*: [ABC$^+$07, SBW08, DVW09, AKK09, AB09, BFKW09, GKKR10, BF11a, AL11, BF11b, CFW12, Fre12]. Such MACs and signautres have interesting applications to *network coding* and *proofs of retrievability*. Boneh and Freeman [BF11a] were the first to consider homomorphic signatures beyond linear functions, and propose a general definition of such signatures. They present a scheme that can evaluate arbitrary *polynomials* over signed data, where the maximal *degree* k of the polynomial is fixed a priori and the size of the evaluated signature grows (polynomially) in k. If we want to translate this to the setting of circuits, then a circuit of depth d can be represented by a polynomial of degree as high as $k = 2^d$, and therefore the signature size can grow exponentially in the depth of the circuit. The construction is based on the hardness of the *Small Integer Solution* (SIS) problem in ideal lattices and has a proof of security in the random-oracle model. The recent work of Catalano et al. [CFW14] gives an alternate solution using multi-linear maps which removes the need for random oracles at the cost of having large public parameters. The main open question left by these works is to construct signatures with greater levels of homomorphism, and ideally a *fully homomorphic* scheme that can evaluate arbitrary circuits and rely on hardness of well-studied assumptions.

Our Results. We construct the first *(leveled) fully homomorphic signature* schemes that can evaluate arbitrary circuits over signed data, where only the maximal depth d of the circuit needs to be fixed a priori. The size of the evaluated signature grows polynomially in d, but is otherwise independent of the data size or the circuit size. In particular, we show:

Theorem 1.3.1 (informal). *Assuming hardness of* small integer solution *(SIS) problem in standard lattices, there exists a fully homomorphic signature scheme for every a-priori circuit depth bound d. Moreover, the size of evaluated signature is* $\text{poly}(d, \lambda)$ *where λ is the security parameter.*

We get a scheme in the standard model, where the maximal dataset size needs to be bounded by some polynomial N during setup and the size of the public parameters is linear in N. In the random-oracle model, we get rid of this caveat and get a scheme with short public parameters and without any a-priori bound on the dataset size. In both cases, the user can sign arbitrarily many different datasets by associating each dataset with some label (e.g., a file name). The verifier must know the label of the dataset on which the computation was supposed to be performed when verifying the output.

1.4 Organization and Sources of This Thesis

The remainder of the thesis is organized as follows.

Chapter 2: Is dedicated to preliminaries and background on lattices. We summarize lattice algorithms needed for our construction in this section.

Chapters 3, 4: These two chapters are dedicated to the privacy of data. In Chapter 3, we present our attribute-based encryption for circuits. We first present definitions of attribute-based encryption. Then, we proceed by describing our new framework (TOR), show how to instantiate it from lattices, and build our attribute-based encryption from it. We also present a separate construction for branching programs from milder lattice assumptions and pairings. In Chapter 4, we present definition and our construction of predicate encryption for circuits. We point out that our Chapter 4 is subsequent to works [GKP$^+$13b] and [BGG$^+$14] which are subsequent and built on top of our Chapter 3.

Chapter 5: This chapter is dedicated to the privacy of programs. We describe our new variant of graph-induced multilinear maps and provide a candidate instantiation from lattices. Then, we show how to use it to construct a non-interactive key-exchange scheme and a general purpose program obfuscator.

Chapter 6: This chapter is dedicated to the integrity of computation. We first present definitions of homomorphic signatures. Then, we define a new tool denoted by homomorphic trapdoor functions, and show how to instantiate it from lattices. Finally, using the new tool we construct homomorphic signature scheme for circuits.

The material for Chapters $3-6$ is taken from the following papers, respectively:

1. Sergey Gorbunov, Vinod Vaikuntanathan, Hoeteck Wee: *Attribute-based encryption for circuits.* STOC 2013: 545-554;

2. Sergey Gorbunov, Vinod Vaikuntanathan, Hoeteck Wee: *Predicate Encryption for Circuits from LWE.* CRYPTO 2015;

3. Craig Gentry, Sergey Gorbunov, Shai Halevi: *Graph-Induced Multilinear Maps from Lattices.* TCC 2015: 498-527;

4. Sergey Gorbunov, Vinod Vaikuntanathan, Daniel Wichs: *Leveled Fully Homomorphic Signatures from Standard Lattices.* STOC 2015.

Chapter 2

Lattice Preliminaries

Notation. Let PPT denote probabilistic polynomial-time. For any integer $q \geq 2$, we let $\mathbb{Z}_q$ denote the ring of integers modulo q and we represent $\mathbb{Z}_q$ as integers in $(-q/2, q/2]$. We let $\mathbb{Z}_q^{n\times m}$ denote the set of $n \times m$ matrices with entries in $\mathbb{Z}_q$. We use bold capital letters (e.g. $\mathbf{A}$) to denote matrices, bold lowercase letters (e.g. $\mathbf{x}$) to denote vectors. The notation $\mathbf{A}^\intercal$ denotes the transpose of the matrix $\mathbf{A}$.

If $\mathbf{A}_1$ is an $n\times m$ matrix and $\mathbf{A}_2$ is an $n\times m'$ matrix, then $[\mathbf{A}_1 \| \mathbf{A}_2]$ denotes the $n\times(m+m')$ matrix formed by concatenating $\mathbf{A}_1$ and $\mathbf{A}_2$. A similar notation applies to vectors. When doing matrix-vector multiplication we always view vectors as column vectors.

We represent elements of $\mathbb{Z}_q$ as integers in the range $(-q/2, q/2]$ and define the absolute value $|x|$ of $x \in \mathbb{Z}_q$ by taking its representative in this range. For a vector $\mathbf{u} \in \mathbb{Z}_q^n$ we write $||\mathbf{u}||_\infty \leq \beta$ if each entry u_i in $\mathbf{u}$ satisfies $|u_i| \leq \beta$. Similarly, for a matrix $\mathbf{U} \in \mathbb{Z}_q^{n\times m}$ we write $||\mathbf{U}||_\infty \leq \beta$ if each entry $u_{i,j}$ in $\mathbf{U}$ satisfies $|u_{i,j}| \leq \beta$.

We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\text{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\text{poly}(n)$ to denote a polynomial function of n. We say an event occurs with *overwhelming probability* if its probability is $1 - \text{negl}(n)$. The function $\lg x$ is the base 2 logarithm of x. The notation $\lfloor x \rceil$ denotes the nearest integer to x, rounding towards 0 for half-integers.

2.1 Lattices

Let $\mathbf{B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_n\} \subset \mathbb{R}^n$ consist of n linearly independent vectors. The n-dimensional lattices Λ generated by the basis $\mathbf{B}$ is

$$\Lambda = \{\mathbf{B}\mathbf{c} = \sum_{i\in[n]} \mathbf{c}_i \cdot \mathbf{b}_i \ : \ \mathbf{c} \in \mathbb{Z}^n\}$$

In this thesis, we will be using integer lattices, namely discrete subgroups of $\mathbb{Z}^m$.

Definition 2.1.1. *For a prime q, $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and $u \in \mathbb{Z}_q^n$, define:*

$$\begin{aligned}
\Lambda_q(\mathbf{A}) &:= \{\mathbf{e} \in \mathbb{Z}^m \ \ s.t.\ \exists \mathbf{s} \in \mathbb{Z}_q^n \ where\ \mathbf{A}^T\mathbf{s} = \mathbf{e} \mod q\} \\
\Lambda_q^{\perp}(\mathbf{A}) &:= \{\mathbf{e} \in \mathbb{Z}^m \ \ s.t.\ \mathbf{A}^T\mathbf{e} = 0 \mod q\} \\
\Lambda_q^{\mathbf{u}}(\mathbf{A}) &:= \{\mathbf{e} \in \mathbb{Z}^m \ \ s.t.\ \mathbf{A}\mathbf{e} = \mathbf{u} \mod q\}
\end{aligned}$$

Observe that if $\mathbf{t} \in \Lambda_q^{\mathbf{u}}(\mathbf{A})$, then $\Lambda_q^{\mathbf{u}}(\mathbf{A}) = \Lambda_q^{\perp}(\mathbf{A}) + \mathbf{t}$ and hence $\Lambda_q^{\mathbf{u}}(\mathbf{A})$ is a shift of $\Lambda_q^{\perp}(\mathbf{A})$.

2.2 Gaussian distribution and Metrics

Gaussian distributions. For any positive parameter $\sigma \in \mathbb{R}_{>0}$, let $\rho_\sigma(\mathbf{x}) := \exp\left(-\pi\|\mathbf{x}\|^2/\sigma^2\right)$ be the Gaussian function on $\mathbb{R}^m$ with parameter σ (and center $\mathbf{0}$). Let $\rho_\sigma(\mathbb{Z}^m) := \sum_{\mathbf{x}\in\mathbb{Z}^m} \rho_\sigma(\mathbf{x})$ be the discrete integral of ρ_σ over $\mathbb{Z}^m$ (which always converges). Let $D_{\mathbb{Z}^m,\sigma}$ be the truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with parameter σ. Specifically, for all $\mathbf{y} \in \mathbb{Z}^m$, we have $D_{\mathbb{Z}^m,\sigma}(\mathbf{y}) = \frac{\rho_\sigma(\mathbf{y})}{\rho_\sigma(\mathbb{Z}^m)}$. Moreover, when we sample from $D_{\mathbb{Z}^m,\sigma}$ and $|| \cdot ||_\infty$ norm exceeds $\sqrt{m} \cdot \sigma$, we replace the output by $\mathbf{0}$. Note that $D_{\mathbb{Z}^m,\sigma}$ is $\sqrt{m} \cdot \sigma$-bounded.

The Gram-Schmidt Norm of a Basis. Let $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_k]$ be a set of vectors in $\mathbb{R}^m$. We use the following notation:

- $||\mathbf{S}||$ denote the L_2 length of the longest vector in S, i.e. $||\mathbf{S}|| := \max_i ||\mathbf{s}_i||$ for $1 \leq i \leq k$.
- $\tilde{\mathbf{S}} := \{\tilde{\mathbf{s}_1}, \ldots, \tilde{\mathbf{s}_k}\} \subset \mathbb{R}^m$ denote the Gram-Schmidt orthogonalization of the vectors $\mathbf{s}_1, \ldots, \mathbf{s}_k$ taken in that order.

We refer to $\|\mathbf{S}\|_{\mathsf{GS}} := ||\tilde{\mathbf{S}}||$ as the Gram-Schmidt norm of $\mathbf{S}$.

Entropy and Statistical Distance. For random variables X, Y with support $\mathcal{X}, \mathcal{Y}$ respectively, we define the *statistical distance* $\triangle(X,Y) \overset{\text{def}}{=} \frac{1}{2}\sum_{u\in\mathcal{X}\cup\mathcal{Y}} |\Pr[X = u] - \Pr[Y = u]|$. We say that two ensembles of random variables $X = \{X_\lambda\}, Y = \{Y_\lambda\}$ are *statistically close*, denoted by $X \overset{\text{stat}}{\approx} Y$, if $\triangle(X_\lambda, Y_\lambda) = \text{negl}(\lambda)$. The *min-entropy* of a random variable X, denoted as $\mathbf{H}_\infty(X)$, is defined as $\mathbf{H}_\infty(X) \overset{\text{def}}{=} -\log(\max_x \Pr[X = x])$. The *(average-)conditional min-entropy* of a random variable X conditioned on a correlated variable Y, denoted as $\mathbf{H}_\infty(X|Y)$, is defined as

$$\mathbf{H}_\infty(X|Y) \overset{\text{def}}{=} -\log\left(\underset{y\leftarrow Y}{\mathbf{E}}\left[\max_x \Pr[X = x|Y = y]\right]\right).$$

The optimal probability of an unbounded adversary guessing X given the correlated value Y is $2^{-\mathbf{H}_\infty(X|Y)}$.

Lemma 2.2.1 ([DORS08]). *Let X, Y be arbitrarily random variables where the support of Y lies in $\mathcal{Y}$. Then $\mathbf{H}_\infty(X|Y) \geq \mathbf{H}_\infty(X) - \log(|\mathcal{Y}|)$.*

2.3 Lattice Assumptions

The Learning With Errors (LWE) Assumption. [Reg09] For an integer $q = q(n) \geq 2$ and a (truncated) discrete Gaussian error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the learning with errors assumption $\mathsf{dLWE}_{n,m,q,\chi}$ states that it is computationally hard to distinguish between the following pairs of distributions:

$$\{\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e}\} \quad \text{and} \quad \{\mathbf{A}, \mathbf{u}\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}, \mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n, \mathbf{e} \xleftarrow{\$} \chi^m, \mathbf{u} \xleftarrow{\$} \mathbb{Z}_q^m$.

Connection to lattices. Let $B = B(n) \in \mathbb{N}$. A family of distributions $\chi = \{\chi_n\}_{n\in\mathbb{N}}$ is called B-bounded if

$$\Pr[\chi \in \{-B, \ldots, B-1, B\}] = 1.$$

There are known quantum [Reg09] and classical [Pei09] reductions between $\mathsf{dLWE}_{n,m,q,\chi}$ and approximating short vector problems in lattices in the worst case, where χ is a B-bounded (truncated) discretized Gaussian for some appropriate B. The state-of-the-art algorithms for these lattice problems run in time nearly exponential in the dimension n [AKS01, MV10]; more generally, we can get a 2^k-approximation in time $2^{\tilde{O}(n/k)}$. Combined with the connection to LWE, this means that the $\mathsf{dLWE}_{n,m,q,\chi}$ assumption is quite plausible for a poly(n)-bounded distribution χ and q as large as 2^{n^ϵ} (for any constant $0 < \epsilon < 1$). Throughout this paper, the parameter $m = \text{poly}(n)$, in which case we will shorten the notation slightly to $\mathsf{LWE}_{n,q,\chi}$.

The Short Integer Solution Problem. [MR07] Let n, m, q, β be integer parameters. In the $\mathsf{SIS}(n, m, q, \beta)$ problem, the attacker is given a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and her goal is to find a vector $\mathbf{u} \in \mathbb{Z}_q^m$ with $\mathbf{u} \neq \mathbf{0}$ and $||\mathbf{u}||_\infty \leq \beta$ such that $\mathbf{A} \cdot \mathbf{u} = \mathbf{0}$.[1] For parameters $n = n(\lambda), m = m(\lambda), q = q(\lambda), \beta = \beta(\lambda)$ defined in terms of the security parameter sec, the $\mathsf{SIS}(n, m, q, \beta)$ hardness assumption states any PPT attacker $\mathcal{A}$ we have

$$\Pr\left[\ \mathbf{A} \cdot \mathbf{u} = \mathbf{0} \ \wedge\ ||\mathbf{u}||_\infty \leq \beta(\lambda) \ \wedge\ \mathbf{u} \neq \mathbf{0} \ :\ \mathbf{A} \xleftarrow{\$} \mathbb{Z}_{q(\lambda)}^{m(\lambda)\times n(\lambda)}, \mathbf{u} \leftarrow \mathcal{A}(1^\lambda, \mathbf{A})\ \right] \leq \text{negl}(\lambda).$$

The SIS problem is known to be as hard as certain worst-case problems (e.g., SIVP) in standard lattices [Ajt96, Mic04, MR07, MP13]. It is is also implied by the hardness of *learning with errors* (LWE). See cited works for exact details of parameters. In this work, we will need to rely on the SIS assumption with super-polynomial β. In particular, we will assume that for any $\beta = 2^{\text{poly}(\lambda)}$ there are some $n = \text{poly}(\lambda)$, $q = 2^{\text{poly}(\lambda)}$ (clearly, $q > \beta$) such that for all $m = \text{poly}(\lambda)$ the $\mathsf{SIS}(n, m, q, \beta)$ hardness assumption holds. The above parameters translate to assuming hardness of worst-case lattice problems with sub-exponential approximation factors, which is widely believed to hold.

[1]Often, the SIS problem is stated with ℓ_2 norm rather than ℓ_∞ norm. It's clear that the two versions are equivalent up to some small losses of parameters. Therefore, we choose to rely on the ℓ_∞ norm for simplicity.

2.4 Lattice Algorithms

Lemma 2.4.1 (Lattice Trapdoors [Ajt99, GPV08, AP09, MP12]). *There exist efficient algorithms* $\mathsf{TrapSamp}$, SamPre *such that the following holds. Given integers $n \geq 1, q \geq 2$ there exists some $m^* = m^*(n,q) = O(n \log q)$ such that for all $m \geq m^*$ and all k (polynomial in n) we have:*

1. *There is an efficient randomized algorithm* $\mathsf{TrapSamp}(1^n, 1^m, q)$ *that outputs a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a trapdoor matrix $\mathbf{T} \in \mathbb{Z}^{m \times m}$ such that the distribution of $\mathbf{A}$ is* $\mathrm{negl}(n)$*-close to uniform.*

2. *Moreover, there is an efficient algorithm* SamPre *that with overwhelming probability over all random choices, does the following: For any $\mathbf{u} \in \mathbb{Z}_q^n$, and large enough $s = \Omega(\sqrt{n \log q})$, the randomized algorithm* $\mathsf{SamPre}(\mathbf{A}, \mathbf{T}, \mathbf{u}, s)$ *outputs a vector $\mathbf{r} \in \mathbb{Z}^m$ with norm $||\mathbf{r}||_\infty \leq ||\mathbf{r}||_2 \leq s\sqrt{n}$ (with probability 1). Furthermore, the following distributions of the tuple $(\mathbf{A}, \mathbf{T}, \mathbf{U}, \mathbf{R})$ are within* $\mathrm{negl}(n)$ *statistical distance of each other for any polynomial $k \in \mathbb{N}$:*

   - $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$; $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n \times k}$; $\mathbf{R} \leftarrow \mathsf{SamPre}(\mathbf{A}, \mathbf{T}, \mathbf{U}, s)$.

   - $(\mathbf{A}, \mathbf{T}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$; $\mathbf{R} \xleftarrow{\$} (D_{\mathbb{Z}^m, s})^k$; $\mathbf{U} := \mathbf{AR} \pmod q$.

3. *Given n, m, q as above, there is an efficiently and deterministically computable matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ and a deterministic polynomial-time algorithm $\mathbf{G}^{-1}$ which takes the input $\mathbf{V} \in \mathbb{Z}_q^{n \times k}$ for any integer k and outputs $\mathbf{R} = \mathbf{G}^{-1}(\mathbf{V})$ such that $\mathbf{R} \in \{0,1\}^{m \times k}$ and $\mathbf{G} \cdot \mathbf{R} = \mathbf{V}$.* [2]

Lemma 2.4.2 ([MG02, Lemma 7.1]). *Let Λ be an m-dimensional lattice. There is a deterministic polynomial time algorithm that, given an arbitrary basis of Λ and full-rank set $\mathbf{S} = [\mathbf{s}_1, \ldots, \mathbf{s}_m]$ in Λ, returns a basis $\mathbf{T}$ of Λ satisfying*

$$||\mathbf{T}||_{\mathsf{GS}} \leq ||\mathbf{S}||_{\mathsf{GS}} \text{ and } ||\mathbf{T}|| \leq ||\mathbf{S}||\sqrt{m}/2$$

We will also use the following algorithms to sample short vectors from specific lattices.

[2]Note that we are abusing notation and $\mathbf{G}^{-1}$ is not a matrix but rather an algorithm. Often, $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ is referred to as a primitive matrix [MP12] which is a powers-of-two decomposition of an $n \times n$ identity matrix (that is, each entry a of $\mathbf{G}$ is decomposed into a vector $[a \cdot 2^0, a \cdot 2^1, \ldots, a \cdot 2^{\log q}]$. The algorithm $\mathbf{G}^{-1}$ then simply uses binary bit decomposition to sample for short pre-images.

Algorithm $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{u}, \alpha)$**:**

> *Inputs:* a full rank matrix $\mathbf{A}$ in $\mathbb{Z}_q^{n\times m}$, a "short" basis $\mathbf{T}_A$ of $\Lambda_q^{\perp}(\mathbf{A})$, a matrix $\mathbf{B}$ in $\mathbb{Z}_q^{n\times m_1}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter α. (2.1)
>
> *Output:* Let $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$. The algorithm outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+m_1}$ in the coset $\Lambda_q^{\mathbf{u}}(\mathbf{F})$.

Theorem 2.4.3 ([ABB10a, Theorem 17], [CHKP12, Lemma 3.2])**.** *Let $q > 2$, $m > n$ and $\alpha > \|\mathbf{T_A}\|_{\mathsf{GS}} \cdot \omega(\sqrt{\log(m + m_1)})$. Then* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{u}, \alpha)$ *taking inputs as in (2.1) outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+m_1}$ distributed statistically close to $D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}),\alpha}$, where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$.*

Where $\|\mathbf{T}\|_{\mathsf{GS}}$ refers to the norm of Gram-Schmidt orthogonalisation of $\mathbf{T}$.

Algorithm $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \mathbf{u}, \alpha)$**:**

> *Inputs:* matrices $\mathbf{A}$ in $\mathbb{Z}_q^{n\times k}$ and $\mathbf{R}$ in $\mathbb{Z}^{k\times m}$, a full rank matrix $\mathbf{B}$ in $\mathbb{Z}_q^{n\times m}$, a "short" basis $\mathbf{T_B}$ of $\Lambda_q^{\perp}(\mathbf{B})$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter α. (2.2)
>
> *Output:* Let $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR} + \mathbf{B})$. The algorithm outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+k}$ in the coset $\Lambda_q^{\mathbf{u}}(\mathbf{F})$.

Often the matrix $\mathbf{R}$ given to the algorithm as input will be a random matrix in $\{1, -1\}^{m\times m}$. Let S^m be the m-sphere $\{\mathbf{x} \in \mathbb{R}^{m+1} \ : \ \|\mathbf{x}\| = 1\}$. We define $s_R := \|\mathbf{R}\| := \sup_{\mathbf{x}\in S^{m-1}} \|\mathbf{R} \cdot \mathbf{x}\|$.

Theorem 2.4.4 ([ABB10a, Theorem 19])**.** *Let $q > 2, m > n$ and $\alpha > \|\mathbf{T_B}\|_{\mathsf{GS}} \cdot s_R \cdot \omega(\sqrt{\log m})$. Then* $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \mathbf{u}, \alpha)$ *taking inputs as in (2.2) outputs a vector $\mathbf{e} \in \mathbb{Z}^{m+k}$ distributed statistically close to $D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}),\alpha}$, where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR} + \mathbf{B})$.*

Proof. The algorithm works in three steps:

1. First, it constructs a set $\mathbf{T_F}$ of $(m + k)$ linearly independent vectors in $\Lambda_q^{\perp}(\mathbf{F})$ such that
$$\|\mathbf{T_F}\|_{\mathsf{GS}} < \|\mathbf{T_B}\|_{\mathsf{GS}}(s_R + 1) < \alpha/\omega(\sqrt{\log m})$$
To do this, let $\mathbf{T_B} = [\mathbf{b}_1, \dots, \mathbf{b}_m]$ and for all $i = 1, \dots, m$ set $\mathbf{t}_i := (-\mathbf{R}\mathbf{b}_i|\mathbf{b}_i) \in \mathbb{Z}^{m+k}$ and view it as a column vector. Then, clearly $\mathbf{F}\mathbf{t}_i = \mathbf{B}\mathbf{b}_i = 0 \mod q$ and therefore $t_i \in \Lambda_q^{\perp}(\mathbf{F})$. Also, for $i = 1, \dots, k$ let $\mathbf{w}_i$ be the i-th column of the identity matrix $\mathbf{I}_k$. Let $\mathbf{u}_i$ be an arbitrary vector in $\mathbb{Z}^m$ satisfying $\mathbf{A}\mathbf{w}_i + \mathbf{B}\mathbf{u}_i = 0 \mod q$. Set $t_{i+m} := (\mathbf{w}_i - \mathbf{R}\mathbf{u}_i|\mathbf{u}_i) \in \mathbb{Z}^{m+k}$ and view it as a column vector. Then, $\mathbf{F}\mathbf{t}_{i+m} = \mathbf{A}\mathbf{w}_i + \mathbf{B}\mathbf{u}_i = 0 \mod q$ and hence $\mathbf{t}_{i+m} \in \Lambda_q^{\perp}(\mathbf{F})$. As it was shown in [ABB10a, Lemma 18], the vectors $\mathbf{T_F} := [\mathbf{t}_1, \dots, \mathbf{t}_{m+k}]$ are linearly independent and satisfy $\|\mathbf{T_F}\|_{\mathsf{GS}} \le \|\mathbf{T_B}\|_{\mathsf{GS}} \cdot (s_R + 1)$.

2. Next, it converts $\mathbf{T_F}$ into a basis $\mathbf{T}'_{\mathbf{F}}$ of $\Lambda_q^{\perp}(\mathbf{F})$ with the same Gram-Schmidt norm as $\mathbf{T_F}$ using Lemma 2.4.2.

3. Finally, it invokes $\mathsf{SamPre}(\mathbf{F}, \mathbf{T}'_{\mathbf{F}}, \mathbf{u}, \alpha)$ to generate a vector $\mathbf{e} \in \Lambda_q^{\mathbf{u}}(\mathbf{F})$. Since $\alpha > \|\mathbf{T}'_{\mathbf{F}}\|_{\mathsf{GS}} \cdot \omega(\sqrt{\log m})$, the vector $\mathbf{e}$ is distributed close to $D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}),\alpha}$.

□

Lemma 2.4.5 (Generalized left-over hash lemma [DORS08, Lemmas 2.2b, 2.4])**.** *Let $\mathcal{H} = \{h : X \to Y\}_{h \in \mathcal{H}}$ be a universal hash family. Let $f : X \to Z$ be some function. Then for any random variable T taking values in X we have that*

$$\triangle\big((h, h(T), f(T)) \ ; \ (h, U, f(T))\big) \leq 1/2 \cdot \sqrt{\gamma(T) \cdot |Y| \cdot |Z|}$$

where U is chosen at random from Y and $\gamma(T) = \max_t Pr[T = t]$.

We will also use the following lemma throughout the thesis.

Lemma 2.4.6 ([ABB10a, Lemma 12])**.** *Suppose that $m \geq (n+1)\log q + \omega(\log n)$ and that $q \geq 2$ is prime. Let $\mathbf{R}$ be an $m \times k$ matrix chosen uniformly from $\{-1, 1\}^{m \times k} \mod q$ where $k = k(n)$ is polynomial in n. Let $\mathbf{A}$ and $\mathbf{B}$ be matrices chosen uniformly in $\mathbb{Z}^{n \times m}$ and $\mathbb{Z}^{n \times k}$ respectively. Then for all vectors $\mathbf{w} \in \mathbb{Z}^m$, the distribution $(\mathbf{A}, \mathbf{AR}, \mathbf{R}^T\mathbf{w})$ is statistically close to distribution $(\mathbf{A}, \mathbf{B}, \mathbf{R}^T\mathbf{w})$.*

Proof. On the high level, the lemma follows from two observations. First, by the standard left-over-hash lemma we know that $(\mathbf{A}, \mathbf{AR})$ and $(\mathbf{A}, \mathbf{B})$ are statistically close. Moreover, Dodis et al. [DORS08] showed that this holds even even little information about $\mathbf{R}$ is leaked (that is $\mathbf{R}^T\mathbf{w}$ in our case). Define the family of hash functions $\mathcal{H} = \{h_{\mathbf{A}} : \mathbb{Z}_q^m \to \mathbb{Z}_q^n\}$ where $h_{\mathbf{A}}(\mathbf{r}) = \mathbf{Ar}$ and $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$. Since q is prime we have that for all $\mathbf{r}_1 \neq \mathbf{r}_2 \in \mathbb{Z}_q^m$ there are exactly $q^{n(m-1)}$ matrices $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ such that $\mathbf{Ar}_1 = \mathbf{Ar}_2$. Hence $\mathcal{H}$ is universal. For a vector $\mathbf{w} \in \mathbb{Z}_q^m$, let $f : \mathbb{Z}_q^m \to \mathbb{Z}_q$ be the function $f(\mathbf{r}) = \mathbf{r}^T \cdot \mathbf{w}$. Observe that for a matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times k}$ whose columns are $\mathbf{r}_1, \ldots, \mathbf{r}_k \in \mathbb{Z}_q^m$ we have that $\mathbf{R}^T\mathbf{w} = (f(\mathbf{r}_1), \ldots, f(\mathbf{r}_k)) \in \mathbb{Z}_q^k$. Similarly, the columns of the matrix $\mathbf{AR}$ are k columns vectors $(h_A(\mathbf{r}_1), \ldots, h_A(\mathbf{r}_k))$.

Using the notation of Lemma 2.4.5, observe that the k columns of $\mathbf{R}$ are independent vectors uniform in $\{1, -1\}^m$. Therefore, letting $T_1, \ldots, T_m$ be m columns of $\mathbf{R}$ and setting $X = \mathbb{Z}_q^m, Y = \mathbb{Z}_q^n$ and $Z = \mathbb{Z}_q$ we obtain that

$$\triangle\big((\mathbf{A}, \mathbf{AR}, \mathbf{R}^T\mathbf{w}) \ ; \ (\mathbf{A}, \mathbf{B}, \mathbf{R}^T\mathbf{w})\big) \leq k/2 \cdot \sqrt{2^{-m} \cdot q^n \cdot q} = k/2 \cdot \sqrt{2^{-m+(n+1)\log q}}.$$

When $m > (n+1)\log q + \omega(\log n)$ and k is polynomial in n, the quantity on the right is $\text{negl}(n)$, as required. □

Chapter 3

Attribute-Based Encryption for Circuits

Attribute-based encryption [SW05, GPSW06] is an emerging paradigm for public-key encryption which enables fine-grained control of access to encrypted data. In traditional public-key encryption, access to the encrypted data is all or nothing: given the secret key, one can decrypt and read the entire message, but without it, nothing about the message is revealed (other than its length). In attribute-based encryption, an encryption of a message m is labeled with a *public* attribute vector a (also called the "index"), and secret keys are associated with predicates P. A secret key sk_P decrypts the ciphertext and recovers the message m if and only if a satisfies the predicate, namely if and only if $P(a) = 1$.

Attribute-based encryption captures as a special case previous cryptographic notions such as identity-based encryption (IBE) [Sha84, BF01, Coc01] and fuzzy IBE [SW05]. It has also found applications in scenarios that demand complex policies to control access to encrypted data (such as medical or multimedia data [APG$^+$11, LYZ$^+$13, PPM$^+$14]), as well as in designing cryptographic protocols for verifiably outsourcing computations [PRV12] (see Section 3.1.2 for further discussion).

The crucial component in the security requirement for attribute-based encryption stipulates that it resists *collusion attacks*, namely any group of users collectively learns nothing about the message m if none of them is individually authorized to decrypt the ciphertext.

In the past few years, there has been significant progress in attribute-based encryption in terms of efficiency, security guarantees, and diversifying security assumptions [GPSW06, Wat09, LW10, LOS$^+$10, CHKP12, ABB10a, OT10]. On the other hand, little progress has been made in terms of supporting larger classes of predicates. The state of the art is Boolean formulas [GPSW06, LOS$^+$10, OT10], which is a subclass of log-space computations. Constructing a secure attribute-based encryption for all polynomial-time predicates was posed as a central challenge by Boneh, Sahai and Waters [BSW11]. We resolve this problem affirmatively in this chapter.

3.1 Our Contributions, Techniques and Related Work

We construct attribute-based encryption schemes for circuits of every a-priori bounded depth, based on the learning with errors (LWE) assumption. In the course of our construction, we present a new framework for constructing attribute-based encryption schemes, based on a primitive that we call "two-to-one recoding" (TOR). Our methodology departs significantly from the current line of work on attribute-based encryption [GPSW06, LOS$^+$10] and instead, builds upon the connection to garbled circuits developed in the context of *bounded* collusions [SS10b, GVW12]. Along the way, we make the first substantial progress towards the 25-year-old open problem of constructing reusable garbled circuits. In particular, we obtain the first construction of reusable garbled circuits that satisfies authenticity, but not privacy. In a follow-up work, Goldwasser et al. [GKP$^+$13b] completely resolved this open problem by constructing reusable garbled circuits with authenticity and privacy; moreover, their construction relies crucially on our ABE scheme as an intermediate building block.

More specifically, for every class of predicate circuits with depth bounded by a polynomial function $d = d(\lambda)$ (where λ is the security parameter), we construct an ABE scheme that supports this class of circuits, under the learning with errors (LWE) assumption. Informally, the (decisional) LWE problem [Reg09] asks to distinguish between "noisy" random linear combinations of n numbers $\mathbf{s} = (s_1, \ldots, s_n) \in \mathbb{Z}_q^n$ from uniformly random numbers over $\mathbb{Z}_q$.

Regev [Reg09] showed that solving the LWE problem *on the average* is as hard as (quantumly) solving several notoriously difficult lattice problems *in the worst case.* Since then, the LWE assumption has become a central fixture in cryptography. We now have a large body of work building cryptographic schemes under the LWE assumption, culminating in the construction of a fully homomorphic encryption scheme [BV11b].

The key parameter that determines the hardness of LWE is the ratio between the modulus q and the maximum absolute value of the noise B; as such, we refer to q/B as the hardness factor of LWE. The problem becomes easier as this ratio grows, but is believed to be hard for 2^{n^ϵ}-time algorithms when $q/B = 2^{O(n^\epsilon)}$, where $0 < \epsilon < 1/2$. Our results will hold as long as the latter holds for *some constant* ϵ.

In particular, our main result constructing selectively secure attribute based encryption (where the adversary must commit to the challenge public attribute vector a before seeing the public parameters) can be summarized as follows:

Theorem 3.1.1 (informal)**.** *Assume that there is a constant $0 < \epsilon < 1$ for which the LWE problem is hard for a $\exp(n^\epsilon)$ factor in dimension n, for all large enough n. Then, for any polynomial d, there is a* selectively secure *attribute encryption scheme for general circuits of depth d.*

Moreover, our scheme has succinct ciphertexts, in the sense that the ciphertext size depends polynomially on the depth d and the length ℓ of the attribute vector a, but not on the size of the circuits in the class. The construction as stated achieves the weaker notion of selective security, but we can easily obtain a fully secure scheme following [BB04] (but using sub-exponential hardness in a crucial way):

Corollary 3.1.2. *Assume that there is a constant $0 < \epsilon < 1/2$ such that the LWE problem with a factor of $\exp(n^{\epsilon})$ is hard in dimension n for $\exp(n^{\epsilon})$-time algorithms. Then, for any polynomial d, there is a* fully secure *attribute-based encryption scheme for general circuits of depth d.*

We also obtain a new ABE scheme for branching programs (which correspond to the complexity class *LOGSPACE*) under the weaker quasi-polynomial hardness of LWE:

Theorem 3.1.3 (informal). *There exist attribute-based encryption schemes for the class of branching programs under either (1) the hardness of the LWE problem with an $n^{\omega(1)}$ factor, or (2) the bilinear decisional Diffie-Hellman assumption.*

Here, there is no a-prori bound on the size or the depth of the branching program. In addition, we achieve succinct ciphertexts of size $O(\ell)$ where ℓ is the number of bits in the index. Prior to this work, under the same $n^{\omega(1)}$-hardness assumption of LWE that we use, the state of the art constructions were limited to IBE and inner product encryption [CHKP12, ABB10a, AFV11] (that is, special cases of branching programs). However, under the same bilinear maps assumption, Goyal et al. [GPSW06] achieved a similar result to ours using secret sharing for general access structures. Our bilinear construction is a different way to achieve the same result, but exploits a combinatorial property of branching programs. The construction is inspired by a pairings-based scheme for regular languages in [Wat12].

We now move on to provide a technical roadmap of our construction: first, we define a new primitive that we call a *two-to-one recoding* (TOR) scheme; we then show how TOR gives us an attribute-based encryption scheme for circuits, and how to construct a TOR scheme from the LWE assumption.

3.1.1 New Framework: TOR

A Two-to-One Recoding (TOR) scheme is a family of (probabilistic) functions $\{\mathsf{Encode}(\mathsf{pk}, \cdot)\}$ indexed by pk, together with a "two-to-one" recoding mechanism. The basic computational security guarantee for $\mathsf{Encode}(\mathsf{pk}, \cdot)$ is that of (correlated) pseudorandomness [RS10]: $\mathsf{Encode}(\mathsf{pk}, s)$ should be pseudorandom given $\mathsf{Encode}(\mathsf{pk}_i, s)$ for polynomially many pk_i's, where s is a uniformly random "seed".

The recoding mechanism guarantees that given any triple of public keys $(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{pk}_{\text{tgt}})$, there is a recoding key rk that allows us to perform the transformation

$$(\mathsf{Encode}(\mathsf{pk}_0, s), \mathsf{Encode}(\mathsf{pk}_1, s)) \mapsto \mathsf{Encode}(\mathsf{pk}_{\text{tgt}}, s).$$

Such a recoding key rk can be generated using either of the two secret keys sk_0 or sk_1. Furthermore, the recoding mechanism must satisfy a natural simulation requirement: namely, we can generate rk given just $\mathsf{pk}_0, \mathsf{pk}_1$ (and neither of the two secret keys), if we are allowed to "program" pk_{tgt}. That is, there are three ways of generating the pair $(\mathsf{pk}_{\text{tgt}}, \mathsf{rk})$ that are (statistically) indistinguishable: (1) given pk_{tgt}, generate rk using the secret key sk_0; (2) given pk_{tgt}, generate rk using the secret key sk_1; and (3) generate rk without either secret key, by "programming" the output public key pk_{tgt}.

This requirement demonstrates the *intuitive guarantee* that we expect from a two-to-one recoding mechanism: namely, the recoding key is "useless" given only one encoding, but not both encodings. For example, it is easy to see that given $\mathsf{Encode}(\mathsf{pk}_0, s)$ and rk (but not $\mathsf{Encode}(\mathsf{pk}_1, s)$), the output $\mathsf{Encode}(\mathsf{pk}_{\mathrm{tgt}}, s)$ is pseudorandom. Indeed, this is because rk could as well have been "simulated" using sk_1, in which case it is of no help in the distinguishing task.

The simulation requirement also rules out the trivial construction from trapdoor functions where rk is a trapdoor for inverting $\mathsf{Encode}(\mathsf{pk}_0, \cdot)$ or $\mathsf{Encode}(\mathsf{pk}_1, \cdot)$.

From TOR to Garbled Circuits. We start from the observation that our TOR primitive implies a form of *reusable* garbled circuits *with no input or circuit privacy*, but instead, with a form of *authenticity guarantee.* As we will see, this leads directly into our attribute-based encryption scheme.

Consider a two-input boolean gate with input wires u, v and output wire w, computing a function $G : \{0,1\} \times \{0,1\} \to \{0,1\}$. In Yao's garbled circuit construction, we associate each wire with a pair of strings (called "labels"). In addition, there is an associated translation table comprising of four values $v_{b,c}$ for $b, c \in \{0,1\}$, where $v_{b,c}$ allows us to perform the transformation:

$$L_{u,b}, L_{v,c} \mapsto L_{w,G(b,c)}$$

The garbled circuits construction guarantees that given the translation table and labels L_{u,b^*} and L_{v,c^*} for specific input bits b^* and c^*, we can obtain $L_{w,G(b^*,c^*)}$; however, the other label at the output, namely $L_{w,1-G(b^*,c^*)}$ remains hidden.[1]

In our setting, we replace labels with public keys, so that each wire is associated with a pair of public keys. As before, we also provide a translation table comprising four values $\mathsf{rk}_{b,c}$ where the recoding key $\mathsf{rk}_{b,c}$ allows us to perform the transformation

$$\mathsf{Encode}(\mathsf{pk}_{u,b}, s), \mathsf{Encode}(\mathsf{pk}_{v,c}, s) \mapsto \mathsf{Encode}(\mathsf{pk}_{w,G(b,c)}, s)$$

The security properties of the TOR scheme then give us the following guarantee: Given the translation table and encodings of s corresponding to b^*, c^*, we clearly compute the encoding of s corresponding to $G(b^*, c^*)$. However, the encoding corresponding to $1 - G(b^*, c^*)$ remains pseudorandom.

Moreover, crucially, the translation table is independent of s, so we can now "reuse" the translation table by providing fresh encodings with different choices of s. In a sentence, replacing strings by functions gives us the power of reusability.

In the garbled circuits construction, the four entries of the table are permuted and thus, one can perform the translation even without knowing what the input bits b^* and c^* are. This is possible because there is an efficient way to verify when the "correct" translation

[1]In the standard instantiation of Yao's garbled circuits, each label is a secret key of a semantically secure encryption. Moreover, each translation value $v_{b,c}$ is an encryption of key $L_{w,G(b,c)}$ under a pair of keys $L_{u,b}, L_{v,c}$. Given two keys L_{u,b^*}, L_{v,c^*} corresponding to input bits b^*, c^*, it is possible to decrypt and learn key $L_{w,G(b,c)}$ in clear, whereas the key $L_{w,1-G(b,c)}$ remains hidden.

key is being used. In contrast, in the reusable construction above, one has to know exactly which of the recoding keys to use. This is part of the reason why we are *unable to provide circuit or input privacy, but instead, only guarantee authenticity*, namely that an adversary can obtain only one of the two possible encodings at the output wire.

This construction forms the cornerstone of the subsequent work of Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich [GKP$^+$13b] who construct reusable garbled circuits with input and circuit privacy, by additionally leveraging the power of fully homomorphic encryption [Gen09, BV11b].

From TOR to Attribute-Based Encryption. How is all this related to attribute-based encryption? In our attribute-based encryption scheme for circuits, the encodings of s are provided in the ciphertext, and the translation tables are provided in the secret key. More precisely, each wire is associated with two TOR public keys, and the encryption of a message m under an index a is obtained by computing $\mathsf{Encode}(\mathsf{pk}_{i,a_i}, s)$ for every input wire i. The output encoding $\mathsf{Encode}(\mathsf{pk}_{\text{out}}, s)$ is then used to mask the message. We obtain the secret key corresponding to a circuit C by "stitching" multiple translation tables together, where the public keys for the input and output wires are provided in the public parameters, and we pick fresh public keys for the internal wires during key generation. In a nutshell, this gives us the guarantee that given a secret key sk_C and an encryption $\mathsf{Enc}(a, m)$ such that $C(a) = 1$, we can compute $\mathsf{Encode}(\mathsf{pk}_{\text{out}}, s)$ and thus recover the message. On the other hand, this value looks pseudorandom if $C(a) = 0$.

In our outline of reusable garbled circuits with authenticity, we wanted to reuse the garbled circuit $G(C)$ across multiple encryptions with indices $a_1, a_2, \ldots$ on which C always evaluates to 0. In attribute-based encryption, we also want reusability across multiple circuits $C_1, C_2, \ldots$ all of which evaluate to 0 on a fixed index a (in addition to multiple indices). Fortunately, the strong security properties of the TOR primitive provide us with this guarantee.

To obtain attribute-based encryption for branching programs, we are able to support a different notion of translation tables, which we can realize using a slightly weaker notion of TOR. In branching programs, the transition function depends on an input variable and the current state. The fact that one of these two values is always an input variable makes things simpler; in circuits, both of the input values to a gate could be internal wires.

TOR from LWE. We show how to instantiate TOR from LWE, building upon previous lattice-based IBE techniques in [GPV08, CHKP12, ABB10a]. The public key is given by a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, and

$$\mathsf{Encode}(\mathbf{A}, \mathbf{s}) = \mathbf{A}^T\mathbf{s} + \mathbf{e}$$

where $\mathbf{s} \in \mathbb{Z}_q^n$, $\mathbf{e} \in \mathbb{Z}_q^m$ is an error vector, and $\mathbf{A}^T$ denotes the transpose of the matrix $\mathbf{A}$. (Correlated) pseudorandomness follows directly from the LWE assumption. Given $\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_{\text{tgt}} \in \mathbb{Z}_q^{n\times m}$, the recoding key rk is given by a low-norm matrix $\mathbf{R} \in \mathbb{Z}_q^{2m\times m}$ such that

$$[\ \mathbf{A}_0 \parallel \mathbf{A}_1\ ]\,\mathbf{R} = \mathbf{A}_{\text{tgt}}$$

Note that

$$\mathbf{R}^T \left[\begin{array}{c} \mathbf{A}_0^T\mathbf{s} + \mathbf{e}_0 \\ \mathbf{A}_1^T\mathbf{s} + \mathbf{e}_1 \end{array} \right] \approx \mathbf{A}_{\text{tgt}}^T\mathbf{s}$$

which gives us the recoding mechanism. There are three ways of generating the public key $\mathbf{A}_{\text{tgt}}$ together with the recoding key $\mathbf{R}$: (1) using the trapdoor for $\mathbf{A}_0$, (2) using the trapdoor for $\mathbf{A}_1$, or (3) first generating $\mathbf{R}$ and then "programming" $\mathbf{A}_{\text{tgt}} := [\mathbf{A}_0 || \mathbf{A}_1]\ \mathbf{R}$. These three ways are statistically indistinguishable by the "bonsai trick" of [CHKP12]. In fact, our recoding mechanism is very similar to the lattice delegation mechanism introduced in [ABB10b], which also uses random low norm matrices to move from one lattice to another.

The multiplicative mechanism for recoding means that the noise grows exponentially with the number of sequential recodings. This, in turn, limits the depth of the circuits we can handle. In particular, the noise grows by a multiplicative $\mathsf{poly}(n)$ factor on each recoding, which means that after depth d, it becomes $n^{O(d)}$. Since $n^{O(d)} < q/4 < 2^{n^\epsilon}$, we can handle circuits of depth $\tilde{O}(n^\epsilon)$ (here, the first inequality is for correctness and the second for security). Viewed differently, setting the LWE dimension $n = d^{1/\epsilon}$ lets us handle circuits of maximum depth $d = d(\ell)$.

Our weak TOR for branching programs uses an additive mechanism, namely the recoding key is given by a low-norm matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{A}_0\mathbf{R} = \mathbf{A}_{\text{tgt}} - \mathbf{A}_1$. Note that $\mathbf{R}^T(\mathbf{A}_0^T\mathbf{s} + \mathbf{e}_0) + (\mathbf{A}_1^T\mathbf{s} + \mathbf{e}_1) \approx \mathbf{A}_{\text{tgt}}^T\mathbf{s}$ which gives us our recoding mechanism. Since in our branching program construction, $\mathbf{A}_0^T\mathbf{s} + \mathbf{e}_0$ will always be a fresh encoding provided in the ciphertext, the noise accumulation is additive rather than multiplicative.

3.1.2 Applications

Let us now explain the application of our result to the problem of publicly verifiable delegation of computation without input privacy.

A verifiable delegation scheme allows a computationally weak client to delegate expensive computations to the cloud, with the assurance that a malicious cloud cannot convince the client to accept an incorrect computation [Mic00, GKN08, GGP10a, CKP10, AIK10]. Recent work of Parno, Raykova and Vaikuntanathan [PRV12] showed that any attribute-based encryption scheme for a class of circuits with encryption time at most linear in the length of the index immediately yields a two-message delegation scheme for the class in the pre-processing model. Namely, there is an initial pre-processing phase which fixes the circuit C the client wishes to compute, produces a circuit key and sends it to the server. Afterwards, to delegate computation on an input x, the client only needs to send a single message. Moreover, the ensuing delegation scheme satisfies public delegatability, namely anyone can delegate computations to the cloud; as well as public verifiability, namely anyone can check the cloud's work (given a "verification" key published by the client). The previous delegation schemes that satisfy both these properties (secure in the standard model) supported the class NC^1 [PRV12, GPSW06, LW12]. Our attribute-based encryption schemes for circuits gives us a verifiable delegation scheme for all circuits, where the computation time of the client in the online phase is polynomial in the length of its input and the depth of the circuit, but is otherwise independent of the circuit size. We note that this scheme does not guarantee privacy of the input. Building on this work, Goldwasser et al. [GKP$^+$13b] show how to achieve a publicly verifiable delegation scheme with input privacy.

3.1.3 Related Work

Prior to this work, the state-of-art for lattice-based predicate encryption was threshold and inner product predicates [ABV$^+$12, AFV11]; realizing Boolean formula was itself an open problem. A different line of work considers definitional issues in the more general realm of functional encryption [BSW11, O'N10], for which general feasibility results are known for the restricted setting of a-priori bounded collusions developed from classical "one-time" garbled circuits [SS10a, GVW12] (the ciphertext size grows with both the circuit size and the collusion bound). Our methodology takes a fresh perspective on how to achieve reusability of garbled circuits with respect to authenticity. Our primitive (TOR) can be thought of as a generalization of the notion of proxy re-encryption [BBS98, AFGH06, HRSV11] which can be thought of as a one-to-one re-encryption mechanism.

Independent work. Boyen [Boy13b] gave a construction of an ABE scheme for Boolean formulas based on LWE; our result for LWE-based branching program subsumes the result since Boolean formulas are a subclass of branching programs. Garg, Gentry, Halevi, Sahai and Waters [GGH$^+$13c] gave a construction of attribute-based encryption for general circuits under a DBDH-like assumption in multi-linear groups; the construction extends to so-called graded encodings, for which we have candidates under non-standard assumptions in ideal lattices [GGH13a, CLT13]. The public parameters in the construction also grow with the depth of the circuit.

Subsequent Work. Our attribute-based encryption scheme has been used as the crucial component in the subsequent work of [GKP$^+$13b] to construct a (private index) functional encryption scheme with succinct ciphertexts. They also show a number of applications of their construction, including reusable garbled circuits *with input and circuit privacy.* Also subsequently, Boneh et al. [BGG$^+$14] gave asymptotic improvements on the sizes of secret keys and ciphertexts in two different constructions respectively. Their main construction is built from a new *fully key-homomorphic encryption* reduces the size of the secret key for a predicate P from $|P| \times \text{poly}(\lambda, d)$, shown in this work, to $|P| + \text{poly}(\lambda, d)$ where λ is the security parameter and d is the circuit depth. Moreover, their construction supports arithmetic circuits and key delegation. Our Chapter 4 is also a subsequent work of [GKP$^+$13b] and [BGG$^+$14]. Pandey et al. [PRW14] showed how to instantiate our two-to-one recoding scheme from multi-linear maps. Goldwasser et al. [GKP$^+$13a] construct ABE and other variants of functional encryption for Turing Machines using very strong intractability assumptions.

3.1.4 Chapter Organization

We present preliminaries in Section 3.2. We present our TOR framework and its instantiation in Sections 3.3 and 3.4. We present our ABE scheme in Section 3.5. We present the scheme for branching programs in Section 3.6. In Section 3.7, we present some extensions for our basic construction for circuits. In Section 3.8, we summarize and present some open problems.

3.2 Preliminaries

3.2.1 Attribute-Based Encryption

We define attribute-based encryption (ABE), following [GPSW06]. An ABE scheme ABE for a class of predicate circuits $\mathcal{C}$ (namely, circuits with a single bit output) consists of four PPT algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$:

$\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{mpk}, \mathsf{msk})$: The setup algorithm gets as input the security parameter λ, the length ℓ of the index, and outputs the master public key (mpk), and the master key msk.

$\mathsf{Enc}(\mathsf{mpk}, a, m) \to \mathsf{ct}_a$: The encryption algorithm gets as input mpk, an index $a \in \{0,1\}^\ell$ and a message $m \in \mathcal{M}$. It outputs a ciphertext ct_a. Note that a is public given ct_a.

$\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$: The key generation algorithm gets as input msk and a predicate specified by $C \in \mathcal{C}$. It outputs a secret key sk_C (where C is also public).

$\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_a) \to m$: The decryption algorithm gets as input sk_C and ct_a, and outputs either $\perp$ or a message $m \in \mathcal{M}$.

Correctness. We define and realize perfect correctness of the ABE scheme. Namely, for all (a, C) such that $C(a) = 1$, all $m \in \mathcal{M}$ and $\mathsf{ct}_a \leftarrow \mathsf{Enc}(\mathsf{mpk}, a, m)$, $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_a) = m$.

Security Definition For a stateful adversary $\mathcal{A}$ (that can maintain a global state information throughout the execution of the experiment), we define the advantage function $\mathsf{Adv}^{\mathrm{PE}}_{\mathcal{A}}(\lambda)$ to be

$$\Pr\left[b = b' : \begin{array}{l} a \leftarrow \mathcal{A}(1^\lambda, 1^\ell); \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell); \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(\mathsf{mpk}), |m_0| = |m_1|; \\ b \xleftarrow{\$} \{0,1\}; \\ \mathsf{ct}_a \leftarrow \mathsf{Enc}(\mathsf{mpk}, a, m_b); \\ b' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk}, \cdot)}(\mathsf{ct}_a) \end{array} \right] - \frac{1}{2}$$

with the restriction that all queries C that $\mathcal{A}$ makes to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ satisfies $C(a) = 0$ (that is, sk_C does not decrypt ct_a). An attribute-based encryption scheme is *selectively secure* if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\text{PE}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ. We call an attribute-based encryption scheme *fully secure* if the adversary $\mathcal{A}$ is allowed to choose the challenge index a after seeing secret keys, namely, along with choosing (m_0, m_1).

3.3 Two-to-One Recoding Schemes

An overview of TOR is provided in Section 3.1.1. Below, we first informally point out some properties of the encryption/decryption algorithms that are a part of TOR. Then, in Section 3.3.1 we provide a formal definition of all TOR algorithms and properties.

Symmetric encryption. In our construction, we will use $\mathsf{Encode}(\mathsf{pk}, s)$ as a one-time key for a symmetric-key encryption scheme (E, D). If Encode is deterministic, then we could simply use a one-time pad. However, since Encode is probabilistic, the one-time pad will not guarantee correctness. Instead, we require (E, D) to satisfy a stronger correctness guarantee, namely for all messages m and for all ψ, ψ' in the support of $\mathsf{Encode}(\mathsf{pk}, s)$, $\mathsf{D}(\psi', \mathsf{E}(\psi, m)) = m$.

Allowing degradation. With each recoding operation, the "quality" of encoding potentially degrades. In order to formalize this, we allow the initial global public parameters to depend on $d_{\max}$, an a-prior upper bound on the number of nested recoding operations. We then require that given any encodings ψ and ψ' that are a result of at most $d_{\max}$ nested recodings, $\mathsf{D}(\psi', \mathsf{E}(\psi, m)) = m$. We stress that we allow $d_{\max}$ to be super-polynomial, and in fact, provide such instantiations for a relaxed notion of TOR.

3.3.1 Definition of TOR

Formally, a TOR scheme over the input space $\mathcal{S} = \{\mathcal{S}_\lambda\}$ and outputs space $\mathcal{K} = \{\mathcal{K}_\lambda\}$ consists of six polynomial-time algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encode}, \mathsf{ReKeyGen}$, $\mathsf{SimReKeyGen}, \mathsf{Recode}$) and a symmetric-key encryption scheme (E, D) with the following properties:

- $\mathsf{Setup}(1^\lambda, d_{\max})$ is a probabilistic algorithm that takes as input the security parameter λ and an upper bound $d_{\max}$ on the number of nested recoding operations (written in binary), outputs "global" public parameters pp.

- $\mathsf{KeyGen}(\mathsf{pp})$ is a probabilistic algorithm that outputs a public/secret key pair $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{Encode}(\mathsf{pk}, s)$ is a probabilistic algorithm that takes pk and an input $s \in \mathcal{S}$, and outputs an encoding ψ.

In addition, there is a recoding mechanism together with two ways to generate recoding keys: given one of the two secret keys, or by programming the output public key.

- $\mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, \mathsf{pk}_{\text{tgt}})$ is a probabilistic algorithm that takes a key pair $(\mathsf{pk}_0, \mathsf{sk}_0)$, another public key pk_1, a "target" public key pk_{tgt}, and outputs a recoding key rk.

- $\mathsf{SimReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1)$ is a probabilistic algorithm that takes two public keys $\mathsf{pk}_0, \mathsf{pk}_1$ and outputs a recoding key rk together with a "target" public key $\mathsf{pk}_{\mathrm{tgt}}$.

- $\mathsf{Recode}(\mathsf{rk}, \psi_0, \psi_1)$ is a deterministic algorithm that takes the recoding key rk, two encodings ψ_0 and ψ_1, and outputs an encoding ψ_{tgt}.

Remark 3.3.1. *For our instantiation from lattices, we can in fact invert* $\mathsf{Encode}(\mathsf{pk}, s)$ *to recover* s *using the corresponding* sk. *However, we will not require this property in our generic constructions from TOR. Indeed, realizing this property over bilinear groups would be hard, since* s *is typically encoded in the exponent.*

Correctness. Correctness of a TOR scheme requires two things. First, for sufficiently large λ, for every pk and $s \in \mathcal{S}$, there exists a family of sets $\Psi_{\mathsf{pk},s,j}, j = 0, 1, \ldots, d_{\max}$:

- $\Pr[\mathsf{Encode}(\mathsf{pk}, s) \in \Psi_{\mathsf{pk},s,0}] = 1$, where the probability is taken over the coin tosses of Encode;

- $\Psi_{\mathsf{pk},s,0} \subseteq \Psi_{\mathsf{pk},s,1} \subseteq \cdots \subseteq \Psi_{\mathsf{pk},s,d_{\max}}$.

- for all $\psi, \psi' \in \Psi_{\mathsf{pk},s,d_{\max}}$ and all $m \in \mathcal{M}$, $\mathsf{D}(\psi', \mathsf{E}(\psi, m)) = m$.

Note that these properties hold trivially if Encode is deterministic and (E, D) is the one-time pad. Secondly, the correctness of recoding requires that for any triple of key pairs $(\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_{\mathrm{tgt}})$, and any encodings $\psi_0 \in \Psi_{\mathsf{pk}_0,s,j_0}$ and $\psi_1 \in \Psi_{\mathsf{pk}_1,s,j_1}$,

$$\mathsf{Recode}(\mathsf{rk}, \psi_0, \psi_1) \in \Psi_{\mathsf{pk}_{\mathrm{tgt}},s,\max(j_0,j_1)+1}$$

Statistical Security Properties. Note that we have three ways of sampling recoding keys: using $\mathsf{ReKeyGen}$ along with one of two secret keys sk_0 or sk_1; using $\mathsf{SimReKeyGen}$ while programming $\mathsf{pk}_{\mathrm{tgt}}$. We require that for all λ, all three ways lead to the same distribution of recoding keys, up to some statistical error.

Key Indistinguishability : Let $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $b = 0, 1$ and $(\mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_{\mathrm{tgt}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$.

Then, for all λ, the following two ensembles must be statistically indistinguishable:

$$\Big[\mathsf{Aux}, \mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1, \boxed{\mathsf{sk}_0}, \mathsf{pk}_{\mathrm{tgt}})\Big] \overset{s}{\approx} \Big[\mathsf{Aux}, \mathsf{ReKeyGen}(\mathsf{pk}_1, \mathsf{pk}_0, \boxed{\mathsf{sk}_1}, \mathsf{pk}_{\mathrm{tgt}})\Big]$$

where $\mathsf{Aux} = ((\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_{\mathrm{tgt}}))$. Informally, this says that sampling recoding keys using sk_0 or sk_1 yields the same distribution.

Recoding Simulation : Let $(\mathsf{pk}_b, \mathsf{sk}_b) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $b = 0, 1$. Then, for all λ, the following two ways of sampling the tuple $\left[(\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1), \mathsf{pk}_{\mathrm{tgt}}, \mathsf{rk}\right]$ must be statistically indistinguishable:

$$\left[(\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1), \mathsf{pk}_{\mathrm{tgt}}, \mathsf{rk} : (\mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_{\mathrm{tgt}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}); \mathsf{rk} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, \mathsf{pk}_{\mathrm{tgt}})\right] \overset{s}{\approx}$$
$$\left[(\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1), \mathsf{pk}_{\mathrm{tgt}}, \mathsf{rk} : (\mathsf{pk}_{\mathrm{tgt}}, \mathsf{rk}) \leftarrow \mathsf{SimReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1)\right]$$

In addition, we require one-time semantic security for (E, D):

One-time Semantic Security : For all $m_0, m_1 \in \mathcal{M}$ and all λ, the following two distributions must be statistically indistinguishable:

$$\left[\ \mathsf{E}(\psi, m_0) : \psi \overset{\$}{\leftarrow} \mathcal{K}\ \right] \overset{s}{\approx} \left[\ \mathsf{E}(\psi, m_1) : \psi \overset{\$}{\leftarrow} \mathcal{K}\ \right]$$

For all three properties, computational indistinguishability is sufficient for our applications, but we will achieve the stronger statistical indistinguishability in our instantiations.

Computational Security Property. We require that for all λ, given the encoding of a random s on $\ell = \mathrm{poly}(\lambda)$ keys, the evaluation at a fresh key is pseudorandom.

Correlated Pseudorandomness : For every polynomial $\ell = \ell(\lambda)$, let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for $i \in [\ell + 1]$. Let $s \overset{\$}{\leftarrow} \mathcal{S}$, and let $\psi_i \leftarrow \mathsf{Encode}(\mathsf{pk}_i, s)$ for $i \in [\ell + 1]$. Then, the following two ensembles must be computationally indistinguishable:

$$\left[\ (\mathsf{pk}_i, \psi_i)_{i \in [\ell]}, \mathsf{pk}_{\ell+1}, \boxed{\psi_{\ell+1}}\ \right] \overset{c}{\approx}$$
$$\left[\ (\mathsf{pk}_i, \psi_i)_{i \in [\ell]}, \mathsf{pk}_{\ell+1}, \boxed{\psi} : \psi \overset{\$}{\leftarrow} \mathcal{K}\ \right]$$

That is, we define the advantage function $\mathsf{Adv}^{\mathrm{CP}}_{\mathcal{A}}(\lambda)$ to be:

$$\Pr\left[b = b' : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); s \leftarrow \mathcal{S}; \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}), \\ \psi_i \leftarrow \mathsf{Encode}(\mathsf{pk}_i, s), i = 1, \ldots, \ell; \\ \psi'_0 \leftarrow \mathsf{Encode}(\mathsf{pk}_{\ell+1}, s); \\ b \overset{\$}{\leftarrow} \{0, 1\}; \psi'_1 \overset{\$}{\leftarrow} \mathcal{K} \\ b' \leftarrow \mathcal{A}(\mathsf{pk}_1, \ldots, \mathsf{pk}_{\ell+1}, \psi_1, \ldots, \psi_\ell, \psi'_b) \end{array} \right] - \frac{1}{2}$$

and we require that for all PPT $\mathcal{A}$, the advantage function $\mathsf{Adv}^{\mathrm{CP}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.

3.3.2 Simple Applications of TOR

In this section, we provide high-level sketches of how to use TOR to construct an analogue of Yao's garbled circuits (without privacy) and Identity-Based Encryption (IBE).

First example. We revisit the example from Section 3.1.1. Consider a two-input boolean gate g with input wires u, v and output wire w, computing a function $G : \{0,1\} \times \{0,1\} \to \{0,1\}$. Analogous to Yao's garbled circuit, we provide a translation table Γ comprising four values

$$\Gamma := (\ \mathsf{rk}_{b,c}\ :\ b, c \in \{0,1\}\ )$$

where $\mathsf{rk}_{b,c}$ allows us to perform the transformation

$$\mathsf{Encode}(\mathsf{pk}_{u,b}, s), \mathsf{Encode}(\mathsf{pk}_{v,c}, s) \mapsto \mathsf{Encode}(\mathsf{pk}_{w,G(b,c)}, s)$$

Now, fix b^*, c^* and $d^* := G(b^*, c^*)$. Given an encoding of s corresponding to b^* and c^*, we can compute that for d^* using the recoding key rk_{b^*,c^*}; in addition, we claim that the encoding corresponding to $1-d^*$ remains pseudorandom. To prove this, it suffices to simulate Γ given $\mathsf{pk}_{u,b^*}, \mathsf{pk}_{v,c^*}, \mathsf{pk}_{w,1-d^*}$ as follows:

- we sample $(\mathsf{pk}_{w,d^*}, \mathsf{rk}_{b^*,c^*})$ using $\mathsf{SimReKeyGen}$;
- we sample $\mathsf{pk}_{u,1-b^*}$ and $\mathsf{pk}_{v,1-c^*}$ along with the corresponding secret keys; using these secret keys, we can sample the other three recoding keys $\mathsf{rk}_{1-b^*,c^*}, \mathsf{rk}_{b^*,1-c^*}, \mathsf{rk}_{1-b^*,1-c^*}$.

IBE from TOR. As a warm-up, we show how to build a selectively secure IBE for identity space $\{0,1\}^\ell$.

$$\mathsf{mpk} := \begin{pmatrix} \mathsf{pk}_{1,0} & \mathsf{pk}_{2,0} & \cdots & \mathsf{pk}_{\ell,0} & \mathsf{pk}_{\mathrm{start}} \\ \mathsf{pk}_{1,1} & \mathsf{pk}_{2,1} & \cdots & \mathsf{pk}_{\ell,1} & \mathsf{pk}_{\mathrm{out}} \end{pmatrix}$$

The ciphertext for identity a and message m is given by:

$$\Big(\mathsf{Encode}(\mathsf{pk}_{1,a_1}, s), \ldots, \mathsf{Encode}(\mathsf{pk}_{\ell,a_\ell}, s), \mathsf{Encode}(\mathsf{pk}_{\mathrm{start}}, s), \mathsf{E}(\mathsf{Encode}(\mathsf{pk}_{\mathrm{out}}, s), m) \Big)$$

The secret key for identity a is given by $(\mathsf{rk}_1, \ldots, \mathsf{rk}_\ell)$ where we first sample

$$(\mathsf{pk}'_1, \mathsf{sk}'_1), \ldots, (\mathsf{pk}'_{\ell-1}, \mathsf{sk}'_{\ell-1}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$$

and then sample

$$\begin{array}{l} \mathsf{rk}_1 \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_{\mathrm{start}}, \mathsf{pk}_{1,a_1}, \mathsf{sk}_{\mathrm{start}}, \mathsf{pk}'_1) \\ \mathsf{rk}_2 \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}'_1, \quad \mathsf{pk}_{2,a_2}, \mathsf{sk}'_1, \quad \mathsf{pk}'_2) \\ \qquad\qquad\qquad\qquad \vdots \\ \mathsf{rk}_\ell \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}'_{\ell-1}, \ \mathsf{pk}_{\ell,a_\ell}, \mathsf{sk}'_{\ell-1}, \ \mathsf{pk}_{\mathrm{out}}) \end{array}$$

To prove selective security, we need to generate secret keys for any $a \neq a^*$, given $\mathsf{sk}_{1,1-a_1^*}, \ldots, \mathsf{sk}_{\ell,1-a_\ell^*}$ but not $\mathsf{sk}_{\text{start}}$ or sk_{out}. We can achieve this as follows: pick an i for which $a_i \neq a_i^*$;

- pick $(\mathsf{rk}_1, \mathsf{pk}'_1), \ldots, (\mathsf{rk}_{i-1}, \mathsf{pk}'_{i-1})$ using $\mathsf{SimReKeyGen}$;
- pick $(\mathsf{pk}'_i, \mathsf{sk}'_i), \ldots, (\mathsf{pk}'_{\ell-1}, \mathsf{sk}'_{\ell-1})$ using KeyGen;
- pick $\mathsf{rk}_i, \mathsf{rk}_{i+1}, \ldots, \mathsf{rk}_\ell$ using $\mathsf{ReKeyGen}$ with secret keys $\mathsf{sk}_{1-a_i^*}, \mathsf{sk}'_i, \ldots, \mathsf{sk}'_{\ell-1}$ respectively.

We note that our IBE construction from TOR seems incomparable to existing IBE constructions from lattices [CHKP12, ABB10a], as we follow more of a "sequential" binding to the identity a.

3.4 TOR from LWE

In this section, we present an instantiation of TOR from LWE, building upon ideas previously introduced in [GPV08, CHKP12, ABB10a].

Lemma 3.4.1. *Assuming $\mathsf{dLWE}_{n,q,\chi}$ there is a TOR scheme that is correct up to $d_{\max}$ levels, where $n = n(\lambda)$, $\chi = D_{\mathbb{Z},\sqrt{n}}$, $q = n^{\Theta(d_{\max})}$, $m = O(n \log q)$ and $q = n^{\Theta(d_{\max})}$.*

- $\mathsf{Setup}(1^\lambda, d_{\max})$: The parameters n, m, χ, q, are given from the assumption. Set the error bound $B = B(n) = O(n)$ and the Gaussian parameter $s = s(n) = O(\sqrt{n \log q})$. Output the global public parameters $\mathsf{pp} = (n, \chi, B, q, m, s)$.
- $\mathsf{KeyGen}(\mathsf{pp})$: Run the trapdoor generation algorithm $\mathsf{TrapSamp}(1^n, 1^m, q)$ to obtain a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with the trapdoor matrix $\mathbf{T} \in \mathbb{Z}^{m \times m}$. Output $\mathsf{pk} := \mathbf{A}$ and $\mathsf{sk} := \mathbf{T}$.
- $\mathsf{Encode}(\mathsf{pk}, \mathbf{s})$: Sample an error vector $\mathbf{e} \leftarrow \chi^m$ and output the encoding $\boldsymbol{\psi} := \mathbf{A}^T\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m$.

The recoding algorithms work as follows:

- $\mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_b, \mathsf{pk}_{\text{tgt}})$: Let $\mathsf{pk}_0 = \mathbf{A}_0$, $\mathsf{pk}_1 = \mathbf{A}_1$, $\mathsf{sk}_b = \mathbf{T}_b$ and $\mathsf{pk}_{\text{tgt}} = \mathbf{A}_{\text{tgt}}$. Compute the matrix $\mathbf{R} \in \mathbb{Z}^{2m \times m}$ in the following way:
  - Choose a discrete Gaussian matrix $\mathbf{R}_{1-b} \leftarrow (D_{\mathbb{Z},s})^{m \times m}$. Namely, each entry of the matrix is an independent sample from the discrete Gaussian distribution $D_{\mathbb{Z},s}$.
  - Compute $\mathbf{U} := \mathbf{A}_{\text{tgt}} - \mathbf{A}_{1-b}\mathbf{R}_{1-b} \in \mathbb{Z}_q^{n \times m}$.
  - Compute the matrix $\mathbf{R}_b$ by running the algorithm SamPre to compute a matrix $\mathbf{R}_b \in \mathbb{Z}^{m \times m}$ as follows:
  $$\mathbf{R}_b \leftarrow \mathsf{SamPre}(\mathbf{A}_b, \mathbf{T}_b, \mathbf{U})$$

Output

$$\mathsf{rk}_{0,1}^{\mathrm{tgt}} := \begin{bmatrix} \mathbf{R}_0 \\ \mathbf{R}_1 \end{bmatrix} \in \mathbb{Z}^{2m \times m}$$

(We remark that $\mathbf{A}_b\mathbf{R}_b = \mathbf{U} = \mathbf{A}_{\mathrm{tgt}} - \mathbf{A}_{1-b}\mathbf{R}_{1-b}$, and thus, $\mathbf{A}_0\mathbf{R}_0 + \mathbf{A}_1\mathbf{R}_1 = \mathbf{A}_{\mathrm{tgt}}$).

- $\mathsf{SimReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1)$: Let $\mathsf{pk}_0 = \mathbf{A}_0$ and $\mathsf{pk}_1 = \mathbf{A}_1$.
  - Sample a matrix $\mathbf{R} \leftarrow (D_{\mathbb{Z},s})^{2m\times m}$ by sampling each entry from the discrete Gaussian distribution $D_{\mathbb{Z},s}$.
  - Define
  $$\mathbf{A}_{\mathrm{tgt}} := [\mathbf{A}_0 \mid\mid \mathbf{A}_1]\ \mathbf{R} \in \mathbb{Z}_q^{n\times m}$$
  Output the pair $(\mathsf{pk}_{\mathrm{tgt}} := \mathbf{A}_{\mathrm{tgt}}, \mathsf{rk}_{0,1}^{\mathrm{tgt}} := \mathbf{R})$.

- $\mathsf{Recode}(\mathsf{rk}_{0,1}^{\mathrm{tgt}}, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1)$: Let $\mathsf{rk}_{0,1}^{\mathrm{tgt}} = \mathbf{R}$. Compute the recoded ciphertext
$$\boldsymbol{\psi}_{\mathrm{tgt}} = [\boldsymbol{\psi}_0^T \mid\mid \boldsymbol{\psi}_1^T]\ \mathbf{R}$$

We also need a one-time symmetric encryption scheme (E, D) which we will instantiate as an *error-tolerant* version of the one-time pad with $\mathcal{K} = \mathbb{Z}_q^m, \mathcal{M} = \{0,1\}^m$, as follows:

- $\mathsf{E}(\boldsymbol{\psi}, \boldsymbol{m})$ takes as input a vector $\boldsymbol{\psi} \in \mathbb{Z}_q^m$ and a bit string $\boldsymbol{m} \in \mathcal{M}^m$ and outputs the encryption
$$\boldsymbol{\gamma} := \boldsymbol{\psi} + \lceil q/2 \rceil\ \boldsymbol{m} \pmod q$$

- $\mathsf{D}(\boldsymbol{\psi}', \boldsymbol{\gamma})$ takes as input a vector $\boldsymbol{\psi}' = (\psi_1', \ldots, \psi_m') \in \mathbb{Z}_q^m$, an encryption $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_n) \in \mathbb{Z}_q^m$ and does the following. Define a function $\mathsf{Round}(x)$ where $x \in [-(q-1)/2, \ldots, (q-1)/2]$ as:
$$\mathsf{Round}(x) = \begin{cases} 0 & \text{if } |x| < q/4 \\ 1 & \text{otherwise} \end{cases}$$
The decryption algorithm outputs a vector $\boldsymbol{m} = (\mathsf{Round}(\gamma_1 - \psi_1'), \ldots, \mathsf{Round}(\gamma_n - \psi_n'))$.

The scheme is information-theoretically secure for a *one-time* use. As for correctness, we will show that for every two $\boldsymbol{\psi}$ and $\boldsymbol{\psi}'$ that are "close", $\mathsf{D}(\boldsymbol{\psi}', \mathsf{E}(\boldsymbol{\psi}, \boldsymbol{m})) = \boldsymbol{m}$. More precisely, call $\boldsymbol{\psi}$ and $\boldsymbol{\psi}'$ close if for every $i \in [n]$, $|\psi_i - \psi_i'| < q/4$.

$$\begin{aligned}\mathsf{D}(\boldsymbol{\psi}', \mathsf{E}(\boldsymbol{\psi}, \boldsymbol{m})) &= \left[\mathsf{Round}(\gamma_1 - \psi_1'), \ldots, \mathsf{Round}(\gamma_n - \psi_n')\right] \\ &= \left[\mathsf{Round}(m_1\lceil q/2\rceil + \psi_1 - \psi_1'), \ldots, \mathsf{Round}(m_n\lceil q/2\rceil + \psi_n - \psi_n')\right]\end{aligned}$$

Observe that if $|\psi_i - \psi_i'| < q/4$, then $\mathsf{Round}(m_i\lceil q/2\rceil + \psi_i - \psi_i') = m_i$. This completes the proof of correctness.

3.4.1 Analysis

Correctness. We define the sets $\Psi_{\mathbf{A},\mathbf{s},j}$ for $\mathsf{pk} := \mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \in \mathbb{Z}_q^n$ and $j \in [1 \dots d_{\max}]$ as follows:

$$\Psi_{\mathbf{A},\mathbf{s},j} = \left\{\mathbf{A}^T\mathbf{s} + \mathbf{e} : \ ||\mathbf{e}||_\infty \le B \cdot (2sm\sqrt{m})^j\right\}$$

Given this definition:

- Observe that when $\mathbf{e} \leftarrow \chi^m$, $||\mathbf{e}||_\infty \le B$ by the definition of χ and B. $\Pr[\mathsf{Encode}(\mathbf{A},\mathbf{s}) \in \Psi_{\mathbf{A},\mathbf{s},0}] = 1$.

- $\Psi_{\mathbf{A},\mathbf{s},0} \subseteq \Psi_{\mathbf{A},\mathbf{s},1} \subseteq \ldots \subseteq \Psi_{\mathbf{A},\mathbf{s},d_{\max}}$, by definition of the sets above.

- For any two encodings $\boldsymbol{\psi} = \mathbf{A}^T\mathbf{s} + \mathbf{e}, \boldsymbol{\psi}' = \mathbf{A}^T\mathbf{s} + \mathbf{e}' \in \Psi_{\mathbf{A},\mathbf{s},d_{\max}}$,

$$||\boldsymbol{\psi} - \boldsymbol{\psi}'||_\infty = ||\mathbf{e} - \mathbf{e}'||_\infty \le 2 \cdot B \cdot (2sm\sqrt{m})^{d_{\max}} < q/4,$$

  which holds as long as $n \cdot O(n^2 \log q)^{d_{\max}} < q/4$. Thus, $\boldsymbol{\psi}$ and $\boldsymbol{\psi}'$ are "close", and by the correctness property of the symmetric encryption scheme (E,D) described above, $\mathsf{D}(\boldsymbol{\psi}', \mathsf{E}(\boldsymbol{\psi}, \boldsymbol{m})) = \boldsymbol{m}$ for any $\boldsymbol{m} \in \{0,1\}^n$.

- Consider two encodings $\boldsymbol{\psi}_0 \in \Psi_{\mathbf{A}_0,\mathbf{s},j_0}$ and $\boldsymbol{\psi}_1 \in \Psi_{\mathbf{A}_1,\mathbf{s},j_1}$ for any $j_0, j_1 \in \mathbb{N}$, any $\mathbf{A}_0, \mathbf{A}_1 \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{s} \in \mathbb{Z}_q^n$. Then, $\boldsymbol{\psi}_0 = \mathbf{A}_0^T\mathbf{s} + \mathbf{e}_0$ and $\boldsymbol{\psi}_1 := \mathbf{A}_1^T\mathbf{s} + \mathbf{e}_1$ where $||\mathbf{e}_0||_\infty \le B \cdot (2sm\sqrt{m})^{j_0}$ and $||\mathbf{e}_1||_\infty \le B \cdot (2sm\sqrt{m})^{j_1}$.

  Then, the recoded ciphertext $\boldsymbol{\psi}_{\text{tgt}}$ is computed as follows:

$$\begin{aligned}
\boldsymbol{\psi}_{\text{tgt}}^T &:= \left[\boldsymbol{\psi}_0^T \ || \ \boldsymbol{\psi}_1^T\right] \mathbf{R}_{0,1}^{\text{tgt}} \\
&= \left[\mathbf{s}^T\mathbf{A}_0 + \mathbf{e}_0^T \ || \ \mathbf{s}^T\mathbf{A}_1 + \mathbf{e}_1^T\right] \mathbf{R}_{0,1}^{\text{tgt}} \\
&= \mathbf{s}^T \left[\mathbf{A}_0 \ || \ \mathbf{A}_1\right] \mathbf{R}_{0,1}^{\text{tgt}} + \left[\mathbf{e}_0^T \ || \ \mathbf{e}_1^T\right] \mathbf{R}_{0,1}^{\text{tgt}} \\
&= \mathbf{s}^T\mathbf{A}_{\text{tgt}} + \mathbf{e}_{\text{tgt}}
\end{aligned}$$

  where the last equation is because $\mathbf{A}_{\text{tgt}} = \left[\mathbf{A}_0 \ || \ \mathbf{A}_1\right] \mathbf{R}_{0,1}^{\text{tgt}}$ and we define $\mathbf{e}_{\text{tgt}} := \left[\mathbf{e}_0^T \ || \ \mathbf{e}_1^T\right] \mathbf{R}_{0,1}^{\text{tgt}}$. Thus,

$$\begin{aligned}
||\mathbf{e}_{\text{tgt}}||_\infty &\le m \cdot ||\mathbf{R}_{0,1}^{\text{tgt}}||_\infty \cdot (||\mathbf{e}_0||_\infty + ||\mathbf{e}_1||_\infty) \\
&\le m \cdot s\sqrt{m} \cdot (B \cdot (2sm\sqrt{m})^{j_0} + B \cdot (2sm\sqrt{m})^{j_1}) \\
&\le B \cdot (2sm\sqrt{m})^{\max(j_0,j_1)+1}
\end{aligned}$$

  exactly as required. Here, the second inequality is because $||\mathbf{R}_{0,1}^{\text{tgt}}||_\infty \le s\sqrt{m}$ by Lemma 2.4.1. This finishes our proof of correctness.

Key Indistinguishability. Recall that in $\mathsf{ReKeyGen}$, we are given samplings $(\mathbf{R}_0, \mathbf{R}_1)$ satisfying $\mathbf{A}_0\mathbf{R}_0 + \mathbf{A}_1\mathbf{R}_1 = \mathbf{A}_{\text{tgt}}$. In key indistinguishability we must argue that the distributions produced by sampling using a trapdoor for $\mathbf{A}_0$ or that for $\mathbf{A}_1$ are indistinguishable. Indeed, this follows directly from the following statement in [CHKP12, GPV08] (see also [CHKP12, Theorem 3.4]): for every $(\mathbf{A}_0, \mathbf{T}_0)$, $(\mathbf{A}_1, \mathbf{T}_1)$ generated by $\mathsf{TrapSamp}(1^n, 1^m, q)$, every matrix $\mathbf{A}_{\text{tgt}} \in \mathbb{Z}_q^{n \times m}$, and any $s = \Omega(\sqrt{n \log q})$, the following two experiments generate distributions with $\text{negl}(n)$ statistical distance:

- Sample $\mathbf{R}_0 \leftarrow (D_{\mathbb{Z}^m,s})^m$, compute $\mathbf{U} := \mathbf{A}_{\text{tgt}} - \mathbf{A}_0\mathbf{R}_0 \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{R}_1 \leftarrow \mathsf{SamPre}(\mathbf{A}_1, \mathbf{T}_1, \mathbf{U}, s)$. Output $(\mathbf{A}_0, \mathbf{T}_0, \mathbf{A}_1, \mathbf{T}_1, \mathbf{A}_{\text{tgt}}, \mathbf{R}_0, \mathbf{R}_1)$.

- Sample $\mathbf{R}_1 \leftarrow (D_{\mathbb{Z}^m,s})^m$, compute $\mathbf{U} := \mathbf{A}_{\text{tgt}} - \mathbf{A}_1\mathbf{R}_1 \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{R}_0 \leftarrow \mathsf{SamPre}(\mathbf{A}_0, \mathbf{T}_0, \mathbf{U}, s)$. Output $(\mathbf{A}_0, \mathbf{T}_0, \mathbf{A}_1, \mathbf{T}_1, \mathbf{A}_{\text{tgt}}, \mathbf{R}_0, \mathbf{R}_1)$.

Note that in both distributions, it clearly holds that $\mathbf{A}_0\mathbf{R}_0 + \mathbf{A}_1\mathbf{R}_1 = \mathbf{A}_tgt$, as required.

The recoding simulation property follows readily from lemma 2.4.1, as is done in [CHKP12]. In particular, given $[(\mathbf{A}_0, \mathbf{T}_0), (\mathbf{A}_1, \mathbf{T}_1), \mathbf{A}_{\text{tgt}}, (\mathbf{R}_0, \mathbf{R}_1)]$, from lemma 2.4.1, it follows that if we are given $\mathbf{A}_{\text{tgt}}$, then we may sample $(\mathbf{R}_0, \mathbf{R}_1)$ using the trapdoor $\mathbf{T}_0$. Alternatively, sampling $(\mathbf{R}_0, \mathbf{R}_1)$ first and setting $\mathbf{A}_{\text{tgt}} = \mathbf{A}_0\mathbf{R}_0 + \mathbf{A}_1\mathbf{R}_1$ produces the same distribution.

Correlated pseudorandomness directly from the decisional LWE assumption $\mathsf{dLWE}_{n,(\ell+1)\cdot m,q,\chi}$ where $q = n^{\Theta(d_{\max})}$. In particular, by the dLWE, given $\mathbf{A}_1, \ldots, \mathbf{A}_{\ell+1}, \psi_1, \ldots, \psi_\ell, \psi^*$, no adversary can distinguish between the cases when ψ^* is a valid LWE sample under key $\mathbf{A}_{\ell+1}$ or a randomly chosen value from $\mathbb{Z}_q^m$.

3.5 Attribute-Based Encryption for Circuits

In this section, we show how to construct attribute-based encryption for circuits from any TOR scheme.[2] Let TOR be the scheme consisting of algorithms ($\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encode}$) with the "two-to-one" recoding mechanism ($\mathsf{Recode}, \mathsf{ReKeyGen}, \mathsf{SimReKeyGen}$) with input space $\mathcal{S}$. For every $d_{\max}$, let $d_{\max}$-TOR denote a secure "two-to-one" recoding scheme that is correct for $d_{\max}$ recoding levels.

Theorem 3.5.1. *For every ℓ and polynomial $d_{\max} = d_{\max}(\lambda)$, let $\mathcal{C}_{\ell,d_{\max}}$ denote a family of polynomial-size circuits of depth at most $d_{\max}$ that take ℓ bits of input. Assuming the existence of a $d_{\max}$-TOR scheme, there exists a selectively secure attribute-based encryption scheme $\mathcal{ABE}$ for $\mathcal{C}_{\ell,d_{\max}}$.*

Combining Theorem 3.5.1 and Lemma 3.4.1, we obtain a selectively secure attribute-based encryption scheme from LWE. Furthermore, invoking an argument from [BB04, Theorem 7.1] and using subexponential hardness of LWE, we obtain a fully secure scheme:

Corollary 3.5.2. *For all ℓ and polynomial $d_{\max} = d_{\max}(\ell)$, there exists a selectively secure attribute-based encryption scheme $\mathcal{ABE}$ for any family of polynomial-size circuits with ℓ*

[2]We point out that our construction and the proof of security generalizes the sample IBE construction presented in Section 3.3.2.

inputs and depth at most $d_{\max}$, assuming the hardness of $\mathsf{dLWE}_{n,q,\chi}$ for sufficiently large $n = \mathrm{poly}(\lambda, d_{\max})$, $q = n^{O(d_{\max})}$ and some $\mathrm{poly}(n)$-bounded error distribution χ.

Moreover, assuming $2^{O(\ell)}$-hardness of $\mathsf{dLWE}_{n,q,\chi}$ for parameters $n = \mathrm{poly}(\lambda, d_{\max}, \ell)$, and q and χ as above, the attribute-based encryption scheme $\mathcal{ABE}$ is fully secure.

The reader is referred to the text after the construction for further explanation of how to choose the LWE parameters. It remains an intriguing open problem to construct fully secure without complexity leveraging (whereas a generic transformation from selective to fully secure functional encryption (more powerful primitive than ABE) is already known [ABSV14]).

Observe that if we start with a TOR scheme that supports $d_{\max} = \ell^{\omega(1)}$, then our construction immediately yields an attribute-based encryption scheme for arbitrary polynomial-size circuit families (without any restriction on the depth). This can be achieved if, for example, we had an LWE-based TOR scheme where q grows polynomially instead of exponentially in $d_{\max}$ as in our LWE-based weak TOR.

We now prove Theorem 3.5.1.

Circuit Representation. Let $\mathcal{C}_\lambda$ be a collection of circuits each having $\ell = \ell(\lambda)$ input wires and one output wire. Define a collection $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$. For each $C \in \mathcal{C}_\lambda$, we index the wires of C in the following way. The input wires are indexed 1 to ℓ, the internal wires have indices $\ell + 1, \ell + 2, \ldots, |C| - 1$ and the output wire has index $|C|$, which also denotes the size of the circuit. We assume that the circuit is composed of arbitrary two-to-one gates. Each gate g is indexed as a tuple (u, v, w) where u and v are the incoming wire indices, and $w > \max\{u, v\}$ is the outgoing wire index. The gate computes the function $g_w : \{0,1\} \times \{0,1\} \to \{0,1\}$. The "fan-out wires" in the circuit are given a single number. That is, if the outgoing wire of a gate feeds into the input of multiple gates, then all these wires are indexed the same. (See e.g. [BHR12, Fig 4].)

3.5.1 Construction from TOR

The ABE scheme $\mathcal{ABE} = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec})$ is defined as follows.

$\mathsf{Setup}(1^\lambda, 1^\ell, d_{\max})$: For each of the ℓ input wires, generate two public/secret key pairs. Also, generate an additional public/secret key pair:

$$\begin{aligned} (\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}) \qquad \text{for } i \in [\ell], b \in \{0,1\} \\ (\mathsf{pk}_{\mathrm{out}}, \mathsf{sk}_{\mathrm{out}}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}) \end{aligned}$$

Output

$$\mathsf{mpk} := \begin{pmatrix} \mathsf{pk}_{1,0} & \mathsf{pk}_{2,0} & \ldots & \mathsf{pk}_{\ell,0} & \\ \mathsf{pk}_{1,1} & \mathsf{pk}_{2,1} & \ldots & \mathsf{pk}_{\ell,1} & \mathsf{pk}_{\mathrm{out}} \end{pmatrix} \qquad \mathsf{msk} := \begin{pmatrix} \mathsf{sk}_{1,0} & \mathsf{sk}_{2,0} & \ldots & \mathsf{sk}_{\ell,0} \\ \mathsf{sk}_{1,1} & \mathsf{sk}_{2,1} & \ldots & \mathsf{sk}_{\ell,1} \end{pmatrix}$$

$\mathsf{Enc}(\mathsf{mpk}, a, m)$: For $a \in \{0,1\}^\ell$, choose a uniformly random $s \xleftarrow{\$} \mathcal{S}$ and encode it under the public keys specified by the index bits:

$$\psi_i \leftarrow \mathsf{Encode}(\mathsf{pk}_{i,a_i}, s) \text{ for all } i \in [\ell]$$

Encrypt the message m:

$$\tau \leftarrow \mathsf{E}(\mathsf{Encode}(\mathsf{pk}_{\mathrm{out}}, s), m)$$

Output the ciphertext

$$\mathsf{ct}_a := \big(\ \psi_1,\ \ \psi_2,\ \ \ldots,\ \ \psi_\ell,\ \ \tau,\ \ a\ \big)$$

$\mathsf{KeyGen}(\mathsf{msk}, C)$:

1. For every non-input wire $w = \ell+1, \ldots, |C|$ of the circuit C, and every $b \in \{0,1\}$, generate public/secret key pairs:

$$(\mathsf{pk}_{w,b}, \mathsf{sk}_{w,b}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}) \text{ if } w < |C| \text{ or } b = 0$$

   and set $\mathsf{pk}_{|C|,1} := \mathsf{pk}_{\mathrm{out}}$.

2. For the gate $g = (u, v, w)$ with outgoing wire w, compute the four recoding keys $\mathsf{rk}^w_{b,c}$ (for $b, c \in \{0,1\}$):

$$\mathsf{rk}^w_{b,c} \leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{u,b}, \mathsf{pk}_{v,c}, \mathsf{sk}_{u,b}, \mathsf{pk}_{w,g_w(b,c)}\Big)$$

Output the secret key which is a collection of $4(|C| - \ell)$ recoding keys

$$\mathsf{sk}_C := \Big(\ \mathsf{rk}^w_{b,c}\ :\ w \in \big[\,\ell+1, |C|\,\big],\ b, c \in \{0,1\}\ \Big)$$

$\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_a)$: For $w = \ell+1, \ldots, |C|$, let $g = (u, v, w)$ denote the gate with outgoing wire w. Suppose wires u and v carry the values b^* and c^* when C evaluated on a, so that wire w carries the value $d^* := g_w(b^*, c^*)$. Compute

$$\psi_{w,d^*} \leftarrow \mathsf{Recode}\Big(\mathsf{rk}^w_{b^*,c^*}, \psi_{u,b^*}, \psi_{v,c^*}\Big)$$

If $C(a) = 1$, then we would have computed $\psi_{|C|,1}$. Output the message

$$m \leftarrow \mathsf{D}\big(\ \psi_{|C|,1}, \tau\ \big)$$

If $C(a) = 0$, output $\perp$.

LWE Parameters. Fix $\ell = \ell(\lambda)$ and $d_{\max} = d_{\max}(\ell)$, and suppose the $\mathsf{dLWE}_{n,m,q,\chi}$ assumption holds for $q = 2^{n^\epsilon}$ for some $0 < \epsilon < 1$. Then, in our LWE-based TOR, we will set:

$$n = \tilde{\Theta}(d_{\max}^{1/\epsilon}) \quad \text{and} \quad q = n^{\Theta(d_{\max})}$$

By Corollary 3.5.2, we get security under 2^{n^ϵ}-LWE.

3.5.2 Correctness

Lemma 3.5.3 (correctness). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be family where each $\mathcal{C}_\lambda$ is a finite collection of polynomial-size circuits each of depth at most* $d_{\max}$. *Let* TOR *be a correct "two-to-one" recoding scheme for* $d_{\max}$ *levels. Then, the construction presented above is a correct attribute-based encryption scheme.*

Proof. Fix a circuit C of depth at most $d_{\max}$ and an input a such that $C(a) = 1$. Informally, we rely on recoding correctness for $d_{\max}$ recodings to show that $w = 1, \ldots, |C|$, we have

$$\psi_{w,d^*} = \mathsf{Encode}(\mathsf{pk}_{w,d^*}, s),$$

where d^* is the value carried by the wire w and ψ_{w,d^*} is computed as in Dec. Formally, we proceed via induction on w to show that

$$\psi_{w,d^*} \in \Psi_{\mathsf{pk}_{w,d^*},s,j}.$$

where j is the depth of wire w. The base case $w = 1, \ldots, \ell$ follows immediately from correctness of Encode. For the inductive step, consider a wire w at depth j for some gate $g = (u, v, w)$ where $u, v < w$. By the induction hypothesis,

$$\psi_{u,b^*} \in \Psi_{\mathsf{pk}_{u,b^*},s,j_0}, \quad \psi_{u,c^*} \in \Psi_{\mathsf{pk}_{v,c^*},s,j_1}$$

where $j_0, j_1 < j$ denote the depths of wires u and v respectively. It follows immediately from the correctness of Recode that

$$\psi_{w,d^*} \in \Psi_{\mathsf{pk}_{w,d^*},s,\max(i_0,i_1)+1} \subseteq \Psi_{\mathsf{pk}_{w,d^*},s,j}$$

which completes the inductive proof. Since $C(a) = 1$ and $\mathsf{pk}_{|C|,1} = \mathsf{pk}_{\text{out}}$, we have $\psi_{|C|,1} \in \Psi_{\mathsf{pk}_{\text{out}},s,d_{\max}}$. Finally, by the correctness of (E, D), $\mathsf{D}(\psi_{|C|,1}, \tau) = m$. □

3.5.3 Security

Lemma 3.5.4 (selective security). *For any adversary $\mathcal{A}$ against selective security of the attribute-based encryption scheme, there exist an adversary $\mathcal{B}$ against correlated pseudorandomness of TOR whose running time is essentially the same as that of $\mathcal{A}$, such that*

$$\mathsf{Adv}^{\text{PE}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{CP}}_{\mathcal{B}}(\lambda) + \text{negl}(\lambda)$$

where $\text{negl}(\lambda)$ *captures the statistical security terms in TOR.*

We begin by describing alternative algorithms, which would be useful later for constructing the adversary $\mathcal{B}$ for the correlated pseudorandomness security game.

Alternative algorithms. Fix the selective challenge a. We get from the "outside" the challenge $\mathsf{pp}, (\mathsf{pk}_i, \psi_i)_{i\in[\ell+1]}$ for correlated pseudorandomness, The main challenge is to design an alternative algorithm KeyGen^* for answering secret key queries without knowing $\mathsf{sk}_{1,a_1}, \dots, \mathsf{sk}_{\ell,a_\ell}$ or $\mathsf{sk}_{\mathrm{out}}$. The algorithm KeyGen^* will maintain the following invariant: on input C with $C(a) = 0$,

- for every non-output wire $w = 1, \dots, |C|-1$ carrying the value b^*, we will know $\mathsf{sk}_{w,1-b^*}$ but not sk_{w,b^*}.

Moreover, we do not know $\mathsf{sk}_{|C|,0}$ or $\mathsf{sk}_{|C|,1} = \mathsf{sk}_{\mathrm{out}}$.

$\mathsf{Setup}^*(a, 1^\lambda, 1^\ell, d_{\max})$: Let

$$\begin{aligned} (\mathsf{pk}_{i,1-a_i}, \mathsf{sk}_{i,1-a_i}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}) \text{ for } i \in [\ell] \\ \mathsf{pk}_{\mathrm{out}} &:= \mathsf{pk}_{\ell+1} \\ \mathsf{pk}_{i,a_i} &:= \mathsf{pk}_i \text{ for } i \in [\ell] \end{aligned}$$

$$\text{Output } \mathsf{mpk} = \begin{pmatrix} \mathsf{pk}_{1,0} & \mathsf{pk}_{2,0} & \dots & \mathsf{pk}_{\ell,0} & \\ \mathsf{pk}_{1,1} & \mathsf{pk}_{2,1} & \dots & \mathsf{pk}_{\ell,1} & \mathsf{pk}_{\mathrm{out}} \end{pmatrix}$$

$\mathsf{Enc}^*(\mathsf{mpk}, a, m)$: Set $\tau \leftarrow \mathsf{E}(\psi_{\ell+1}, m)$ and output the ciphertext

$$\mathsf{ct}_a = (\ \psi_1, \ \psi_2, \ \dots, \ \psi_\ell, \ \tau, \ a\ )$$

where $\psi_1, \dots, \psi_{\ell+1}$ are provided in the challenge.

$\mathsf{KeyGen}^*(a, \mathsf{msk}, C)$: where $C(a) = 0$,

1. For each internal wire $w \in [\ell+1, |C|-1]$ of the circuit C carrying the value b^* when evaluating C on input a, generate public/secret key pairs:

$$(\mathsf{pk}_{w,1-b^*}, \mathsf{sk}_{w,1-b^*}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$$

   We will generate pk_{w,b^*} using $\mathsf{SimReKeyGen}$ as described next.

2. For $w = \ell+1, \dots, |C|$, let $g = (u, v, w)$ denote the gate for which w is the outgoing wire. Suppose wires u and v carry the values b^* and c^* when C evaluated on a, so that wire w carries the value $d^* := g_w(b^*, c^*)$. By the invariant above, we know $\mathsf{sk}_{u,1-b^*}$ and $\mathsf{sk}_{v,1-c^*}$ but not sk_{u,b^*} and sk_{v,c^*}. We start by generating

$$(\mathsf{pk}_{w,d^*}, \mathsf{rk}^w_{b^*,c^*}) \leftarrow \mathsf{SimReKeyGen}(\mathsf{pk}_{u,b^*}, \mathsf{pk}_{v,c^*})$$

We generate the other three recoding keys using $\mathsf{ReKeyGen}$ as follows:

$$\begin{aligned}
\mathsf{rk}^w_{1-b^*,c^*} &\leftarrow \mathsf{ReKeyGen}\big(\mathsf{pk}_{u,1-b^*}, \mathsf{pk}_{v,c^*}, \quad \mathsf{sk}_{u,1-b^*}, \mathsf{pk}_{w,g_w(1-b^*,c^*)}\big)\\
\mathsf{rk}^w_{b^*,1-c^*} &\leftarrow \mathsf{ReKeyGen}\big(\mathsf{pk}_{v,1-c^*}, \mathsf{pk}_{u,b^*}, \quad \mathsf{sk}_{v,1-c^*}, \mathsf{pk}_{w,g_w(b^*,1-c^*)}\big)\\
\mathsf{rk}^w_{1-b^*,1-c^*} &\leftarrow \mathsf{ReKeyGen}\big(\mathsf{pk}_{u,1-b^*}, \mathsf{pk}_{v,1-c^*}, \mathsf{sk}_{u,1-b^*}, \mathsf{pk}_{w,g_w(1-b^*,1-c^*)}\big)
\end{aligned}$$

Note that $\mathsf{rk}^w_{1-b^*,c*}, \mathsf{rk}^w_{1-b^*,1-c*}$ are generated the same way in both KeyGen and KeyGen^* using $\mathsf{sk}_{u,1-b^*}$.

Output the secret key

$$\mathsf{sk}_C := \Big(\mathsf{rk}^w_{b,c} \ : \ w \in \big[\,\ell+1, |C|\,\big], \, b,c \in \{0,1\} \Big)$$

Informally, the recoding key $\mathsf{rk}^w_{b^*,1-c^*}$ looks the same as in KeyGen because of key indistinguishability, and $\mathsf{rk}^w_{b^*,c^*}$ (together with the simulated pk_{w,d^*}) looks the same as in KeyGen because of the recoding simulation property.

Game sequence. Next, consider the following sequence of games. We use $\mathsf{Adv}_0, \mathsf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc. Game 0 is the real experiment. We then show that the games are either statistically or computationally indistinguishable.

Game i for $i = 1, 2, \ldots, q$ As in Game 0, except the challenger answers the first i key queries using KeyGen^* and the remaining $q - i$ key queries using KeyGen. For the i'th key query C_i, we consider sub-Games $i.w$ as follows:

Game $i.w$, for $w = \ell + 1, \ldots, |C_i|$ The challenger switches $(\mathsf{rk}^w_{b,c} : b, c \in \{0,1\})$ from KeyGen to KeyGen^*. More precisely:

- We switch $(\mathsf{pk}_{w,d^*}, \mathsf{rk}^w_{b^*,c*})$ from KeyGen to KeyGen^*.
- We switch $\mathsf{rk}^w_{b^*,1-c*}$ from KeyGen to KeyGen^*.
- The other two keys $\mathsf{rk}^w_{1-b^*,c*}, \mathsf{rk}^w_{1-b^*,1-c*}$ are generated the same way in both KeyGen and KeyGen^*.

From lemma 3.5.5, we have

$$|\mathsf{Adv}_i - \mathsf{Adv}_{i+1}| \leq \text{negl}(\lambda) \text{ for all } i$$

Note that in Game q, the challenger runs Setup^* and answers all key queries using KeyGen^* with the selective challenge a and generates the challenge ciphertext using Enc.

Game $q + 1$ Same as Game q, except the challenger generates the challenge ciphertext using Enc^* with $\psi_{\ell+1} = \mathsf{Encode}(\mathsf{pk}_{\ell+1}, s)$. Clearly,

$$\mathsf{Adv}_{q+1} = \mathsf{Adv}_q$$

Game $q+2$ Same as Game $q+1$, except $\psi_{\ell+1} \xleftarrow{\$} \mathcal{K}$. From lemma 3.5.6, there is an adversary $\mathcal{B}$ such that

$$|\mathsf{Adv}_{q+1} - \mathsf{Adv}_{q+2}| \leq \mathsf{Adv}_{\mathcal{B}}^{\text{CP}}(\lambda)$$

Finally, by the one-time semantic security of (E, D) for all $m_0, m_1 \in \mathcal{M}$, $\mathsf{E}(\psi_{\ell+1}, m_0)$ is indistinguishable from $\mathsf{E}(\psi_{\ell+1}, m_1)$ (for $\psi_{\ell+1} \xleftarrow{\$} \mathcal{K}$). Therefore, it follows that $\mathsf{Adv}_{q+2} \leq \text{negl}(\lambda)$, concluding the proof of security.

Lemma 3.5.5. *Games i and $i+1$ are (statistically) indistinguishable for all $i = 0, \ldots, q-1$.*

Proof. In Game $i+1$, first $i+1$ key queries are answered using KeyGen^* and the remaining $q-(i+1)$ key queries using KeyGen. Consider a key query C_{i+1} and for all $w = \ell, \ldots, |C_i|$, sub-Games $(i+1).w$ defined as above. We argue that for all $w = \ell, \ldots, |C_i| - 1$, sub-Games $(i+1).w$ and $(i+1).(w+1)$ are statistically indistinguishable. Clearly, sub-Game $(i+1).\ell$ corresponds to Game i, where $i+1$'th key query is sampled using KeyGen and sub-Game $(i+1).|C_i|$ this corresponds to Game $i+1$, where $i+1$'th query sampled using KeyGen^* (the remaining queries are sampled identically). Now, the sub-Games $(i+1).w$ and $(i+1).(w+1)$ differ in how recoding keys $(\mathsf{rk}_{b,c}^{w+1} : b, c \in \{0,1\})$ are sampled for query C_{i+1}. The statistical indistinguishability of keys $\mathsf{rk}_{b^*,c^*}^{w+1}$ follows from recoding simulation, $\mathsf{rk}_{b^*,1-c^*}^{w+1}$ on indistinguishability w.r.t. sampling using keys sk_{b^*} and sk_{1-c^*}, and other two keys $\mathsf{rk}_{1-b^*,c^*}^{w+1}, \mathsf{rk}_{1-b^*,1-c^*}^{w+1}$ are generated identically in two experiments. It follows that the two sub-Games $(i+1).w$ and $(i+1).(w+1)$ are statistically indistinguishable, concluding the lemma. □

Lemma 3.5.6. *Games $q+1$ and $q+2$ are computationally indistinguishable.*

Proof. Suppose there exists an adversary $\mathcal{B}^*$ that distinguishes between the Games $q+1$ and $q+2$. We construct and adversary $\mathcal{B}$ that breaks correlated pseudorandomness property. The adversary $\mathcal{B}$ gets as input $(\mathsf{pk}_i, \psi_i)_{i\in[\ell]}, \mathsf{pk}_{\ell+1}, \psi^*$, where ψ^* is either a valid encoding using key $pk_{\ell+1}$ or a randomly chosen element. The adversary $\mathcal{B}$ uses these values to invoke algorithms $\mathsf{Setup}^*, \mathsf{Enc}^*, \mathsf{KeyGen}^*$, where it uses $\psi_{\ell+1} := \psi^*$. It invokes the adversary $\mathcal{B}^*$ and the same thing. Clearly, if $\psi^* = \mathsf{Encode}(\mathsf{pk}_{\ell+1}, s)$, then this experiment corresponds to Game $q+1$. On the other hand, if $\psi^* \xleftarrow{\$} \mathcal{K}$ then this experiment corresponds to Game $q+2$. This concludes the proof of the lemma. □

3.6 Attribute-Based Encryption for Branching Programs

In this section, we present weak TOR and attribute-based encryption for branching programs, which capture the complexity class log-space. As noted in Section 3.1.1, we exploit the fact that in branching programs, the transition function depends on an input variable and the current state; this means that one of the two input encodings during recoding is always a "depth 0" encoding.

Branching programs. Recall that a branching program Γ is a directed acyclic graph in which every nonterminal node has exactly two outgoing edges labeled $(i, 0)$ and $(i, 1)$ for some $i \in [\ell]$. Moreover, there is a distinguished terminal accept node. Every input $x \in \{0,1\}^\ell$ naturally induces a subgraph Γ_x containing exactly those edges labeled (i, x_i). We say that Γ accepts x iff there is a path from the start node to the accept node in Γ_x. At the cost of possibly doubling the number of edges and vertices, we may assume that there is at most one edge connecting any two nodes in Γ.

3.6.1 Weak TOR

A weak "two-to-one" encoding (wTOR) scheme consists of the same algorithms as TOR, except that $\mathsf{KeyGen}(\mathsf{pp}, j)$ takes an additional input $j \in \{0, 1\}$. That is, KeyGen may produce different distributions of public/secret key pairs depending on j. Moreover, in $\mathsf{ReKeyGen}$, the first public key is always generated using $\mathsf{KeyGen}(\mathsf{pp}, 0)$ and the second using $\mathsf{KeyGen}(\mathsf{pp}, 1)$; similarly, in Recode, the first encoding is always generated with respect to a public key from $\mathsf{KeyGen}(\mathsf{pp}, 0)$ and the second from $\mathsf{KeyGen}(\mathsf{pp}, 1)$. Similarly, the correctness and statistical security properties are relaxed.

Correctness. First, for every pk and $s \in \mathcal{S}$, there exists a family of sets $\Psi_{\mathsf{pk},s,j}, j = 0, 1, \ldots, d_{\max}$:

- $\Psi_{\mathsf{pk},s,1} \subseteq \cdots \subseteq \Psi_{\mathsf{pk},s,d_{\max}}$.
- for all $\psi, \psi' \in \Psi_{\mathsf{pk},s,d_{\max}}$ and all $m \in \mathcal{M}$,

$$\mathsf{D}(\psi', \mathsf{E}(\psi, m)) = m$$

Secondly, the correctness of recoding requires that for any triple of key pairs $(\mathsf{pk}_0, \mathsf{sk}_0), (\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_{\mathrm{tgt}})$ respectively in the support of $\mathsf{KeyGen}(\mathsf{pp}, 0), \mathsf{KeyGen}(\mathsf{pp}, 1), \mathsf{KeyGen}(\mathsf{pp}, 1)$ and any encodings $\psi_0 \in \mathsf{Encode}(\mathsf{pk}_0, s)$ and $\psi_1 \in \Psi_{\mathsf{pk}_1,s,j_1}$ where $0 < j_1$,

$$\mathsf{Recode}(\mathsf{rk}, \psi_0, \psi_1) \in \Psi_{\mathsf{pk}_{\mathrm{tgt}},s,j_1+1}$$

Statistical Security Properties. We require recoding simulation as before, but not key indistinguishability. However, we require the following additional property:

Back-tracking : For all $(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 0)$ and all $(\mathsf{pk}_1, \mathsf{sk}_1), (\mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_{\mathrm{tgt}}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1)$, the following distributions are identical:

$$\mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, \mathsf{pk}_{\mathrm{tgt}}) \equiv -\mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_0, \mathsf{pk}_1)$$

Informally, this says that switching the order of pk_1 and $\mathsf{pk}_{\mathrm{tgt}}$ as inputs to $\mathsf{ReKeyGen}$ is the same as switching the "sign" of the output. In our instantiations, the output of $\mathsf{ReKeyGen}$ lies in a group, so negating the output simply refers to applying the group inverse operation.

Computational Security Property. We define the advantage function $\mathsf{Adv}^{\text{CP}}_{\mathcal{A}}(\lambda)$ (modified to account for the additional input to KeyGen) to be the absolute value of:

$$\Pr\left[b = b' : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda); s \leftarrow \mathcal{S}; \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 0), \\ \psi_i \leftarrow \mathsf{Encode}(\mathsf{pk}_i, s), i = 1, \ldots, \ell; \\ (\mathsf{pk}_{\ell+1}, \mathsf{sk}_{\ell+1}) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1); \\ \psi'_0 \leftarrow \mathsf{Encode}(\mathsf{pk}_{\ell+1}, s); \\ b \xleftarrow{\$} \{0,1\}; \psi'_1 \xleftarrow{\$} \mathcal{K} \\ b' \leftarrow \mathcal{A}(\mathsf{pk}_1, \ldots, \mathsf{pk}_{\ell+1}, \psi_1, \ldots, \psi_\ell, \psi'_b) \end{array} \right] - \frac{1}{2}$$

and we require that for all PPT $\mathcal{A}$, the advantage function $\mathsf{Adv}^{\text{CP}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.

Remark 3.6.1. *Due to the additional back-tracking property, it is not the case that a TOR implies a weak TOR. However, we are able to instantiate weak TOR under weaker and larger classes of assumptions than TOR.*

3.6.2 Weak TOR from LWE

We provide an instantiation of weak TOR from LWE. The main advantage over our construction of TOR in Section 3.4 is that the dependency of q on $d_{\max}$ is linear in $d_{\max}$ instead of exponential. Therefore, if q is quasi-polynomial, we can handle any polynomial $d_{\max}$, as opposed to an a-prior bounded $d_{\max}$.

Lemma 3.6.1. *Assuming $\mathsf{dLWE}_{n,(\ell+2)m,q,\chi}$ where $q = O(d_{\max} n^3 \log n)$, there is a weak TOR scheme that is correct up to $d_{\max}$ levels, where $n = n(\lambda)$, the error distribution $\chi = \chi(n) = D_{\mathbb{Z},\sqrt{n}}$, the modulus $q = q(n) = d_{\max} \cdot O(n^3 \log n)$ and the number of samples $m = m(n) = O(n \log q)$.*

Note that the parameters here are better than in Lemma 3.4.1. The construction of weak TOR from learning with errors follows:

- $\mathsf{Setup}(1^\lambda, d_{\max})$: We are given parameters n, m, χ, q and we set the error bound $B = B(n) = O(n)$ and the Gaussian parameter $s = s(n) = O(\sqrt{n \log q})$. Output the global public parameters $\mathsf{pp} = (n, \chi, B, q, m, s)$. Define the domain $\mathcal{S}$ of the encoding scheme to be $\mathbb{Z}_q^n$.
- $\mathsf{KeyGen}(\mathsf{pp}, j)$: Run the trapdoor generation algorithm $\mathsf{TrapSamp}(1^n, 1^m, q)$ to obtain a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ together with the trapdoor $\mathbf{T}$. Output

$$\mathsf{pk} = \mathbf{A}; \quad \mathsf{sk} = \mathbf{T}.$$

- $\mathsf{Encode}(\mathbf{A}, \mathbf{s})$: Sample an error vector $\mathbf{e} \leftarrow \chi^m$ and output the encoding $\boldsymbol{\psi} := \mathbf{A}^T\mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^m$.

- $\mathsf{ReKeyGen}(\mathbf{A}_0, \mathbf{A}_1, \mathbf{A}_{\text{tgt}}, \mathbf{T})$: Outputs a low-norm matrix $\mathbf{R}$ such that $\mathbf{A}_0\mathbf{R} = \mathbf{A}_{\text{tgt}} - \mathbf{A}_1$. In particular,
$$\mathbf{R} \leftarrow \mathsf{SamPre}(\mathbf{A}_0, \mathbf{T}_0, \mathbf{A}_{\text{tgt}} - \mathbf{A}_1, s)$$

- $\mathsf{SimReKeyGen}(\mathbf{A}_0, \mathbf{A}_1)$: Sample a matrix $\mathbf{R} \leftarrow (D_{\mathbb{Z},s})^{m\times m}$ by sampling each entry from the discrete Gaussian distribution $D_{\mathbb{Z},s}$. Output
$$\mathsf{rk} := \mathbf{R}; \quad \mathbf{A}_{\text{tgt}} := \mathbf{A}_0\mathbf{R} + \mathbf{A}_1$$

- $\mathsf{Recode}(\mathsf{rk}, \boldsymbol{\psi}_0, \boldsymbol{\psi}_1)$: Outputs $\mathsf{rk}^T\boldsymbol{\psi}_0 + \boldsymbol{\psi}_1$.

Correctness. We define the sets $\Psi_{\mathbf{A},\mathbf{s},j}$ for $\mathsf{pk} := \mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \in \mathbb{Z}_q^n$ and $j \in [1 \ldots d_{\max}]$ as follows:
$$\Psi_{\mathbf{A},\mathbf{s},j} = \left\{\mathbf{A}^T\mathbf{s} + \mathbf{e} : \ ||\mathbf{e}||_\infty \le B \cdot j \cdot (sm\sqrt{m})\right\}$$
The analysis is similar to that in the previous section. In particular, we observe right away that

- $\Psi_{\mathbf{A},\mathbf{s},1} \subseteq \Psi_{\mathbf{A},\mathbf{s},1} \subseteq \ldots \subseteq \Psi_{\mathbf{A},\mathbf{s},d_{\max}}$.

- For any two encodings $\boldsymbol{\psi}, \boldsymbol{\psi}' \in \Psi_{\mathbf{A},\mathbf{s},d_{\max}}$ and $\boldsymbol{m} \in \{0,1\}^n$, $\mathsf{D}(\boldsymbol{\psi}', \mathsf{E}(\boldsymbol{\psi}, \boldsymbol{m})) = \boldsymbol{m}$, as long as
$$B \cdot d_{\max} \cdot (sm\sqrt{m}) \le q/4.$$

- Consider two encodings $\mathbf{A}^T\mathbf{s} + \mathbf{e} \in \mathsf{Encode}(\mathbf{A}, \mathbf{s})$ and $\boldsymbol{\psi}_1 \in \Psi_{\mathbf{A}_1,\mathbf{s},j_1}$ for any $j_1 \in \mathbb{N}$. Then, $\boldsymbol{\psi}_0 = \mathbf{A}_0^T\mathbf{s} + \mathbf{e}_0$ and $\boldsymbol{\psi}_1 := \mathbf{A}_1^T\mathbf{s} + \mathbf{e}_1$ where $||\mathbf{e}_0||_\infty \le B$ and $||\mathbf{e}_1||_\infty \le j_1 \cdot B \cdot (sm\sqrt{m})$.

  Then, the recoded ciphertext $\boldsymbol{\psi}_{\text{tgt}}$ is computed as follows:
$$\begin{aligned} \boldsymbol{\psi}_{\text{tgt}} &:= \mathbf{R}^T\boldsymbol{\psi}_0 + \boldsymbol{\psi}_1 \\ &= \mathbf{R}^T(\mathbf{A}_0^T\mathbf{s} + \mathbf{e}_0) + (\mathbf{A}_1^T\mathbf{s} + \mathbf{e}_1) \\ &= \mathbf{A}_{\text{tgt}}^T\mathbf{s} + \mathbf{e}_{\text{tgt}} \end{aligned}$$
  where the last equation is because $\mathbf{A}_{\text{tgt}} = \mathbf{A}_0\mathbf{R} + \mathbf{A}_1$ and we define $\mathbf{e}_{\text{tgt}} := \mathbf{R}^T\mathbf{e}_0 + \mathbf{e}_1$. Thus,
$$\begin{aligned} ||\mathbf{e}_{\text{tgt}}||_\infty &\le m \cdot ||\mathbf{R}||_\infty ||\mathbf{e}_0||_\infty + ||\mathbf{e}_1||_\infty \\ &\le m \cdot s\sqrt{m} \cdot B + B \cdot j_1 \cdot (sm\sqrt{m}) \\ &= (j_1 + 1) \cdot B \cdot (sm\sqrt{m}) \end{aligned}$$
  exactly as required. Here, the second inequality is because $||\mathbf{R}||_\infty \le s\sqrt{m}$ by Lemma 2.4.1. This finishes our proof of correctness.

Security. Correlated pseudorandomness follows from $\mathsf{dLWE}_{n,(\ell+2)m,q,\chi}$ where $q = n \cdot d_{\max}$. In particular, by the dLWE, given $\mathbf{A}_1, \dots, \mathbf{A}_{\ell+1}, \psi_1, \dots, \psi_\ell, \psi^*$, no adversary can distinguish between the cases when ψ^* is a valid LWE sample under key $\mathbf{A}_{\ell+1}$ or a randomly chosen value from $\mathbb{Z}_q^m$. Recoding simulation follows readily from Lemma 2.4.1 by an argument identical to the one for the construction of TOR in Section 3.4. For back-tracking, negation of a recoding matrix $\mathbf{R}$ is simply the additive inverse over $\mathbb{Z}_q^m$. That is, the output of $\mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_1, \mathsf{sk}_0, \mathsf{pk}_{\mathrm{tgt}})$ is a matrix $\mathbf{R}$ such that $\mathbf{A}_0\mathbf{R} = \mathbf{A}_{\mathrm{tgt}} - \mathbf{A}_1$ and the output of $\mathsf{ReKeyGen}(\mathsf{pk}_0, \mathsf{pk}_{\mathrm{tgt}}, \mathsf{sk}_0, \mathsf{pk}_1)$, is also a matrix $\mathbf{R}'$ such that $\mathbf{A}_0\mathbf{R}' = \mathbf{A}_1 - \mathbf{A}_{\mathrm{tgt}}$. It is easy to see that the output $\mathbf{R}$ is distributed identically to $-\mathbf{R}'$.

3.6.3 Weak TOR from Bilinear Maps

We use asymmetric groups for maximal generality and for conceptual clarity. Let $G_1, G_2, G_T \leftarrow \mathsf{GroupGen}(1^\lambda)$ be descriptions cyclic groups (with corresponding generators) of prime order q and $e : G_1 \times G_2 \to G_T$ is a non-degenerate bilinear map. We require that the group operations in G and G_T as well the bilinear map e are computable in deterministic polynomial time with respect to λ. Let g_1, g_2 denote random generators of G_1, G_2 respectively. The DBDH assumption says that the following two distributions are computationally indistinguishable:

$$\left[g_1, g_2, g_1^a, g_2^a, g_2^b, g_1^s, e(g_1, g_2)^{abs}\right] \overset{c}{\approx} \left[g_1, g_2, g_1^a, g_2^a, g_2^b, g_1^s, g_T^c\right]$$

where a, b, s and c are randomly chosen from $\mathbb{Z}_q$.

- $\mathsf{Setup}(1^\lambda, d_{\max})$: Outputs $\mathsf{pp} := (g_1, g_2, g_1^a, g_2^a)$.
- $\mathsf{KeyGen}(\mathsf{pp}, j)$:
  - If $j = 0$, then samples $t \xleftarrow{\$} \mathbb{Z}_q$ and outputs
  $$(\mathsf{pk}, \mathsf{sk}) := ((g_1^{a/t}, g_2^{a/t}), t)$$
  - If $j \geq 1$, output $\mathsf{pk} \xleftarrow{\$} G_2$.
- $\mathsf{Encode}(\mathsf{pk}, s)$:
  - If $\mathsf{pk} = (g_1^{a/t}, g_2^{a/t}) \in G_1 \times G_2$, output $(g_1^{a/t})^s$
  - If $\mathsf{pk} \in G_2$, output $e(g_1^a, \mathsf{pk})^s$
- $\mathsf{Recode}(\mathsf{rk}, c_0, c_1)$: Outputs $e(c_0, \mathsf{rk}) \cdot c_1$.
- $\mathsf{ReKeyGen}((g_1^{a/t}, g_2^{a/t}), \mathsf{pk}_1, \mathsf{pk}_{\mathrm{tgt}}, t)$: Outputs $\mathsf{rk} := (\mathsf{pk}_{\mathrm{tgt}} \cdot \mathsf{pk}_1^{-1})^t \in G_2$.

- $\mathsf{SimReKeyGen}((g_1^{a/t}, g_2^{a/t}), \mathsf{pk}_1)$: Picks $z \xleftarrow{\$} Z_q$ and outputs

$$\mathsf{rk} := (g_2^{a/t})^z, \quad \mathsf{pk}_{\mathrm{tgt}} := \mathsf{pk}_1 \cdot (g_2^a)^z$$

Correctness. Define $\Psi_{\mathsf{pk},s,j} := \{\mathsf{Encode}(\mathsf{pk}, s)\}$. For recoding, observe that:

$$\begin{aligned}
&\mathsf{Recode}\big((\mathsf{pk}_{\mathrm{tgt}} \cdot \mathsf{pk}_1^{-1})^t, g_1^{as/t}, e(g_1^a, \mathsf{pk}_1)^s\big)\\
&= e(g_1^{as/t}, (\mathsf{pk}_{\mathrm{tgt}} \cdot \mathsf{pk}_1^{-1})^t) \cdot e(g_1^a, \mathsf{pk}_1)^s\\
&= e(g_1^a, (\mathsf{pk}_{\mathrm{tgt}} \cdot \mathsf{pk}_1^{-1})^s) \cdot e(g_1^a, \mathsf{pk}_1)^s\\
&= e(g_1^a, \mathsf{pk}_{\mathrm{tgt}})^s = \mathsf{Encode}(\mathsf{pk}_{\mathrm{tgt}}, s)
\end{aligned}$$

Security. For back-tracking, negation is simply the multiplicative inverse over G_q. Correlation pseudorandomness follows readily from the DBDH assumption. In particular, suppose there is an adversary $\mathcal{D}$ that can distinguish between

$$\big(g_1, g_1, g_1^a, g_2^a, \big(g_1^{a/t}, g_2^{a/t}, (g_1^{a/t})^s\big), \big(g_2^{b_1}, e(g_1^a, g_2^{b_1})^s, \ldots, g_2^{b_\ell}, e(g_1^a, g_2^{b_\ell})^s, g_2^{b_{\ell+1}}, e(g_1^a, g_2^{b_{\ell+1}})^s\big)$$

and

$$\big(g_1, g_1, g_1^a, g_2^a, \big(g_1^{a/t}, g_2^{a/t}, (g_1^{a/t})^s\big), \big(g_2^{b_1}, e(g_1^a, g_2^{b_1})^s, \ldots, g_2^{b_\ell}, e(g_1^a, g_2^{b_\ell})^s, g_2^{b_{\ell+1}}, g_T^c\big)$$

for a randomly chosen c from $\mathbb{Z}_q$. Then, we can construct an adversary $\mathcal{A}$ that breaks DBDH assumption. In particular, $\mathcal{A}$ is given $g_1, g_2, g_1^a, g_2^a, g_2^b, g_1^s, \alpha$ where α is either either g_T^{abs} or g_T^c for a randomly chosen c. Then, the adversary $\mathcal{A}$ can simulate components $\big(g_1^{a/t}, g_2^{a/t}, (g_1^{a/t})^s\big)$ by sampling t' and setting $t = a/t'$. Then, $g_1^{a/t} = g_1^{t'}, g_2^{a/t} = g_2^{t'}$ and $(g_1^{a/t})^s = g_1^{t's}$ (which can be computed given g_1^s and t'). It can also simulate $g_2^{b_i}, e(g_1^a, g_2^{b_i})^s$ for all $i \leq \ell$ by sampling b_i in clear, and then using g_1^s, g_2^a and b_i to compute $e(g_1^s, g_2^a)^{b_i} = e(g_1^a, g_2^{b_i})^s$. Finally, it sets $b_{\ell+1} := b$ from the challenge together with α, and calls the adversary $\mathcal{D}$ to distinguish.

3.6.4 Attribute-Based Encryption from weak TOR

$\mathsf{Setup}(1^\lambda, 1^\ell, d_{\max})$: For each one of ℓ input bits, generate two public/secret key pairs. Also, generate a public/secret key pair for the start and accept states:

$$\begin{aligned}
(\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}, 0) \quad \text{for } i \in [\ell], b \in \{0,1\}\\
(\mathsf{pk}_{\mathrm{start}}, \mathsf{sk}_{\mathrm{start}}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1)\\
(\mathsf{pk}_{\mathrm{accept}}, \mathsf{sk}_{\mathrm{accept}}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1)
\end{aligned}$$

Output

$$\mathsf{mpk} := \begin{pmatrix} \mathsf{pk}_{1,0} & \mathsf{pk}_{2,0} & \ldots & \mathsf{pk}_{\ell,0} & \mathsf{pk}_{\mathrm{start}} \\ \mathsf{pk}_{1,1} & \mathsf{pk}_{2,1} & \ldots & \mathsf{pk}_{\ell,1} & \mathsf{pk}_{\mathrm{accept}} \end{pmatrix}$$

$$\mathsf{msk} := \begin{pmatrix} \mathsf{sk}_{1,0} & \mathsf{sk}_{2,0} & \ldots & \mathsf{sk}_{\ell,0} & \mathsf{sk}_{\mathrm{start}} \\ \mathsf{sk}_{1,1} & \mathsf{sk}_{2,1} & \ldots & \mathsf{sk}_{\ell,1} & \mathsf{sk}_{\mathrm{accept}} \end{pmatrix}$$

$\mathsf{Enc}(\mathsf{mpk}, a, m)$: For $a \in \{0,1\}^\ell$, choose a uniformly random $s \xleftarrow{\$} \mathcal{S}$ and encode it under the public keys specified by the index bits and the start state:

$$\psi_i \leftarrow \mathsf{Encode}(\mathsf{pk}_{i,a_i}, s) \text{ for all } i \in [\ell]$$
$$\psi_{\text{start}} \leftarrow \mathsf{Encode}(\mathsf{pk}_{\text{start}}, s)$$

Encrypt the message:

$$\tau \leftarrow \mathsf{E}(\mathsf{Encode}(\mathsf{pk}_{\text{accept}}, s), m)$$

Output the ciphertext:

$$\mathsf{ct}_a = \big(\ \psi_1,\ \ \psi_2,\ \ \ldots,\ \ \psi_\ell,\ \ \psi_{\text{start}},\ \ \tau\ \big)$$

$\mathsf{KeyGen}(\mathsf{msk}, \Gamma)$: $\Gamma : \{0,1\}^\ell \to \{0,1\}$ is a branching program that takes a ℓ-bit input and outputs a single bit.

- For every node u, except the start and accept nodes, sample public/secret key pair:
$$(\mathsf{pk}_u, \mathsf{sk}_u) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1)$$
- For every edge (u, v) labeled (i, b) in Γ, sample a recoding key $\mathsf{rk}_{u,v}$ as follows:
$$\mathsf{rk}_{u,v} \leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{i,b}, \mathsf{pk}_u, \mathsf{sk}_{i,b}, \mathsf{pk}_v\Big)$$

The secret key sk_Γ is the collection of all the recoding keys $\mathsf{rk}_{u,v}$ for every edge (u, v) in Γ.

$\mathsf{Dec}(\mathsf{sk}_\Gamma, \mathsf{ct}_a)$: Suppose $\Gamma(a) = 1$; output $\perp$ otherwise. Let Π denote the (directed) path from the start node to the accept node in Γ_a. For every edge (u, v) labeled (i, a_i) in Π, apply the recoding algorithm on the two encodings ψ_i, ψ_u and the recoding key $\mathsf{rk}_{u,v}$:

$$\psi_v \leftarrow \mathsf{Recode}\Big(\mathsf{rk}_{u,v}, \psi_i, \psi_u\Big)$$

If $\Gamma(a) = 1$, we obtain ψ_{accept}. Decrypt and output the message:

$$m \leftarrow \mathsf{D}(\psi_{\text{accept}}, \tau)$$

Lemma 3.6.2 (Correctness). *Let $\mathcal{G} = \{\Gamma\}_\lambda$ be a collection of polynomial-size branching programs of depth at most $d_{\max}$ and let* wTOR *be a weak "two-to-one" recoding scheme for $d_{\max}$ levels. Then, the construction presented above is a correct attribute-based encryption scheme for $\mathcal{G}$.*

Proof. Let Π denote the directed path from the start to the accept nodes in Γ_a. We show via induction on nodes v along the path Π that

$$\psi_v \in \Psi_{\mathsf{pk}_v, s, j}$$

where j is the depth of node v along the path. The base case for $v :=$ start node follows immediately from correctness of Encode. For the inductive step, consider a node v along the path Π at depth j for some edge (u, v) labeled (i, a_i). By the induction hypothesis,

$$\psi_u \in \Psi_{\mathsf{pk}_u, s, j_0}$$

where $j_0 < j$ denote the depths of node u. Also by the correctness of the Encode algorithm, for all $i \in [\ell]$

$$\psi_i \in \Psi_{\mathsf{pk}_{i,a_i}, s, 0}$$

It follows immediately from the correctness of Recode that

$$\psi_v \in \Psi_{\mathsf{pk}_v, s, j_0+1} \subseteq \Psi_{\mathsf{pk}_v, s, j}$$

which completes the inductive proof. Since $C(a) = 1$, we have

$$\psi_{\text{accept}} \in \Psi_{\mathsf{pk}_{\text{accept}}, s, d_{\max}}$$

Recall that $\tau \leftarrow \mathsf{E}(\mathsf{Encode}(\mathsf{pk}_{\text{accept}}, s), m)$. Finally, by the correctness of (E, D),

$$\mathsf{D}(\psi_{\text{accept}}, \tau) = m$$

□

Lemma 3.6.3 (Selective Security). *For any adversary $\mathcal{A}$ against selective security of the attribute-based encryption scheme for branching programs, there exist an adversary $\mathcal{B}$ against correlated pseudorandomness of wTOR whose running time is essentially the same as that of $\mathcal{A}$, such that*

$$\mathsf{Adv}^{\text{PE}}_{\mathcal{A}}(\lambda) \leq \mathsf{Adv}^{\text{CP}}_{\mathcal{B}}(\lambda) + \text{negl}(\lambda)$$

where $\text{negl}(\lambda)$ *captures the statistical security terms in TOR.*

In the proof of security, we will rely crucially on the following combinatorial property of branching programs: for any input x, the graph Γ_x does not contain any cycles as an *undirected* graph.

Alternative algorithms. Fix the selective challenge a. We also get a collection of public keys, corresponding encodings from the "outside": $(\mathsf{pk}_i, \psi_i)_{i \in [\ell+2]}$, where the challenge is to decide whether $\psi_{\ell+1}$ is $\mathsf{Encode}(\mathsf{pk}_{\ell+2}, s)$ or random. The main challenge is design an alternative algorithm KeyGen^* for answering secret key queries without knowing $\mathsf{sk}_{1,a_1}, \ldots, \mathsf{sk}_{\ell,a_\ell}$ or $\mathsf{sk}_{\text{start}}, \mathsf{sk}_{\text{accept}}$. We consider the following "alternative" algorithms.

$\mathsf{Setup}^*(1^\lambda, 1^\ell, d_{\max})$: Let

$$\begin{aligned}(\mathsf{pk}_{i,1-a_i}, \mathsf{sk}_{i,1-a_i}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}, 0) \text{ for } i \in [\ell] \\ \mathsf{pk}_{i,a_i} &:= \mathsf{pk}_i \text{ for } i \in [\ell] \\ \mathsf{pk}_{\text{start}} &:= \mathsf{pk}_{\ell+1} \\ \mathsf{pk}_{\text{accept}} &:= \mathsf{pk}_{\ell+2}\end{aligned}$$

Define and output the master public key as follows:

$$\mathsf{mpk} = \begin{pmatrix} \mathsf{pk}_{1,0} & \mathsf{pk}_{2,0} & \dots & \mathsf{pk}_{\ell,0} & \mathsf{pk}_{\text{start}} \\ \mathsf{pk}_{1,1} & \mathsf{pk}_{2,1} & \dots & \mathsf{pk}_{\ell,1} & \mathsf{pk}_{\text{accept}} \end{pmatrix}$$

$\mathsf{Enc}^*(\mathsf{mpk}, a, m)$: Define

$$\begin{aligned}\psi_{i,a_i} &:= \psi_i \quad \text{for all } i \in [\ell] \\ \psi_{\text{start}} &:= \psi_{\ell+1} \\ \psi_{\text{accept}} &:= \psi_{\ell+2}\end{aligned}$$

Encrypt the message m:

$$\tau \leftarrow \mathsf{E}(\psi_{\text{accept}}, b)$$

Output the simulated ciphertext

$$\mathsf{ct}_a = \begin{pmatrix} \psi_1, & \psi_2, & \dots, & \psi_\ell, & \psi_{\text{start}}, & \tau \end{pmatrix}$$

$\mathsf{KeyGen}^*(\mathsf{msk}, \Gamma)$: Let Γ'_a denote the *undirected* graph obtained from Γ_a by treating every directed edge as an undirected edge (while keeping the edge label). Observe that Γ'_a satisfies the following properties:

- Γ'_a contains no cycles. This is because Γ_a is acyclic and every nonterminal node contains exactly one outgoing edge.
- The start node and the accept node lie in different connected components in Γ'_a, since $\Gamma(a) = 0$.

Simulation invariant: for each "active" edge labeled (i, a_i) from node u to node v, simulate the recoding key. Choose our own public/secret key pair for each "inactive" edges $(i, 1 - a_i)$ and generate the recoding key honestly.

- Run a DFS in Γ'_a starting from the start node. Whenever we visit a new node v from a node u along an edge labeled (i, a_i), we set:

$$\begin{aligned}(\mathsf{pk}_v, \mathsf{rk}_{u,v}) &\leftarrow \mathsf{SimReKeyGen}\big(\mathsf{pk}_{i,a}, \mathsf{pk}_u\big) \quad \text{if } (u, v) \text{ is a directed edge in } \Gamma \\ (\mathsf{pk}_v, -\mathsf{rk}_{v,u}) &\leftarrow \mathsf{SimReKeyGen}\big(\mathsf{pk}_{i,a}, \mathsf{pk}_u\big) \quad \text{if } (v, u) \text{ is a directed edge in } \Gamma\end{aligned}$$

Here, we exploit the back-tracking property in wTOR.

Note that since $\Gamma(a) = 0$, then the accept node is not assigned a public key by this process.

- For all nodes u without an assignment, run $(\mathsf{pk}_u, \mathsf{sk}_u) \leftarrow \mathsf{KeyGen}(\mathsf{pp}, 1)$.
- For every remaining edge (u, v) labeled $(i, 1 - a_i)$ in Γ, sample a recoding key $\mathsf{rk}_{u,v}$ as in KeyGen using $\mathsf{sk}_{i,1-a}$ as follows:

$$\mathsf{rk}_{u,v} \leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{i,1-a}, \mathsf{pk}_u, \mathsf{sk}_{i,1-a}, \mathsf{pk}_v\Big)$$

The secret key sk_Γ is simply the collection of all the recoding keys $\mathsf{rk}_{u,v}$ for every edge (u, v) in Γ.

Game sequence. Next, consider the following sequence of games. We use $\mathsf{Adv}_0, \mathsf{Adv}_1, \ldots$ to denote the advantage of the adversary $\mathcal{A}$ in Games 0, 1, etc. Let n denote the number of edges in a branching program Γ labeled (i, a_i) for some i, and for all $j \in [n]$ let e_j denote the actual edge.

Game 0 Real experiment.

Game i **for** $i = 1, 2, \ldots, q$ As in Game 0, except the challenger answers the first i key queries using KeyGen^* and the remaining $q - i$ key queries using KeyGen. For the i'th key query Γ_i, we also consider sub-Games $i.e$ as follows:

Game $i.j$**, for** $j = 1, \ldots, n$ For edge $e_j = (u, v)$ labeled (i, a_i), the challenger switches the simulated recoding key $\mathsf{rk}_{u,v}$ from KeyGen to KeyGen^*.

From lemma 3.6.4, it follows that

$$|\mathsf{Adv}_i - \mathsf{Adv}_{i+1}| \leq \text{negl}(\lambda) \text{ for all } i$$

Note that in Game q, the challenger runs Setup^* and answers all key queries using KeyGen^* with the selective challenge a and generates the challenge ciphertext using Enc.

Game $q + 1$ Same as Game q, except the challenger generates the challenge ciphertext using Enc^* with $\psi_{\ell+2} \leftarrow \mathsf{Encode}(\mathsf{pk}_{\ell+2}, s)$.

$$\mathsf{Adv}_{q+1} = \mathsf{Adv}_q$$

Game $q + 2$ Same as Game $q + 1$, except $\psi_{\ell+2} \xleftarrow{\$} \mathcal{K}$. It is straight-forward to construct an adversary $\mathcal{B}$ such that

$$|\mathsf{Adv}_{q+1} - \mathsf{Adv}_{q+2}| \leq \mathsf{Adv}_{\mathcal{B}}^{\text{CP}}(\lambda)$$

Finally, by the one-time semantic security of (E, D) for all $m_0, m_1 \in \mathcal{M}$, $\mathsf{E}(\psi_{\ell+2}, m_0)$ is indistinguishable from $\mathsf{E}(\psi_{\ell+2}, m_1)$ (for $\psi_{\ell+2} \xleftarrow{\$} \mathcal{K}$). Therefore, it follows that $\mathsf{Adv}_{q+2} \leq \text{negl}(\lambda)$, concluding the proof of security.

Lemma 3.6.4. *Games i and $i+1$ are (statistically) indistinguishable for all $i = 0, \ldots, q-1$.*

Proof. The difference in two games is in how $i+1$'st key query is answered. In Game i, it is answered using KeyGen, whereas in Game $i+1$ it is answered using KeyGen^*. Now, consider sub-Games $(i+1).j$ and $(i+1).(j+1)$. The difference in two sub-Games is in how the recoding key $rk_{u,v}$. Now, by the recoding simulation and back-tracking, it follows that the keys sampled statistically indistinguishable, concluding the lemma. □

Lemma 3.6.5. *Games $q+1$ and $q+2$ are computationally indistinguishable.*

Proof. Suppose there exists an adversary $\mathcal{B}^*$ that distinguishes between the Games $q+1$ and $q+2$. We construct and adversary $\mathcal{B}$ that breaks correlated pseudorandomness property. The adversary $\mathcal{B}$ gets as input $(\mathsf{pk}_i, \psi_i)_{i \in [\ell+1]}, \mathsf{pk}_{\ell+2}, \psi^*$, where ψ^* is either a valid encoding using key $pk_{\ell+2}$ or a randomly chosen element. The adversary $\mathcal{B}$ uses these values to invoke algorithms $\mathsf{Setup}^*, \mathsf{Enc}^*, \mathsf{KeyGen}^*$, where it uses $\psi_{\ell+2} := \psi^*$. It invokes the adversary $\mathcal{B}^*$ and the same thing. Clearly, if $\psi^* = \mathsf{Encode}(\mathsf{pk}_{\ell+2}, s)$, then this experiment corresponds to Game $q+1$. On the other hand, if $\psi^* \xleftarrow{\$} \mathcal{K}$ then this experiment corresponds to Game $q+2$. This concludes the proof of the lemma. □

3.7 Extensions

3.7.1 Outsourcing Decryption

In this section we show how to modify our main construction of attribute-based encryption to support outsourcing of decryption circuits, similar to [GHW11]. Syntactically, ABE with outsourcing decryption consists of algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Eval}, \mathsf{Enc}, \mathsf{Dec})$, satisfying the same semantics as standard ABE, except:

1. we require that the KeyGen algorithm on input (msk, C) returns two keys:
   - the evaluation key ek_C, that is given to a computationally powerful proxy,
   - and a decryption key dk, given to the client.
2. Given a ciphertext ct_a, the proxy must perform the "bulk" of the computation by running $\mathsf{Eval}(\mathsf{ek}_C, \mathsf{ct}_a) \to \mathsf{ct}'_a$. The client, holding the decryption key dk, can then compute $\mathsf{Dec}(\mathsf{dk}, \mathsf{ct}'_a) \to m$ iff $C(a) = 1$.

Formally, we require that the runtime of Dec is $\text{poly}(\lambda, d_{\max})$, where λ is the security parameter and $d_{\max}$ is the maximum circuit depth (in particular, it is independent on the size of C). The other correctness properties are the same as in standard ABE.

We envision a semi-honest proxy server, where it performs all algorithms correctly but tries to learn the message payload. That is, the security ensures that an adversary (that may play proxy's role) should learn nothing about the message, conditioned on that it queries for decryption keys dk's for predicates that are not satisfied by the challenge index (note, the adversary *can* query for evaluation keys separately for predicates that are satisfied). More formally, for a stateful adversary $\mathcal{A}$ (that can maintain a global state information throughout the execution of the experiment), we consider the following game:

- **Phase 1:** The adversary specifies the challenge index a, gives it to the challenger who runs the Setup algorithm and returns the master public key mpk to the adversary.
- **Phase 2:** The adversary may issue oracle queries of the form C to KeyGen algorithm and specify whether to obtain ek_C only or both $(\mathsf{ek}_C, \mathsf{dk})$. It outputs two challenge messages m_0, m_1.
- **Phase 3:** The challenger runs $\mathsf{ct}_a \leftarrow \mathsf{Enc}(\mathsf{mpk}, a, m_b)$ for a randomly chosen bit b, and gives ct_a to the adversary.
- **Phase 4:** Is the same as Phase 2, where eventually the adversary outputs a guess bit b'.

The ABE with outsourcing decryption capabilities is said to be secure if no adversary can guess the bit b with probability significantly better than 1/2, conditioned on that for all decryption keys dk that it obtained for predicates C, $C(a) = 0$.

We now give a high-level overview of the changes applied to our main construction and then provide a formal construction. As before, the key-generation algorithm assigns two keys for each circuit wire. The evaluation key consists of all the recoding keys for the circuit. In addition, the output wire has another key pk_{out} which now plays a special role. The recoding key from $\mathsf{pk}_{|C|,1}$ to pk_{out} is only given to the client as the decryption key. If $C(a) = 1$, the the proxy computes an encoding under the $\mathsf{pk}_{|C|,1}$ and forwards it to the client. The client applies the transformation, and decrypts the message. For technical reasons, since we are using "two-to-one" recoding mechanism, we need to introduce an auxiliary public key pk_{in} and a corresponding encoding.

$\mathsf{Setup}(1^\lambda, 1^\ell, d_{\max})$: For each of the ℓ input wires, generate two public/secret key pairs. Also, generate an additional public/secret key pair:

$$\begin{aligned} (\mathsf{pk}_{i,b}, \mathsf{sk}_{i,b}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}) \qquad \text{for } i \in [\ell], b \in \{0,1\} \\ (\mathsf{pk}_{\text{out}}, \mathsf{sk}_{\text{out}}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}) \\ (\mathsf{pk}_{\text{in}}, \mathsf{sk}_{\text{in}}) &\leftarrow \mathsf{KeyGen}(\mathsf{pp}) \end{aligned}$$

Output

$$\mathsf{mpk} := \begin{pmatrix} \mathsf{pk}_{1,0} & \mathsf{pk}_{2,0} & \dots & \mathsf{pk}_{\ell,0} & \mathsf{pk}_{\text{in}} \\ \mathsf{pk}_{1,1} & \mathsf{pk}_{2,1} & \dots & \mathsf{pk}_{\ell,1} & \mathsf{pk}_{\text{out}} \end{pmatrix} \qquad \mathsf{msk} := \begin{pmatrix} \mathsf{sk}_{1,0} & \mathsf{sk}_{2,0} & \dots & \mathsf{sk}_{\ell,0} & \mathsf{sk}_{\text{in}} \\ \mathsf{sk}_{1,1} & \mathsf{sk}_{2,1} & \dots & \mathsf{sk}_{\ell,1} & \mathsf{sk}_{\text{out}} \end{pmatrix}$$

$\mathsf{Enc}(\mathsf{mpk}, a, m)$: For $a \in \{0,1\}^\ell$, choose a uniformly random $s \xleftarrow{\$} \mathcal{S}$ and encode it under the public keys specified by the index bits:

$$\psi_i \leftarrow \mathsf{Encode}(\mathsf{pk}_{i,a_i}, s) \text{ for all } i \in [\ell]$$

Encode s under the input public key:

$$\psi_{\text{in}} \leftarrow \mathsf{Encode}(\mathsf{pk}_{\text{in}}, s)$$

Encrypt the message m:

$$\tau \leftarrow \mathsf{E}(\mathsf{Encode}(\mathsf{pk}_{\text{out}}, s), m)$$

Output the ciphertext

$$\mathsf{ct}_a := \big(\ \psi_1,\ \ \psi_2,\ \ \ldots,\ \ \psi_\ell,\ \ \psi_{\text{in}},\ \ \tau\ \big)$$

$\mathsf{KeyGen}(\mathsf{msk}, C)$:

1. For every non-input wire $w = \ell+1, \ldots, |C|$ of the circuit C, and every $b \in \{0,1\}$, generate public/secret key pairs:

$$(\mathsf{pk}_{w,b}, \mathsf{sk}_{w,b}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$$

2. For the gate $g = (u, v, w)$ with output wire w, compute the four recoding keys $\mathsf{rk}^w_{b,c}$ (for $b, c \in \{0,1\}$):

$$\mathsf{rk}^w_{b,c} \leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{u,b}, \mathsf{pk}_{v,c}, \mathsf{sk}_{u,b}, \mathsf{pk}_{w,g_w(b,c)}\Big)$$

3. Also, compute the recoding key

$$\mathsf{rk}^{\text{out}} \leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{|C|,1}, \mathsf{pk}_{\text{in}}, \mathsf{sk}_{|C|,1}, \mathsf{pk}_{\text{out}}\Big)$$

Output the evaluation key which is a collection of $4(|C| - \ell)$ recoding keys

$$\mathsf{ek}_C := \Big(\ \mathsf{rk}^w_{b,c}\ :\ w \in \big[\,\ell+1, |C|\,\big],\ b, c \in \{0,1\}\ \Big)$$

and the decryption key $\mathsf{dk} := \mathsf{rk}^{\text{out}}$.

$\mathsf{Eval}(\mathsf{ek}_C, \mathsf{ct}_a)$: We tacitly assume that ct_a contains the index a. For $w = \ell+1, \ldots, |C|$, let $g = (u, v, w)$ denote the gate with output wire w. Suppose wires u and v carry the values b^* and c^*, so that wire w carries the value $d^* := g_w(b^*, c^*)$. Compute

$$\psi_{w,d^*} \leftarrow \mathsf{Recode}\Big(\mathsf{rk}^w_{b^*,c^*}, \psi_{u,b^*}, \psi_{v,c^*}\Big)$$

If $C(a) = 1$, then we would have computed $\psi_{|C|,1}$. Output

$$\mathsf{ct}'_a := (\psi_{|C|,1}, \psi_{\text{in}}, \tau)$$

If $C(a) = 0$, output $\perp$.

$\mathsf{Dec}(\mathsf{dk}, \mathsf{ct}'_a)$: Apply the transformation

$$\psi_{\text{out}} \leftarrow \mathsf{Recode}\Big(\mathsf{rk}^{\text{out}}, \psi_{\text{in}}, \psi_{|C|,1}\Big)$$

and output the message

$$m \leftarrow \mathsf{D}(\ \psi_{\text{out}}, \tau\ )$$

Security We informally state how to modify the simulator in the proof of security in Section-3.5.4. The simulator gets $\{\mathsf{pk}_i, \psi_i\}_{i\in[\ell+2]}$ from the "outside". It assigns $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$ as the public keys specified by the bits of a and $\mathsf{pk}_{\text{in}} := \mathsf{pk}_{\ell+1}, \mathsf{pk}_{\text{out}} := \mathsf{pk}_{\ell+2}$. It is easy to see how to simulate the ciphertext: all the input encodings become a part of it, as well as an encryption of the message using $\psi_{\text{out}} := \psi_{\ell+2}$. Now, the evaluation key ek is simulated by applying the TOR simulator.

- For query C such that $C(a) = 0$, the simulator can choose $(\mathsf{pk}_{|C|,1}, \mathsf{sk}_{|C|,1})$ by itself (the public key $\mathsf{pk}_{|C|,0}$ is "fixed" by the TOR simulator). Hence, the decryption key dk can be computed using $\mathsf{sk}_{|C|,1}$.

- On the other hand, for query C such that $C(a) = 1$, the adversary is not allowed to obtain the decryption key dk, hence there is not need to simulate it.

3.7.2 Extending Secret Keys

Consider the following problem: a users holds two (or more) secret keys sk_{C_1} and sk_{C_2}. C_1 allows to decrypt all ciphertexts addressed to *human resources* department and C_2 allows to decrypt ciphertexts addressed to *share holders*. The user wishes to create (and delegate) another secret key sk_{C^*} that allows to decrypt ciphertexts addressed to *human resources* and *share holders*. The question that we study is whether it is possible to allow the user to compute sk_{C^*} without calling the authority holding the master secret key msk.

More formally, given $\{\mathsf{sk}_{C_i}\}_{i\in[q]}$ we require an additional algorithm $\mathsf{Extend}(\{\mathsf{sk}_{C_i}\}_{i\in[q]}, C^*)$ that returns a secret key sk_{C^*} for any circuit C^* that is an black-box monotone composition of C_i's. The correctness properties are that of the standard ABE, in addition to that sk_{C^*} can be used to decrypt ciphertexts ct_a if $C^*(a) = 1$. The security properties also remain similar to standard ABE: no adversary should be able to learn anything from the ciphertext ct_a about the message m given many secret keys sk_C for queries C such that $C(a) = 0$, where secret key sk_C may be generated via KeyGen or Extend algorithms. Note that only monotone compositions are realizable, since otherwise a users holding a secret keys sk_C where $C(a) = 0$

could come up with a secret key for $\overline{C}$ and hence break the security game.

To suppose monotone extensions, it is enough to show how to obtain (1) $\mathsf{sk}_{C_1 \text{ AND } C_2}$ given $\mathsf{sk}_{C_1}, \mathsf{sk}_{C_2}$, and (2) $\mathsf{sk}_{C_1 \text{ OR } C_2}$ given $\mathsf{sk}_{C_1}, \mathsf{sk}_{C_2}$. We start from the construction presented in Section-3.7.1 and explain how to modify it.

- First, instead of fixing $\mathsf{pk}_{|C_i|,1} = \mathsf{pk}_{\text{out}}$ in this construction, we sample $(\mathsf{pk}_{|C_i|,1}, \mathsf{sk}_{|C_i|,1}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$ for every query C_i. In the secret key sk_{C_i}, in additional to all recoding keys for the internal gates, we publish

  $$\mathsf{rk}_{\mathsf{aux}} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_{|C_i|,1}, \mathsf{pk}_{|C_i|,1}, \mathsf{sk}_{|C_i|,1}, \mathsf{pk}_{\text{out}})$$

  and also include the secret key $\mathsf{sk}_{|C_i|,1}$. Clearly, this does not affect the correctness, because given an encoding under $\mathsf{pk}_{|C_i|,1}$ the user can compute the encoding under pk_{out} and record the message if $C_i(a) = 1$. Moreover, this does not affect the security since our simulation proceeds by sampling $(\mathsf{pk}_{|C_i|,1}, \mathsf{sk}_{|C_i|,1})$ honestly using KeyGen algorithm and the fact the adversary is restricted to queries C_i such that $C_i(a) = 0$.

- Now, suppose the user holds sk_{C_1} and sk_{C_2}, let $C^* = C_1$ AND C_2 (binary OR-gate is handled analogously). Then, we define $\mathsf{Extend}(\mathsf{sk}_{C_1}, \mathsf{sk}_{C_2}, C^* = C_1 \text{ AND } C_2)$ as follows. Sample keys associated with the active value (i.e., "1") at the output wire of C^*:

  $$(\mathsf{pk}_{|C^*|,1}, \mathsf{sk}_{|C^*|,1}) \leftarrow \mathsf{KeyGen}(\mathsf{pp})$$

  Also, sample four recoding keys $\mathsf{rk}^{C^*}_{b,c}$ (for $b, c \in \{0,1\}$):

  $$\begin{aligned}
  (\mathsf{pk}_{|C^*|,0}, \mathsf{rk}^{C^*}_{0,0}) &\leftarrow \mathsf{SimReKeyGen}(\mathsf{pk}_{|C_1|,0}, \mathsf{pk}_{|C_2|,0}) \\
  \mathsf{rk}^{C^*}_{0,1} &\leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{|C_1|,0}, \mathsf{pk}_{|C_2|,1}, \mathsf{sk}_{|C_2|,1}, \mathsf{pk}_{|C^*|,0}\Big) \\
  \mathsf{rk}^{C^*}_{1,0} &\leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{|C_1|,1}, \mathsf{pk}_{|C_2|,0}, \mathsf{sk}_{|C_1|,1}, \mathsf{pk}_{|C^*|,0}\Big) \\
  \mathsf{rk}^{C^*}_{1,1} &\leftarrow \mathsf{ReKeyGen}\Big(\mathsf{pk}_{|C_1|,1}, \mathsf{pk}_{|C_2|,1}, \mathsf{sk}_{|C_1|,1}, \mathsf{pk}_{\text{out}}\Big)
  \end{aligned}$$

  And an auxiliary recoding key

  $$\mathsf{rk}_{\mathsf{aux}} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_{|C^*|,1}, \mathsf{pk}_{|C^*|,1}, \mathsf{sk}_{|C^*|,1}, \mathsf{pk}_{\text{out}})$$

  The secret key sk_{C^*} consists of all recoding keys for the internal gates of C_1 and C_1, recoding keys $\mathsf{rk}^{C^*}_{b,c}$ for $b, c \in \{0,1\}$, $\mathsf{rk}_{\mathsf{aux}}$ and secret key $\mathsf{sk}_{|C^*|,1}$.

It is easy to see that the correctness is easy to satisfy as before. In particular, given encoding under output public key $\mathsf{pk}_{|C^*|,1}$, the user can obtain encoding under $\mathsf{pk}_{\mathsf{aux}}$, and therefore recover the message if $C^*(a) = 1$. Also, the security holds similarly, by satisfying the following invariant: for all inactive wires w (wires carrying value "0" when evaluating $C^*(a)$) of the circuit, the simulator knows the secret key $\mathsf{sk}_{w,0}$. Hence, it knows the secret key $\mathsf{sk}_{|C^*|,1}$

for all C^* such that $C^*(a) = 0$ and can use it to compute $\mathsf{rk}_{\mathsf{aux}}$ by running:

$$\mathsf{rk}_{\mathsf{aux}} \leftarrow \mathsf{ReKeyGen}(\mathsf{pk}_{|C^*|,1}, \mathsf{pk}_{|C^*|,1}, \mathsf{sk}_{|C^*|,1}, \mathsf{pk}_{\mathrm{out}}).$$

The remaining internal recoding keys (and the ciphertext) are sampled by the simulator identically as in the security proof in Section 3.5.3.

3.8 Conclusions and Open Problems

In this chapter we showed a construction of attribute-based encryption from lattices. The main drawback of our construction is the dependence on the depth. It remains an open problem to remove this dependence, perhaps by providing a general bootstrapping transformation. It remains open to construct ABE for other models of computation such as Turing machines and RAM programs from standard assumptions. It also remains open to construct ciphertext-policy attribute based encryption with unbounded size predicates – where the ciphertext is associated with a predicate P and message m, and the secret key is associated with attributes $\mathbf{a}$. Using our construction, it is possible to realize only a-priori bounded ciphertext-policy attribute based encryption (that is, the description of the predicates must be a-priori bounded by the setup).

Chapter 4

Predicate Encryption for Circuits

Predicate encryption [BW07, SBC$^+$07, KSW08] is a new paradigm for public-key encryption that supports searching on encrypted data. In predicate encryption, ciphertexts are associated with descriptive attribute values a in addition to plaintexts m, secret keys are associated with a predicate P, and a secret key decrypts the ciphertext to recover m if and only if $P(a) = 1$. The security requirement for predicate encryption enforces privacy of a and the plaintext even amidst multiple secret key queries: an adversary holding secret keys for different query predicates learns nothing about the attribute a and the plaintext if none of them is individually authorized to decrypt the ciphertext.

Motivating applications. We begin with several motivating applications for predicate encryption [BW07, SBC$^+$07]:

- For inspecting recorded log files for network intrusions, we would encrypt network flows labeled with a set of attributes from the network header, such as the source and destination addresses, port numbers, time-stamp, and protocol numbers. We could then issue auditors with restricted secret keys that can only decrypt the network flows that fall within a particular range of IP addresses and some specific time period.

- For credit card fraud investigation, we would encrypt credit card transactions labeled with a set of attributes such as time, costs and zipcodes. We could then issue investigators with restricted secret keys that decrypt transactions over \$1,000 which took place in the last month and originated from a particular range of zipcodes.

- For anti-terrorism investigation, we would encrypt travel records labeled with a set of attributes such as travel destination and basic traveller data. We could then issue investigators with restricted secret keys that match certain suspicious travel patterns.

- For online dating, we would encrypt personal profiles labeled with dating preferences pertaining to age, height, weight, salary and hobbies. Secret keys are associated with specific attributes and can only decrypt profiles for which the attributes match the dating preferences.

In all of these examples, it is important that unauthorized parties do not learn the contents of the ciphertexts, nor of the meta-data associated with the ciphertexts, such as the network header or dating preferences. On the other hand, it is often okay to leak the meta-data to authorized parties. We stress that privacy of the meta-data is an additional security requirement provided by predicate encryption but not by the related and weaker notion of attribute-based encryption (ABE) [SW05, GPSW06]; the latter only guarantees the privacy of the plaintext m and not the attribute a.

Utility and expressiveness. The utility of predicate encryption is intimately related to the class of predicates for which we could create secret keys. Ideally, we would like to support the class of all circuits. Over the past decade, substantial advances were made for the weaker primitive of ABE, culminating most recently in schemes supporting any policy computable by general circuits [GVW13, BGG$^+$14] under the standard LWE assumption [Reg09]. However, the state-of-the-art for predicate encryption is largely limited to very simple functionalities related to computing an inner product [BW07, SBC$^+$07, KSW08, AFV11, GMW15].

4.1 Our Contributions, Techniques and Related Work

Our Contributions. In this work, we substantially advance the state of the art to obtain predicate encryption for all circuits:

> **Theorem (informal).** Under the LWE assumption, there exists a predicate encryption scheme for all circuits, with succint ciphertexts and secret keys independent of the size of the circuit.

As with prior LWE-based ABE for circuits [GVW13, BGG$^+$14], to support circuits of depth d, the parameters of the scheme grow with poly(d), and we require sub-exponential $n^{\Omega(d)}$ hardness of the LWE assumption. In addition, the security guarantee is selective, but can be extended to adaptive security via complexity leveraging [BB04].

Privacy guarantees. The privacy notion we achieve is a *simulation-based* variant of "attribute-hiding" from the literature [SBC$^+$07, OT10, AFV11]. That is, we guarantee privacy of the attribute a and the plaintext m against collusions holding secret keys for predicates P such that $P(a) = 0$. An even stronger requirement would be to require privacy of a even against authorized keys corresponding to predicates P where $P(a) = 1$; in the literature, this stronger notion is referred to as "full attribute-hiding" [BW07, KSW08]. This stronger requirement is equivalent to "full-fledged" functional encryption [BSW11], for which we cannot hope to achieve simulation-based security for all circuits as achieved in this work [BSW11, AGVW13].

Relation to prior works. Our result subsumes all prior works on predicate encryption under standard cryptographic assumptions, apart from a few exceptions pertaining to the inner product predicate [BW07, KSW08, OT12a]. These results achieve a stronger security notion for predicate encryption, known as full (or strong) security (please refer to Sections 4.3.1, 4.2.1 for definitions).

In a recent break-through work, Garg et al. [GGH$^+$13b] gave a beautiful candidate construction of functional encryption (more general primitive than predicate encryption) for arbitrary circuits. However, the construction relies on "multi-linear maps" [GGH13a, CLT13, GGH15], for which we have few candidates and which rely on complex intractability assumptions that are presently poorly understood and not extensively studied in the literature. It remains an intriguing open problem to construct a functional encryption scheme from a standard assumption, such as LWE.

In contrast, if we consider functional encryption with *a-priori bounded collusions size* (that is, the number of secret keys any collusion of adversaries may obtain is fixed by the scheme at the setup phase), then it is possible to obtain functional encryption for general circuits under a large class of standard assumptions [SS10a, GVW12, GKP$^+$13b]. This notion is *weaker* than standard notion of functional encryption, yet remains very meaningful for many applications.

4.1.1 Overview of Our Construction

Our starting point is the work of Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich [GKP$^+$13b] who show how to convert an attribute-based encryption (ABE) scheme into a *single key secure* functional encryption (FE) scheme. Recall that in an attribute-based encryption scheme [GPSW06], a ciphertext is associated with a descriptive value (a public "attribute") a and plaintext m, and it hides m, but not a. The observation of Goldwasser et al. [GKP$^+$13b] is to hide a by encrypting it using a fully homomorphic encryption (FHE) scheme [Gen09, BV11b], and then using the resulting FHE ciphertext as the public "attribute" in an ABE scheme for general circuits [GVW13, BGG$^+$14]. This has the dual benefit of guaranteeing privacy of a, while at the same time allowing homomorphic computation of predicates P on the encryption of a.

This initial idea quickly runs into trouble. The decryptor who is given the predicate secret key for P and a predicate encryption of (a, m) can indeed compute an *FHE encryption* of $P(a)$. However, the decryption process is confronted with a decision, namely whether to release the message m or not, and this decision depends on whether the *plaintext* $P(a)$ is 0 or 1.[1] Clearly, resolving this conundrum requires obtaining $P(a)$, which requires knowledge of the FHE secret key. Goldwasser et al. [GKP$^+$13b] solved this by employing a (single use) Yao garbling of the FHE decryption circuit, however this limited them to obtaining *single key secure* predicate/functional encryption schemes.[2]

[1]In fact, there is a syntactic mismatch since $\hat{P}(\cdot)$ is not a predicate, as it outputs an FHE ciphertext.

[2]A reader familiar with [GKP$^+$13b] might wonder whether replacing single-use garbled circuits in their construction with reusable garbled circuits (also from [GKP$^+$13b]) might remove this limitation. We remark that this does not seem possible, essentially because the construction in [GKP$^+$13b] relies crucially on the simplicity of computing garbled inputs from the "garbling key". In particular, in Yao's garbled circuit scheme, the garbling key is (many) pairs of "strings" L_0 and L_1, and a garbling of an input bit b is simply L_b. This fits

Our first key idea is to embed the FHE secret key as part of the attributes in the ABE ciphertext. That is, in order to encrypt a plaintext m with attributes a in the predicate encryption scheme, we first choose a symmetric key $\mathsf{fhe.sk}$ for the FHE scheme, encrypt a into a FHE ciphertext $\hat{a}$, and encrypt m using the ABE scheme with $(\mathsf{fhe.sk}, \hat{a})$ as the attributes to obtain an ABE ciphertext ct. Our predicate encryption ciphertext is then given by

$$(\hat{a}, \mathsf{ct})$$

To generate the predicate secret key for a predicate P, one simply generates the ABE secret key for the function g that takes as input $(\mathsf{fhe.sk}, \hat{a})$ and computes

$$g(\mathsf{fhe.sk}, \hat{a}) = \mathsf{FHE.Dec}(\mathsf{fhe.sk}; \mathsf{FHE.Eval}(P, \hat{a}))$$

That is, g first homomorphically computes a FHE encryption of $P(a)$, and then decrypts it using the FHE secret key to output $P(a)$.

At first glance, this idea evokes strong and conflicting emotions as it raises two problems. The first pertains to correctness: we can no longer decrypt the ciphertext since the ABE decryption algorithm needs to know all of the attributes ($\hat{a}$ and $\mathsf{fhe.sk}$), but $\mathsf{fhe.sk}$ is missing. The second pertains to security: the ABE ciphertext ct is not guaranteed to protect the privacy of the attributes, and could leak all of $\mathsf{fhe.sk}$ which together with $\hat{a}$ would leak all of a. Solving both of these problems seems to require designing a predicate encryption scheme from scratch!

Our next key observation is that the bulk of the computation in g, namely the homomorphic evaluation of the predicate P, is performed on the *public* attribute $\hat{a}$. The only computation performed on the secret value $\mathsf{fhe.sk}$ is FHE decryption which is a fairly lightweight computation. In particular, with all known FHE schemes [Gen09, BV11b, BV11a, BGV12, GSW13, BV14, AP14], decryption corresponds to computing an inner product followed by a threshold function. Furthermore, we do know how to construct lattice-based predicate encryption schemes for threshold of inner product [AFV11, GMW15]. We stress that the latter do not correspond to FHE decryption since the inner product is computed over a vector in the ciphertext and one in the key, whereas FHE decryption requires computing an inner product over two vectors in the ciphertext; nonetheless, we will build upon the proof techniques in achieving attribute-hiding in [AFV11, GMW15] in the proof of security.

In other words, if we could enhance ABE with a modicum of secrecy so that it can perform a heavyweight computation on public attributes followed by a lightweight privacy-preserving computation on *secret* attributes, we are back in business. Our first contribution is to define such an object, that we call *partially hiding predicate encryption.*

perfectly with the semantics of ABE (rather, a variant termed two-input ABE in [GKP$^+$13b]) that releases one of two possible "messages" L_0 or L_1 depending on the outcome of a computation. In contrast, computing a garbled input in the reusable garbling scheme is a more complex and randomized function of the garbling key, and does not seem to align well with the semantics of ABE.

Partially Hiding Predicate Encryption. We introduce the notion of partially hiding predicate encryption (PHPE), an object that interpolates between attribute-based encryption and predicate encryption (analogously to partial garbling in [IW14]). In PHPE, the ciphertext, encrypting message m, is associated with an attribute (x, y) where x is private but y is always public. The secret key is associated with a function f, and decryption succeeds iff $f(x, y) = 1$. On the one extreme, considering a dummy x or functions f that ignore x and compute on y, we recover attribute-based encryption. On the other end, considering a dummy y or functions f that ignore y and compute on x, we recover predicate encryption.

We will be interested in realizing PHPE for functions ϕ of the form $\phi(x, y) = g(x, h(y))$ for some functions g and h where h may perform arbitrary heavy-weight computation on the public y and g only performs light-weight computation on the private x. Mapping back to our discussion, we would like to achieve PHPE for the "evaluate-then-decrypt" class of functions, namely where g is the FHE decryption function, h is the FHE evaluation function, x is the FHE secret key, and y is the FHE ciphertext. In general, we would like g to be simple and will allow h to be complex. It turns out that we can formalize the observation above, namely that PHPE for this class of functions gives us a predicate encryption scheme. The question now becomes: can we construct PHPE schemes for the "evaluate-then-decrypt" class of functions?

Assuming the subexponential hardness of learning with errors (LWE), we show how to construct a partially hiding predicate encryption for the class of functions $f : \mathbb{Z}_q^t \times \{0,1\}^\ell \to \{0,1\}$ of the form

$$f_\gamma(\mathbf{x}, \mathbf{y}) = \mathsf{IP}_\gamma(\mathbf{x}, h(\mathbf{y})),$$

where $h : \{0,1\}^\ell \to \{0,1\}^t$, $\gamma \in \mathbb{Z}_q$, and $\mathsf{IP}_\gamma(\mathbf{x}, \mathbf{z}) = 1$ iff $\langle \mathbf{x}, \mathbf{z} \rangle = \left(\sum_{i \in [t]} \mathbf{x}[i] \cdot \mathbf{z}[i] \right) = \gamma \mod q$.

This is almost what we want, but not quite. Recall that FHE decryption in many recent schemes [BV11b, BGV12, GSW13, BV14, AP14] is a function that checks whether an inner product of two vectors in $\mathbb{Z}_q^t$ (one of which could be over $\{0,1\}^t$) lies in a certain range. Indeed, if $\mathbf{z} \in \{0,1\}^t$ is an encryption of 1 and $\mathbf{x} \in \mathbb{Z}_q^t$ is the secret key, we know that $\langle \mathbf{x}, \mathbf{z} \rangle \in [q/2 - B, q/2 + B] \pmod q$, where B is the noise range. Applying the so-called "modulus reduction" [BV11b] transformation to all these schemes, we can assume that this range is polynomial in size.

In other words, we will manage to construct a partially hiding PE scheme for the function

$$f_\gamma(\mathbf{x}, \mathbf{y}) : \langle \mathbf{x}, h(\mathbf{y}) \rangle \stackrel{?}{=} \gamma \pmod q$$

whereas we need a partially hiding PE scheme for the FHE decryption function which is

$$f'_R(\mathbf{x}, \mathbf{y}) : \langle \mathbf{x}, h(\mathbf{y}) \rangle \overset{?}{\in} R \pmod q$$

where R is the polynomial size range $[q/2 - B, q/2 + B]$ from above. How do we reconcile this disparity?

The "Lazy OR" Trick. The solution, called the "lazy OR trick" [SBC+07, GMW15] is to publish secret keys for all functions f_γ for $\gamma \in R := [q/2-B, q/2+B]$. This will indeed allow us to test if the FHE decryption of the evaluated ciphertext is 1 (and reveal the message m if it is), but it is also worrying. Publishing these predicate secret keys for the predicates f_γ reveals more information than whether $\langle \mathbf{x}, h(\mathbf{y})\rangle \overset{?}{\in} R$. In particular, it reveals what $\langle \mathbf{x}, h(\mathbf{y})\rangle$ is. This means that an authorized key would leak partial information about the attribute, which we do allow for predicate encryption. On the other hand, for an unauthorized key where the FHE decryption is 0, each of these $f_\gamma, \gamma \in R$ is also an unauthorized key in the PHPE and therefore leaks no information about the attribute. This extends to the collection of keys in R since the PHPE is secure against collusions. For simplicity, we assume in the rest of this overview that FHE decryption corresponds to exactly to inner product.

Asymmetry to the Rescue: Constructing Partially Hiding PE. Our final contribution is the construction of a partially hiding PE for the function class $f_\gamma(\mathbf{x}, \mathbf{y})$ above. We will crucially exploit the fact that f_γ computes an inner product on the private attribute $\mathbf{x}$. There are two challenges here: first, we need to design a decryption algorithm that knows f_γ and $\mathbf{y}$ but not $\mathbf{x}$ (this is different from decryption in ABE where the algorithm also knows $\mathbf{x}$); second, show that the ciphertext does not leak too much information about $\mathbf{x}$. We use the fully key-homomorphic encryption techniques developed by Boneh et al [BGG+14] in the context of constructing an "arithmetic" ABE scheme. The crucial observation about the ABE scheme of [BGG+14] is that while it was not designed to hide the attributes, it can be made to partially hide them in exactly the way we want. In particular, the scheme allows us to carry out an inner product of a public attribute vector (corresponding to the evaluated FHE ciphertext) and a private attribute vector (corresponding to the FHE secret key $\mathsf{fhe.sk}$), thanks to an inherent asymmetry in homomorphic evaluation of a multiplication gate on ABE ciphertexts. More concretely, in the homomorphic evaluation of a ciphertext for a multiplication gate in [BGG+14], the decryption algorithm works even if one of the attribute remains private, and for addition gates, the decryption algorithms works even if both attributes remain private. This addresses the first challenge of a decryption algorithm that is oblivious to $\mathbf{x}$. For the second challenge of security, we rely on techniques from inner product predicate encryption [AFV11] to prove the privacy of $\mathbf{x}$ Note that in the latter, the inner product is computed over a vector in the ciphertext and one in the key, whereas in our scheme, the inner product is computed over two vectors in the ciphertext. Interestingly, the proof still goes through since the ciphertext in the ABE [BGG+14] has the same structure as the ciphertext in [AFV11]. We refer the reader to Section 4.3.2 for a detailed overview of the partial hiding PE, and to Section 4.6 for an overview of how we combine the partial hiding PE with FHE to obtain our main result.

Finally, we remark that exploiting asymmetry in multiplication has been used in fairly different contexts in both FHE [GSW13, BV14] and in ABE [GVW13, GV14]. In [GSW13] and in this work, the use of asymmetry was crucial for realizing the underlying cryptographic primitive; whereas in [GVW13, BV14, GV14], asymmetry was used to reduce the noise growth during homomorphic evaluation, thereby leading to quantitative improvements in the underlying assumptions and hence improved efficiency.

4.1.2 Discussion

Comparison with other approaches. The two main alternative approaches for realizing predicate and functional encryption both rely on multi-linear maps either implicitly, or explicitly. The first is to use indistinguishability obfuscation as in [GGH$^+$13b], and the second is to extend the dual system encryption framework to multi-linear maps [Wat09, GGHZ14]. A crucial theoretical limitation of these approaches is that they all rely on non-standard assumptions; we have few candidates for multi-linear maps [GGH13a, CLT13, GGH15] and the corresponding assumptions are presently poorly understood and not extensively studied in cryptanalysis, and in some cases, broken [CLT14]. In particular, the latest attack in [CLT14] highlight the importance of obtaining constructions and developing techniques that work under standard cryptographic assumptions, as is the focus of this work.

Barriers to functional encryption from LWE. We note the two main barriers to achieving full-fledged functional encryption from LWE using our framework. First, the lazy conjunction approach to handle threshold inner product for FHE decryption leaks the exact inner product and therefore cannot be used to achieve full attribute-hiding. Second, we do not currently know of a fully attribute-hiding inner product encryption scheme under the LWE assumption, although we do know how to obtain such schemes under standard assumptions in bilinear groups [OT12a, KSW08].

4.1.3 Chapter Organization

In Section 5.2 we provide preliminaries needed for our construction, such as fully-homomorphic encryption. In Section 4.3 we present syntax of partially-hiding predicate encryption and show how to instantiate it from lattices in Sections 4.4, 4.5. In Section 4.6, we present a construction of predicate encryption for circuits from partially-hiding predicate encryption and fully-homomorphic encryption. In Section 4.7, we summarize and provide some open problems.

4.2 Preliminaries

4.2.1 Predicate Encryption

We present the definitions of predicate encryption for general circuits [BW07, KSW08, AFV11]. A predicate encryption scheme $\mathcal{PE}$ with respect to an attribute universe $\mathcal{A}$, predicate universe $\mathcal{C}$ and a message universe $\mathcal{M}$ consists of four algorithms ($\mathsf{Setup}, \mathsf{Enc}, \mathsf{KeyGen}, \mathsf{Dec}$):

$\mathsf{Setup}(1^\lambda, 1^k, d) \to (\mathsf{mpk}, \mathsf{msk})$. The setup algorithm gets as input the security parameter λ, the length of the attributes k and the maximum depth d and outputs the public parameter mpk, and the master key msk.

$\mathsf{Enc}(\mathsf{mpk}, a, m) \to \mathsf{ct}$. The encryption algorithm gets as input mpk, an attribute $a \in \mathcal{A}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext ct.

$\mathsf{KeyGen}(\mathsf{msk}, C) \to \mathsf{sk}_C$. The key generation algorithm gets as input msk and a predicate $C \in \mathcal{C}$. It outputs a secret key sk_C.

$\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) \to m$. The decryption algorithm gets as input sk_C and a ciphertext ct. It outputs a message m.

Correctness. We require that for all $(a, C) \in \mathcal{A} \times \mathcal{C}$ such that $C(a) = 1$ and all $m \in \mathcal{M}$,

$$\Pr\left[\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C), \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, a, m); \mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = m\right] \geq 1 - \mathrm{negl}(\lambda)$$

And for all $(a, C) \in \mathcal{A} \times \mathcal{C}$ such that $C(a) = 0$ and all $m \in \mathcal{M}$,

$$\Pr\left[\mathsf{sk}_C \leftarrow \mathsf{KeyGen}(\mathsf{msk}, C), \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, a, m); \mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}) = \perp\right] \geq 1 - \mathrm{negl}(\lambda)$$

where the probability is taken over $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, d)$ and the coins of $\mathsf{Enc}, \mathsf{KeyGen}$.

4.2.2 Security Model

Definition 4.2.1 (SIM-AH)**.** *For every stateful p.p.t. adversary* Adv*, and a p.p.t. simulator* Sim*, consider the following two experiments:*

```
$\mathrm{exp}^{\mathsf{real}}_{\mathcal{PE},\mathrm{Adv}}(1^\lambda)$:

1: $a \leftarrow \mathrm{Adv}(1^\lambda)$
2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, d)$
3: $m \leftarrow \mathrm{Adv}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$
4: $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, a, m)$
5: $\alpha \leftarrow \mathrm{Adv}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct})$
6: Output $(a, m, \alpha)$
```

```
$\mathrm{exp}^{\mathsf{ideal}}_{\mathcal{PE},\mathrm{Sim}}(1^\lambda)$:

1: $a \leftarrow \mathrm{Adv}(1^\lambda)$
2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, d)$
3: $m \leftarrow \mathrm{Adv}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$
4: $\mathsf{ct} \leftarrow \mathrm{Sim}(\mathsf{mpk}, 1^{|a|}, 1^{|m|})$
5: $\alpha \leftarrow \mathrm{Adv}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct})$
6: Output $(a, m, \alpha)$
```

We say an adversary Adv *is admissible if for all oracle queries that it makes $C \in \mathcal{C}$, $C(a) = 0$. The Predicate Encryption scheme $\mathcal{PE}$ is then said to be simulation-based* attribute-hiding

(SIM-AH) *if there is a p.p.t. simulator* Sim *such that for every stateful p.p.t. adversary* Adv*, the following two distributions are computationally indistinguishable:*

$$\left\{ \exp^{\mathsf{real}}_{\mathcal{PE},\mathrm{Adv}}(1^\lambda) \right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{ \exp^{\mathsf{ideal}}_{\mathcal{PE},\mathrm{Sim}}(1^\lambda) \right\}_{\lambda\in\mathbb{N}}$$

4.2.3 Relations to Other Security Models

Single Message Implies Many Message Security. We point out a simple composition result for our definition. The definition for many message security remains virtually identical, except the adversary declares a list of tuples (a_i, m_i) for which it sees either real or simulated ciphertexts. Given a one-message simulator Sim_1, we can construct a many message simulator Sim_m which just invokes the one-message simulator to simulate each ciphertext independently. The security follows via the standard hybrid argument and crucially relies on the fact that the adversary is restricted to queries that *do not allow* to decrypt. Similar result follows for partially hiding predicate encryption.

Impossibility of Strong-Simulation Security We point out the many-messages strong-simulation security is impossible to realize. In the strong-simulation security (or strong attribute hiding (SAH)), the adversary is *allowed* to query for secret keys that allow to decrypt. The simulator is given the result oracle access to functionality that returns the outputs of the predicates.

Definition 4.2.2 (Many-Messages SIM-SAH)**.** *For every stateful p.p.t. adversary* Adv*, and a p.p.t. simulator* Sim*, consider the following two experiments:*

```
$\exp^{\mathsf{real}}_{\mathcal{PE},\mathrm{Adv}}(1^\lambda)$:
1: $\vec{a} \leftarrow \mathrm{Adv}(1^\lambda)$
2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, d)$
3: $\vec{m} \leftarrow \mathrm{Adv}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$
4: $\vec{\mathsf{ct}} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \vec{a}, \vec{m})$
5: $\alpha \leftarrow \mathrm{Adv}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\vec{\mathsf{ct}})$
6: Output $(\vec{a}, \vec{m}, \alpha)$
```

```
$\exp^{\mathsf{ideal}}_{\mathcal{PE},\mathrm{Sim}}(1^\lambda)$:
1: $\vec{a} \leftarrow \mathrm{Adv}(1^\lambda)$
2: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, d)$
3: $\vec{m} \leftarrow \mathrm{Adv}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk})$
4: $\vec{\mathsf{ct}} \leftarrow \mathrm{Sim}^{O(\cdot)}(\mathsf{mpk}, 1^{|a_i|}, 1^{|m_i|})$
5: $\alpha \leftarrow \mathrm{Adv}^{O'(\mathsf{msk},\cdot)}(\vec{\mathsf{ct}})$
6: Output $(\vec{a}, \vec{m}, \alpha)$
```

Where the oracles are defines as:

- $O(C)$*: returns a list* $(C(a_i), b_i)$ *(for all* $i \in |\vec{a}|$*) where* $b_i = m_i$ *if* $C(a_i) = 1$ *and* $b_i = \perp$ *otherwise. In words, it returns the results of the decryption queries that the adversary should learn by the correctness of the scheme.*

- $O'(\mathsf{msk}, C)$*: is the second stage of the simulator, namely* $\mathrm{Sim}^{O(\cdot)}(\mathsf{msk})$*. It is given access to the same oracle that returns results of the decryption queries that must be satisfied by the correctness.*

We say the simulator is admissible if it queries its oracle $O(\cdot)$ on the identical ordered set of queries issued by Adv*. The Predicate Encryption scheme $\mathcal{PE}$ is then said to be simulation-based* strong attribute-hiding *for many-messages if there is an admissible stateful p.p.t. simulator* Sim *such that for every admissible stateful p.p.t. adversary* Adv*, the following two distributions are computationally indistinguishable:*

$$\left\{ \exp^{\mathsf{real}}_{\mathcal{PE},\mathrm{Adv}}(1^\lambda) \right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{ \exp^{\mathsf{ideal}}_{\mathcal{PE},\mathrm{Sim}}(1^\lambda) \right\}_{\lambda\in\mathbb{N}}$$

The impossibility of this notion follows from the compression arguments of [BSW11, AGVW13]. In particular, consider an adversary that outputs a random list of messages $\overrightarrow{m}$ and arbitrary identities $\overrightarrow{a}$. It sets the lengths of these lists to be much greater than the length of secret keys of the scheme. Then, upon receiving the challenge ciphertext, it asks for a query that allows to decrypt all messages. The simulator, on the other side, must first commit to a long string $\overrightarrow{\mathsf{ct}}$ and then later fake a short secret key that must decrypt the entire ciphertext correctly, which is impossible by the standard information theoretic argument.

4.2.4 Indistinguishability Security of PE

For comparison, we include here the indistinguishability-based formulation of selective, weakly attribute-hiding predicate encryption.

Definition 4.2.3 (IND-AH)**.** *For a stateful adversary $\mathcal{A}$, we define the advantage function*

$$\mathsf{Adv}^{\mathrm{PE}}_{\mathcal{A}}(\lambda) := \Pr\left[\beta = \beta' : \begin{array}{l} (x_0, x_1) \leftarrow \mathcal{A}(1^\lambda); \\ \beta \xleftarrow{\$} \{0,1\}; \\ (\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^k, d); \\ (m_0, m_1) \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{mpk}); \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x_\beta, m_\beta); \\ \beta' \leftarrow \mathcal{A}^{\mathsf{KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}) \end{array}\right] - \frac{1}{2}$$

with the restriction that all queries C that $\mathcal{A}$ makes to $\mathsf{KeyGen}(\mathsf{msk}, \cdot)$ *satisfies $C(x_0) = C(x_1) = 0$ (that is, sk_C does not decrypt* ct*). A predicate encryption scheme is* selectively secure *if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathrm{PE}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.*

4.2.5 Fully-Homomorphic Encryption

We present a fairly minimal definition of fully homomorphic encryption (FHE) which is sufficient for our constructions. A leveled homomorphic encryption scheme is a tuple of polynomial-time algorithms $(\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Eval}, \mathsf{HE.Dec})$:

- **Key generation.** $\mathsf{HE.KeyGen}(1^\lambda, 1^d, 1^k)$ is a probablistic algorithm that takes as input the security parameter λ, a depth bound d and message length k and outputs a secret key sk.

- **Encryption.** $\mathsf{HE.Enc}(\mathsf{sk}, m)$ is a probabilistic algorithm that takes as input sk and a message $m \in \{0,1\}^k$ and outputs a ciphertext ct.

- **Homomorphic evaluation.** $\mathsf{HE.Eval}(f, \mathsf{ct})$ is a deterministic algorithm that takes as input a boolean circuit $C : \{0,1\}^k \to \{0,1\}$ of depth at most d and a ciphertext ct and outputs another ciphertext ct'.

- **Decryption.** $\mathsf{HE.Dec}(\mathsf{sk}, \mathsf{ct}')$ is a deterministic algorithm that takes as input sk and ciphertext ct' and outputs a bit.

Correctness. We require perfect decryption correctness with respect to homomorphically evaluated ciphertexts: namely for all λ, d, k and all $\mathsf{sk} \leftarrow \mathsf{HE.KeyGen}(1^\lambda, 1^d, 1^k)$, all $m \in \{0,1\}^k$ and for all boolean circuits $C : \{0,1\}^k \to \{0,1\}$ of depth at most d:

$$\Pr\Big[\mathsf{HE.Dec}(\mathsf{sk},\ \mathsf{HE.Eval}(C,\ \mathsf{HE.Enc}(\mathsf{sk}, m))) = C(m)\Big] = 1$$

where the probablity is taken over $\mathsf{HE.Enc}$ and $\mathsf{HE.KeyGen}$.

Security. We require semantic security for a single ciphertext: namely for every stateful p.p.t. adversary $\mathcal{A}$ and for all $d, k = \mathrm{poly}(\lambda)$, the following quantity

$$\Pr\left[b = b' : \begin{array}{l} \mathsf{sk} \leftarrow \mathsf{Setup}(1^\lambda, 1^d, 1^k); \\ (m_0, m_1) \leftarrow \mathcal{A}(1^\lambda, 1^d, 1^k); \\ b \xleftarrow{\$} \{0,1\}; \\ \mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{sk}, m_b); \\ b' \leftarrow \mathcal{A}(\mathsf{ct}) \end{array} \right] - \frac{1}{2}$$

is negligible in λ.

4.2.6 FHE from LWE

We will rely on an instantiation of FHE from the LWE assumption:

Theorem 4.2.1 (FHE from LWE [BV11b, BGV12, GSW13, BV14, AP14])**.** *There is a FHE scheme* $\mathsf{HE.KeyGen}, \mathsf{HE.Enc}, \mathsf{HE.Eval}, \mathsf{HE.Dec}$ *that works for any q with $q \geq O(\lambda^2)$ with the following properties:*

- $\mathsf{HE.KeyGen}$ *outputs a secret key* $\mathsf{sk} \in \mathbb{Z}_q^t$ *where $t = \mathrm{poly}(\lambda)$;*

- $\mathsf{HE.Enc}$ *outputs a ciphertext* $\mathsf{ct} \in \{0,1\}^\ell$ *where $\ell = \mathrm{poly}(k, d, \lambda, \log q)$;*

- $\mathsf{HE.Eval}$ *outputs a ciphertext* $\mathsf{ct}' \in \{0,1\}^t$*;*

- *for any boolean circuit of depth* d*,* $\mathsf{HE.Eval}(C,\cdot)$ *is computed by a boolean circuit of depth* $\mathrm{poly}(d,\lambda,\log q)$*.*

- $\mathsf{HE.Dec}$ *on input* sk,ct' *outputs a bit* $b \in \{0,1\}$*. If* ct' *is an encryption of* 1 *then*

$$\sum_{i=1}^{t} \mathsf{sk}[i]\cdot \mathsf{ct}'[i] \in [\lfloor q/2 \rfloor - B, \lfloor q/2 \rfloor + B]$$

  for some fixed $B = \mathrm{poly}(\lambda)$*. Otherwise, if* ct' *is an encryption of* 0*, then*

$$\sum_{i=1}^{t} \mathsf{sk}[i]\cdot \mathsf{ct}'[i] \notin [\lfloor q/2 \rfloor - B, \lfloor q/2 \rfloor + B];$$

- *security relies on* $\mathsf{dLWE}_{\Theta(t),q,\chi}$*.*

We highlight several properties of the above scheme: (1) the ciphertext is a bit-string, (2) the bound B is a polynomial independent of q (here, we crucially exploit the new results in [BV14] together with the use of leveled bootstrapping)[3], (3) the size of normal ciphertexts is independent of the size of the circuit (this is the typical compactness requirement).

4.3 Partially Hiding Predicate Encryption

4.3.1 Definitions

We introduce the notation of partially hiding predicate encryption (PHPE), which interpolates attribute-based encryption and predicate encryption (analogously to partial garbling in [IW14]). In PHPE, the ciphertext, encrypting message m, is associated with an attribute (x,y) where x is private but y is always public. The secret key is associated with a predicate C, and decryption succeeds iff $C(x,y)=1$. The requirement is that a collusion learns nothing about (x,m) if none of them is individually authorized to decrypt the ciphertext. Attribute-based encryption corresponds to the setting where x is empty, and predicate encryption corresponds to the setting where y is empty. We refer the reader to Section 4.2.1 for the standard notion of predicate encryption.

Looking ahead to our construction, we show how to:

- construct PHPE for a restricted class of circuits that is "low complexity" with respect to x and allows arbitrarily polynomial-time computation on y;

- bootstrap this PHPE using FHE to obtain PE for all circuits.

[3]Recall that no circular security assumption needs to be made for leveled bootstrapping.

Syntax. A Partially-Hiding Predicate Encryption scheme $\mathcal{PHPE}$ for a pair of input-universes $\mathcal{X}, \mathcal{Y}$, a predicate universe $\mathcal{C}$, a message space $\mathcal{M}$, consists of four algorithms $(\mathsf{PH.Setup}, \mathsf{PH.Enc}, \mathsf{PH.KeyGen}, \mathsf{PH.Dec})$:

$\mathsf{PH.Setup}(1^\lambda, 1^t, 1^\ell, d) \to (\mathsf{ph.mpk}, \mathsf{ph.msk})$. The setup algorithm gets as input the security parameter λ, the lengths of private/public attributes t, ℓ and the maximum circuit depth d, and outputs the public parameter $\mathsf{ph.mpk}$, and the master key $\mathsf{ph.msk}$.

$\mathsf{PH.Enc}(\mathsf{ph.mpk}, (x, y), m) \to \mathsf{ct}_y$. The encryption algorithm gets as input $\mathsf{ph.mpk}$, an attribute $(x, y) \in \mathcal{X} \times \mathcal{Y}$ and a message $m \in \mathcal{M}$. It outputs a ciphertext ct_y.

$\mathsf{PH.KeyGen}(\mathsf{ph.msk}, C) \to \mathsf{sk}_C$. The key generation algorithm gets as input $\mathsf{ph.msk}$ and a predicate $C \in \mathcal{C}$. It outputs a secret key sk_C.

$\mathsf{PH.Dec}(\mathsf{sk}_C, (\mathsf{ct}_y, y)) \to m$. The decryption algorithm gets as input the secret key sk_C, and a ciphertext ct_y and the public part of the attribute y. It outputs a message $m \in \mathcal{M}$ or $\perp$.

Correctness. We require that for all $\mathsf{PH.Setup}(1^\lambda, 1^t, 1^\ell, d) \to (\mathsf{ph.mpk}, \mathsf{ph.msk})$, for all $(x, y, C) \in \mathcal{X} \times \mathcal{Y} \times \mathcal{C}$, for all $m \in \mathcal{M}$,

- if $C(x, y) = 1$,
$$\Pr\Big[\mathsf{PH.Dec}(\mathsf{sk}_C, (\mathsf{ct}_y, y)) = m\Big] \geq 1 - \text{negl}(\lambda),$$
- if $C(x, y) = 0$,
$$\Pr\Big[\mathsf{PH.Dec}(\mathsf{sk}_C, (\mathsf{ct}_y, y)) = \perp\Big] \geq 1 - \text{negl}(\lambda),$$

where the probabilities are taken over $\mathsf{sk}_C \leftarrow \mathsf{PH.KeyGen}(\mathsf{ph.msk}, C)$, $\mathsf{ct}_y \leftarrow \mathsf{PH.Enc}(\mathsf{ph.mpk}, (x, y), m)$ and coins of $\mathsf{PH.Setup}$.

Definition 4.3.1 (PHPE Attribute-Hiding)**.** *Fix* $(\mathsf{PH.Setup}, \mathsf{PH.Enc}, \mathsf{PH.KeyGen}, \mathsf{PH.Dec})$*. For every stateful p.p.t. adversary* Adv*, and a p.p.t. simulator* Sim*, consider the following two experiments:*

```
$\underline{\text{exp}^{\mathsf{real}}_{\mathcal{PHPE},\text{Adv}}(1^\lambda)\text{:}}$
1: $(x, y) \leftarrow \text{Adv}(1^\lambda, 1^t, 1^\ell, d)$
2: $(\mathsf{ph.mpk}, \mathsf{ph.msk}) \leftarrow$
   $\mathsf{PH.Setup}(1^\lambda, 1^t, 1^\ell, d)$
3: $m \leftarrow \text{Adv}^{\mathsf{PH.KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ph.mpk})$
4: $\mathsf{ct}_y \leftarrow \mathsf{PH.Enc}(\mathsf{ph.mpk}, (x, y), m)$
5: $\alpha \leftarrow \text{Adv}^{\mathsf{PH.KeyGen}(\mathsf{ph.msk},\cdot)}(\mathsf{ct}_y)$
6: Output $(x, y, m, \alpha)$
```

```
$\underline{\text{exp}^{\mathsf{ideal}}_{\mathcal{PHPE},\text{Sim}}(1^\lambda)\text{:}}$
1: $(x, y) \leftarrow \text{Adv}(1^\lambda, 1^t, 1^\ell, d)$
2: $(\mathsf{ph.mpk}, \mathsf{ph.msk}) \leftarrow$
   $\mathsf{PH.Setup}(1^\lambda, 1^t, 1^\ell, d)$
3: $m \leftarrow \text{Adv}^{\mathsf{PH.KeyGen}(\mathsf{ph.msk},\cdot)}(\mathsf{ph.mpk})$
4: $\mathsf{ct}_y \leftarrow \text{Sim}(\mathsf{mpk}, y, 1^{|x|}, 1^{|m|})$
5: $\alpha \leftarrow \text{Adv}^{\mathsf{PH.KeyGen}(\mathsf{msk},\cdot)}(\mathsf{ct}_y)$
6: Output $(x, y, m, \alpha)$
```

We say an adversary Adv *is admissible if all oracle queries that it makes* $C \in \mathcal{C}$ *satisfy* $C(x, y) = 0$. *The Partially-Hiding Predicate Encryption scheme* $\mathcal{PHPE}$ *is then said to be* attribute-hiding *if there is a p.p.t. simulator* Sim *such that for every stateful p.p.t. adversary* Adv*, the following two distributions are computationally indistinguishable:*

$$\left\{ \exp^{\mathsf{real}}_{\mathcal{PHPE},\mathrm{Adv}}(1^\lambda) \right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{ \exp^{\mathsf{ideal}}_{\mathcal{PHPE},\mathrm{Sim}}(1^\lambda) \right\}_{\lambda\in\mathbb{N}}$$

Remarks. We point out some remarks of our definition (SIM-AH) and how it compares to other definitions in the literature.

- We note the simulator for the challenge ciphertext gets y but not x; this captures the fact that y is public whereas x is private. In addition, the simulator is not allowed to program the public parameters or the secret keys. In the ideal experiment, the simulator does not explicitly learn any information about x (apart from its length); nonetheless, there is implicit leakage about x from the key queries made by an admissible adversary. Finally, we note that we can efficiently check whether an adversary is admissible.

- Our security notion is "selective", in that the adversary "commits" to (x, y) before it sees $\mathsf{ph.mpk}$. It is possible to bootstrap selectively-secure scheme to full security using standard complexity leveraging arguments [BB04, GVW13], at the price of a $2^{|x|}$ loss in the security reduction.

- Our definition refers to a single challenge message, but the definition extends readily to a setting with multiple challenge messages. Moreover, our definition composes in that security for a single message implies security with multiple messages (see Section 4.2.3). The following remarks refer to many messages setting.

- We distinguish between two notions of indistinguishability-based (IND) definitions used in the literature: attribute-hiding (IND-AH)[4] and strong attribute-hiding (IND-SAH)[5] [BW07, SBC+07, KSW08, AFV11]. In the IND-AH, the adversary should not be able to distinguish between two pairs of attributes/messages given that it is restricted to queries which do not decrypt the challenge ciphertext (See Section 4.2.4). It is easy to see that our SIM-AH definition is stronger than IND-AH. Furthermore, IND-SAH also ensures that adversary cannot distinguish between the attributes even when it is allowed to ask for queries that decrypt the messages (in this case, it must output $m_0 = m_1$). Our SIM-AH definition is weaker than IND-SAH, since we explicitly restrict the adversary to queries that do not decrypt the challenge ciphertext.

- In the context of arbitrary predicates, *strong* variants of definitions (that is, IND-SAH and SIM-SAH) are equivalent to security notions for functional encryption (the

[4]Sometimes also referred as weak attribute-hiding.
[5]Sometimes also referred as full attribute-hiding.

simulation definition must be adjusted to give the simulated the outputs of the queries). However, the strong variant of notion (SIM-SAH) is impossible to realize for many messages [BSW11, AGVW13]. We refer the reader to Section 4.2.3 for a sketch of the impossibility. Hence, SIM-AH is the best-possible simulation security for predicate encryption which we realize in this work. The only problem which we leave open is to realize IND-SAH from standard LWE.

4.3.2 Our Construction

We refer the reader to Sections 4.4 and 4.5 for the complete description of our construction. Below, we provide an overview.

Overview. We construct a partially hiding predicate encryption for the class of predicate circuits $C : \mathbb{Z}_q^t \times \{0,1\}^\ell \to \{0,1\}$ of the form $\widehat{C} \circ \mathsf{IP}_\gamma$ where $\widehat{C} : \{0,1\}^\ell \to \{0,1\}^t$ is a boolean circuit of depth d, $\gamma \in \mathbb{Z}_q$, and

$$(\widehat{C} \circ \mathsf{IP}_\gamma)(\mathbf{x}, \mathbf{y}) = \mathsf{IP}_\gamma(\mathbf{x}, \widehat{C}(\mathbf{y})),$$

where $\mathsf{IP}_\gamma(\mathbf{x}, \mathbf{z}) = 1$ iff $\langle \mathbf{x}, \mathbf{z} \rangle = \left(\sum_{i\in[t]} \mathbf{x}[i] \cdot \mathbf{z}[i] \right) = \gamma \mod q$. We refer to circuit IP as the generic inner-product circuit of two vectors.

Looking ahead, $\widehat{C}$ corresponds to FHE evaluation of an arbitrary circuit C, whereas IP_γ corresponds to roughly to FHE decryption; in the language of the introduction in Section 4, $\widehat{C}$ corresponds to heavy-weight computation h, whereas IP_γ corresponds to light-weight computation g.

The scheme. The public parameters are matrices

$$\left(\mathbf{A},\ \mathbf{A}_1, \ldots, \mathbf{A}_\ell,\ \mathbf{B}_1, \ldots, \mathbf{B}_t\right)$$

An encryption corresponding to the attribute $(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}_q^t \times \{0,1\}^\ell$ is a GPV ciphertext (an LWE sample) corresponding to the matrix

$$\left[\,\mathbf{A} \mid \mathbf{A}_1 + \mathbf{y}[1]\cdot\mathbf{G} \mid \cdots \mid \mathbf{A}_\ell + \mathbf{y}[\ell]\cdot\mathbf{G} \mid \mathbf{B}_1 + \mathbf{x}[1]\cdot\mathbf{G} \mid \cdots \mid \mathbf{B}_t + \mathbf{x}[t]\cdot\mathbf{G}\,\right]$$

To decrypt the ciphertext given $\mathbf{y}$ and a key for $\widehat{C} \circ \mathsf{IP}_\gamma$, we apply the BGGHNSVV algorithm to first transform the first part of the ciphertext into a GPV ciphertext corresponding to the matrix

$$\left[\,\mathbf{A} \mid \mathbf{A}_{\widehat{C}_1} + \mathbf{z}[1]\cdot\mathbf{G} \mid \cdots \mid \mathbf{A}_{\widehat{C}_t} + \mathbf{z}[t]\cdot\mathbf{G}\right]$$

where $\widehat{C}_i$ is the circuit computing the i'th bit of $\widehat{C}$ and $\mathbf{z} = \widehat{C}(\mathbf{y}) \in \{0,1\}^t$. Next, observe that

$$-\Big((\mathbf{A}_{\widehat{C}_i} + \mathbf{z}[i]\cdot\mathbf{G})\cdot\mathbf{G}^{-1}(\mathbf{B}_i)\Big) + \mathbf{z}[i]\cdot\Big(\mathbf{B}_i + \mathbf{x}[i]\cdot\mathbf{G}\Big) = -\mathbf{A}_{\widehat{C}_i}\mathbf{G}^{-1}(\mathbf{B}_i) + \mathbf{x}[i]\cdot\mathbf{z}[i]\cdot\mathbf{G}.$$

Summing over i, we have

$$\sum_{i=1}^{\ell} -\Big((\mathbf{A}_{\widehat{C}_i} + \mathbf{z}[i]\cdot\mathbf{G})\cdot\mathbf{G}^{-1}(\mathbf{B}_i)\Big) + \mathbf{z}[i]\cdot\Big(\mathbf{B}_i + \mathbf{x}[i]\cdot\mathbf{G}\Big) = \mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \langle\mathbf{x},\mathbf{z}\rangle\cdot\mathbf{G}$$

where

$$\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} := -\Big(\mathbf{A}_{\widehat{C}_1}\mathbf{G}^{-1}(\mathbf{B}_1) + \cdots + \mathbf{A}_{\widehat{C}_t}\mathbf{G}^{-1}(\mathbf{B}_t)\Big).$$

Therefore, given only the public matrices and $\mathbf{y}$ (but not $\mathbf{x}$), we may transform the ciphertext into a GPV ciphertext corresponding to the matrix

$$\big[\,\mathbf{A} \mid \mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \langle\mathbf{x},\mathbf{z}\rangle\cdot\mathbf{G}\,\big].$$

The secret key corresponding to $\widehat{C} \circ \mathsf{IP}_\gamma$ is essentially a "short basis" for the matrix

$$\big[\,\mathbf{A} \mid \mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \gamma\cdot\mathbf{G}\,\big]$$

which can be sampled using a short trapdoor $\mathbf{T}$ of the matrix $\mathbf{A}$.

Proof strategy. There are two main components to the proof. Fix the selective challenge attribute $\mathbf{x},\mathbf{y}$. First, we will simulate the secret keys without knowing the trapdoor for the matrix $\mathbf{A}$: here, we rely on the simulated key generation for the ABE [BGG$^+$14]. Roughly speaking, we will need to generate a short basis for the matrix

$$\big[\,\mathbf{A} \mid \mathbf{A}\mathbf{R}_{\widehat{C}\circ\mathsf{IP}} + (\gamma - \widehat{C} \circ \mathsf{IP}(\mathbf{x},\mathbf{y}))\cdot\mathbf{G}\,\big]$$

where $\mathbf{R}_{\widehat{C}\circ\mathsf{IP}}$ is a small-norm matrix known to the simulator. Now, whenever $\widehat{C}\circ\mathsf{IP}(\mathbf{x},\mathbf{y}) \neq \gamma$ as is the case for admissible adversaries, we will be able to simultae secret keys using the puncturing techniques in [ABB10a, AFV11, MP12]

Next, we will show that the attribute $\mathbf{x}$ is hidden in the challenge ciphertext. Here, we adopt the proof strategy for attribute-hiding inner product encryption in [AFV11, GMW15]. In the proof, we simulate the matrices $\mathbf{A},\mathbf{B}_1,\ldots,\mathbf{B}_t$ using

$$\mathbf{A}, \mathbf{A}\mathbf{R}'_1 - \mathbf{x}[1]\mathbf{G}, \ldots, \mathbf{A}\mathbf{R}'_t - \mathbf{x}[t]\mathbf{G}$$

where $\mathbf{R}'_1,\ldots,\mathbf{R}'_t \overset{\$}{\leftarrow} \{\pm 1\}^{m\times m}$. In addition, we simulate the corresponding terms in the challenge ciphertext by:

$$\mathbf{c}, \mathbf{c}^{\mathsf{T}}\mathbf{R}'_1, \ldots, \mathbf{c}^{\mathsf{T}}\mathbf{R}'_t$$

where $\mathbf{c}$ is a uniformly random vector, which we switched from $\mathbf{A}^{\mathsf{T}}\mathbf{s} + \mathbf{e}$ using the LWE assumption. Here we crucially rely on the fact that switched to simulation of secret keys without knowing the trapdoor of $\mathbf{A}$. Going further, once $\mathbf{c}$ is random, we can switch back to simulating secret keys using the trapdoor $\mathbf{T}$. Hence, the secret keys now do not leak any information about $\mathbf{R}'_1,\ldots,\mathbf{R}'_t$. Therefore, we may then invoke the left-over hash lemma to argue that $\mathbf{x}$ is information-theoretically hidden.

4.4 Auxiliary evaluation algorithms

In order to formally describe our scheme, we first need to recall two algorithms $(\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}})$ from the BGGHNSVV14 ABE [BGG+14], which we may use as a "black box" and then extend to our setting. Given a boolean predicate $C : \{0,1\}^\ell \to \{0,1\}$ and $\mathbf{y} \in \{0,1\}^\ell$, the algorithm $\mathsf{Eval}_{\mathsf{ct}}$ transforms a GPV ciphertext for the matrix

$$\big[\, \mathbf{A}_1 + \mathbf{y}[1] \cdot \mathbf{G} \mid \cdots \mid \mathbf{A}_\ell + \mathbf{y}[\ell] \cdot \mathbf{G} \big]$$

into one for the matrix

$$\big[\, \mathbf{A}_C + C(\mathbf{y}) \cdot \mathbf{G} \big],$$

where the matrix $\mathbf{A}_C$ is deterministically derived from $(C, \mathbf{A}_1, \ldots, \mathbf{A}_\ell)$ via $\mathsf{Eval}_{\mathsf{pk}}$. We then extend $\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}$ to handle circuits $\widehat{C} \circ \mathsf{IP}$ as outlined above, with the additional property that $\mathsf{Eval}_{\mathsf{ct}}$ is oblivious to $\mathbf{x}$. We exploit the fact that for a multiplication gate, $\mathsf{Eval}_{\mathsf{ct}}$ works even if one of the attribute remains private, and for addition gates, $\mathsf{Eval}_{\mathsf{ct}}$ works even if both attributes remain private. Concretely, $\mathsf{Eval}_{\mathsf{ct}}$ transforms a GPV ciphertext for the matrix

$$\big[\, \mathbf{A}_1 + \mathbf{y}[1] \cdot \mathbf{G} \mid \cdots \mid \mathbf{A}_\ell + \mathbf{y}[\ell] \cdot \mathbf{G} \mid \mathbf{B}_1 + \mathbf{x}[1] \cdot \mathbf{G} \mid \cdots \mid \mathbf{B}_t + \mathbf{x}[t] \cdot \mathbf{G} \,\big]$$

into one for the matrix

$$\big[\, \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + (\widehat{C} \circ \mathsf{IP})(\mathbf{x}, \mathbf{y}) \cdot \mathbf{G} \big],$$

where the matrix $\mathbf{A}_{\widehat{C} \circ \mathsf{IP}}$ is deterministically derived from $(\widehat{C}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{B}_1, \ldots, \mathbf{B}_t)$ via $\mathsf{Eval}_{\mathsf{pk}}$, and where $\mathsf{Eval}_{\mathsf{ct}}$ gets $\mathbf{y}$ but not $\mathbf{x}$.

Two basic algorithms. The BGGHNSVV14 ABE provides two deterministic algorithms $\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}$ with the following properties:

- $\mathsf{Eval}_{\mathsf{pk}}$ takes as input ℓ matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n \times m}$ and a predicate $C : \{0,1\}^\ell \to \{0,1\}$, outputs a matrix $\mathbf{A}_C \in \mathbb{Z}_q^{n \times m}$;
- $\mathsf{Eval}_{\mathsf{ct}}$ takes as input $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$ and C as before, along with $\mathbf{y} \in \{0,1\}^\ell$ and ℓ vectors $\mathbf{u}_1, \ldots, \mathbf{u}_\ell \in \mathbb{Z}_q^m$, outputs a vector $\mathbf{u}_C \in \mathbb{Z}_q^m$.

The algorithms satisfy the following properties:

- if $(\mathbf{u}_1, \ldots, \mathbf{u}_\ell) \approx \big((\mathbf{A}_1 + \mathbf{y}[1] \cdot \mathbf{G})^\intercal \mathbf{s}, \ldots, (\mathbf{A}_\ell + \mathbf{y}[\ell] \cdot \mathbf{G})^\intercal \mathbf{s}\big)$, then $\mathbf{u}_C \approx (\mathbf{A}_C + C(\mathbf{y}) \cdot \mathbf{G})^\intercal \mathbf{s}$.
- if $(\mathbf{A}_1, \ldots, \mathbf{A}_\ell) = (\mathbf{A}\mathbf{R}_1 - \mathbf{y}[1] \cdot \mathbf{G}, \ldots, \mathbf{A}\mathbf{R}_\ell - \mathbf{y}[\ell] \cdot \mathbf{G})$ where $\mathbf{R}_1, \ldots, \mathbf{R}_\ell$ are small-norm matrices, then we have
$$\mathbf{A}_C = \mathbf{A}\mathbf{R}_C - C(\mathbf{y}) \cdot \mathbf{G}$$
where $\mathbf{R}_C$ is also a small-norm matrix with a roughly n^{2d} multiplicative blow-up.

These two properties are formalized quantitatively in the following lemma:

Lemma 4.4.1 (properties of $\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}$ [BGG+14])**.** *The algorithms* $\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}$ *satisfy the following properties. For all* $\mathbf{A}_1, \ldots, \mathbf{A}_\ell \in \mathbb{Z}_q^{n\times m}$*, all* $\mathbf{y} \in \{0,1\}^\ell$*, all boolean predicate* C *of depth* d*, let* $\mathbf{A}_C := \mathsf{Eval}_{\mathsf{pk}}(\mathbf{A}_1, \ldots, \mathbf{A}_\ell, C)$*. Then,*

- *for all* $\mathbf{u}_1, \ldots, \mathbf{u}_\ell \in \mathbb{Z}_q^m$ *and all* $\mathbf{s} \in \mathbb{Z}_q^n$*,*

$$\|\mathbf{u}_C - (\mathbf{A}_C + C(\mathbf{y}) \cdot \mathbf{G})^\intercal \mathbf{s}\|_\infty \leq O(\ell n \log q)^{O(d)} \cdot \max_{i\in[\ell]}\big\{\|\mathbf{u}_i - (\mathbf{A}_i + \mathbf{y}[i] \cdot \mathbf{G})^\intercal \mathbf{s}\|_\infty\big\}$$

  where $\mathbf{u}_C := \mathsf{Eval}_{\mathsf{ct}}(\mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{u}_1, \ldots, \mathbf{u}_\ell, \mathbf{y}, C)$*.*

- *if* $(\mathbf{A}_1, \ldots, \mathbf{A}_\ell) = (\mathbf{A}\mathbf{R}_1 - \mathbf{y}[1] \cdot \mathbf{G}, \ldots, \mathbf{A}\mathbf{R}_\ell - \mathbf{y}[\ell] \cdot \mathbf{G})$ *where* $\mathbf{R}_1, \ldots, \mathbf{R}_\ell \in \mathbb{Z}_q^{m\times m}$*, then we have*

$$\mathbf{A}_C = \mathbf{A}\mathbf{R}_C - C(\mathbf{y}) \cdot \mathbf{G}$$

  where $\mathbf{R}_C$ *is efficiently computable given* $(C, \mathbf{A}, \mathbf{R}_1, \ldots, \mathbf{R}_\ell)$ *and*

$$\|\mathbf{R}_C\|_\infty \leq O(\ell n \log q)^{O(d)} \cdot \max\{\|\mathbf{R}_1\|_\infty, \ldots, \|\mathbf{R}_\ell\|_\infty\}$$

Extension to $\widehat{C} \circ \mathsf{IP}$. We extend the above algorithms to obtain the circuits of the form $\widehat{C} \circ \mathsf{IP}$. Let $\widehat{C}_i$ denote the circuit computing the i'th bit of $\widehat{C}$.

- $\mathsf{Eval}_{\mathsf{pk}}$ takes as input $\ell+t$ matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{B}_1, \ldots, \mathbf{B}_t \in \mathbb{Z}_q^{n\times m}$ and a circuit $\widehat{C} \circ \mathsf{IP} : \{0,1\}^\ell \times \mathbb{Z}_q^t \to \mathbb{Z}_q$, outputs a matrix $\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} \in \mathbb{Z}_q^{n\times m}$ computed as follows:

  1. For $i = 1, \ldots, t$, compute $\mathbf{A}_{\widehat{C}_i} := \mathsf{Eval}_{\mathsf{pk}}(\mathbf{A}_1, \ldots, \mathbf{A}_\ell, \widehat{C}_i)$;
  2. Output $\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} := -\Big(\mathbf{A}_{\widehat{C}_1}\mathbf{G}^{-1}(\mathbf{B}_1) + \cdots + \mathbf{A}_{\widehat{C}_t}\mathbf{G}^{-1}(\mathbf{B}_t)\Big)$.

- $\mathsf{Eval}_{\mathsf{ct}}$ takes as input $\mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{B}_1, \ldots, \mathbf{B}_t$ and $\widehat{C} \circ \mathsf{IP}$ as before, along with $\mathbf{y} \in \{0,1\}^\ell$ and $\ell + t$ vectors $\mathbf{u}_1, \ldots, \mathbf{u}_\ell, \mathbf{v}_1, \ldots, \mathbf{v}_t \in \mathbb{Z}_q^m$, outputs a vector $\mathbf{u}_{\widehat{C}\circ\mathsf{IP}} \in \mathbb{Z}_q^m$ computed as follows:

  1. For $i = 1, \ldots, t$, compute $\mathbf{u}'_{\widehat{C}_i} := \mathsf{Eval}_{\mathsf{ct}}(\mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{u}_1, \ldots, \mathbf{u}_\ell, \mathbf{y}, \widehat{C}_i)$;
  2. Output $\mathbf{u}^\intercal_{\widehat{C}\circ\mathsf{IP}} := \Big((\mathbf{z}[1] \cdot \mathbf{v}_1 - \mathbf{G}^{-1}(\mathbf{B}_1)^\intercal \cdot \mathbf{u}'_1) + \cdots + (\mathbf{z}[t] \cdot \mathbf{v}_t - \mathbf{G}^{-1}(\mathbf{B}_t)^\intercal \cdot \mathbf{u}'_{\widehat{C}_i})\Big)$, where $\mathbf{z} = \widehat{C}(\mathbf{y})$.

We stress that $\mathsf{Eval}_{\mathsf{ct}}$ does not get $\mathbf{x}$. We obtain the following extension of Lemma 4.4.1:

Lemma 4.4.2 (properties of extended $\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}$)**.** *The algorithms* $\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}$ *satisfy the following properties. For all* $\mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{B}_1, \ldots, \mathbf{B}_t \in \mathbb{Z}_q^{n\times m}$*, all* $(\mathbf{x}, \mathbf{y}) \in \mathbb{Z}_q^t \times \{0,1\}^\ell$*, all boolean circuits* $\widehat{C}$ *of depth* d*, let* $\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} := \mathsf{Eval}_{\mathsf{pk}}(\mathbf{A}_1, \ldots, \mathbf{A}_\ell, \mathbf{B}_1, \ldots, \mathbf{B}_t, \widehat{C} \circ \mathsf{IP})$*. Then,*

- *for all* $\mathbf{u}_1, \dots, \mathbf{u}_\ell, \mathbf{v}_1, \dots, \mathbf{v}_t \in \mathbb{Z}_q^m$ *and all* $\mathbf{s} \in \mathbb{Z}_q^n$,

  $$\left\| \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} - (\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle \cdot \mathbf{G})^\intercal \mathbf{s} \right\|_\infty \le O(\ell n \log q)^{O(d)} \cdot \max_{i \in [\ell]} \{ \|\mathbf{u}_i - (\mathbf{A}_i + \mathbf{y}[i] \cdot \mathbf{G})^\intercal \mathbf{s}\|_\infty, ...\}$$

  where $\mathbf{u}_{\widehat{C} \circ \mathsf{IP}} := \mathsf{Eval}_{\mathsf{ct}}(\mathbf{A}_1, \dots, \mathbf{A}_\ell, \mathbf{B}_1, \dots, \mathbf{B}_t, \mathbf{u}_1, \dots, \mathbf{u}_\ell, \mathbf{v}_1, \dots, \mathbf{v}_t, \mathbf{y}, \widehat{C} \circ \mathsf{IP})$.

- *if* $(\mathbf{A}_1, \dots, \mathbf{A}_\ell) = (\mathbf{A}\mathbf{R}_1 - \mathbf{y}[1] \cdot \mathbf{G}, \dots, \mathbf{A}\mathbf{R}_\ell - \mathbf{y}[\ell] \cdot \mathbf{G})$ *and* $(\mathbf{B}_1, \dots, \mathbf{B}_t) = (\mathbf{B}\mathbf{R}'_1 - \mathbf{x}[1] \cdot \mathbf{G}, \dots, \mathbf{B}\mathbf{R}'_\ell - \mathbf{x}[t] \cdot \mathbf{G})$, *where* $\mathbf{R}_1, \dots, \mathbf{R}_\ell, \mathbf{R}'_1, \dots, \mathbf{R}'_t \in \mathbb{Z}_q^{m \times m}$, *then we have*

  $$\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} = \mathbf{A}\mathbf{R}_{\widehat{C} \circ \mathsf{IP}} + \langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle \mathbf{G}$$

  where $\mathbf{R}_{\widehat{C} \circ \mathsf{IP}}$ *is efficiently computable and*

  $$\left\| \mathbf{R}_{\widehat{C} \circ \mathsf{IP}} \right\|_\infty \le O(\ell n \log q)^{O(d)} \cdot \max\{\|\mathbf{R}_1\|_\infty, \dots, \|\mathbf{R}_\ell\|_\infty, \|\mathbf{R}'_1\|_\infty, \dots, \|\mathbf{R}'_t\|_\infty\}$$

Proof. The proof follows readily from Lemma 4.4.1, along with the calculation

$$\mathbf{R}_{\widehat{C} \circ \mathsf{IP}} := \mathbf{R}_{\widehat{C}_1} \mathbf{G}^{-1}(\mathbf{B}_1) + \cdots + \mathbf{R}_{\widehat{C}_t} \mathbf{G}^{-1}(\mathbf{B}_t) - \big(\hat{\mathbf{y}}[1]\mathbf{R}'_1 + \cdots + \hat{\mathbf{y}}[t]\mathbf{R}'_t\big).$$

□

4.5 Our PHPE scheme

For simplicity, we present our scheme for 1-bit message spaces.

- $\mathsf{PH.Setup}\big(1^\lambda, 1^t, 1^\ell, d\big)$: The setup algorithm takes the security parameter λ, the length of the secret attribute t, the length of the public attribute ℓ, and the circuit depth bound d. Define the lattice parameters $n = n(\lambda), m = m(n, d), q = q(n, d), \chi = \chi(n)$ as per Section 4.5.3.

  1. Choose random matrices $\mathbf{A}_i \in \mathbb{Z}_q^{n \times m}$ for $i = 1, \dots, \ell$, $\mathbf{B}_i \in \mathbb{Z}_q^{n \times m}$ for $i = 1, \dots, t$ and $\mathbf{P} \in \mathbb{Z}_q^{n \times m}$.[6]
  2. Sample a matrix with associated trapdoor:

     $$\big(\mathbf{A}, \mathbf{T}\big) \leftarrow \mathsf{TrapSamp}(1^m, 1^n, q)$$

  3. Let $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ be the powers-of-two matrix with a public trapdoor basis $\mathbf{T}_{\mathbf{G}}$.
  4. Output the master public key $\mathsf{ph.mpk} := \big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \mathbf{A}, \mathbf{P}\big)$ and the master secret key as $\mathsf{ph.msk} := (\mathsf{mpk}, \mathbf{T})$.

[6]To simplify notation, we always denote the collections $\{\mathbf{A}_i\} := \{\mathbf{A}_i\}_{i \in [\ell]}$ and $\{\mathbf{B}_i\} = \{\mathbf{B}_i\}_{i \in [t]}$.

- $\mathsf{PH.KeyGen}\big(\mathsf{ph.msk}, \widehat{C} \circ \mathsf{IP}_\gamma\big)$: The key-generation algorithms takes as input the master secret key msk, a circuit $\widehat{C} \circ \mathsf{IP}_\gamma$. It outputs a secret key $\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}$ computed as follows.

  1. Let
  $$\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \widehat{C} \circ \mathsf{IP}\big)$$
  be the homomorphically computed "public key" as per the evaluation algorithm in Section 4.4.
  2. Sample a matrix $\mathbf{R} \in \mathbb{Z}_q^{2m \times m}$ such that $[\mathbf{A}|\mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}] \cdot \mathbf{R} = \mathbf{P} \mod q$, where
  $$\mathbf{R} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}, \mathbf{T}, \mathbf{P}, s)$$
  3. Output the secret key $\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma} := (\mathbf{R})$.

- $\mathsf{PH.Enc}\big(\mathsf{ph.mpk}, (\mathbf{x}, \mathbf{y}), m\big)$: The encryption algorithm takes as input the public key $\mathsf{ph.mpk}$, attribute vectors $\mathbf{x} \in \mathbb{Z}_q^t$, $\mathbf{y} \in \{0,1\}^\ell$ and a message $m \in \{0,1\}$. It computes ciphertext $\mathsf{ct}_\mathbf{y}$ as follows.

  1. Choose a secret vector $\mathbf{s} \leftarrow (\chi)^n$ and error terms $\mathbf{e}, \mathbf{e}' \leftarrow (\chi)^m$.
  2. Let $\mathbf{b} = \big[0, \ldots, 0, \lceil q/2 \rceil m\big]^\mathsf{T} \in \mathbb{Z}_q^m$. Compute encodings
  $$\beta_0 = (\mathbf{A})^\mathsf{T}\mathbf{s} + \mathbf{e} \text{ and } \beta_1 = \mathbf{P}^\mathsf{T}\mathbf{s} + \mathbf{e}' + \mathbf{b}$$
  3. For all $i = 1, \ldots, \ell$ compute an encoding
  $$\mathbf{u}_i = (\mathbf{A}_i + \mathbf{y}[i] \cdot \mathbf{G})^\mathsf{T}\mathbf{s} + \mathbf{R}_i^\mathsf{T}\mathbf{e}$$
  where $\mathbf{R}_i \leftarrow \{-1,1\}^{m \times m}$.
  4. For all $i = 1, \ldots, t$ compute an encoding
  $$\mathbf{v}_i = \big(\mathbf{B}_i + \mathbf{x}[i] \cdot \mathbf{G}\big)^\mathsf{T}\mathbf{s} + (\mathbf{R}_i')^\mathsf{T}\mathbf{e}_i$$
  where $\mathbf{R}_i' \leftarrow \{-1,1\}^{m \times m}$.
  5. Output the ciphertext
  $$\mathsf{ct}_\mathbf{y} := \Big(\{\mathbf{u}_i\}_{i \in [\ell]}, \{\mathbf{v}_i\}_{i \in [t]}, \beta_0, \beta_1\Big)$$

- $\mathsf{PH.Dec}(\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}, (\mathsf{ct}_\mathbf{y}, \mathbf{y}))$: The decryption algorithm takes as input the secret key $\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}$ for a circuit $\widehat{C} \circ \mathsf{IP}_\gamma$ and the ciphertext $\mathsf{ct}_\mathbf{y}$ along with the public attribute $\mathbf{y}$. It proceeds as follows.

  1. Using $\{\mathbf{u}_i\}, \{\mathbf{v}_i\}$ and $\mathbf{y}$, apply the encoding evaluation algorithm (See Section 4.4)

to obtain a ciphertext

$$\mathbf{u}_{\widehat{C}\circ\mathsf{IP}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}\big(\{\mathbf{A}_i, \mathbf{u}_i\}, \{\mathbf{B}_i, \mathbf{v}_i\}, \widehat{C} \circ \mathsf{IP}, \mathbf{y}\big)$$

where $\mathbf{u}_{\widehat{C}\circ\mathsf{IP}} \approx (\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \rho \cdot \mathbf{G})^{\intercal}\mathbf{s} + \mathbf{e}$ for some $\rho \in \mathbb{Z}_q$.

2. Now, compute

$$\eta = \beta_1 - \mathbf{R}^{\intercal} \cdot \begin{bmatrix} \beta_0 \\ \mathbf{u}_{\widehat{C}\circ\mathsf{IP}} \end{bmatrix} \in \mathbb{Z}_q^m$$

Output $m = \mathsf{Round}(\eta[m])$ if $\big[\mathsf{Round}(\eta[1]), \ldots, \mathsf{Round}(\eta[m-1])\big] = \mathbf{0}$, where

$$\mathsf{Round}(c) = \begin{cases} 0 & \text{if } |c| < q/4 \\ 1 & \text{otherwise} \end{cases}$$

Otherwise, output $\perp$.

4.5.1 Analysis and Correctness

Lemma 4.5.1. *Let $\mathcal{C}$ be a family of circuits bounded by depth d and let $\mathcal{PHPE}$ be our scheme defined above. Assume that for LWE dimension $n = n(\lambda)$, the parameters are instantiated as follows:*

$$\begin{aligned} \chi &= D_{\mathbb{Z},\sqrt{n}} \\ q &= \tilde{O}(tnd)^{O(d)} \\ m &= O(n \log q) \\ B &= B(n) \\ s &= O(tn \log q)^{O(d)} \end{aligned}$$

Then, the scheme is correct according to Definition 4.3.1.

Proof. We proceed proving correctness of the scheme in two steps. First, we bound the error term $\mathbf{e}$ in the final homomorphically computed encoding $\mathbf{u}_{\widehat{C}\circ\mathsf{IP}}$. By Lemma 4.4.2, the error in $\mathbf{u}_{\widehat{C}\circ\mathsf{IP}}$ satisfies

$$\left\|\mathbf{u}_{\widehat{C}\circ\mathsf{IP}} - (\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \langle \mathbf{x}, \widehat{C}(\mathbf{y})\rangle \cdot \mathbf{G})^{\intercal}\mathbf{s}\right\|_{\infty} \leq O(tB \cdot O(n \log q)^{O(d+1)}).$$

Recall that $[\mathbf{A} \mid \mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \gamma \cdot \mathbf{G}] \cdot \mathbf{R} = \mathbf{P} \mod q$ and $\|\mathbf{R}^{\intercal}\|_{\infty} \leq s\sqrt{m}$. After multiplying by $\mathbf{R}^{\intercal}$, we obtain the final error bound of $O(tB \cdot O(n \log q)^{O(d+1)})$. We then consider two cases:

- if $\langle \mathbf{x}, \widehat{C}(\mathbf{y})\rangle = \gamma \mod q$, then

$$||\eta = \beta_1 - \mathbf{R}^{\intercal} \cdot \begin{bmatrix} \beta_0 \\ \mathbf{u}_{\widehat{C}\circ\mathsf{IP}} \end{bmatrix}||_{\infty} = O(tB \cdot O(n \log q)^{O(d+1)}) \leq q/4$$

in the first $m-1$ entries for sufficiently large $q = \tilde{O}(tnd)^{O(d)}$. Hence, the message m is recovered correctly.

- Otherwise, say $\langle \mathbf{x}, \widehat{C}(\mathbf{y}) \rangle = \gamma' \neq \gamma \mod q$ and $\gamma' = \gamma + \gamma^*$. Then, then multiplying by $\mathbf{R}^\intercal = [\mathbf{R}_1, \mathbf{R}_2]$ we obtain

$$\eta = \beta_1 - \mathbf{R}^\intercal \cdot \begin{bmatrix} \beta_0 \\ \mathbf{u}_{\widehat{C} \circ \mathsf{IP}} \end{bmatrix} = \mathbf{R}_2^\intercal \cdot \gamma^* \cdot \mathbf{G} + \mathbf{e}^*$$

for some error vector $\mathbf{e}^*$. Hence, with all but negligible probability all first $m-1$ coefficients of η will be below $q/4$.

This concludes the correctness proof. □

4.5.2 Security

Theorem 4.5.2. *Let $\mathcal{PHPE}$ be our partially-hiding predicate encryption scheme. Then, it is secure according to Definition 4.3.1 assuming hardness of Learning With Errors problem.*

Proof. We describe a p.p.t. simulator Sim algorithm and then claim that the output of the ideal experiment is indistinguishable from real via a series of hybrids.

- Sim$(\mathsf{ph.mpk}, \mathbf{y})$: samples $\beta_0, \beta_1, \mathbf{u}_i, \mathbf{v}_i$ randomly and independently from $\mathbb{Z}_q^m$ and outputs the ciphertext

$$\mathsf{ct} := \left(\{\mathbf{u}_i\}_{i \in [\ell]}, \{\mathbf{v}_i\}_{i \in [t]}, \mathbf{y}, \beta_0, \beta_1 \right)$$

Hybrid Sequence. We now claim that security of our scheme scheme via a series of hybrids, where Hybrid 0 corresponds to the real experiment and Hybrid 6 corresponds to the simulated experiment using algorithms Sim.

- **Hybrid 0:** The real experiment.
- **Hybrid 1:** The real game algorithms $\mathsf{PH.Setup}$, $\mathsf{PH.Enc}$ are replaced with $\mathsf{PH.Setup}_1^*$, $\mathsf{PH.Enc}_1^*$ defined below. Informally, these algorithms use the knowledge of $\mathbf{x}, \mathbf{y}$ to setup the public parameters in a special form.
- **Hybrid 2:** The real game $\mathsf{PH.KeyGen}$ is replaced with $\mathsf{PH.KeyGen}_1^*$, where instead of using the trapdoor $\mathbf{T}$ of the matrix $\mathbf{A}$, the secret keys are sampled using the public trapdoor $\mathbf{T_G}$ along with the trapdoor information generated using $\mathsf{PH.Setup}_1^*$.
- **Hybrid 3:** Same as above, except the $\mathsf{PH.Enc}_1^*$ is replaced with $\mathsf{PH.Enc}_2^*$ defined below.
- **Hybrid 4:** Same as above, except $\mathsf{PH.KeyGen}_1^*$ is replaced with real key-generation $\mathsf{PH.KeyGen}$.

- **Hybrid 5:** Same as above, except $\mathsf{PH.Enc}_2^*$ is replaced with $\mathsf{PH.Enc}_3^*$ define below, which informally replaces all ciphertext components with random elements.

- **Hybrid 6:** The simulated experiment, that is, same as above, except $\mathsf{PH.Setup}_1^*$ is replaced with $\mathsf{PH.Setup}$.

Auxiliary Algorithms. We now define the auxiliary algorithms similar to those in [AFV11, GMW15] and then argue that the hybrids are either statistically or computationally indistinguishable.

- $\mathsf{PH.Setup}_1^*\big(1^\lambda, d, \mathbf{x}, \mathbf{y}\big)$: In addition to the system parameters, the setup algorithms use the knowledge of the challenge attribute vectors $(\mathbf{x}, \mathbf{y})$. Define the lattice parameters $n = n(\lambda), m = m(n,d), q = q(n,d), \chi = \chi(n)$ (See Section 4.5.3).

  1. Sample a matrix with associated trapdoor:
  $$\big(\mathbf{A}, \mathbf{T}\big) \leftarrow \mathsf{TrapSamp}(1^m, 1^n, q)$$
  Let $\mathbf{G}$ be the powers-of-two matrix with a public trapdoor $\mathbf{T_G}$.
  2. Let $\mathbf{A}_i = \mathbf{A}\cdot\mathbf{R}_i - \mathbf{y}[i]\cdot\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ for $i = 1, \ldots, \ell$ where $\mathbf{R}_i \xleftarrow{\$} \{-1,1\}^{m\times m}$.
  3. Let $\mathbf{B}_i = \mathbf{A}\cdot\mathbf{R}'_i - \mathbf{x}[i]\cdot\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ for $i = 1, \ldots, t$ where $\mathbf{R}'_i \xleftarrow{\$} \{-1,1\}^{m\times m}$.
  4. Choose a random matrix $\mathbf{P} \in \mathbb{Z}_q^{n\times m}$.
  5. Output the master public key $\mathsf{ph.mpk} := \big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \mathbf{A}, \mathbf{P}\big)$ and the master secret key as $\mathsf{ph.msk} := (\mathsf{mpk}, \mathbf{T}, \{\mathbf{R}_i\}, \{\mathbf{R}'_i\})$.

  In Hybrids 1, 4, 5, we will generate secret keys using $\mathsf{PH.KeyGen}$, which requires knowing $\mathbf{T}$. In Hybrids 2 and 3, we will generate secret keys using $\mathsf{PH.KeyGen}_1^*$, which does not require knowing $\mathbf{T}$.

- $\mathsf{PH.KeyGen}_1^*\big(\mathsf{ph.msk}, \widehat{C}\circ\mathsf{IP}_\gamma\big)$: The key-generation algorithms takes as input the master secret key $\mathsf{ph.msk}$, a pair of circuits C, IP_γ. It outputs a secret key $\mathsf{sk}_{\widehat{C}\circ\mathsf{IP}_\gamma}$ computed as follows.

  1. Let
  $$\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(\{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \widehat{C}\circ\mathsf{IP}\big)$$
  be the homomorphically computed "public key" as per the evaluation algorithm in Section 4.4.
  2. By Lemma 4.4.2, $\mathbf{A}_{\widehat{C}\circ\mathsf{IP}} = \mathbf{A}\cdot\mathbf{R}_{\widehat{C}\circ\mathsf{IP}} - \langle\mathbf{x}, \widehat{C}(\mathbf{y})\rangle\cdot\mathbf{G}$. Now, an admissible adversary is restricted to queries on circuits $\widehat{C}\circ\mathsf{IP}_\gamma$ such that $\langle\mathbf{x}, \widehat{C}(\mathbf{y})\rangle \neq \gamma$. Hence, we can sample $\mathbf{R} \in \mathbb{Z}_q^{2m\times m}$ such that
  $$[\mathbf{A} \mid \mathbf{A}_{\widehat{C}\circ\mathsf{IP}} + \gamma\cdot\mathbf{G}]\cdot\mathbf{R} = [\mathbf{A} \mid \mathbf{A}\mathbf{R}_{\widehat{C}\circ\mathsf{IP}} + (\gamma - \langle\mathbf{x}, \widehat{C}(\mathbf{y})\rangle)\cdot\mathbf{G}]\cdot\mathbf{R} = \mathbf{P} \mod q,$$

using

$$\mathbf{R} \leftarrow \mathsf{SampleRight}(\mathbf{A}, (\gamma - \langle \mathbf{x}, \widehat{C}(\mathbf{y})\rangle) \cdot \mathbf{G}, \mathbf{R}_{\widehat{C} \circ \mathsf{IP}}, \mathbf{T}_{\mathbf{G}}, \mathbf{P}, s)$$

3. Output the secret key $\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma} := (\mathbf{R})$.

- $\mathsf{PH.Enc}_1^*(\mathsf{ph.mpk}, (\mathbf{x}, \mathbf{y}), m)$: The encryption algorithm takes as input the public key $\mathsf{ph.mpk}$, challenge vectors $\mathbf{x} \in \mathbb{Z}_q^t$, $\mathbf{y} \in \mathbb{Z}_q^\ell$ and a message m. It computes ciphertext $\mathsf{ct}_{\mathbf{y}}$ as follows.

  1. Choose a vector $\mathbf{s} \in (\chi)^n$ and compute encodings

  $$\beta_0 = (\mathbf{A})^\intercal \mathbf{s} + \mathbf{e} \text{ and } \beta_1 = \mathbf{P}^\intercal \mathbf{s} + \mathbf{e}' + \mathbf{b}$$

     where $\mathbf{b} = [0, \ldots, 0, \lceil q/2 \rceil m]^\intercal \in \mathbb{Z}_q^m$.
  2. For all $i = 1, \ldots, \ell$ compute an encoding

  $$\begin{aligned} \mathbf{u}_i &= \mathbf{R}_i^\intercal \cdot \beta_0 \\ &= (\mathbf{A} \cdot \mathbf{R}_i)^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \cdot \mathbf{e} \\ &= (\mathbf{A}_i + \mathbf{y}[i] \cdot \mathbf{G})^\intercal \mathbf{s} + \mathbf{R}_i^\intercal \cdot \mathbf{e} \end{aligned}$$

  3. For all $i = 1, \ldots, t$ compute an encoding

  $$\begin{aligned} \mathbf{v}_i &= (\mathbf{R}_i')^\intercal \cdot \beta_0 \\ &= (\mathbf{A} \cdot \mathbf{R}_i')^\intercal \mathbf{s} + (\mathbf{R}_i')^\intercal \cdot \mathbf{e} \\ &= (\mathbf{B}_i + \mathbf{x}[i] \cdot \mathbf{G})^\intercal \mathbf{s} + (\mathbf{R}_i')^\intercal \cdot \mathbf{e} \end{aligned}$$

  4. Output the ciphertext

  $$\mathsf{ct} := \Big(\{\mathbf{u}_i\}_{i \in [\ell]}, \{\mathbf{v}_i\}_{i \in [t]}, \mathbf{y}, \beta_0, \beta_1\Big)$$

- $\mathsf{PH.Enc}_2^*(\mathsf{ph.mpk}, (\mathbf{x}, \mathbf{y}), m)$:

  1. Choose random elements β_0, β_1 from $\mathbb{Z}_q^m$.
  2. For all $i = 1, \ldots, \ell$ compute an encoding $\mathbf{u}_i = \mathbf{R}_i^\intercal \cdot \beta_0$.
  3. For all $i = 1, \ldots, t$ compute an encoding $\mathbf{v}_i = (\mathbf{R}_i')^\intercal \cdot \beta_0$.
  4. Output the ciphertext

  $$\mathsf{ct} := \Big(\{\mathbf{u}_i\}_{i \in [\ell]}, \{\mathbf{v}_i\}_{i \in [t]}, \mathbf{y}, \beta_0, \beta_1\Big)$$

- $\mathsf{PH.Enc}_3^*(\mathsf{ph.mpk}, (\mathbf{x}, \mathbf{y}), m)$: The final auxiliary encryption algorithm computes ciphertext $\mathsf{ct}_{\mathbf{y}}$ as a collection of random encodings. That is, $\beta_0, \beta_1, \mathbf{u}_i, \mathbf{v}_i$ are all randomly and independently chosen from $\mathbb{Z}_q^m$. Output the ciphertext

$$\mathsf{ct} := \Big(\{\mathbf{u}_i\}_{i\in[\ell]}, \{\mathbf{v}_i\}_{i\in[t]}, \mathbf{y}, \beta_0, \beta_1\Big)$$

Lemma 4.5.3. *The output of Hybrid 0 is statistically indistinguishable from the output of Hybrid 1.*

Proof. The proof follows closely to [AFV11, Lemma 4.3]. For completeness, we first summarize the difference between the two hybrids:

1. In Hybrid 0, matrices $\mathbf{A}_i, \mathbf{B}_i$ are uniformly chosen in $\mathbb{Z}_q^{n\times m}$. However, in Hybrid 1, $\mathbf{A}_i = \mathbf{A}\cdot\mathbf{R}_i - \mathbf{y}[i]\cdot\mathbf{G}$ and $\mathbf{B}_i = \mathbf{A}\cdot\mathbf{R}'_i - \mathbf{x}[i]\cdot\mathbf{G}$.

2. In Hybrid 0, the ciphertext encodings are computed as

$$\mathbf{u}_i = (\mathbf{A}_i + \mathbf{y}[i]\cdot\mathbf{G})^{\intercal}\mathbf{s} + \mathbf{R}_i^{\intercal}\mathbf{e} \quad \text{and} \quad \mathbf{v}_i = (\mathbf{A}_i + \mathbf{x}[i]\cdot\mathbf{G})^{\intercal}\mathbf{s} + (\mathbf{R}'_i)^{\intercal}\mathbf{e}$$

   whereas in Hybrid 1 it is computed as

$$\mathbf{u}_i = \mathbf{R}_i^{\intercal}\beta_0 \quad \text{and} \quad \mathbf{v}_i = (\mathbf{R}'_i)^{\intercal}\beta_0$$

   where $\beta_0 = (\mathbf{A})^{\intercal}\mathbf{s} + \mathbf{e}$.

We now argue that the joint distribution of the public parameters, the ciphertext and the secret keys

$$\Big(\mathbf{A}, \{\mathbf{A}_i\}, \{\mathbf{B}_i\}, \{\mathbf{u}_i\}, \{\mathbf{v}_i\}, \{\mathsf{sk}_{\widehat{C}\circ\mathsf{IP}}\}\Big)$$

is statistically indistinguishable between the two hybrids. Note that the secret keys are produced in both using trapdoor $\mathbf{T}$ and the public matrices. Now, observe that by Lemma 2.4.6,

$$\big(\mathbf{A}, \mathbf{A}\cdot\mathbf{R}_i - \mathbf{y}[i]\cdot\mathbf{G}, \mathbf{R}_i^{\intercal}\mathbf{e}, \mathbf{T}\big) \overset{s}{\approx} \big(\mathbf{A}, \mathbf{A}_i, \mathbf{R}_i^{\intercal}\mathbf{e}, \mathbf{T}\big).$$

This holds for matrices $\mathbf{B}_i$ as well. And since for all i, $\mathbf{R}_i$ (resp. $\mathbf{R}'_i$) is randomly and independently chosen, it follows that

$$\Big(\mathbf{A}, \big\{\mathbf{A}_i\big\}, \big\{\mathbf{B}_i\big\}, \big\{\mathbf{R}_i^{\intercal}\mathbf{e}\big\}, \big\{(\mathbf{R}'_i)^{\intercal}\mathbf{e}\big\}, \mathbf{T}\Big) \overset{s}{\approx}$$

$$\Big(\mathbf{A}, \big\{\mathbf{A}_i\mathbf{R}_i - \mathbf{y}[i]\cdot\mathbf{G}\big\}, \big\{\mathbf{B}_i\mathbf{R}'_i - \mathbf{x}[i]\cdot\mathbf{G}\big\}, \big\{\mathbf{R}_i^{\intercal}\mathbf{e}\big\}, \big\{(\mathbf{R}'_i)^{\intercal}\mathbf{e}\big\}, \mathbf{T}\Big).$$

The ciphertext components $\mathbf{u}_i$ and $\mathbf{v}_i$ are derived simply by adding $(\mathbf{A}_i + \mathbf{y}[i]\cdot\mathbf{G})^{\intercal}\mathbf{s}$ and $(\mathbf{B}_i + \mathbf{x}[i]\cdot\mathbf{G})^{\intercal}\mathbf{s}$ to $\mathbf{R}_i^{\intercal}\mathbf{e}$ and $(\mathbf{R}'_i)^{\intercal}\mathbf{e}$, respectively. And the secret keys are generated from the matrices and the trapdoor $\mathbf{T}$. Since applying a function to two statistically indistinguishable distributions produces two statistically indistinguishable distributions, this shows that the public parameters, the ciphertext and the secret keys are statistically close in both hybrids. □

Lemma 4.5.4. *The output of Hybrid 1 is statistically indistinguishable from the output of Hybrid 2.*

Proof. From Hybrid 1 to Hybrid 2, we switch between between KeyGen and $\mathsf{PH.KeyGen}^*$. Fix a secret key query $\widehat{C} \circ \mathsf{IP}_\gamma$ made by an admissible adversary:

- In Hybrid 1, the secret key is sampled using $\mathsf{SampleLeft}$ with trapdoor $\mathbf{T}$, and its distribution only depends on the public matrices in $\mathsf{ph.mpk}$ by Theorem 2.4.3 (provided the parameter s is sufficiently large);
- In Hybrid 2, the secret key is sampled using $\mathsf{SampleRight}$ with trapdoor $\mathbf{T_G}$ along with $\mathbf{R}_i, \mathbf{R}'_i$ (which we can do since the adversary is admissible) and its distribution only depends on the public matrices in $\mathsf{ph.mpk}$ by Theorem 2.4.4 (again, provided s is sufficiently large).

In particular, in both hybrids, the distribution of the secret key only depends on the matrices $\mathbf{A}, \mathbf{A}_{\widehat{C} \circ \mathsf{IP}} + \gamma \cdot \mathbf{G}, \mathbf{P}$ and are in turn completely determined by $\mathsf{ph.mpk}$. Since $\mathsf{ph.mpk}$ has exactly the same distribution in Hybrids 1 and 2, it follows that the output of both hybrids are statistically indistinguishable. □

Lemma 4.5.5. *The output of Hybrid 2 is computationally indistinguishable from the output of Hybrid 3, under the LWE assumption.*

Proof. We show how to break the security of LWE given an adversary that distinguishes between the two hybrids. We are given matrices $(\mathbf{A}, \mathbf{P}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{n\times m}$ and samples $\mathbf{u}, \mathbf{w} \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m$ which are either LWE samples for some secret vector $\mathbf{s}$ or randomly chosen. We simulate the experiments as follows.

- Runs $\mathsf{PH.Setup}_1^*, \mathsf{PH.KeyGen}_1^*$ algorithms using the matrices $\mathbf{A}, \mathbf{P}$ from the challenge.
- To simulate the ciphertext encodings, let $\beta_0 = \mathbf{u}$ and $\beta_1 = \mathbf{w} + \mathbf{b}$, where $\mathbf{b} = \big[0, \ldots, 0, \lceil q/2 \rceil m\big]^\mathsf{T} \in \mathbb{Z}_q^m$. The ciphertext encodings $\mathbf{u}_i, \mathbf{v}_i$ are computed using $\mathbf{R}_i, \mathbf{R}'_i$ as
$$\mathbf{u}_i = \mathbf{R}_i^\mathsf{T} \beta_0 \text{ and } \mathbf{v}_i = (\mathbf{R}'_i)^\mathsf{T} \beta_0$$
Output
$$\mathsf{ct} := \Big(\{\mathbf{u}_i\}_{i\in[\ell]}, \{\mathbf{v}_i\}_{i\in[t]}, \mathbf{y}, \beta_0, \beta_1\Big)$$

Now clearly, if $\mathbf{u} = \mathbf{A}^\mathsf{T}\mathbf{s} + \mathbf{e}$ and $\mathbf{w} = \mathbf{P}^\mathsf{T}\mathbf{s} + \mathbf{e}'$, then the simulation is identical to Hybrid 2. Otherwise, if $\mathbf{u}, \mathbf{w}$ are random elements then the experiment corresponds exactly to Hybrid 3. Hence, given an adversary that distinguishes between Hybrids 2 and 3, we can break the security of the standard LWE problem. □

Lemma 4.5.6. *The output of Hybrid 3 is statistically indistinguishable from the output of Hybrid 4.*

Proof. The proof follows similarly to that of Lemma 4.5.4, where we switch between $\mathsf{PH.KeyGen}^*$ and KeyGen. $\square$

Lemma 4.5.7. *The output of Hybrid 4 is statistically indistinguishable from the output of Hybrid 5.*

Proof. In Hybrid 4, the ciphertext encodings are computed as $\mathbf{u}_i = \mathbf{R}_i^\intercal \cdot \beta_0$ and $\mathbf{v}_i = (\mathbf{R}'_i)^\intercal \cdot \beta_0$. However, in Hybrid 5 these are randomly chosen from the encoding space. The indistinguishability of two hybrids follows from the standard leftover hash lemma, since:

- the secret keys are generated using $\mathsf{PH.KeyGen}$, which do not use any information about $\mathbf{R}_i, \mathbf{R}'_i$;
- the only additional leakage on $\mathbf{R}_i, \mathbf{R}'_i$ comes from $(\mathbf{A}\mathbf{R}_i, \mathbf{B}\mathbf{R}'_i)$ in $\mathsf{ph.mpk}$.

Therefore, for all $\mathbf{A}, \mathbf{B}, \beta_0$ and for all i,

$$\Big(\mathbf{A}, \mathbf{B}, \beta_0, \{\mathbf{A}\mathbf{R}_i, \mathbf{R}_i^\intercal \cdot \beta_0\}, \{\mathbf{B}\mathbf{R}'_i, (\mathbf{R}'_i)^\intercal \cdot \beta_0\}\Big) \overset{s}{\approx} \Big(\mathbf{A}, \mathbf{B}, \beta_0, \{\mathbf{A}\mathbf{R}_i, \mathbf{u}_i\}, \{\mathbf{B}\mathbf{R}'_i, \mathbf{v}_i\}\Big)$$

for randomly chosen $\mathbf{R}_i, \mathbf{R}'_i, \mathbf{u}_i, \mathbf{v}_i$ and sufficiently large $m = O(n \log q)$. $\square$

Lemma 4.5.8. *The output of Hybrid 5 is statistically indistinguishable from the output of Hybrid 6.*

Proof. The proof follows similarly to that of Lemma 4.5.3, where we switch between $\mathsf{PH.Setup}_1^*$ and $\mathsf{PH.Setup}$. $\square$

This completes the security proof. $\square$

4.5.3 Parameters Selection of Our PHPE Scheme

We must set the parameters to satisfy correctness and security of the scheme. For correctness, we must ensure that the magnitude of the final error $\mathbf{e}$ is below $q/4$. For security, we must ensure the statistical indistingushability of matrix $\mathbf{A}$ from uniform and indistinguishability of $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ algorithms by setting parameter s large enough. We start by setting LWE dimension $n = n(\lambda)$, the error distribution $\chi = \chi(n) = D_{\mathbb{Z},\sqrt{n}}$ and the error bound $B = B(n) = O(n)$. We set the modulus $q = \tilde{O}(tnd)^{O(d)}$ and lattice dimension $m = O(n \log q)$ to apply Lemma 2.4.1. Finally, we set $s = O(tn \log q)^{O(d)}$ to apply Lemmas 2.4.3, 2.4.4. As shown in Section 4.5.1, these parameters also satisfy the correctness requirements of the scheme. That is, the master public key, ciphertext and secret keys all have size $\mathrm{poly}(\lambda, t, \ell, d)$ where $(1^\lambda, 1^t, 1^\ell, 1^d)$ is the input to $\mathsf{PH.Setup}$ and we achieve security under $\mathsf{LWE}_{n,q,\chi}$ where $q = \tilde{O}(tnd)^{O(d)}$ and the modulus-to-noise ratio is $\tilde{O}(tnd)^{O(d)}$.

4.6 Predicate Encryption for Circuits

In this section, we present our main construction of predicate encryption for circuits by bootstrapping on top of the partially-hiding predicate encryption. That is,

- We construct a Predicate Encryption scheme $\mathcal{PE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ for boolean predicate family $\mathcal{C}$ bounded by depth d over k bit inputs.

starting from

- an FHE scheme $\mathcal{FHE} = (\mathsf{FH.Keygen}, \mathsf{FH.Enc}, \mathsf{FH.Dec}, \mathsf{FH.Eval})$ with properties as described in Section 4.2.5. Define ℓ as the size of the initial ciphertext encrypting k bit messages, and t as the size of the FHE secret key and evaluated ciphertext vectors;

- a partially-hiding predicate encryption scheme $\mathcal{PHPE} = (\mathsf{PH.Setup}, \mathsf{PH.KeyGen}, \mathsf{PH.Enc}, \mathsf{PH.Dec})$ for the class $\mathcal{C}_{\mathsf{PHPE}}$ of predicates bounded by some depth parameter $d' = \mathrm{poly}(d, \lambda, \log q)$. Recall that

$$(\widehat{C} \circ \mathsf{IP}_\gamma)(\mathbf{x} \in \mathbb{Z}_q^t, \mathbf{y} \in \{0,1\}^t) = 1 \text{ iff } \left(\sum_{i \in [t]} \mathbf{x}[i] \cdot \widehat{C}(\mathbf{y})[i]\right) = \gamma \mod q$$

  where $\widehat{C} : \{0,1\}^\ell \to \{0,1\}^t$ is a circuit of depth at most d'.

Overview. At a high level, the construction proceeds as follows:

- the $\mathcal{PE}$ ciphertext corresponding to an attribute $\mathbf{a} \in \{0,1\}^k$ is a $\mathcal{PHPE}$ ciphertext corresponding to an attribute $(\mathsf{fhe.sk}, \mathsf{fhe.ct})$ where $\mathsf{fhe.sk} \xleftarrow{\$} \mathbb{Z}_q^t$ is private and $\mathsf{fhe.ct} := \mathsf{FH.Enc}(\mathbf{a}) \in \{0,1\}^\ell$ is public;

- the $\mathcal{PE}$ secret key for a predicate $C : \{0,1\}^k \to \{0,1\} \in \mathcal{C}$ is a collection of $2B+1$ $\mathcal{PHPE}$ secret keys for the predicates $\{\widehat{C} \circ \mathsf{IP}_\gamma : \mathbb{Z}_q^t \times \{0,1\}^\ell \to \{0,1\}\}_{\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B}$ where $\widehat{C} : \{0,1\}^\ell \to \{0,1\}$ is the circuit:

$$\widehat{C}(\mathsf{fhe.ct}) := \mathsf{FH.Eval}(\mathsf{fhe.ct}, C),$$

  so $\widehat{C}$ is a circuit of depth at most $d' = \mathrm{poly}(d, \lambda, \log q)$;

- decryption works by trying all possible $2B+1$ secret keys.

Note that the construction relies crucially on the fact that B (the bound on the noise in the FHE evaluated ciphertexts) is polynomial. For correctness, observe that for all $C, \mathbf{a}$:

$$
\begin{aligned}
& C(\mathbf{a}) = 1 \\
\Leftrightarrow \quad & \mathsf{FH.Dec}(\mathsf{fhe.sk}, \mathsf{FH.Eval}(C, \mathsf{fhe.ct})) = 1 \\
\Leftrightarrow \quad & \exists\, \gamma \in [\lfloor q/2 \rfloor - B, \lfloor q/2 \rfloor + B] \text{ such that } \left(\sum_{i \in [t]} \mathsf{fhe.sk}[i] \cdot \mathsf{fhe.ct}[i] \right) = \gamma \mod q \\
\Leftrightarrow \quad & \exists\, \gamma \in [\lfloor q/2 \rfloor - B, \lfloor q/2 \rfloor + B] \text{ such that } (\widehat{C} \circ \mathsf{IP}_\gamma)(\mathsf{fhe.sk}, \mathsf{fhe.ct}) = 1
\end{aligned}
$$

where $\mathsf{fhe.sk}, \mathsf{fhe.ct}, \hat{C}$ are derived from $C, \mathbf{a}$ as in our construction.

4.6.1 Our Predicate Encryption scheme

Our construction proceeds as follows:

- $\mathsf{Setup}(1^\lambda, 1^k, d)$: The setup algorithm takes the security parameter λ, the attribute length k and the predicate depth bound d.
  1. Run the partially-hiding PE scheme for family $\mathcal{C}_{\mathsf{PHPE}}$ to obtain a pair of master public and secret keys:
  $$(\mathsf{ph.mpk}, \mathsf{ph.msk}) \leftarrow \mathsf{PH.Setup}(1^\lambda, 1^t, 1^\ell, d')$$
  where for k-bit messages and depth d circuits: t is the length of FHE secret key, ℓ is the bit-length of the initial FHE ciphertext and d' is the bound on FHE evaluation circuit (as described at the beginning of this section).
  2. Output $(\mathsf{mpk} := \mathsf{ph.mpk}, \mathsf{msk} := \mathsf{ph.msk})$.

- $\mathsf{KeyGen}(\mathsf{msk}, C)$: The key-generation algorithms takes as input the master secret key msk and a predicate C. It outputs a secret key sk_C computed as follows.
  1. Let $\widehat{C}(\cdot) := \mathsf{FH.Eval}(\cdot, C)$ and let $(\widehat{C} \circ \mathsf{IP}_\gamma)$ be the predicates for $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$.
  2. For all $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$, compute
  $$\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma} \leftarrow \mathsf{PH.KeyGen}\big(\mathsf{ph.msk}, \widehat{C} \circ \mathsf{IP}_\gamma\big)$$
  3. Output the secret key as $\mathsf{sk}_C := \big(\{\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}}\}_{\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B}\big)$.

- $\mathsf{Enc}(\mathsf{mpk}, \mathbf{a}, m)$: The encryption algorithm takes as input the public key mpk, the input attribute vector $\mathbf{a} \in \{0,1\}^k$ and message $m \in \{0,1\}$. It proceeds as follow.
  1. Samples a fresh FHE secret key $\mathsf{fhe.sk} \in \mathbb{Z}_q^t$ by running $\mathsf{FH.Keygen}(1^\lambda, 1^{d'}, 1^k)$.

2. Encrypt the input to obtain

$$\mathsf{fhe.ct} \leftarrow \mathsf{FH.Enc}(\mathsf{fhe.sk}, \mathbf{a}) \in \{0,1\}^{\ell}$$

3. Compute

$$\mathsf{ct}_{\mathsf{fhe.ct}} \leftarrow \mathsf{PH.Enc}\big(\mathsf{mpk}, (\mathsf{fhe.sk}, \mathsf{fhe.ct}), m\big)$$

   Note that the $\mathsf{fhe.sk}$ corresponds to the hidden attribute and $\mathsf{fhe.ct}$ corresponds to the public attribute.

4. Output the ciphertext $\mathsf{ct} = (\mathsf{ct}_{\mathsf{fhe.ct}}, \mathsf{fhe.ct})$.

- $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct})$: The decryption algorithm takes as input the secret key sk_C with corresponding predicate C and the ciphertext ct. If there exists $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$ such that

$$\mathsf{PH.Dec}(\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}, (\mathsf{ct}_{\mathsf{fhe.ct}}, \mathsf{fhe.ct})) = m \neq \perp$$

  then output m. Otherwise, output $\perp$.

4.6.2 Correctness

Lemma 4.6.1. *Let $\mathcal{C}$ be a family of predicates bounded by depth d and let $\mathcal{PHPE}$ be the partially-hiding PE and $\mathcal{FHE}$ be a fully-homomorphic encryption as per scheme description. Then, our predicate encryption scheme $\mathcal{PE}$ is correct according to Definition 4.2.1. Moreover, the size of each secret key is* $\mathrm{poly}(d, \lambda)$ *and the size of each ciphertext is* $\mathrm{poly}(d, \lambda, k)$.

Proof. Fix an arbitrary attribute vector $\mathbf{a}$ and a predicate C.

- If $C(\mathbf{a}) = 1$, we claim that decryption returns m with all but negligible probability. By the correctness of FHE decryption, we have

$$\langle \mathsf{fhe.sk}, \mathsf{FH.Eval}(\mathsf{fhe.ct}, C) \rangle = \rho \mod q$$

  for some scalar ρ in range $[\lfloor q/2 \rfloor - B, \lfloor q/2 \rfloor + B]$. Hence,

$$\mathsf{PH.Dec}((\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}, \widehat{C} \circ \mathsf{IP}_\gamma), (\mathsf{ct}_{\mathsf{fhe.ct}}, \mathsf{fhe.ct})) = \begin{cases} m & \text{if } \gamma = \rho \\ \perp & \text{otherwise} \end{cases}$$

  by the correctness of partially-hiding scheme.

- If $C(\mathbf{a}) = 0$, then by the correctness of FHE decryption

$$\langle \mathsf{fhe.sk}, \mathsf{FH.Eval}(\mathsf{fhe.ct}, C) \rangle = \rho \mod q$$

for a scalar ρ outside of range $[\lfloor q/2 \rfloor - B, \lfloor q/2 \rfloor + B]$. Hence, for all $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$,

$$\mathsf{PH.Dec}((\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}, \widehat{C} \circ \mathsf{IP}_\gamma), (\mathsf{ct}_{\mathsf{fhe.ct}}, \mathsf{fhe.ct})) = \perp$$

The correctness of the scheme follows. □

4.6.3 Security

Theorem 4.6.2. *Let $\mathcal{C}$ be a family of predicates bounded by depth d and let $\mathcal{PHPE}$ be the secure partially-hiding PE and $\mathcal{FHE}$ be the secure fully-homomorphic encryption as per scheme description. Then, our predicate encryption scheme $\mathcal{PE}$ is secure according to Definition 4.2.2.*

Proof. We define p.p.t. simulator algorithms $\mathsf{Enc}_{\mathrm{Sim}}$ and argue that its output is indistinguishable from the output of the real experiment. Let $\mathsf{PH.Enc}_{\mathrm{Sim}}$ be the p.p.t. simulator for partially-hiding predicate encryption scheme.

- $\mathsf{Enc}_{\mathrm{Sim}}(\mathsf{mpk}, 1^{|\mathbf{a}|}, 1^{|m|})$: To compute the encryption, the simulator does the following. It samples FHE secret key $\mathsf{fhe.sk}$ by running $\mathsf{FH.Keygen}(1^\lambda, 1^{d'}, 1^k)$. It encrypts a zero-string $\mathsf{fhe.ct} \leftarrow \mathsf{FH.Enc}(\mathsf{fhe.sk}, \mathbf{0})$. It obtains the ciphertext as

$$\mathsf{ct}_{\mathsf{fhe.ct}} \leftarrow \mathsf{PH.Enc}_{\mathrm{Sim}}(\mathsf{mpk}, \mathsf{fhe.ct}, 1^{|\mathsf{fhe.sk}|}, 1^{|m|}).$$

We now argue via a series of hybrids that the output of the ideal experiment.

- **Hybrid 0**: The real experiment.
- **Hybrid 1**: The real encryption algorithm is replaced with Enc^*, where Enc^* is an auxiliary algorithm defined below. On the high level, Enc^* computes the FHE ciphertext honestly by sampling a secret key and using the knowledge of $\mathbf{a}$. It then invokes $\mathsf{PH.Enc}_{\mathrm{Sim}}$ on the honestly generated ciphertext.
- **Hybrid 2**: The simulated experiment.

Auxiliary Algorithms. We define the auxiliary algorithm Enc^* used in Hybrid 1.

- $\mathsf{Enc}^*(\mathbf{a}, 1^{|m|})$: The auxiliary encryption algorithm takes as input the attribute vector $\mathbf{a}$ and message length.
  1. Sample a fresh FHE secret key $\mathsf{fhe.sk}$ by running $\mathsf{FH.Keygen}(1^\lambda, 1^{d'}, 1^k)$.
  2. Encrypt the input attribute vector to obtain a ciphertext

$$\mathsf{fhe.ct} \leftarrow \mathsf{FH.Enc}(\mathsf{fhe.sk}, \mathbf{a}) \in \{0,1\}^\ell$$

3. Run $\mathsf{PH.Enc}_{\text{Sim}}$ on input $(\mathsf{mpk}, \mathsf{fhe.ct}, 1^{|\mathsf{fhe.sk}|}, 1^{|m|})$ to obtain the ciphertext $\mathsf{ct}_{\mathsf{fhe.ct}}$.

Lemma 4.6.3. *The output of Hybrid 0 is computationally indistinguishable from the Hybrid 1, assuming security of Partially-Hiding Predicate Encryption.*

Proof. Assume there is an adversary Adv and a distinguisher $\mathcal{D}$ that distinguishes the output $(\mathbf{a}, m, \alpha)$ produced in either of the two hybrids. We construct an adversary Adv$'$ and a distinguisher $\mathcal{D}'$ that break the security of the Partially-Hiding Predicate Encryption. The adversary Adv$'$ does the following.

1. Invoke the adversary Adv to obtain an attribute vector $\mathbf{a}$.

2. Sample a fresh FHE secret key $\mathsf{fhe.sk}$ using $\mathsf{FH.Keygen}(1^\lambda, 1^{d'}, 1^k)$. Encrypt the attribute vector
$$\mathsf{fhe.ct} \leftarrow \mathsf{FH.Enc}(\mathsf{fhe.sk}, \mathbf{a})$$
and output the pair $(\mathsf{fhe.sk}, \mathsf{fhe.ct})$ as the "selective" challenge attribute.

3. Upon receiving mpk, it forwards it to Adv.

4. For each oracle query C that Adv makes which satisfies $C(\mathbf{a}) \neq 0$, Adv$'$ uses its oracle to obtain secret keys $\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}$ for $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$. It outputs $\mathsf{sk}_C = \left(\{\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}\}_{\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B}\right)$.

5. It outputs message m that Adv produces, obtains a ciphertext $\mathsf{ct}_{\mathsf{fhe.ct}}$ and sends $\mathsf{ct} = (\mathsf{ct}_{\mathsf{fhe.ct}}, \mathsf{fhe.ct})$ back to Adv to obtain α.

We note that given Adv that is admissible, Adv$'$ is also admissible. That is, for all queries $\widehat{C} \circ \mathsf{IP}_\gamma$ that Adv$'$ makes satisfies $(\widehat{C} \circ \mathsf{IP}_\gamma)(\mathsf{fhe.sk}, \mathsf{fhe.ct}) = 0$ since $\langle \mathsf{fhe.sk}, \widehat{C}(\mathsf{fhe.ct}) \rangle \neq \gamma$ for $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$ by the correctness of FHE in Section 4.2.5 and the fact that $C(\mathbf{a}) \neq 0$. Finally, the distinguisher $\mathcal{D}'$ on input $(\mathsf{fhe.sk}, \mathsf{fhe.ct}, m, \alpha)$ invokes $\mathcal{D}$ and outputs whatever it outputs. Now, in Hybrid 0 the algorithms used as $\mathsf{PH.Setup}, \mathsf{PH.KeyGen}, \mathsf{PH.Enc}$ which corresponds exactly to the real security game of PHPE. However, in Hybrid 1 the algorithms correspond exactly to the simulated security game. Hence, we can distinguish between the real and simulated experiments contradicting the security of PHPE scheme. $\square$

Lemma 4.6.4. *The output of Hybrid 1 and Hybrid 2 are computationally indistinguishable, assuming semantic security of Fully-Homomorphic Encryption Scheme.*

Proof. The only difference in Hybrids 1 and 2 is how the FHE ciphertext is produced. In one experiment, it is computed honestly by encrypting the attribute vector $\mathbf{a}$, while in the other experiment it is always an encryption of $\mathbf{0}$. Hence, we can readily construct an FHE adversary that given $\mathbf{a}$, distinguishes encryption of $\mathbf{a}$ from encryption of $\mathbf{0}$ as follows:

1. Invoke the admissible PE adversary Adv to obtain an attribute vector $\mathbf{a}$.

2. Run the honest $\mathsf{PH.Setup}$ and forwards mpk to Adv.

3. For each oracle query C that Adv makes which satisfies $C(\mathbf{a}) \neq 0$, return $\mathsf{sk}_C = \left(\{\mathsf{sk}_{\widehat{C} \circ \mathsf{IP}_\gamma}\}_{\gamma=\lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B}\right)$ as computed using the honest $\mathsf{PH.KeyGen}$ algorithm.

4. To simulate the ciphertext, first forward the pair $(\mathbf{a}, \mathbf{0})$ to the FHE challenger to obtain a ciphertext $\mathsf{fhe.ct}$. Then, run $\mathsf{PH.Enc}_{\mathrm{Sim}}(\mathsf{mpk}, \mathsf{fhe.ct}, 1^{|\mathsf{fhe.sk}|}, 1^m)$ to obtain a ciphertext $\mathsf{ct}_{\mathsf{fhe.ct}}$ and forward it to Adv

5. Finally, it runs the PE distinguisher on input $(\mathbf{a}, m, \alpha)$ and outputs its guess.

The lemma then follows from semantic security of the FHE. □

This completes the proof of the security of the scheme. □

We refer the reader to Section 4.6.4 for the summary of parameters selection.

4.6.4 Parameters Selection

We summarize the lattice parameters selection for our construction. First, we set the LWE dimension $n = \mathrm{poly}(\lambda)$ and the error distribution $\chi = \chi(n) = D_{\mathbb{Z},\sqrt{n}}$. Now, we set FHE secret key size $t = \mathrm{poly}(\lambda)$ and modulo $q = \tilde{O}(tnd)^{O(d)}$. To encrypt k-bit attribute vector and support FHE evaluation of arbitrary depth-d circuits, we set $\ell = \mathrm{poly}(k, d, \lambda, \log q)$ and $d' = \mathrm{poly}(d, \lambda, \log q)$. That is, the master public key, ciphertext and secret keys all have size $\mathrm{poly}(\lambda, k, d)$ and we achieve security under $\mathsf{LWE}_{n,q,\chi}$ where $q = 2^{\mathrm{poly}(d,n,\log k)}$ and the modulus-to-noise ratio is $2^{\mathrm{poly}(d,n,\log k)}$.

4.7 Conclusions and Open Problems

In this chapter, we presented a construction of predicate encryption for arbitrary circuits. The main limitation of our construction is the dependence on the depth. It remains open to remove this dependency – possibly by providing a general bootstrapping transformation. It remains an intriguing open problem to realize strong security notion (that is, realize indistinguishability-based functional encryption) from standard learning with errors problem. Additionally, it remains open to construct predicate encryption for other models of computation, such as Turing machines or RAM programs from standard assumption. It also remains to construct ciphertext-policy predicate encryption for circuits – where the secret keys are associated with attributes $\mathbf{a}$ and ciphertexts are associated with predicates P and messages m, and the size of the predicates is not a-priori bounded by the scheme.

Chapter 5

Graph-Induced Multilinear Maps

Cryptographic multilinear maps are an amazingly powerful tool: like homomorphic encryption schemes, they let us encode data in a manner that simultaneously hides it and permits processing on it. But they go even further and let us recover some limited information (such as equality) on the processed data without needing any secret key. Even in their simple bi-linear form (that only supports quadratic processing) they already give us pairing-based cryptography [jou04, SOK00, BF01], enabling powerful applications such as identity- and attribute-based encryption [BF01, Wat05, GPSW06], broadcast encryption [BGW05] and many others. In their general form, cryptographic multilinear maps are so useful that we had a body of work examining their applications even before we knew of any candidate constructions to realize them [BS03, RS09, PTT10, Rot13].

Formally, a non-degenerate map between order-q algebraic groups, $e : G^d \to G_T$, is $d-$multilinear if for all $a_1, \ldots, a_d \in \mathbb{Z}_q$ and $g \in G$,

$$e(g^{a_1}, \ldots, g^{a_d}) = e(g, \ldots, g)^{a_1 \cdot \ldots \cdot a_d}.$$

We say that the map e is "cryptographic" if we can evaluate it efficiently and at least the discrete-logarithm in the groups G, G_T is hard.

In a recent breakthrough, Garg, Gentry and Halevi [GGH13a] gave the first candidate construction of multilinear maps from ideal lattices, followed by a second construction by Coron, Lepoint and Tibouchi [CLT13] over the integers. (Some optimizations to the GGH scheme were proposed in [LSS14]). Due to certain differences between their construction and "ideal" multilinear maps, Garg et al. (and Coron et al.) called their constructions "graded encoding schemes." These graded encoding schemes realize an approximate version of multilinear maps with no explicit algebraic groups, where the transformation $a \mapsto g^a$ is replaced by some (randomized) encoding function.

Moreover, these constructions are "graded", in the sense that they allow intermediate computation. One way to think of these intermediate computations is as a sequence of levels (or groups) $G_1, \ldots, G_d$ and a set of maps e_{ij} such that for all $g_i^a \in G_i, g_j^b \in G_j$ (satisfying $i + j \leq d$), $e_{ij}(g_i^a, g_j^b) = g_{i+j}^{ab}$. Asymmetric variant of graded multilinear maps provides additional structure on how these encodings can be combined. Each encoding is assigned with a set of levels $S \subseteq [N]$. Given two encodings $g_S^a, g_{S'}^b$ the map allows to compute $g_{S\cup S'}^{ab}$ only if $S \cap S' = \emptyset$.

Both [GGH13a] and [CLT13] constructions begin from some variant of homomorphic encryption and use public-key encryption as the encoding method. The main new ingredient, however, is that they also publish a defective version of the secret key, which cannot be used for decryption but can be used to test if a ciphertext encrypts a zero. (This defective key is called the "zero-test parameter".) Over the last two years, the applications of (graded) multilinear maps have expanded much further, supporting applications such as witness encryption, general-purpose obfuscation, functional encryption, and many more [GGSW13, GGH$^+$13c, GGH$^+$13b, BGG$^+$14, BZ14].

5.1 Our Contributions, Techniques and Applications

We present a new "graph-induced" variant of multilinear maps. In this variant, the multilinear map is defined with respect to a directed acyclic graph. Namely, encoded value are associated with paths in the graph, and it is only possible to add encoding relative to the same paths, or to multiply encodings relative to "connected paths" (i.e., one ends where the other begins) Our candidate construction of graph-induced multilinear maps does not rely on ideal lattices or hard-to-factor integers. Rather, *we use standard random lattices* such as those used in LWE-based cryptography. We follow a similar outline to the previous constructions, except our instance generation algorithm takes as input a description of a graph. Furthermore, our zero-tester *does not* include any secrets about the relevant lattices. Rather, in our case the zero-tester is just a random matrix, similar to a *public key* in common LWE-based cryptosystems.

Giving up the algebraic structure of ideal lattices and integers could contribute to a better understanding of the candidate itself, reducing the risk of unforeseen algebraic crypt-analytical attacks. On the flip side, using our construction is sometimes harder than previous construction, exactly because we give up some algebraic structure. For that same reason, we were not able so far to reduce any of our new construction to "nice" hardness assumptions, currently they are all just candidate constructions, that withstood our repeated cryptanalytic attempts at breaking them. Still we believe that our new construction is a well needed addition to our cryptographic toolbox, providing yet another avenue for implementing multilinear maps.

Our Techniques Our starting point is the new homomorphic encryption (HE) scheme of Gentry, Sahai and Waters [GSW13]. The secret key in that scheme is a vector $\mathbf{a} \in \mathbb{Z}_q^m$, and a ciphertext encrypting $m \in \mathbb{Z}_q$ is a matrix $\mathbf{C} \in \mathbb{Z}_q^{m\times m}$ with small entries such that $\mathbf{C} \cdot \mathbf{a} = m \cdot \mathbf{a} + \mathbf{e}$ for some small error vector $\mathbf{e}$. In other words, valid ciphertexts all have the secret key $\mathbf{a}$ as an "approximate eigenvector", and the eigenvalue is the message. Given the secret eigenvector $\mathbf{a}$, decoding arbitrary m's becomes easy.

This HE scheme supports addition and multiplication, but we also need a *public* equivalent of the approximate eigenvector for zero-testing. The key idea is to replace the "approximate eigenvector" with an "approximate eigenspace" by increasing the dimensions. Instead of having a single approximate eigenvectors, our "approximate eigenspace" is described by n vectors $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$. The approximate eigenvalues will not merely be elements of $\mathbb{Z}_q$, but rather matrices $\mathbf{S} \in \mathbb{Z}_q^{n\times n}$ with small entries. An encoding of $\mathbf{S}$ is a matrix $\mathbf{C} \in \mathbb{Z}^{m\times m}$ with small entries such that

$$\mathbf{C}^T \cdot \mathbf{A}^T = \mathbf{A}^T \cdot \mathbf{S} + \mathbf{E}$$

for small noise matrix $\mathbf{E} \in \mathbb{Z}_q^{m\times n}$. In other words, $\mathbf{C}$ is a matrix that maps any column vector in $\mathbf{A}$ to a vector that is very close to the span of $\mathbf{A}$. In that sense, $\mathbf{A}$ is an approximate eigenspace. In the HE scheme, $\mathbf{a}$ was a secret key that allowed us to easily recover m. However, for the eigenspace setting, assuming $\mathbf{A}$ is just a uniformly random matrix and $\mathbf{S}$ is a random small matrix, $\mathbf{A}^T \cdot \mathbf{S} + \mathbf{E}$ is an LWE instance that looks uniform even when given $\mathbf{A}$.

Overview of Our Construction. Our construction is parametrized by a directed acyclic graph $G = (V, E)$. For each node $v \in V$, we assign a random matrix $\mathbf{A}_v \in \mathbb{Z}_q^{n\times m}$. Any path $u \rightsquigarrow v$ (which can be a single edge) can be assigned with an encoding $\mathbf{D} \in \mathbb{Z}_q^{m\times m}$ of some plaintext secret $\mathbf{S} \in \mathbb{Z}_q^{n\times n}$ satisfying

$$\mathbf{D}^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{E} \tag{5.1}$$

for some small error $\mathbf{E} \in (\chi)^{m\times n}$.

Adding and multiplying encodings corresponds to addition and multiplication of matrices. Addition of encodings can only be performed relative to the same path $u \rightsquigarrow v$. For example, given encodings $\mathbf{D}_1, \mathbf{D}_2$ at path $u \rightsquigarrow v$, we have that:

$$(\mathbf{D}_1^T + \mathbf{D}_2^T) \cdot \mathbf{A}_u^T \;\approx\; \mathbf{A}_v^T \cdot \mathbf{S}_1 + \mathbf{A}_v^T \cdot \mathbf{S}_2 \;=\; \mathbf{A}_v^T \cdot (\mathbf{S}_1 + \mathbf{S}_2).$$

Multiplication of encodings can only be performed when they form a complete path. That is, given encodings $\mathbf{D}_1$ and $\mathbf{D}_2$ relative to paths $u \rightsquigarrow v$ and $v \rightsquigarrow w$ respectively, we have:

$$\begin{aligned} \mathbf{D}_2^T \cdot \mathbf{D}_1^T \cdot \mathbf{A}_u^T &= \mathbf{D}_2^T \cdot (\mathbf{A}_v^T \cdot \mathbf{S}_1 + \mathbf{E}_1) \\ &= (\mathbf{A}_w^T \cdot \mathbf{S}_2 + \mathbf{E}_2) \cdot \mathbf{S}_1 + \mathbf{D}_2^T \cdot \mathbf{E}_1 \;=\; \mathbf{A}_w^T \cdot \mathbf{S}_2 \cdot \mathbf{S}_1 + \underbrace{\mathbf{E}_2 \cdot \mathbf{S}_1 + \mathbf{D}_2^T \cdot \mathbf{E}_1}_{\mathbf{E}'} \end{aligned} \tag{5.2}$$

where $\mathbf{E}'$ is small since the errors and matrices $\mathbf{S}_1, \mathbf{D}_2$ have small entries. Furthermore, it is possible to compare two encodings with the same sink node. That is, given $\mathbf{D}_1$ and $\mathbf{D}_2$ relative to paths $u \rightsquigarrow v$ and $w \rightsquigarrow v$, it is sufficient to check if $\mathbf{D}_1^T \cdot \mathbf{A}_u^T - \mathbf{D}_2^T \cdot \mathbf{A}_w^T$ is small since if $\mathbf{S}_1 = \mathbf{S}_2$, then we have

$$\mathbf{D}_1^T \cdot \mathbf{A}_u^T - \mathbf{D}_2^T \cdot \mathbf{A}_w^T \;=\; (\mathbf{A}_v^T \cdot \mathbf{S}_1 + \mathbf{E}_1) - (\mathbf{A}_v^T \cdot \mathbf{S}_2 + \mathbf{E}_2) \;=\; \mathbf{E}_1 - \mathbf{E}_2 \tag{5.3}$$

Hence, the random matrices $\mathbf{A}_u, \mathbf{A}_w \in \mathbb{Z}_q n \times m$, which are commonly available in the public parameters, is sufficient for comparison and zero-testing.

As we explain in 5.4, generating the encoding matrices requires knowing a trapdoor for the matrices $\mathbf{A}_i$. But for the public-sampling setting, it is possible to generate encodings of many random matrices during setup, and later anyone can take a random linear combinations of them to get "fresh" random encodings.

We remark that since $\mathbf{S}$ needs to be small in Eqn. (5.2), our scheme only supports encoding of *small plaintext elements*, as opposed to arbitrary plaintext elements as in previous schemes.[1] Another difference is that in the basic construction our plaintext space is a non-commutative ring (i.e. square matrices). We extend to the commutative setting in 5.4.2.

Variations and parameters. We also describe some variations of the basic scheme above, aimed at improving the parameters or offering different trade-offs. One standard way of improving parameters is to switch to a ring-LWE setting, where scalars are taken from a large polynomial ring (rather than being just integers), and the dimension of vectors and matrices is reduced proportionally. In our context, we can also use the same approach to move to a commutative plaintext space, see 5.4.2.

5.1.1 Applications

Our new constructions support many of the known cryptographic uses of graded encoding. Here we briefly sketch two of them.

Non-interactive Multipartite Key-Exchange. Consider k-partite key-exchange. We design a graph in a star topology with k-branches each of length $k-1$ nodes. All branches meet at the common sink node $\mathbf{A}_0$. For each branch i, we associate encodings of small LWE secrets $t_1, \ldots, \ldots, t_k$ in a specific order. The public parameters consists of many such plaintext values t_is and their associated encodings. Each player j takes a random linear combination of these encodings. It stores one of the encodings along the path as the secret key and broadcasts the rest of to other players. Assume some canonical ordering of the players. Each player computes the $k-1$ product of the other players' encodings along the path with index j and its own secret encoding. This yields an encoding $\mathbf{D}$ of $\mathbf{T}^* = \prod_{i \in [k]} s_i$, satisfying

$$\mathbf{D}^T \cdot \mathbf{A}_{j,1}^T = \mathbf{A}_0^T \cdot \prod_{i \in [k]} s_i + \text{noise}$$

And the players obtain the shared secret key by applying a randomness extractor on the most significant bits.

Branching-program obfuscation. Perhaps the "poster application" of cryptographic graded encodings is to obtain general-purpose obfuscation [GGH$^+$13c, BR14a, BGK$^+$14, PST14, GLSW14], with the crucial step being the use of graded encoding to obfuscate

[1]The only exception is that the leftmost plaintext matrix $\mathbf{S}$ in a product could encode a large element, as Eqn. (5.2) is not affected by the size of $\mathbf{S}_1$. Similarly the rightmost encoding matrix $\mathbf{D}$ in a product need not be small. We do not use these exceptions in the current paper, however.

branching programs. These branching programs are represented as a sequence of pairs of encoded matrices, and the user just picks one matrix from each pair and then multiply them all in order.

This usage pattern of graded encoding fits very well into our graph-induced scheme since these matrices are given in a pre-arranged order. We describe a candidate obfuscation construction from our multilinear map based on a path graph. Informally, to obfuscate a length-L matrix branching program $\{\mathbf{B}_{i,b}\}$, we first perform Kilian's randomization and then encode values $\mathbf{R}_{i-1}^{-1}\mathbf{B}_{i,0}\mathbf{R}_i$ and $\mathbf{R}_{i-1}^{-1}\mathbf{B}_{i,1}\mathbf{R}_i$ relative to the edge i. The user can then compute an encoding of a product of matrices corresponding to its input. If the product $\prod_{i\in[L]}\mathbf{B}_{i,x_{\mathsf{vari}}} = \mathbf{I}$, then the user obtains an encoding $\mathbf{D}$ satisfying:

$$\mathbf{D}^T \cdot \mathbf{A}_0^T = \mathbf{A}_L^T \cdot \mathbf{I} + \mathsf{noise}$$

Given $\mathbf{A}_L^T \cdot \mathbf{I} + \mathsf{noise}'$ in the public parameters (or its encoding), the user can then learn the result of the computation by a simple comparison. We note that our actual candidate construction is more involved as we deploy additional safeguards from the literature (See Section 5.6.2).

5.1.2 Chapter Organization

In Section 5.2, we provide some background and present the syntax of graph-induced multilinear maps. In Section 5.4, we describe our basic construction in the non-commutative variant. In Subsection 5.4.2 we show how to extend our basic construction to commutative variant. In Section 5.5, we analyze the security of our construction. In Section 5.6 we present applications of our construction to key-exchange and obfuscation.

5.2 Preliminaries

Extractors. An efficient (n, m, ℓ, ϵ)-strong extractor is a poly-time algorithm $\mathsf{Extract} : \{0,1\}^n \to \{0,1\}^\ell$ such that for any random variable W over $\{0,1\}^n$ with min-entropy m, it holds that the statistical distance between $(\mathsf{Extract}_\alpha(W), \alpha)$ and (U_ℓ, α) is at most ϵ. Here, α denotes the random bits used by the extractor. Universal hash functions [CW79, WC81] can extract $\ell = m - 2\log\frac{1}{\epsilon} + 2$ nearly random bits, as given by the leftover hash lemma [HILL99]. This will be sufficient for our applications.

Leftover Hash Lemma Over Gaussians Recent works [AGHS13, AR13] considered the setting where the columns of a matrix $\mathbf{X} \in \mathbb{Z}^{t\times k}$ are drawn independently from a "wide enough" Gaussian distribution over a lattice $L \subset \mathbb{Z}^t$, $\mathbf{x}_i \leftarrow D_{L,S}$. Once these columns are fixed, we consider the distribution $\mathcal{D}_{\mathbf{X},\sigma}$, induced by choosing an integer vector $\mathbf{r}$ from a discrete spherical Gaussian over $\mathbb{Z}^t$ with parameter σ and outputting $y = \mathbf{X}^T\mathbf{r}$, $\mathcal{D}_{\mathbf{X},\sigma} := \{\mathbf{X}^T\mathbf{r} \ : \ \mathbf{r} \leftarrow D_{\mathbb{Z}^t,\sigma}\}$. It turns out that with high probability over the choice of $\mathbf{X}$, the distribution $\mathcal{D}_{\mathbf{X},\sigma}$ is statistically close to ellipsoid Gaussian $D_{L,\sigma\mathbf{X}}$ (and moreover the singular values of $\mathbf{X}$ are of size roughly $\sigma\sqrt{t}$).

Theorem 5.2.1 ([AGHS13, AR13]). *For integers $k \geq 1, t = \mathrm{poly}(k), \sigma = \Omega(\sqrt{\log(k/\epsilon)})$ and $\sigma' = \tilde{\Omega}(k\sigma\sqrt{\log(1/\epsilon)})$, we have that with probability $1 - 2^{-k}$ over the choice $\mathbf{X} \leftarrow (\mathcal{D}_{\mathbb{Z}^k,\sigma})^t$ that the statistical distance between $\mathcal{D}_{\mathbf{X},\sigma'}$ and $D_{\mathbb{Z}^k,\sigma'\mathbf{X}^T}$ is smaller than ϵ.*

5.3 Graded Multilinear Encodings

The notion of graded encoding scheme that we relaize is similar (but not exactly identical) to the GGH notion from [GGH13a]. Very roughly, a graded encoding scheme for an algebraic "plaintext ring R" provides methods for encoding ring elements and manipulating these encodings. Namely we can sample random plaintext elements together with their encoding, can add and multiply encoded elements, can test if a given encoding encodes zero, and can also extract a "canonical representation" of a plaintext element from an encoding of that element.

Syntax of Graph-Induced Graded Encoding Schemes

There are several variations of graded-encoding systems in the literature, such as public/secret encoding, with/without re-randomization, symmetric/asymmetric, etc. Below we define the syntax for our scheme, which is still somewhat different than all of the above. The main differences are that our encodings are defined relative to edges of a directed graph (as opposed to levels/sets/vectors as in previous schemes), and that we only encode "small elements" from the plaintext space. Below we provide the relevant definitions, modifying the ones from [GGH13a].

Definition 5.3.1 (Graph-Induced Encoding Scheme). *A graph-based graded encoding scheme with secret sampling consists of the following (polynomial-time) procedures, $\mathcal{G}_{\mathsf{es}} = (\mathsf{PrmGen}, \mathsf{InstGen}, \mathsf{Sample}, \mathsf{Encode}, \mathsf{add}, \mathsf{neg}, \mathsf{mult}, \mathsf{ZeroTest}, \mathsf{Extract})$:*

- $\mathsf{PrmGen}(1^\lambda, G, \mathcal{C})$*: The parameter-generation procedure takes the security parameter λ, underlying directed graph $G = (V, E)$, and the class $\mathcal{C}$ of supported circuits. It outputs some global parameters of the system* gp*, which includes in particular the graph G, a specification of the plaintext ring R and also a distribution χ over R.*

  For example, in our case the global parameters consists of the dimension n of matrices, the modulus q and the Gaussian parameter σ.

- $\mathsf{InstGen}(\mathsf{gp})$*: The randomized instance-generation procedure takes the global parameters* gp*, and outputs the public and secret parameters* sp, pp.

- $\mathsf{Sample}(\mathsf{pp})$*: The sampling procedure samples an element in the the plaintext space, according to the distribution χ.*

- $\mathsf{Encode}(\mathsf{sp}, p, \alpha)$*: The encoding procedure takes the secret parameters* pp*, a path* $p = u \rightsquigarrow v$ *in the graph, and an element* $\alpha \in R$ *from the support of the* Sample *procedure, and outputs an encoding* u_p *of* α *relative to* p*.* [2]

- $\mathsf{neg}(\mathsf{pp}, u)$*,* $\mathsf{add}(\mathsf{pp}, u, u')$*,* $\mathsf{mult}(\mathsf{pp}, u, u')$*. The arithmetic procedures are deterministic, and they all take as input the public parameters and use them to manipulate encodings.*

  Negation takes an encoding of $\alpha \in R$ *relative to some path* $p = u \rightsquigarrow v$ *and outputs encoding of* $-\alpha$ *relative to the same path. Addition takes* u, u' *that encode* $\alpha, \alpha' \in R$ *relative to the same path* p*, and outputs an encoding of* $\alpha + \alpha$ *relative to* p*. Multiplication takes* u, u' *that encode* $\alpha, \alpha' \in R$ *relative to consecutive paths* $p = u \rightsquigarrow v$ *and* $p' = v \rightsquigarrow w$*, respectively. It outputs an encoding of* $\alpha \cdot \alpha'$ *relative to the combined path* $u \rightsquigarrow w$*.*

- $\mathsf{ZeroTest}(\mathsf{pp}, u)$*: Zero testing is a deterministic procedure that takes the public parameters* pp *and an encoding* u *that is tagged by its path* p*. It outputs 1 if* u *is an encoding of zero and 0 if it is an of a non-zero element.*

- $\mathsf{Extract}(\mathsf{pp}, u)$*: The extraction procedure takes as input the public parameters* pp *and an encoding* u *that is tagged by its path* p*. It outputs a* λ*-bit string that serves as a "random canonical representation" of the underlying plaintext element* α *(see below).*

Correctness. The graph G, in conjunction with the procedures for sampling, encoding, and arithmetic operations, and the class of supported circuits, implicitly define the set S_G of "valid encodings" and its partition into sets $S_G^{(\alpha)}$ of "valid encoding of α".

Namely, we consider arithmetic circuits whose wires are labeled by paths in G in a way that respects the permitted operations of the scheme (i.e., negation and addition have all the same labels, and multiplication has consecutive input paths and the output is labeled by their concatenation). Then S_G consists of all the encoding that can be generated by using the sampling/encoding procedures to sample plaintext elements and compute their encoding, then compute the operations of the scheme according to Π, and collect the encoding at the output of Π. An encoding $u \in S_G$ belongs to $S_G^{(\alpha)}$ is there exists such circuit Π and inputs for which Π outputs α when evaluated on plaintext elements. Of course, to be useful we require that the sets $S_G^{(\alpha)}$ form a partition of S_G.

We can also sub-divide each $S_G^{(\alpha)}$ into $S_p^{(\alpha)}$ for different paths p in the graph, depending on the label of the output wire of Π (but here it is not important that these sets are disjoint), and define $S_p = \bigcup_{\alpha \in R} S_p^{(\alpha)}$.

Note that the sets $S_p^{(\alpha)}$ can be empty, for example in our construction the sampling procedure only outputs "small" plaintext values α, so a "large" β would have $S_p^{(\beta)} = \emptyset$. Below we denote the set of α's with non-empty encoding sets (relative to path p) by $\mathsf{SMALL}_p \overset{\text{def}}{=} \{\alpha \in R : S_p^{(\alpha)} \neq \emptyset\}$, and similarly $\mathsf{SMALL}_G \overset{\text{def}}{=} \{\alpha \in R : S_G^{(\alpha)} \neq \emptyset\}$.

[2] See the description below for the meaning of "u_p is an encoding of α relative to p", formally u_p is just a bit string, which is tagged with its path p.

We assume for simplicity that the sets SMALL *depend only on the global parameters* gp *and not the specific parameters* sp, pp. (This assumption holds for our construction and it simplifies the syntax below.)

We can now state the correctness conditions for zero-testing and extraction. For zero-testing we require that $\mathsf{ZeroTest}(\mathsf{pp}, u) = 1$ for every $u \in S^{(0)}$ (with probability one), and for every $\alpha \in \mathsf{SMALL}_G$, $\alpha \neq 0$ it holds with overwhelming probability over instance-generation that $\mathsf{ZeroTest}(\mathsf{pp}, u) = 0$ for every encoding $u \in S_G^{(\alpha)}$.

For extraction,we roughly require that $\mathsf{Extract}$ outputs the same string on all the encodings of the same α, different strings on encodings of different α's, and random strings on encodings of "random α's." Formally, we require the following for any global parameters gp output by PrmGen:

- For any plaintext element $\alpha \in \mathsf{SMALL}_G$ and path p in G, with overwhelming probability over the parameters $(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(\mathsf{gp})$, there exists a single value $x \in \{0,1\}^\lambda$ such that $\mathsf{Extract}(\mathsf{pp}, u) = x$ holds for all $u \in S_p^{(\alpha)}$.
- For any $\alpha \neq \alpha' \in \mathsf{SMALL}_G$ and path p in G, it holds with overwhelming probability over the parameters $(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(\mathsf{gp})$ that for any $u \in S_p^{(\alpha)}$ and $u' \in S_p^{(\alpha')}$, $\mathsf{Extract}(\mathsf{pp}, u) \neq \mathsf{Extract}(\mathsf{pp}, u')$.
- For any path p in G and distribution $\mathcal{D}$ over SMALL_p with min-entropy 3λ or more, it holds with overwhelming probability over the parameters $(\mathsf{sp}, \mathsf{pp}) \leftarrow \mathsf{InstGen}(\mathsf{gp})$ that the induced distribution $\{\mathsf{Extract}(\mathsf{pp}, u) : \alpha \leftarrow \mathcal{D}, u \in S_d^{(\alpha)}\}$ is nearly uniform over $\{0,1\}^\lambda$.

In some applications these conditions can be weakened. For example we often only need them to hold for some paths in G rather than all of them (e.g., we only care about source-to-sink paths).

5.3.1 Variations

Public sampling of encoded elements. One useful variation allows a public sampling procedure that takes as input pp rather than sp and outputs both a plaintext α and its encoding u_p relative to some path p. In many cases it is easy to go from secret-encoding to public sampling. Specifically, given a scheme that supports secret encoding we can augment the instance-generation procedure by sampling many tuples (α_i, u_i) relative to relevant paths (e.g., the edges in G) and adding them to the public parameters. Then a public sampling procedure can just use a subset sum of these tuples as a new sample, which would have some other distribution χ'.

If the distribution χ of the secret sampling procedure was uniform over R, then by the leftover hash lemma so is the distribution χ' of the public sampling procedure. Similarly, if χ was a Gaussian then using the Gaussian leftover-lemma Theorem 5.2.1 also χ' is a Gaussian (with somewhat different parameters).

In our construction we have a Gaussian distribution χ, so we can use this method to transform our scheme to one with a public sampling procedure.

Re-randomization. In some cases one may want to re-randomize a given encoding with changing the encoded value or the path in G, or to compute given a plaintext element the corresponding encoding relative to come path. Our construction does not support re-randomization (see 5.5).

5.4 Our Graph-Induced Multilinear Maps

The plaintext space in our basic scheme is the non-commutative ring of matrices $R = \mathbb{Z}_q^{n \times n}$, later in 5.4.2 we describe a commutative variant. In this section we only deal with correctness of these schemes, their security is discussed in Section 5.5.

As sketched in the introduction, for the basic scheme we have an underlying directed acyclic graph $G = (V, E)$, we identify a random matrix $\mathbf{A}_v \in \mathbb{Z}_q^{n \times m}$ with each node $v \in V$, and encodings in the scheme are defined relative to paths. A small plaintext matrix $\mathbf{S} \in R$ is encoded wrt to the path $u \rightsquigarrow v$ via another small matrix $\mathbf{D} \in \mathbb{Z}_q^{m \times m}$ such that $\mathbf{D}^T \cdot \mathbf{A}_u^T \approx \mathbf{A}_v^T \cdot \mathbf{S}$. In more detail, we have the following graded encoding scheme $\mathcal{G}_{\mathsf{es}} = (\mathsf{PrmGen}, \mathsf{InstGen}, \mathsf{Sample}, \mathsf{Encode}, \mathsf{add}, \mathsf{neg}, \mathsf{mult}, \mathsf{ZeroTest}, \mathsf{Extract})$:

- $\mathsf{PrmGen}(1^\lambda, G, \mathcal{C})$: On input the security parameter λ, an underlying DAG $G = (V, E)$, and class $\mathcal{C}$ of supported circuits, we compute:

  1. LWE parameters n, m, q and error distribution $\chi = D_{\mathbb{Z},s}$.
  2. A Gaussian parameters σ for SamPre.
  3. Another parameter t for the number of most significant bits used for zero-test and extraction.

  The constraints that dictate these parameters are described in Appendix 4.5.3. The resulting parameters for a DAG of diameter d are $n = \Theta(d\lambda \log(d\lambda))$, $q = (d\lambda)^{\Theta(d)}$, $m = \Theta(nd \log q)$, $s = \sqrt{n}$, $\sigma = \sqrt{n(d+1)\log q}$, and $t = \lfloor (\log q)/4 \rfloor - 1$. These global parameters gp (including the graph G) are given to all the procedures below.

- $\mathsf{InstGen}(\mathsf{gp})$: Given the global parameters, instance-generation proceeds as follows:

  1. Use trapdoor-sampling to generate $|V|$ matrices with trapdoors, one for each node.
  $$\forall v \in V, \quad \big(\mathbf{A}_v, \tau_v\big) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$$
  2. Choose the randomness-extractor seed β from a pairwise-independent function family, and a uniform "shift matrix" $\mathbf{\Delta} \in \mathbb{Z}_q^{m \times n}$.

  The public parameters are $\mathsf{pp} := \big(\{\mathbf{A}_v : v \in V\}, \beta, \mathbf{\Delta}\big)$ and the secret parameters include also the trapdoors $\{\tau_v : v \in V\}$.

- $\mathsf{Sample}(\mathsf{pp})$: This procedure just samples an LWE secret $\mathbf{S} \leftarrow (\chi)^{n \times n}$ as the plaintext.

- $\mathsf{Encode}(\mathsf{sp}, p, \mathbf{S})$: On input the matrices $\mathbf{A}_u, \mathbf{A}_v$, the trapdoor τ_u, and the small matrix $\mathbf{S}$, sample an LWE error matrix $\mathbf{E}_i \leftarrow (\chi)^{m\times n}$, set $\mathbf{V} = \mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{E} \in \mathbb{Z}_q^{m\times n}$, and then use the trapdoor τ_u to compute the encoding $\mathbf{D}_p$ s.t. $\mathbf{D}_p^T \cdot \mathbf{A}_u^T = \mathbf{V}$, $\mathbf{D}_p \leftarrow \mathsf{SamPre}(\mathbf{A}_u, \tau_u, \mathbf{V}, \sigma)$. The output is the plaintext S and encoding $\mathbf{D}_p$.

- The arithmetic operations are just matrix operations in $\mathbb{Z}_q^{m\times m}$:

  $$\mathsf{neg}(\mathsf{pp}, \mathbf{D}) := -\mathbf{D}, \quad \mathsf{add}(\mathsf{pp}, \mathbf{D}, \mathbf{D}') := \mathbf{D} + \mathbf{D}', \quad \text{and} \quad \mathsf{mult}(\mathsf{pp}, \mathbf{D}, \mathbf{D}') := \mathbf{D}' \cdot \mathbf{D}.$$

  To see that negation and addition maintain the right structure, let $\mathbf{D}, \mathbf{D}' \in \mathbb{Z}_q^{m\times m}$ be two encodings reltive to the same path $u \rightsquigarrow v$. Namely $\mathbf{D}^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{E}$ and $\mathbf{D}'^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot \mathbf{S}' + \mathbf{E}'$, with the matrices $\mathbf{D}, \mathbf{D}', \mathbf{E}, \mathbf{E}', \mathbf{S}, \mathbf{S}'$ all small. Then we have

  $$\begin{array}{rcl} -\mathbf{D}^T \cdot \mathbf{A}_u^T &=& \mathbf{A}_v^T \cdot (-\mathbf{S}) + (-\mathbf{E}), \\ \text{and } (\mathbf{D} + \mathbf{D}')^T \cdot \mathbf{A}_u &=& (\mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{E}) + (\mathbf{A}_v^T \cdot \mathbf{S}' + \mathbf{E}') \;=\; \mathbf{A}_v^T \cdot (\mathbf{S} + \mathbf{S}') + (\mathbf{E} + \mathbf{E}'), \end{array}$$

  and all the matrices $-\mathbf{D}, -\mathbf{S}, -\mathbf{E}$, $\mathbf{D} + \mathbf{D}'$, $\mathbf{S} + \mathbf{S}'$, $\mathbf{E} + \mathbf{E}'$ are still small. For multiplication, consider encodings $\mathbf{D}, \mathbf{D}'$ relative to paths $v \rightsquigarrow w$ and $u \rightsquigarrow v$, respectively, then we have

  $$\begin{array}{rcl} (\mathbf{D}' \cdot \mathbf{D})^T \cdot \mathbf{A}_u^T &=& \mathbf{D}^T \cdot \big(\mathbf{A}_v^T \cdot \mathbf{S}' + \mathbf{E}'\big) \\ &=& \big(\mathbf{A}_w^T \cdot \mathbf{S} + \mathbf{E}\big) \cdot \mathbf{S}' + \mathbf{D}^T \cdot \mathbf{E}' \;=\; \mathbf{A}_w \cdot (\mathbf{S} \cdot \mathbf{S}') + \underbrace{(\mathbf{E} \cdot \mathbf{S}' + \mathbf{D}^T \cdot \mathbf{E}')}_{\mathbf{E}''}, \end{array}$$

  and the matrices $\mathbf{D} \cdot \mathbf{D}'$, $\mathbf{S} \cdot \mathbf{S}'$, and $\mathbf{E}''$ are still small.

  Of course, the matrices $\mathbf{D}, \mathbf{S}, \mathbf{E}$ all grow with arithmetic operations, but our parameter-choice enures that for any encoding relative to any path in the graph $u \rightsquigarrow v$ (of length $\leq d$) we have $\mathbf{D}^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{E}$ where $\mathbf{E}$ is still small, specifically $\|\mathbf{E}\| < q^{3/4} \leq q/2^{t+1}$.

- $\mathsf{ZeroTest}(\mathsf{pp}, \mathbf{D})$. Given an encoding $\mathbf{D}$ relative to path $u \rightsquigarrow v$ and the matrix $\mathbf{A}_u$, our zero-test procedure outputs 1 if and only if $\|\mathbf{D}^T \cdot \mathbf{A}_u^T\| < q/2^{t+1}$.

- $\mathsf{Extract}(\mathsf{pp}, \mathbf{D})$: Given an encoding $\mathbf{D}$ relative to path $u \rightsquigarrow v$, the matrix $\mathbf{A}_u$ and shift-matrix $\mathbf{\Delta}$, and the extrator seed β, we compute $\mathbf{D}^T \cdot \mathbf{A}_0^T + \mathbf{\Delta}$, collect the t most-significant bits from each entry (when mapped to the interval $[0, q-1]$), and apply the randomness extractor, outputting

  $$w := \mathsf{RandExt}_\beta\big(\mathsf{msb}_t(\mathbf{D}^T \cdot \mathbf{A}_u^T + \mathbf{\Delta})\big)$$

5.4.1 Correctness

Correctness of the scheme follows from our invariant, which says that encoding of some plaintext matrix $\mathbf{S}$ relative to any path $u \rightsquigarrow v$ of legnth $\leq d$ satisfies $\mathbf{D}^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{E}$

for $\|\mathbf{E}\| < q/2^{t+1}$.

Correctness of Zero-Test. An encoding of zero satisfies $\mathbf{D}^T \cdot \mathbf{A}_u = \mathbf{E}$, hence $\|\mathbf{D}^T \cdot \mathbf{A}_u^T\| < q/2^{t+1}$. On the other hand, since $\mathbf{A}_v$ is uniform then for any nonzero $\mathbf{S}$ we only get $\|\mathbf{A}_v^T \cdot \mathbf{S}\| \leq q/2^t$ with exponentially small probability, and since $\|\mathbf{E}\| < q/2^{t+1}$ then

$$\|\mathbf{D}^T \cdot \mathbf{A}_u^T\| \geq \|\mathbf{A}_v^T \cdot \mathbf{S}\| - \|\mathbf{E}\| > q/2^t - q/2^{t+1} \geq q/2^{t+1}.$$

Hence with overwhelming probability over the choise of $\mathbf{A}_v$, our zero-test will output 0 on *all* the encoding of $\mathbf{S}$.

Correctness of Extraction. We begin by proving that for any plaintext matrix $\mathbf{S}$ and any encoding $\mathbf{D}$ of $\mathbf{S}$ (relative to $u \rightsquigarrow v$), with overwhelming probability over the parameters we have that $\mathsf{msb}_t(\mathbf{D}^T \cdot \mathbf{A}_u^T + \mathbf{\Delta}) = \mathsf{msb}_t(\mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{\Delta})$.

Since the two matrices $\mathbf{M} = \mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{\Delta}$ and $\mathbf{M}' = \mathbf{D}^T \cdot \mathbf{A}_u + \mathbf{\Delta}$ differ in each entry by at most $q/2^{t+1}$ modulo q, they can only differ in their top t bits due to the mod-q reduction, i.e., if for some entry we have $[\mathbf{M}]_{k,\ell} \approx 0$ but $[\mathbf{M}']_{k,\ell} \approx q$ or the other way around. (Recall that here we reduce mod-q into the interval $[0, q-1]$.) Clearly, this only happens when $\mathbf{M} \approx \mathbf{M}' \approx 0 \pmod q$, in particular we need

$$- < q/2^{t+1} < [\mathbf{A}_v^T\mathbf{S} + \mathbf{\Delta}]_{k,\ell} < q/2^{t+1}.$$

For any $\mathbf{S}$ and $\mathbf{A}_v$, the last condition occurs only with exponentially small probability over the choise of $\mathbf{\Delta}$. We conclude that if all the entries of $|\mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{\Delta}|$ are larger than $q/2^{t+1}$ (modulo q), which happens with overwhelming probability, then for all level-i encodings $\mathbf{D}$ of $\mathbf{S}$, the top t bits of $\mathbf{D}^T \cdot \mathbf{A}_u^T$ agree with the top t bits of $\mathbf{A}_v^T \cdot \mathbf{S}$. We call a plaintext matrix $\mathbf{S}$ "v-good" if the above happens, and denote their set by GOOD_v. With this notation, the arguments above say that for any fixed $\mathbf{S}, v$, we have $\mathbf{S} \in \mathsf{GOOD}_v$ with overwhelming probability over the instance-generation randomness.

Same input implies same extracted value. For any plaintext matrix $\mathbf{S} \in \mathsf{GOOD}_v$, clearly all its encodings relative to $u \rightsquigarrow v$ agree on the top t bits of $\mathbf{D}^T \cdot \mathbf{A}_u^T$ (since they all agree with $\mathbf{A}_v^T \cdot \mathbf{S}$). Hence they all have the same extracted value.

Different inputs imply different extracted values. If $\mathbf{D}, \mathbf{D}'$ encode different plaintext matrices then $\mathbf{D} - \mathbf{D}'$ is an encoding of non-zero, hence $\|(\mathbf{D} - \mathbf{D}')^T \cdot \mathbf{A}_u^T\| \gg q/2^t$ except with negligible probability, $\mathbf{D}^T \cdot \mathbf{A}_u^T + \mathbf{\Delta}$ and $\mathbf{D}'^T \cdot \mathbf{A}_u^T + \mathbf{\Delta}$ must differ somewhere in their top t bits. Since we use universal hashing for our randomness extractor, then with high probability (over the hash function β) we get $\mathsf{RandExt}_\beta\big(\mathsf{msb}_t(\mathbf{D} \cdot \mathbf{A}_u + \mathbf{\Delta})\big) \neq \mathsf{RandExt}_\beta\big(\mathsf{msb}_t(\mathbf{D}'^T \cdot \mathbf{A}_u^T + \mathbf{\Delta})\big)$.

Random input implies random extracted value. Fix some high-entropy distribution $\mathcal{D}$ over inputs $\mathbf{S}$. Since for every $\mathbf{S}$ we have $\Pr[\mathbf{S} \in \mathsf{GOOD}_v] = 1 - \mathrm{negl}(\lambda)$ then also with overwheling probability over the parameters we have $\Pr_{\mathbf{S} \leftarrow \mathcal{D}}[\mathbf{S} \in \mathsf{GOOD}_v] = 1 - \mathrm{negl}(\lambda)$.

It is therefore enough to show that $\mathsf{RandExt}_\beta(\mathsf{msb}_t(\mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{\Delta}))$ is nearly uniform on $\mathbf{S} \leftarrow \mathcal{D}$.

We observe that the function $H(\mathbf{S}) = \mathbf{A}_v^T \cdot \mathbf{S} + \mathbf{\Delta}$ is itself pairwise independent on each column of the output separately, and therefore so is the function $H'(\mathbf{S}) = \mathsf{msb}_t(H(\mathbf{S}))$. [3] We note that H' has very low collision probability, its range has many more than 6λ bits in every column, so for every $\mathbf{S} \neq \mathbf{S}'$ we get $\Pr_{H'}[H'(\mathbf{S}) = H'(\mathbf{S}')] \ll 2^{-6\lambda}$. Therefore H' is a good condenser, i.e., if the min-entropy of $\mathcal{D}$ is above 3λ, then with overwhelming probability over the choise of H, the min-entropy of $H'(\mathcal{D})$ is above $3\lambda - 1$ (say). By the extraction properties of $\mathsf{RandExt}$, this implies that $\mathsf{RandExt}_\beta(H'(\mathcal{D}))$ is close to uniform (whp over β).

5.4.2 A Commutative Variant

In some applications it may be convenient or even necessary to work with a commutative plaintext space. Of course, simply switching to a commutative sub-ring of the ring of matrices (such as $s \cdot I$ for a scalar s and the identity I) would be insecure, but we can make it work by moving to a larger ring.

Cyclotomic rings. We switch from working over the ring of integers to working over polynomial rings, $R = \mathbb{Z}[x]/(F(X))$ and $R_q = R/qR$ for some degree n irreducible integer polynomial $F(X) \in \mathbb{Z}[X]$ and an integer $q \in \mathbb{Z}$. Elements of this ring correspond to degree-$(n-1)$ polynomials, and hence they can be represented by n-vectors of integers in some convenient basis. The norm of a ring element is the norm of its coefficient vector, and this can be extended as usual for norm of vectors and matrices over R. Addition and multiplication are just polynomial addition and multiplication modulo $F(X)$ (and also modulo q when talking about R_q).

As usual, we need a ring where the norm of a product is not much larger than the product of the norms, and this can be achieved for example by using $F = \Phi_M(X)$, the M'th cyclotomic polynomial (of degree $n = \phi(M)$). All the required operations and lemmas that we need (such as trapdoor and pre-image sampling etc.) can be extended also to this setting, see e.g. [LPR13].

The construction remains nearly identical, except all operations are now performed over the rings R and R_q and the dimensions are changed to match. We now have the "matrices" $\mathbf{A}_v \in R_q^{1\times m}$ with only one row (and similarly the error matrices are $\mathbf{E} \in R_q^{m\times 1}$), and the plaintext space is R_q itself. An encoding of plaintext element $s \in R_q$ relative to path $u \rightsquigarrow v$ is a small matrix $\mathbf{D} \in R_q^{m\times m}$ such that

$$\mathbf{D}^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot s + \mathbf{E}$$

where $\mathbf{E}'$ is some small error term. As before, we only encode small plaintext elements, i.e.,

[3]If q is not a power of two then H' does not produce uniformly random t-bit strings. But still its outputs on any two $\mathbf{S}' \neq \mathbf{S}$ are independent, and each has almost full (min-)entropy, which sufficies for our purposes.

the sampling procedure draws s from a Gaussian distribution with small parameter. The operations all remain the same as in the basic scheme.

We emphasize that it is the plaintext space that is commutative, not the space of encoding. Indeed, if we have $\mathbf{D}, \mathbf{D}'$ that encode s, s' relative to paths $v \rightsquigarrow w$ and $u \rightsquigarrow v$, respectively, we can only multiply them in the order $\mathbf{D}' \cdot \mathbf{D}$. Multiplying in the other order is inconsistent with the graph G and hence is unlikely to yield a meaningful result. What makes the symmetric scheme useful is the ability to multiply the *plaintext elements* in arbitrary order. For example for $\mathbf{D}, \mathbf{D}'$ that encode s, s' relative to paths $u \rightsquigarrow w$ and $v \rightsquigarrow w$, we can compute either $\mathbf{D}^T \cdot \mathbf{A}_u^T \cdot s'$ or $\mathbf{D}'^T \cdot \mathbf{A}_v^T \cdot s$ and the results will both be close $\mathbf{A}_v^T cdotss'$ (and hence also close to each other).

5.4.3 Public Sampling and Some Other Variations

As mentioned in Appendix 5.3, we can provide a public sampling procedure relative to any desired path $p = u \rightsquigarrow v$ by publishing with the public parameters a collection of pairs generated by the secret sampling procedure above, $\{(\mathbf{S}_k, \mathbf{D}_k) : k = 1, \ldots, \ell\}$ (for some large enough ℓ). The public sampling procedure then takes a random linear combination of these pairs as a new sample, namely it chooses $\mathbf{r} \leftarrow D_{\mathbb{Z}^\ell, \sigma'}$ and compute the encoding pair as:

$$(\mathbf{S}, \mathbf{D}) := \left(\sum_{i \in [\ell]} \mathbf{r}_i \mathbf{S}_i \ , \quad \sum_{i \in [\ell]} \mathbf{r}_i \mathbf{D}_i \right).$$

It is easy to see that the resulting $\mathbf{D}$ encodes $\mathbf{S}$ relative to the edge e. Also by Theorem 5.2.1, the plaintext matrix $\mathbf{S}$ is distributed according to a Gaussian distribution whp.

We note that in most applications it is not necessary to include in the public parameters the matrices for all the nodes in the graph. Indeed we typically only need the matrices for the source nodes in the DAG in order to preform zero-testing or extraction.

Some Safeguards. Since our schemes are graph-based, and hence the order of products is known in advance, we can often provide additional safeguards using Kilian-type randomization [Kil88] "on the encoding side". Namely, for each internal node v in the graph we choose a random invertible $m \times m$ matrix modulo q $\mathbf{R}_v$, and for the sinks and sources we set $\mathbf{R}_v = I$. Then we replace each encoding $\mathbf{C}$ relative to the path $u \rightsquigarrow v$ by the masked encoding $\mathbf{C}' := \mathbf{R}_u^{-1} \cdot \mathbf{C} \cdot \mathbf{R}_v$.

Clearly, this randomization step does not affect the product on any source-to-sink path in the graph, but the masked encodings relative to any other path no longer consist of small entries, and this makes it harder to mount the attacks from 5.5. On the down side, it now takes more bits to represent these encodings.

Other safeguards of this type includes the observations that encoding matrices relative to paths that end at a sink node need not have small entries since the size of the last matrix on a path does not contribute to the size of the final error matrix. Similarly the plaintext elements that re encoded on paths that begin at source nodes need not be small, for the same reason.

We remark that applying the safeguards from above comes with a price tag: namely the encoding matrices no longer consist of small entries, hence it takes more bits to represent them.

Finally, we observe that sometimes we do not need to give explicitly the matrices $\mathbf{A}_u$ corresponding to source nodes, and can instead "fold them" into the encoding matrices. That is, instead of providing both $\mathbf{A}$ and $\mathbf{C}$ such that $\mathbf{B} = \mathbf{D}^T \cdot \mathbf{A}^T \approx \mathbf{A}'^T \cdot \mathbf{S}$, we can publish only the matrix $\mathbf{B}$ and keep $\mathbf{A}, \mathbf{D}$ hidden. This essentially amounts to shortening the path by one, starting it at the matrix $\mathbf{B}$. (Of course, trying to repeat this process and further process the path will lead to exponential growth in the number of matrices that we need to publish.)

5.5 Cryptanalysis

Below we describe several attacks and "near attacks" on some variations of our scheme, these attacks guided our choices in designing these scheme.

5.5.1 Encoding of Zero is a Weak Trapdoor

The main observation in this section is that an encoding of zero relative to a path $u \rightsquigarrow v$ can sometimes be used as a weak form of trapdoor for the matrix $\mathbf{A}_u$. Recall from [GPV08] that a full-rank $m \times m$ matrix $\mathbf{T}$ with small entries satisfying $\mathbf{T}^T\mathbf{A}^T = 0 \pmod q$ can be used as a trapdoor for the matrix $\mathbf{A}$ as per Lemma 2.4.1. An encoding of zero relative the path $u \rightsquigarrow v$ is a matrix $\mathbf{C}$ such that $\mathbf{C}^T \cdot \mathbf{A}_u^T = \mathbf{E} \pmod q$ for a small matrix $\mathbf{E}$. This is not quite a trapdoor, but it appears close and indeed we show that if can often be used as if it was a real trapdoor.

Let us denote by $\mathbf{A}_u'^T = (\mathbf{A}_u/I)^T$ the $(m+n) \times n$ matrix whose first m rows are those of $\mathbf{A}_u$ and whose last n rows are the $n \times n$ identity matrix. Given the matrices $\mathbf{A}_u$ and $\mathbf{C}$ as above, we can compute the small matrix $\mathbf{E} = \mathbf{C}^T\mathbf{A}_u^T \bmod q$, then set $\mathbf{C}' = [\mathbf{C}|(-\mathbf{E})]$ to be the $m \times (m+n)$ matrix whose first m columns are the columns of $\mathbf{C}$ and whose last n columns are the negation of the columns of $\mathbf{E}$. Clearly $\mathbf{C}'$ is a small matrix satisfying $\mathbf{C}'^T\mathbf{A}_u'^T = 0 \pmod q$, but it is not a trapdoor yet because it has rank m rather than $m+n$.

However, assume that we have two encodings of zero, relative to two (possibly different) paths that begin at the same node u. Then we can apply the procedure above to get two such matrices $\mathbf{C}_1'$ and $\mathbf{C}_2'$, and now we have $2m$ rows that are all orthogonal to $\mathbf{A}_u'^T \bmod q$, and it is very likely that we can find $m+n$ among them that are linearly independent. This gives a full working trapdoor $\mathbf{T}_u'$ for the matrix $\mathbf{A}_u'$, what can we do with this trapdoor?

Assume now that the application gives us, in addition to the zero encodings for path that begin with u, also an encoding of a plaintext elements $\mathbf{S} \neq 0$ relative to some path that ends at u, say $w \rightsquigarrow u$. This is a matrix $\mathbf{D}$ such that $\mathbf{D}^T \cdot \mathbf{A}_w^T = \mathbf{A}_u^T\mathbf{S} + \mathbf{E}$, namely $\mathbf{B}^T = \mathbf{D}^T\mathbf{A}_w^T \bmod q$ is an LWE instance relative to public matrix $\mathbf{A}_u$, secret $\mathbf{S}$, and error term $\mathbf{E}$. Recalling that the plaintext $\mathbf{S}$ in our scheme must be small, it is easy to convert $\mathbf{B}$ into an LWE instance relative to matrix $\mathbf{A}_u'^T = (\mathbf{A}_u/I)^T$, for which we have a trapdoor: Simply add

n zero rows at the bottom, thus getting $\mathbf{B}'^T = (\mathbf{B}/0)^T$, and we have $\mathbf{B}'^T = \mathbf{A}_u'^T\mathbf{S} + \mathbf{E}'$, with $\mathbf{E}' = (\mathbf{E}/(-\mathbf{S}))$ a small matrix.[4] Given $\mathbf{B}'$ and $\mathbf{A}_u'$, in conjunction with the trapdoor $\mathbf{T}_u'$, we can now recover the plaintext $\mathbf{S}$.

We note that a consequence of this attack is that in our scheme it is unsafe for the application to allow computation of zero-encoding, except perhaps relative to source-nodes in the graph. As we show in 5.6, it is possible to design applications that get around this problem.

Extensions. The attacks from above can be extended even to some cases where we are not given encodings of zero. Suppose that instead we are given pairs $\{(\mathbf{C}_i, \mathbf{C}_i')\}_i$, where the two encodings in each pair encode *the same plaintext* $\mathbf{S}_i$ relative to two paths with a common end point, $u \rightsquigarrow v$ and $u' \rightsquigarrow v$. In this case we can use the same techniques to find a "weak trapdoor" for the concatenated matrix $\mathbf{A}' = (\mathbf{A}_u/\mathbf{A}_{u'})$ of dimension $2m \times n$, using the fact that $[\mathbf{C}_i|(-\mathbf{C}_i')] \cdot \mathbf{A}' = (\mathbf{A}_v\mathbf{S}_i + \mathbf{E}_i) - (\mathbf{A}_v\mathbf{S}_i + \mathbf{E}_i') = \mathbf{E}_i - \mathbf{E}_i'$.

If we are also given a pair $(\mathbf{D}, \mathbf{D}')$ that encodes the same element $\mathbf{S}$ relative to two paths that end at u, u', respectively, then we can use these approximate trapdoors to find $\mathbf{S}$, since $(\mathbf{D}, \mathbf{D}')$ (together with the start points of these paths) yield an LWE instance relative to public matrix $\mathbf{A}'$ and the secret $\mathbf{S}$.

Corollary 1: No Re-randomization. A consequence of the attacks above is that in our scheme we usually cannot provide encoding-of-zero in the public parameters. Hence the re-randomization technique by adding encodings of zero usually cannot be used in our case.

Corollary 2: No Symmetric plaintext/encoding pairs. Another consequence of the attacks above is that at least in the symmetric case it is not safe to provide many pairs (s_i, C_i) s.t. C_i is an encoding of the scalar s_i along a path $u \rightsquigarrow v$. The reason is that given two such pairs $(s_1, C_1), (s_2, C_2)$ we can compute an encoding of zero along the path $u \rightsquigarrow v$ as $s_1C_2 - s_2C_1$.

5.5.2 Recovering Hidden $\mathbf{A}_v$'s.

As we noted earlier, in many applications we only need to know the matrices $\mathbf{A}_u$ for source nodes u and there is no need to publish the matrices $\mathbf{A}_v$ for internal nodes. This raises the possibility that we might get better security by withholding the $\mathbf{A}_v$'s of internal nodes.

Trying to investigate this possibility, we show below two "near attacks" for recovering the public matrices of internal nodes from those of source nodes in the graph. The first attack applies to the commutative setting, and is able to recover an approximate version of the internal matrices (with the approximation deteriorating as we move deeper into the graph). The second attack can recover the internal matrices exactly, but it requires a full trapdoor

[4]$\mathbf{B}'$ does not have the right distribution for an LWE instance, but using the trapdoor we can solve the worst-case BDD, not just the average-case LWE, so the attack still stands.

for the matrices of the source nodes (and we were not able to extend it to work with the "approximate trapdoors" that one gets from an encoding of zero).

The conclusion from these "near attacks" is uncertain. Although is still possible that withholding the internal-node matrices helps security, it seems prudent to examine the security of candidate applications that use our scheme in a setting where the $\mathbf{A}_v$'s are all public.

Recovering the $\mathbf{A}_v$'s in the symmetric setting. For this attack we are given a matrix $\mathbf{A}_u$, and many encodings relative to the path $u \rightsquigarrow v$, *together with the corresponding plaintext elements* (e.g., as needed for the public-encoding variant). Namely, we have $\mathbf{A}_u$, small matrices $\mathbf{C}_1, \ldots, \mathbf{C}_t$ (for $t > 1$) and small ring elements $s_1, \ldots, s_t$ such that $\mathbf{C}_j^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot s_j + \mathbf{E}_j$ holds for all j, with small $\mathbf{E}_j$'s. Our goal is to find $\mathbf{A}_v$.

We note that the matrix $\mathbf{A}_v$ and the error vectors $\mathbf{E}_j$ are only defined upto small additive factors, since adding 1 to any entry in $\mathbf{A}_v$ can be offset by subtracting the s_j's from the corresponding entry in the $\mathbf{E}_j$'s. Hence the best we can hope for is to solve for $\mathbf{A}_v$ upto a small additive factor (resp. for the $\mathbf{E}_j$'s upto a small additive multiple of the s_j's). Denoting $\mathbf{B}_j := \mathbf{C}_j^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot s_j + \mathbf{E}_j$, we compute for $j = 1, \ldots, t-1$,

$$\begin{aligned}\mathbf{F}_j &:= \mathbf{B}_j \cdot s_{j+1} - \mathbf{B}_{j+1} \cdot s_j \\ &= (\mathbf{A}_v^T \cdot s_j + \mathbf{E}_j) \cdot s_{j+1} - (\mathbf{A}_v^T \cdot s_{j+1} + \mathbf{E}_{j+1}) \cdot s_j \;=\; \mathbf{E}_j \cdot s_{j+1} - \mathbf{E}_{j+1} \cdot s_j.\end{aligned}$$

This gives us a non-homogeneous linear system of equations (with the s_j's and $\mathbf{F}_j$'s as coefficients), which we want to solve for the small solution $\mathbf{E}_j$'s. Writing this system explicitly we have

$$\begin{pmatrix} [s_2] & [-s_1] & & & \\ & [s_3] & [-s_2] & & \\ & & \ddots & \ddots & \\ & & & [s_t] & [-s_{t-1}] \end{pmatrix} \begin{pmatrix} \mathbf{X}_1 \\ \mathbf{X}_2 \\ \vdots \\ \mathbf{X}_{t-1} \\ \mathbf{X}_t \end{pmatrix} = \begin{pmatrix} \mathbf{F}_1 \\ \mathbf{F}_2 \\ \vdots \\ \mathbf{F}_{t-1} \end{pmatrix},$$

where $[s]$ denotes the $m \times m$ matrix $I_{m\times m} \cdot s$. Clearly this system is partitioned into m independent systems, each of the form

$$\begin{pmatrix} s_2 & -s_1 & & & \\ & s_3 & -s_2 & & \\ & & \ddots & \ddots & \\ & & & s_t & -s_{t-1} \end{pmatrix} \begin{pmatrix} x_{1,\ell} \\ x_{2,\ell} \\ \vdots \\ x_{t-1,\ell} \\ x_{t,\ell} \end{pmatrix} = \begin{pmatrix} f_{1,\ell} \\ f_{2,\ell} \\ \vdots \\ f_{t,\ell} \end{pmatrix},$$

with $x_{j,\ell}$, $f_{j,\ell}$ being the ℓ'th entries of the vectors $\mathbf{X}_j, \mathbf{F}_j$, respectively. These systems are under-defined, and to get the $\mathbf{E}_i$'s we need to find *small solutions* for them. Suppressing the index ℓ, we denote these systems in matrix form by $\mathbf{M}\vec{x} = \vec{f}$, and show how to find small solutions for them.

At first glance this seems like a SIS problem so one might expect it to be hard, but here we already know a small solution for the corresponding homogeneous system, namely the solution $x_j = s_j$ for all j. Below we assume that the s_j do not all share a prime factor (i.e., that $GCD(s_1, s_2, \ldots, s_t) = 1$), and also that at least one of them has a small inverse in the field of fractions of R. (These two conditions hold with good probability, see discussion in [GGH13a, Sec 4.1].)

To find a small solution for the inhomogeneous system, we begin by computing an arbitrary solution for it over the ring R (not modulo q). We note that a solution exists (in particular the E_j's solve this system over R without mod-q reduction), and we can use Gaussian elimination in the field of fractions of R to find it. Denote that solution that was found by $\vec{g} \in R$, namely we have $\mathbf{M}\vec{g} = \vec{f}$. [5] Since over R this is a $(t-1) \times t$ system then its solution space is one-dimensional. Hence every solution to this system (and in particular the small solution that we seek) is of the form $\vec{e} = \vec{g} + \vec{s} \cdot k$ for some $k \in R$. [6]

Choosing one index j such that the element $1/s_j$ in the field of fractions is small, we compute a candidate for the scalar k simply by rounding, $k' := -\lfloor g_j/s_j \rceil$, where division happens in the field of fractions. We next prove that indeed the vector $\vec{e'} = \vec{g} + \vec{s} \cdot k'$ is a small vector over R. Clearly $\vec{e'} \in R^t$ since $k' \in R$ and $\vec{g}, \vec{s} \in R^t$, we next prove that it must be small by showing that "the right scalar k" must be close to the scalar k' that we computed. First, observe that $e'_j = g_j + s_j \cdot k'$ must be small, since

$$e'_j = g_j + s_j \cdot k' = g_j - \lfloor g_j/s_j \rceil \cdot s_j = g_j - (g_j/s_j + \epsilon_j) \cdot s_j = -\epsilon_j \cdot s_j,$$

with ϵ_j the rounding error. Since both ϵ_j and s_j are small, then so is e'_j.

Now consider the "real value" e_j, it too is small and is obtained as $g_j + s_j \cdot k$ for some $k \in R$. It follows that $e_j - e'_j = s_j \cdot (k - k')$ is small, and since we know that $1/s_j$ is also small then it follows that so is $k - k' = (e_j - e'_j)/s_j$. We thus conclude that $\vec{e'} = \vec{g} + k' \cdot \vec{s} = \vec{e} + (k - k') \cdot \vec{e}$ is also small.

Repeating the same procedure for all the m independent systems, we get a small solution $\{\mathbf{E}'_j, j = 1, \ldots, t\}$ to the system $\mathbf{B}_j = \mathbf{A}_v^T \cdot s_j + \mathbf{E}'_j$. Subtracting the $\mathbf{E}'_j$'s from the $\mathbf{B}_j$'s and dividing by the s_j's give us (an approximation of) $\mathbf{A}_v$.

Recovering the $\mathbf{A}_v$'s using trapdoors. Suppose that we are given $\mathbf{A}_u$, encodings $\mathbf{C}_j$ and the corresponding plaintext matrices $\mathbf{S}_j$, s.t. $\mathbf{B}_j := \mathbf{C}_j^T \cdot \mathbf{A}_u^T = \mathbf{A}_v^T \cdot \mathbf{S}_j + \mathbf{E}_j \pmod q$ for small errors $\mathbf{E}_j$. Suppose that in addition we are also given a *full working trapdoor* for the matrix $\mathbf{A}_v$, say, in the form of a small full-rank matrix $\mathbf{T}$ over R s.t. $\mathbf{T}^T \cdot \mathbf{A}_v^T = 0 \pmod q$. We can then use $\mathbf{T}$ to recover the errors $\mathbf{E}_j$ from the LWE instances $\mathbf{B}_j$, which can be done without knowing $\mathbf{A}_v$: Let $\mathbf{T}^{-1}$ be the inverse of $\mathbf{T}$ over R, we compute $\mathbf{E}_j \leftarrow (\mathbf{T}^{-1})^T \cdot (\mathbf{T}^T \cdot \mathbf{B}_j \bmod q)$. Once we have the error matrices $\mathbf{E}_j$ we can subtract them and get the set of equations $\mathbf{B}_j - \mathbf{E}_j = \mathbf{A}_v^T \cdot \mathbf{S}_j \pmod q$, where the entries of $\mathbf{A}_v$ are the unknowns.

[5] Using Gaussian elimination may yield a fractional solution $\vec{g'}$, but we can "round it" to an integral solution by solving for k' the equation $\vec{g'} + \vec{s} \cdot k' = 0 \pmod 1$, then setting $\vec{g} = \vec{g'} + \vec{s} \cdot k'$.

[6] In general the scalar k may be fractional, but if $GCD(s_1, s_2, \ldots, s_t) = 1$ then k must be integral.

With sufficiently many of these equations, we can then solve for $\mathbf{A}_v$.

We note that so far we were unable to extend this attack to using the "weak trapdoor" that one gets from an encoding of zero wrt paths of the form $v \rightsquigarrow w$. Indeed the procedure from 5.5.1 for recovering a stronger trapdoor from the weak one relies on knowing $\mathbf{A}_v$.

5.6 Applications

5.6.1 Multipartite Key-Agreement

For our first application, we describe a candidate construction for a non-interactive multipartite key-agreement protocol using the commutative variant of our graph-based encoding scheme. As is usual with multipartite key-agreement from multilinear maps, each party i is contributing an encoding of some secret s_i and the shared secret is derived from an encoding of the product $s = \prod_i s_i$. However in our case we need to use extra caution to protect against the "weak trapdoor attacks" from 5.5.1.

To that end, we design our graph to ensure that the adversary is never given encodings of the same element on two paths with a common end-point, and also is not given an encoding and the corresponding plaintext on any edge. For an k-partite protocol we use a graph topology of k directed chains that meet at a common point, where the contribution of any given party appears at different edges on different chains (i.e. the first edge on one chain, the second edge on another, the third edge on a third chain, etc.)

That is, each player i has a directed path of matrices, $\mathbf{A}_{i,1}, \ldots, \mathbf{A}_{i,k+1}$, all sharing the same end-point, i.e., $\mathbf{A}_{i,k+1} = \mathbf{A}_0$ for all i. Note that every chain has k edges, and for the chain "belonging" to party i we will broadcast on its edges encodings of all the secrets $s_j, j \neq i$, but not an encoding of s_i, that last encoding will only be known to party i. Party i will multiply these encodings (the one that only it knows, and all the ones that are publicly available) to get an encoding of $\prod_i s_j$ relative to the path $\mathbf{A}_{i,1} \rightsquigarrow \mathbf{A}_0$. Namely, a matrix $\mathbf{D}_i$ such that $\mathbf{D}_i^T \cdot \mathbf{A}_{i,1}^T \approx \mathbf{A}_0^T \cdot \prod_i s_j$. The shared secret is then obtained by applying the extraction procedure to this $\mathbf{D}_i$.

The assignment of which secret is encoded on what edge of what chain is done in a "round robin" fashion. Specifically, the i'th secret s_i is encoded on the j'th edge of the chain belonging to party $i' = j - i + 1$. In other words, the secret that we encode on the edge $\mathbf{A}_{i,j} \to \mathbf{A}_{i,j+1}$ in the graph is s_{j-i+1}, with index arithmetic modulo k. An example of the assignment of secrets to edges for a 4-partite protocol is depicted in Figure 5-1.

Of course, we must publish encodings that would allow the parties to choose their secrets and provide encodings for them. This means that together with the public parameters we also publish encodings of many plaintext elements $\{t_{i,\ell} : i = 1, \ldots, k,\ \ell = 1, \ldots, N\}$ (for a sufficiently large N), for each $t_{i,\ell}$ we publish encoding of it relative to all the edges $\mathbf{A}_{i',i+i'-1} \to \mathbf{A}_{i',i+i'}$ for all i, i' (index arithmetic modulo $k+1$). Party i then chooses random small coefficients $r_{i,\ell}$ and computes its encoding relative to each edge $\mathbf{A}_{i',i+i'-1} \to \mathbf{A}_{i',i+i'}$ as the linear combination of the encodings on that edge with the coefficient $r_{i,\ell}$. We are now ready to describe our scheme $\mathcal{NMKE} = (\mathsf{KE.Setup}, \mathsf{KE.Publish}, \mathsf{KE.Keygen})$.

$$\begin{tikzcd}
\mathbf{A}_{1,1} \arrow[r, "s_1"] & \mathbf{A}_{1,2} \arrow[r, "s_2"] & \mathbf{A}_{1,3} \arrow[r, "s_3"] & \mathbf{A}_{1,4} \arrow[dr, "s_4"] & & \mathbf{A}_{3,4} \arrow[dl, "s_2"'] & \mathbf{A}_{3,3} \arrow[l, "s_1"] & \mathbf{A}_{3,2} \arrow[l, "s_4"] & \mathbf{A}_{3,1} \arrow[l, "s_3"] \\
\mathbf{A}_{2,1} \arrow[r, "s_4"] & \mathbf{A}_{2,2} \arrow[r, "s_1"] & \mathbf{A}_{2,3} \arrow[r, "s_2"] & \mathbf{A}_{2,4} \arrow[r, "s_3"] & \mathbf{A}_0 & \mathbf{A}_{4,4} \arrow[l, "s_1"] & \mathbf{A}_{4,3} \arrow[l, "s_4"] & \mathbf{A}_{4,2} \arrow[l, "s_3"] & \mathbf{A}_{4,1} \arrow[l, "s_2"]
\end{tikzcd}$$

Figure 5-1: Graph for a 4-partite key-agreement protocol.

- $\mathsf{KE.Setup}(1^\lambda, k)$: The setup algorithm takes as input the security parameter 1^λ and the total number of players k.

  1. Run the parameter-generation and instance-generation of our graph-based encoding scheme for the graph consisting of k chains with a common end-point, each of length k edges. Let $e_{i,j}$ denote the j'th edge on the i'th chain.
  2. Using the secret parameters, run the sampling procedure of the encoding scheme to choose random plaintext elements $t_{i,\ell}$ for $i = 1, \dots, k$ and $\ell = 1, \dots, N$, and for each $t_{i,\ell}$ compute also an encoding of it relative to all the edges $e_{i',j}$ for $j = i + i'$ (mod k). Denote the encoding of $t_{i,\ell}$ on chain i' (relative to edge $e_{i',i+i' \bmod k}$) by $\mathbf{C}_{i,\ell,i'}$.

  The public parameters of the key-agreement protocol include the public parameters of the encoding scheme (i.e., the matrices for all the source nodes $\mathbf{A}_{i,1}$), and also the encoding matrices $\{\mathbf{C}_{i,\ell,i'} : i, i' = 1, \dots, k,\ \ell = 1, \dots, N\}$.

- $\mathsf{KE.Publish}(\mathsf{pp}, i)$: The i'th party chooses random small plaintext elements $r_{i,\ell} \leftarrow \chi$ for $\ell = 1, \dots, N$ and then sets $\mathbf{D}_{i,i'} \leftarrow \sum_\ell \mathbf{C}_{i,\ell,i'} \cdot r_{i,\ell}$ for all i'. It keeps $\mathbf{D}_{i,i}$ as its secret and broadcast all the other $\mathbf{D}_{i,i'}$'s.

- $\mathsf{KE.Keygen}(\mathsf{pp}, i, \mathsf{sk}_i, \{\mathsf{pub}_j\}_{j \neq i})$: Party i collects all the matrices $\mathbf{D}_{i',i}$ (encoding the secrets $s_{i'}$ relative to "its chain" i) and orders them according to $j = i + i'$. Namely, it sets $\mathbf{F}_{j,i} = \mathbf{D}_{i+j \bmod k, i}$ for $j = 1, \dots k$, then computes the product $\mathbf{F}^*_i = (\prod_{j=1}^k F_{j,i}) \cdot \mathbf{A}_{i,1}$. Finally, party i applies the extraction procedure of the encoding scheme to obtain the secret key, setting $\mathsf{ssk} = \mathsf{Extract}\big(\mathbf{F}^*_i\big)$.

Security. Unfortunately, we were not able to reduce the security of this candidate scheme to any "nicer" assumption. As such, at present the only evidence of security that we can offer is the failure of our attempts to cryptanalyze it.

The basic attack from 5.5.1 does not seem to apply here since the public parameters do not provide any encoding of zero (not even relative to $\mathbf{A}_0$). Similarly, the extended attacks do not seem to apply since the only common end-point in the graph is $\mathbf{A}_0$, and no two paths that end at $\mathbf{A}_0$ include an encoding of the same element.

We note that the attacker can use the public parameters to compute an approximate trapdoors for concatenated matrices of the form $(\mathbf{A}_0 \cdot t_{i,\ell,i'}/(-\mathbf{A}_0))$ (or similar), but the broadcast messages of the parties do not provide LWE instances relative to these matrices.

Finally, we note that as for any other application of this encoding scheme, it seems that security would be enhanced by applying the additional safeguards that were discussed at the end of 5.4. That is, we can use Kilian-style randomization on the encoding side, by choosing k invertible matrices for each chain, $\mathbf{R}_{i,1}, \ldots, \mathbf{R}_{i,k}$, where the first and last are set to the identity and the others are chosen at random. Then we can replacing each encoding matrix $\mathbf{C}$ in the public parameters by $\mathbf{C}' := \mathbf{R}^{-1} \cdot \mathbf{C} \cdot \mathbf{R}'$ using the randomizer matrices $\mathbf{R}, \mathbf{R}'$ belonging to the two adjacent nodes. We can also choose the first encoding matrix in each chain to have large entries.

This has no effect on the product of all the encoding matrices along the i'-th chain, but the new matrices $\mathbf{C}'$ no longer have small entries, which seems to aid security. On the down side, this increases the size of the encodings roughly by a $\log q/\log n$ factor.

5.6.2 Candidate Branching-Program Obfuscation

We next describe how to adapt the branching-program obfuscation constructions from previous work [GGH+13b, BR14b, BGK+14, PST14] to use our encoding schemes. We remark that on some level this is the simplest type of constructions to adapt to our setting, since we essentially need only a single chain and there almost no issues of providing zero-encoding in the public parameters (or encodings of the same plaintext relative to different nodes in the graph).

Roughly speaking, previous works all followed a similar strategy for obfuscating branching programs. Starting from a given oblivious branching program, encoded as permutation matrices, they all applied Kilian's randomization strategy to randomized these matrices, then added some extra randomization steps (mostly multiplication by random scalars) to protect against partial-evaluation and mixed-input attacks, and finally encoded the resulting matrices relative to specially-designed sets/levels. The specific details of the extra randomization steps are somewhat different between the previous schemes, but all these techniques have their counterparts in our setting. Below we explain how to adapt the randomization techniques from previous work to our setting, and then describe one specific BP-obfuscation candidate.

Matrices vs. individual elements. Our scheme natively encodes matrices, rather than individual elements. This has some advantages, for example we need not worry about attacks that mix and match encoded elements from different matrices. At the same time it also poses some challenges, in particular some of the prior schemes worked by comparing to zero one element of the resulting matrix at the end of evaluation, an operation which is not available in our case.

To be able to examine sub-matrices (or single elements), we adopt the "bookend encoding" trick from [GGH+13b]. That is, we add to our chain a new source u^* and a new sink v^*, with edges from u^* to the old source and from the old sink to v^*. On the edge from u^* we encode a matrix $\mathbf{T}$ which is only nonzero in the columns that we want to examine, and on the edge to v^* we encode a matrix $\mathbf{S}$ which is only nonzero in the rows that we wish to examine. This way, we should have the matrix $\mathbf{T} \cdot \mathbf{U} \cdot \mathbf{S}$ encoded relative to a path $u^* \rightsquigarrow v^*$, and that matrix is only nonzero in the sub-matrix of interest. In the candidate below we somewhat improve on this by folding the source matrix $\mathbf{A}_{u^*}$ into the encoding of $\mathbf{T}$, publishing instead the matrix $\mathbf{A}_{u^*}^T \cdot \mathbf{T}$ (and in fact making $\mathbf{T}$ a single column vector).

Only small plaintexts. In our scheme we can only encode "small plaintext elements", not every plaintext element. This is particularly relevant for Kilian randomization technique, since it requires that we multiply by both $\mathbf{R}$ and $\mathbf{R}^{-1}$ for each randomizer matrix $\mathbf{R}$. One way to get randomizer matrices with both $\mathbf{R}$ and $\mathbf{R}^{-1}$ small is using the facts that

$$\left(\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline \mathbf{R} & \mathbf{I} \end{array}\right)^{-1} = \left(\begin{array}{c|c} \mathbf{I} & \mathbf{0} \\ \hline -\mathbf{R} & \mathbf{I} \end{array}\right), \quad \left(\begin{array}{c|c} \mathbf{I} & \mathbf{R} \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right)^{-1} = \left(\begin{array}{c|c} \mathbf{I} & -\mathbf{R} \\ \hline \mathbf{0} & \mathbf{I} \end{array}\right)$$
$$\left(\begin{array}{c|c} \mathbf{0} & \mathbf{I} \\ \hline \mathbf{I} & \mathbf{R} \end{array}\right)^{-1} = \left(\begin{array}{c|c} \mathbf{0} & \mathbf{I} \\ \hline \mathbf{I} & -\mathbf{R} \end{array}\right), \quad \left(\begin{array}{c|c} \mathbf{R} & \mathbf{I} \\ \hline \mathbf{I} & \mathbf{0} \end{array}\right)^{-1} = \left(\begin{array}{c|c} -\mathbf{R} & \mathbf{I} \\ \hline \mathbf{I} & \mathbf{0} \end{array}\right).$$

Multiplying a sequence of these types of matrices above yields a high-entropy distribution of randomizer matrices with the desired property, and seemingly without obvious algebraic structure. Another family of matrices where both the matrix and its inverse are small are permutation matrices (and of course we can mix and match these families). Concretely, we speculate that a randomizer of the form

$$\mathbf{R} = \Pi_1 \cdot \begin{pmatrix} \mathbf{0} & \mathbf{I} \\ \mathbf{I} & \mathbf{R}_1 \end{pmatrix} \cdot \Pi_2 \cdot \begin{pmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{R}_2 & \mathbf{I} \end{pmatrix} \cdot \Pi_3 \cdot \begin{pmatrix} \mathbf{R}_3 & \mathbf{I} \\ \mathbf{I} & \mathbf{0} \end{pmatrix} \cdot \Pi_4 \cdot \begin{pmatrix} \mathbf{I} & \mathbf{R}_4 \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \cdot \Pi_5 \tag{5.4}$$

(with the Π_i's random permutations and the $\mathbf{R}_i$'s random small matrices) has sufficient entropy and lack of algebraic structure to server as randomizers for our scheme.

We note that although these randomizers are far from uniform, there may still be hope of using some of the tools developed in [BR14b, BGK$^+$14, PST14] (where the analysis includes a reduction to Kilian's information-theoretic argument). This is because the matrices before randomization are permutation matrices, and hence the random permutations Π_i can be used to perfectly randomize them. In this way, one can view the $\mathbf{R}_i$'s are merely "safeguards" to protect against possible weaknesses in the encoding scheme, and the Π_i's are "ideal model randomizers" than can be used in an ideal-model analysis. So far we did not attempt such analysis, however.

Another way to introduce Kilian-type rerandomization in our setting is the aforementioned option of applying it "on the encoding side," i.e., choosing random $m \times m$ invertible matrices $\mathbf{P}$ modulo q and set $\mathbf{C}' \leftarrow \mathbf{P}^{-1} \cdot \mathbf{C} \cdot \mathbf{P}'$.

Multiplicative binding and sraddling sets. Another difference between our setting and that of GGH or CLT is that the previous schemes support encoding relative to arbitrary subsets of a universe set, so there are exponentially many potential sets to use. In our scheme the encoding is relative to edges of a given graph, and there can be only polynomial many of them. This difference seems particularly critical in the design of sraddling sets [BGK$^+$14, PST14].

On a second look, however, this issue is more a question of modeling, rather than a real difference. The different encoding sets in the "asymmetric variants" of [GGH13a, CLT13] are obtained just by multiplying by different random secret constants (e.g., the z_i's from GGH), and we can similarly multiply our encoding matrices by such random constants mod q (especially when working over a large polynomial ring). We use that option in the candidate scheme that we describe below.

We note that similar effects can be obtained by the multiplicative binding technique of [GGH$^+$13b]. Roughly speaking, the main difference between multiplicative binding and sraddling sets is that the former multiplies by constants "on the plaintext side" while the latter multiplies "on the encoding side." In our setting we can do both, and indeed it seems prudent to do so.

A Concrete BP-Obfuscation Candidate

For our concrete candidate below we work over a large polynomial ring of dimension k, and we will use small-dimension matrices over this ring (roughly as high as the dimension of the underlying branching program).

Let $\mathsf{Sym}(w)$ be the set of $w \times w$ permutation matrices and consider a length-n branching program over ℓ bit inputs:

$$\mathsf{BP} = \{(\mathsf{ind}(i), \mathbf{B}_{i,0}, \mathbf{B}_{i,1} : i \in [n], \mathsf{ind}(i) \in [\ell], \mathbf{B}_{i,b} \in \mathsf{Sym}(w)\}$$

For a bit position $j \in [\ell]$, let I_j be the steps in the branching program that examines j'th input bit: $I_j = \{i \in [n] : \mathsf{ind}(i) = j\}$. We obfuscate BP as follows:

- Following the original construction of [GGH$^+$13c] we embed the $\mathbf{B}_{i,\sigma}$'s inside higher-dimension matrices with random elements on the diagonal, but in our case it is sufficient to have only two such random entries (so the dimension only grows form w to $w+2$). Denote the higher-dimension matrices by $\mathbf{B}'_{i,\sigma}$.

  We also follow the original construction of [GGH$^+$13c] by applying the same transformation to a "dummy program" DP of the same length that consists of only the identity matrices, let $\mathbf{D}'_{i,\sigma}$ be the higher-dimension dummy matrices.

- We proceed to randomize these branching programs a-la-Kilian "on the plaintext side," by choosing randomizing matrices $\mathbf{R}_i$'s as per the form of Eqn. (5.4) such that both $\mathbf{R}_i$ and $\mathbf{R}_i^{-1}$ are small, and setting $\mathbf{B}''_{i,\sigma} = \mathbf{R}_{i-1}\mathbf{B}'_{i,\sigma}\mathbf{R}_i^{-1}$. The dummy program is randomized similarly.

- We then prepare $(w+2) \times (w+2)$ "bookend matrices" $\mathbf{S}, \mathbf{S}'$, and "bookend column vectors" $\mathbf{t}, \mathbf{t}'$. $\mathbf{S}$ is random and small except the first row which is zero, $\mathbf{t}$ is random and small except the second entry which is zero, and similarly for $\mathbf{S}'$ and $\mathbf{t}'$, subject to $\mathbf{S}' \cdot \mathbf{t}' = \mathbf{S} \cdot \mathbf{t}$. Then we set $\tilde{\mathbf{S}} = \mathbf{S}\mathbf{R}_0^{-1}$ and $\tilde{\mathbf{t}} = \mathbf{R}_n\mathbf{t}$, and similarly $\tilde{\mathbf{S}'} = \mathbf{S}'\mathbf{R}_0^{-1}$ and $\tilde{\mathbf{t}'} = \mathbf{R}_n\mathbf{t}'$.

- We also sample random small scalars $\{\alpha_{i,0}, \alpha_{i,1}, \alpha'_{i,0}, \alpha'_{i,1} : i \in [n]\}$, subject to constraint: $\prod_{i \in I_j} \alpha_{i,0} = \prod_{i \in I_j} \alpha'_{i,0}$ and $\prod_{i \in I_j} \alpha_{i,1} = \prod_{i \in I_j} \alpha'_{i,1}$. These are used for the "plaintext-side" multiplicative bundling.

  We set $\mathbf{B}^*_{i,\sigma} = \mathbf{B}''_{i,\sigma} \cdot \alpha_{i,\sigma}$ for the main program and similarly $\mathbf{D}^*_{i,\sigma} = \mathbf{D}''_{i,\sigma} \cdot \alpha'_{i,\sigma}$ for the dummy program.

- Next we use our encoding scheme to encode these matrices relative to a graph with two chains with a common end-point, each of length $n+2$. Namely we have $\mathbf{A}_1 \to \ldots \to \mathbf{A}_{n+2}$ and $\mathbf{A}'_1 \to \ldots \to \mathbf{A}'_{n+1} \to \mathbf{A}_{n+2}$.

  For each $i \in [n]$, we encode the two matrices $\mathbf{B}^*_{n-i+1,b}$ relative to the edge $\mathbf{A}_i \to \mathbf{A}_{i+1}$, i.e., we have
  $$\mathbf{C}^T_{n-i+1,b} \cdot \mathbf{A}^T_i = \mathbf{A}^T_{i+1} \cdot \mathbf{B}^*_{n-i+1,b} + \mathbf{E}_{i,b}$$
  for some small error $\mathbf{E}_{i,b}$. Similarly we encode the dummy program with the two matrices $\mathbf{D}^*_{n-i+1,b}$ encoded relative to the edge $\mathbf{A}'_i \to \mathbf{A}'_{i+1}$, i.e.,
  $$\mathbf{C}'^T_{n-i+1,b} \cdot \mathbf{A}'^T_i = \mathbf{A}'^T_{i+1} \cdot (\mathbf{D}^*_{n-i+1,b})^T + \mathbf{E}'_{i,b}$$

- Encode $\tilde{\mathbf{S}}, \tilde{\mathbf{S}'}$ relative to the edges leading to the common sink, i.e. compute the encoding matrices $\mathbf{C}_S, \mathbf{C}'_{S'}$ such that
  $$\mathbf{C}^T_S \cdot \mathbf{A}^T_{n+1} = \mathbf{A}^T_{n+2} \cdot \tilde{\mathbf{S}} + \mathbf{E}_S \text{ and } \mathbf{C}'^T_{S'} \cdot \mathbf{A}'^T_{n+1} = \mathbf{A}^T_{n+2} \cdot \tilde{\mathbf{S}'} + \mathbf{E}'_{S'}$$

- Compute the encoded bookend vectors, folded into the two sources $\mathbf{A}_1$ and $\mathbf{A}'_1$, namely $\mathbf{a} = \mathbf{A}^T_1 \cdot \tilde{\mathbf{t}} + \mathbf{e}_t$ and $\mathbf{a}' = \mathbf{A}'^T_1 \cdot \tilde{\mathbf{t}'} + \mathbf{e}'_t$.

- We next apply both the multiplicative bundling and the the Kilian-style randomization *also on the encoding side.* Namely we sample random full-rank matrices $\mathbf{P}_0, \ldots, \mathbf{P}_n$ and $\mathbf{P}'_0, \ldots, \mathbf{P}'_n$, and also random scalars modulo q $\{\beta_{i,0}, \beta_{i,1}, \beta'_{i,0}, \beta'_{i,1} : i \in [n]\}$, subject to constraints $\prod_{i \in I_j} \beta_{i,0} = \prod_{i \in I_j} \beta'_{i,0} = \prod_{i \in I_j} \beta_{i,1} = \prod_{i \in I_j} \beta'_{i,1} = 1$.

  We then set $\hat{\mathbf{C}}_{i,\sigma} = \mathbf{P}_i^{-1} \cdot \mathbf{C}_{i,\sigma} \cdot \mathbf{P}_{i-1} \cdot \beta_{i,\sigma}$ and $\hat{\mathbf{C}}'_{i,\sigma} = \mathbf{P}'^{-1}_i \cdot \mathbf{C}'_{i,\sigma} \cdot \mathbf{P}'_{i-1} \cdot \beta'_{i,\sigma}$, and also $\hat{\mathbf{C}}_S = \mathbf{P}_0 \cdot \mathbf{C}_S$ and $\hat{\mathbf{C}}'_{S'} = \mathbf{P}'_0 \cdot \mathbf{C}'_{S'}$ and $\hat{\mathbf{a}} = \mathbf{P}_n^{-1}\mathbf{a}$ and $\hat{\mathbf{a}'} = \mathbf{P}'^{-1}_n\mathbf{a}'$.

- The obfuscation consists of all the matrices and vectors above, namely
  $$\mathcal{O}(\mathsf{BP}) = \left(\left\{\hat{\mathbf{C}}_S, \left\{\hat{\mathbf{C}}_{i,\sigma} : i \in [n], \sigma \in \{0,1\}\right\}, \hat{\mathbf{a}}\right\},\ \left\{\hat{\mathbf{C}}'_{S'}, \left\{\hat{\mathbf{C}}'_{i,\sigma} : i \in [n], \sigma \in \{0,1\}\right\}, \hat{\mathbf{a}'}\right\}\right)$$

Evaluation. On input $\mathbf{x} \in \{0,1\}^\ell$ the user choose the appropriate encoding matrices $\hat{\mathbf{C}}_{i,0}$ or $\hat{\mathbf{C}}_{i,1}$ depending on the relevant input bit (and the same for $\hat{\mathbf{C}}'_{i,0}$ or $\hat{\mathbf{C}}'_{i,1}$) and then multiply in order setting

$$\mathbf{y} \;=\; \hat{\mathbf{C}}_S^T \cdot (\prod_{i=1}^{n} \hat{\mathbf{C}}_{i,x[\mathsf{ind}(i)]}^T) \cdot \mathbf{a} \;=\; \mathbf{A}_{n+2}^T \cdot \big(\mathbf{S} \cdot (\prod_{i=1}^{n} \mathbf{B}''_{i,x[\mathsf{ind}(i)]}) \cdot \mathbf{t}\big) + \mathbf{e}$$

and

$$\mathbf{y}' \;=\; \hat{\mathbf{C}'}_{S'}^T \cdot (\prod_{i=1}^{n} \hat{\mathbf{C}'}_{i,x[\mathsf{ind}(i)]}^T) \cdot \mathbf{a}' \;=\; \mathbf{A}_{n+2}^T \cdot \big(\mathbf{S}' \cdot (\prod_{i=1}^{n} \mathbf{D}''_{i,x[\mathsf{ind}(i)]}) \cdot \mathbf{t}'\big) + \mathbf{e}',$$

The output is 1 if $\|\mathbf{y} - \mathbf{y}'\| < q^{3/4}$ and 0 otherwise. Note that indeed if $\prod_{i=1}^n \mathbf{D}_{i,x[\mathsf{ind}(i)]} = \mathbf{I}$ then both $\mathbf{y}$ and $\mathbf{y}'$ are roughly equal to $\mathbf{A}_{n+2}^T \cdot \mathbf{S} \cdot \mathbf{t} \cdot (\prod_{i=1}^n \alpha_{i,x[\mathsf{ind}(i)]})$, as needed.

Security. As before, this is merely a candidate and we do not know how to reduce its security to any "nice" assumption. However the type of attacks that we know against these scheme do not seem to apply to this candidate.

5.7 Parameter Selection

We now describe the parameter-generation procedure PrmGen, showing how to choose the parameters for our scheme. The procedure takes as input the security parameter λ, a DAG G with diameter d, and the class $\mathcal{C}$ of supported circuits. It outputs n, m, q and the Gaussian parameters s, σ. The constraints that these parameters need to satisfy are the following:

- It should be possible to efficiently sample from the input/error distribution $\chi = D_{\mathbb{Z},s}$, and the LWE problem with parameters n, m, q, χ should be hard. This means that we need (say) $s = \sqrt{n}$ and $q/s < 2^{n/\lambda}$.
- It should possible to generate trapdoor for the $\mathbf{A}_v$'s, that enables sampling from SamPre with parameter σ. By Lemma 2.4.1, this means that we need $m = \Omega(n \log q)$ and $\sigma = \Omega(\sqrt{n \log q})$.
- For any supported circuit, the size of the error $\mathbf{E}$ at the output of the circuit must remain below $q^{3/4}$. Namely if the output is an encoding $\mathbf{D}$ of the plaintext matrix $\mathbf{S}$ relative to path $u \rightsquigarrow v$, then we need $\|[\mathbf{D}^T \mathbf{A}_u^T - \mathbf{A}_b^T \mathbf{S}]_q\| < q^{3/4}$.

Let us now analyze the error size in the system. We assume here that we use truncated Gaussian distributions, i.e. we condition $D_{\mathbb{Z},s}$ on the output being smaller than $b \overset{\text{def}}{=} s\sqrt{\lambda}$ (which only affect the distribution negligibly.) We similarly condition SamPre on the output being shorter than $B \overset{\text{def}}{=} \sigma\sqrt{\lambda}$. With our settings, we get $b \le n$ and $B \le n\sqrt{\log q}$. Hence the sample procedure always outputs $(\mathbf{S}, \mathbf{C}, \mathbf{D})$ with the plaintext satisfying $\|\mathbf{S}\| < b$ and the encoding matrices satisfying $\|\mathbf{C}\|, \|\mathbf{D}\| < B$.

To analyze the noise development, recall that when multiplying $\mathbf{A} \in \mathbb{Z}^{u\times v}$ by $\mathbf{B} \in \mathbb{Z}^{v\times w}$ we have $\|\mathbf{AB}\| \leq \|\mathbf{A}\|\cdot\|\mathbf{B}\|\cdot v$. This means in particular that multiplying i encoding matrices we get an encoding matrix $\mathbf{D} \in \mathbb{Z}_q^{m\times m}$ with $\|\mathbf{D}\| < B^i m^{i-1}$ and similarly multiplying i plaintext matrices we get a plaintext matrix $\mathbf{S} \in \mathbb{Z}_q^{n\times n}$ with $\|\mathbf{S}\| < b^i n^{i-1}$. Regarding the error, one can show by induction that the product of i encoding matrices has an error $\mathbf{E} \in \mathbb{Z}_q^{m\times n}$ with

$$\|\mathbf{E}\| \;<\; b\cdot\sum_{j=0}^{i-1} B^j m^j b^{d-1-j} n^{d-1-j} \;<\; b\cdot i\cdot B^{i-1} m^{i-1}.$$

Given a class $\mathcal{C}$ of permitted circuits, we consider the canonical representation of the polynomials in this class as sums of monomials. Let D be a bound on the degree of these polynomials, R be a bound on the size of the coefficients, and N be a bound on the number of monomials. Note that in our setting, the degree-bound D cannot be larger than the diameter of the graph G (since G is acyclic and hence cannot have directed paths longer than d). The size of the error in this case could grow as much as $N\cdot R\cdot b\cdot D\cdot B^{D-1}m^{D-1}$. With $b \leq n$ and $B \leq n\sqrt{\log q}$, we thus have the constraint

$$\begin{aligned} q^{3/4} \;&>\; N\cdot R\cdot n\cdot D\cdot \left(n\sqrt{\log q}\right)^{D-1} m^{D-1} \\ &=\; N\cdot R\cdot n^D\cdot m^{D-1}\cdot D\cdot \left(\log q\right)^{(D-1)/2}. \end{aligned} \tag{5.5}$$

Substituting $m = \Theta(n\log q)$, and $q = 2^{n/\lambda}$, we can use Eqn. (5.5) to solve for n in terms of λ, N, R and D. With $D \leq d$ and assuming the (typical) case of $R = \text{poly}(\lambda)$ and $N < d^d$, it can be verified that this bound is satisfied using $n = \Theta(d\lambda\log(d\lambda))$. This setting yields $q = 2^{n/\lambda} = 2^{\Theta(d\log(d\lambda))} = (d\lambda)^{\Theta(d)}$ and $m = \Theta(n\log q) = \Theta(d^2\lambda^2\log^2(d\lambda))$.

Note that with this setting, each matrix $\mathbf{A}_v \in \mathbb{Z}_q^{n\times m}$ is of size $mn\log q = \Theta(d^4\lambda^2\log^4(d\lambda))$ bits. The public parameters typically contain just one or a handful of such matrices (corresponding to the source nodes in G), but the secret parameters contain all of them. Hence the secret parameters are of size $\Theta(|V|\times d^4\lambda^2\log^4(d\lambda)) = \Omega(d^5\lambda^2\log^4(d\lambda))$ bits. (We have $|V| > d$ since the diameter of G is d.) The encoding matrices are of dimension $m\times m$, but their entries are small, so they can be represented by roughly $m^2\log n = \Theta(d^4\lambda^2\log^5(d\lambda))$ bits.

Working over a larger ring. As usual, we can get better parameters by working over larger rings, and let n denote the extension degree of the ring. In this case the matrices $\mathbf{A}$ are $m\times 1$ column vectors over the larger ring, and we can find trapdoors for these matrices already for $m = \Theta(\log q)$, and also the plaintext elements are now scalars (or constant-degree matrices).

This only affects Eqn. (5.5) or the solution $n = \Theta(d\lambda\log(d\lambda))$ by a constant factor, and hence shaves only a constant factor from the number of bits in $q = 2^{\Theta(d\log(d\lambda))}$, but now we have $m = \Theta(\log q) = \Theta(d \log(d\lambda))$. With each scalar in R_q represented by $n \log q$ bits, it takes $mn \log q = \Theta(d^3 \lambda \log^3(d\lambda))$ bits to represent each matrix $\mathbf{A}_v \in R_q^{1\times m}$, and $\Theta(m^2 \log n) = \Theta(d^3 \lambda \log^4(d\lambda))$ bits to represent each encoding matrix with small entries.

5.8 Conclusions and Open Problems

In this chapter, we presented a new multilinear map candidate from standard (random) lattices and showed its applications to key-exchange and general purpose program obfuscation. It remains an intriguing open problem to understand the security of our construction further, and, if possible, provide reductions from well-studied lattice problems. Alternatively, it should be possible to reduce the security of our constructions to new stand-alone assumptions and understand their hardness. It remains open to construct multilinear map directly from one of the standard assumptions used in cryptography, such as learning-with-errors.

Chapter 6

Fully Homomorphic Signatures

Motivated by the prevalence of cloud computing, there has been much interest in cryptographic schemes that allow a user Alice to securely outsource her data to an untrusted remote server (e.g., the cloud), while also allowing the server to perform useful computations over this data. The ground-breaking development of *fully homomorphic encryption* (FHE) by Gentry [Gen09] allows Alice to maintain the *privacy* of her data by encrypting it, while allowing the server to homomorphically perform arbitrary computations over the ciphertexts. In this chapter, we are interested in the dual question of *authenticity*.

Homomorphic Signatures. A *homomorphic signature* scheme allows Alice to sign some large dataset x using her secret signing key. She can then distribute the signed data to some untrusted entity called a "data processor". A data processor can perform arbitrary computations $y = f(x)$ over this data and homomorphically derive a signature $\sigma_{f,y}$, which certifies that y is the correct output of the computation f over Alice's data. The derived signature $\sigma_{f,y}$ should be short, with length independent of the size of the data x. Anybody can verify the tuple $(f, y, \sigma_{f,y})$ using Alice's public verification key and become convinced that y is indeed the correct output of the computation f over Alice's signed data x, without needing to download the entire data x. Although the computational effort of verifying a derived signature is proportional to the complexity of computing the function f, this work can be performed *offline* prior to seeing the signature, and can be amortized when verifying the same computation over many datasets.

Application: Computation on Outsourced Data. As the most basic application of homomorphic signatures, a user Alice can outsource her signed data to a remote server acting as a data processor, and later verify various computations performed by the server over her data. Using homomorphic signatures, this can be done with *minimal communication and interaction* consisting of a single short signature sent from the server to Alice. Although verifying the correctness of this computation takes Alice as much time as the computation itself, she avoids having to store this data long term. We refer the reader to Section 6.1.1 for a detailed comparison with related work on delegating memory and computation.

Application: Certified Data Analysis. The *non-interactive* nature of homomorphic signatures and the fact that they provide *public verifiability* also makes them useful in a wide variety of settings beyond the above outsourcing scenario.

For example, consider a scenario where a trusted agency such as the National Institute of Health (NIH), runs a large-scale medical study. It signs the collected data x and distributes it to various research groups and pharmaceutical companies for analysis. Some of these groups may have incentives to lie about the outcomes of their analysis and are not trusted by the public. However, using homomorphic signatures, they can publicly post their methodology for the analysis (a function f), the claimed outcome of the analysis ($y = f(x)$), and a short signature $\sigma_{f,y}$ that certifies the correctness of the outcome. This information can be posted publicly (e.g., on third-party websites) and verified *by the public* using a verification key published by the NIH. The public *does not need to have access to the underlying data* and *does not need to interact* with the NIH or the research groups that performed the computation to verify such signatures – indeed, these entities may go offline and the underlying data may be deleted after the analysis is performed but the signed results remain verifiable. Furthermore, such signatures can be made *context hiding* to ensure that they do not reveal anything about the underlying medical data beyond the outcome of the analysis. In this case, the NIH trusts the research groups to preserve the privacy of the underlying data (from the world), but the world does not trust the research groups to perform the analysis correctly.

6.1 Our Contributions, Techniques and Related Work

In this chapter, we construct the first *(leveled) fully homomorphic signature* schemes that can evaluate arbitrary circuits over signed data, where only the maximal depth d of the circuit needs to be fixed a priori. The size of the evaluated signature grows polynomially in d, but is otherwise independent of the data size or the circuit size. This is an exponential improvement in capabilities over the best prior homomorphic signature schemes which could only evaluate polynomials of some bounded degree k where the efficiency degraded polynomially with k (in general, $k = 2^{O(d)}$).

Our solutions are based on the (subexponential) hardness of the *small integer solution* (SIS) problem in standard lattices, which is in turn implied by the worst-case hardness of standard lattice problems [Ajt96]. This is also a significant improvement in assumptions over the best prior schemes which either relied on ideal lattices or multi-linear maps.

We get a scheme in the standard model, where the maximal dataset size needs to be bounded by some polynomial N during setup and the size of the public parameters is linear in N. In the random-oracle model, we get rid of this caveat and get a scheme with short public parameters and without any a-priori bound on the dataset size. In both cases, the user can sign arbitrarily many different datasets by associating each dataset with some label (e.g., a file name). The verifier must know the label of the dataset on which the computation was supposed to be performed when verifying the output.

Efficient Online/Amortized Verification. Our schemes allow for fast amortized verification of a computation f over many different datasets, even if the datasets belong to different users with different verification keys. In particular, the verifier only needs to perform work proportional to the circuit size of f once to verify arbitrarily many signatures under arbitrary verification keys. This work can be performed offline prior to receiving any of the signatures, and the online verification can then be much more efficient than computing f.

Context Hiding. Our schemes can also be made *context hiding* so that a signature $\sigma_{f,y}$ does not reveal any additional information about the underlying data x beyond the output $y = f(x)$. We show how to achieve this statistically without relying on any additional assumptions.

Composition. Our schemes also allow composition of several different computations over signed data. One evaluator can compute some functions $y_1 = h_1(x), \ldots, y_\ell = h_\ell(x)$ over signed data x and publish the homomorphically computed signatures $\sigma_{h_1,y_1}, \ldots, \sigma_{h_\ell,y_\ell}$. A second evaluator can then come and perform an additional computation $y^* = g(y_1, \ldots, y_\ell)$ on the outputs of the previous computation and combine the signatures $\sigma_{h_1,y_1}, \ldots, \sigma_{h_\ell,y_\ell}$ into $\sigma_{g\circ\bar{h},y^*}$ which certifies y^* as the output of the composed computation $(g \circ \bar{h})(x) \overset{\text{def}}{=} g(h_1(x), \ldots, h_\ell(x))$. The second evaluator does not need to know the original data x or the original signatures. This can continue as long as the total computation is of bounded depth.

6.1.1 Related Work

Linearly Homomorphic Schemes. Many prior works consider the question of homomorphic message authentication codes (MACs with private verification) and signatures (public verification) for restricted homomorphisms, and almost exclusively for *linear functions*: [ABC$^+$07, SBW08, DVW09, AKK09, AB09, BFKW09, GKKR10, BF11a, AL11, BF11b, CFW12, Fre12]. Such MACs and signautres have interesting applications to *network coding* and *proofs of retrievability.*

Homomorphic Signatures Beyond Linear. Boneh and Freeman [BF11a] were the first to consider homomorphic signatures beyond linear functions, and propose a general definition of such signatures. They present a scheme that can evaluate arbitrary *polynomials* over signed data, where the maximal *degree* k of the polynomial is fixed a priori and the size of the evaluated signature grows (polynomially) in k. If we want to translate this to the setting of circuits, then a circuit of depth d can be represented by a polynomial of degree as high as $k = 2^d$, and therefore the signature size can grow exponentially in the depth of the circuit. The construction is based on the hardness of the *Small Integer Solution* (SIS) problem in ideal lattices and has a proof of security in the random-oracle model. The recent work of Catalano et al. [CFW14] gives an alternate solution using multi-linear maps which removes the need for random oracles at the cost of having large public parameters. The main open question left by these works is to construct signatures with greater levels of homomorphism, and ideally a *fully homomorphic* scheme that can evaluate arbitrary circuits.

Homomorphic MACs Beyond Linear. There has also been progress in constructing *homomorphic message authentication (MACs)* with private verification for larger classes of homomorphisms. The work of Gennaro and Wichs [GW13] defines and achieves *fully homomorphic MACs* using fully homomorphic encryption. However, the security of the scheme only holds in a setting *without verification queries* and can completely break down if the attacker has access to a verification oracle allowing him to test whether authentication tags are valid. More recent works [CF13, BFR13, CFGN14] show how to get homomorphic MACs that remain secure in the presence of verification queries, but only for restricted homomorphisms. Currently, the best such schemes allow for the evaluation of polynomials of degree k, where the computational effort grows polynomially with k (but the size of the evaluated authentication tag stays fixed). In other words, the question of (leveled) fully homomorphic authentication, even in the setting of private verification, remained open prior to this work.

Other Types of Homomorphic Authentication. We also mention works on specific types of homomorphic properties such as *redactable signatures* (see e.g., [JMSW02, ABC$^+$12]) where, given a signature on a long message x, it should be possible to derive a signature on a subset/substring x' of x. The work of [ABC$^+$12] proposes a general notion of P-homomorphic signature schemes for various predicates P, but efficient constructions were only known for a few specific instances. [1]

Homomorphic Signatures via SNARKs. There is a very simple construction of fully homomorphic signatures by relying on CS-Proofs [Mic94] or, more generally, *succinct non-interactive arguments of knowledge for* **NP** (SNARKs) [BCCT12, BCCT13, BCI$^+$13, GGPR13, PHGR13, BSCG$^+$13]. This primitive allows us to non-interactively create a short "argument" π for any **NP** statement so that π proves "knowledge" of the corresponding witness. The length of π is bounded by some fixed polynomial in the security parameter and is independent of the statement/witness size. The complexity of verifying π only depends on the size of the statement (but not the witness). Using SNARKs, we can authenticate the output $y = f(x)$ of any computation f over any signed data x (under any standard signature scheme) by creating a short argument $\pi_{f,y}$ that proves the knowledge of "data x along with valid signature of x, such that $f(x) = y$".

[1] There is a potentially confusing syntactic difference between the notion of P-homomorphic signatures and the notion of homomorphic signatures in this work, although the two are equivalent. In P-homomorphic signatures, given a signature of x one should be able to derive a signature of x' as long as $P(x, x') = 1$ for some predicate P (e.g., the substring predicate). The same effect can be achieved using the syntax of homomorphic signatures in this work by defining a function $f_{x'}$ that has x' hard-coded and computes $f_{x'}(x) = P(x, x')$. We would then give a derived signature $\sigma_{f_{x'},1}$ certifying that $y = 1$ is the output of the computation $f_{x'}(x)$ over the originally signed data x.

One advantage of the SNARK-based scheme is that a signature can be verified very efficiently, independently of the complexity of the computation f being verified, as long as f has a short Turing-Machine description. In contrast, in this work, we will only get efficient verification in an amortized sense, when verifying a computation f over many different datasets. Unfortunately, constructing SNARKs, even without a "knowledge" requirement, is known to require the use of *non-standard* assumptions [GW11]. The additional requirement of (non black-box) knowledge extraction makes SNARKs even more problematic and is unlikely to be possible in its full generality [BCPR14]. Known SNARK constructions are either based on the random-oracle heuristic *and* the use of PCP machinery, or on various "knowledge of exponent" assumptions and light-weight PCP variants.

Delegating Computation. Several other works consider the related problem of *delegating computation* to a remote server while maintaining the ability to efficiently verify the result [GKR08, GGP10b, CKV10, AIK10, BGV11, CKLR11, PRV12, PST13, KRR13]. In this scenario, the server needs to convince the user that $f(x) = y$, where the user knows the function f, the input x and the output y, but does not want to do the *work* of computing $f(x)$. In contrast, in our scenario the verifier only knows f and y but does *not* know the previously authenticated data x, which may be huge. As mentioned in [GW13], some of the results for delegating computation in the pre-processing model can also be re-interpreted as giving results for the latter scenario. The latter scenario was also explicitly considered by Chung et al. [CKLR11] in the context of *memory delegation*, where the client can also update the data on the server. Some of these solutions only allow a single party with secret key to verify computation, while others (e.g., [PRV12]) allow anyone to verify. However, all of the above solutions for memory delegation and delegating computation require at least some interaction between the client and server (often just a challenge-response protocol) to verify a computation f. Therefore they do not give us a solution to the problem of homomorphic signatures (or even homomorphic MACs), where we require a *static* certificate which certifies the output of a computation and which can be posted publicly and verified by everyone.

6.1.2 Our Techniques

Our constructions of homomorphic signatures are modular and, as a building block of potentially independent interest, we present a new primitive called a *homomorphic trapdoor function* (HTDF). This primitive allows us to conceptually unite homomorphic encryption and signatures. We now give a high-level overview of our techniques. We start with the notion of HTDFs, then show how to construct homomorphic signatures from HTDFs, and finally show how to construct HTDFs from the SIS problem.

Homomorphic Trapdoor Functions (HTDF). An HTDF consists of a function $v = f_{pk,x}(u)$ described via a public key pk and an index $x \in \{0, 1\}$, with input u and output v. It will be useful to think of this as a commitment scheme where x is the message, v is the commitment and u is the randomness/decommitment. Given some values

$$u_1, x_1, v_1 = f_{pk,x_1}(u_1) \quad , \ldots , \quad u_N, x_N, v_N = f_{pk,x_N}(u_N)$$

and a circuit $g : \{0,1\}^N \to \{0,1\}$, we can homomorphically compute an input $u^* := \mathsf{HTDF.Eval}^{in}(g, (x_1, u_1) \ldots , (x_N, u_N))$ and an output $v^* := \mathsf{HTDF.Eval}^{out}(g, v_1, \ldots , v_N)$ such that:

$$f_{pk,g(x_1,\ldots,x_N)}(u^*) = v^*.$$

Thinking of HTDFs as commitments, the above says that if the values v_i are commitments to the message bits x_i with decommitments u_i, then we can homomorphically combine the v_i to derive a commitment v^* and homomorphically combine the messages/decommitments (x_i, u_i) to get a decommitment u^* which opens v^* to the message $g(x_1, \ldots , x_N)$.

We want to be able to generate the public key pk of the HTDF together with a *trapdoor* sk that allows us to take any output v and efficiently invert it with respect to any index x to get $u \leftarrow \mathsf{Inv}_{sk,x}(v)$ such that $f_{pk,x}(u) = v$. In the language of commitments, this means that we want the scheme to be equivocable (and therefore statistically hiding) with a trapdoor sk that lets us open any commitment to any message.

For security, we simply require that the HTDF is *claw-free*: given pk, it should be hard to come up with inputs u_0, u_1 such that $f_{pk,0}(u_0) = f_{pk,1}(u_1)$. Equivalently, in the language of commitments, we want the scheme to be computationally binding.

As an intellectual curiosity (but of no application to this work), we could also change our requirements and instead ask for an HTDF which is extractable (and therefore statistically binding) while also being computationally hiding. In other words, we would want to generate pk along with a trapdoor sk that would allow us to extract x from $v = f_{pk,x}(u)$. In this case, such an HTDF could also be though of as a fully homomorphic encryption scheme, where v is the ciphertext of a message x and $\mathsf{HTDF.Eval}^{out}$ is the homomorphic evaluation procedure on ciphertexts. Our eventual construction of an HTDF will provide both options by allowing us to choose pk in one of two indistinguishable modes: an equivocable mode and an extractable mode. In this work, we will solely rely on the equivocable mode to construct homomorphic signatures. However, the extractable mode of the HTDF (essentially) corresponds to the Gentry-Sahai-Waters [GSW13] fully homomorphic encryption scheme. Therefore, we view HTDFs as providing an interesting conceptual unification of homomorphic signatures and encryption. We refer the reader to Section 6.6 for more on this grand unification.

Basic Homomorphic Signatures from HTDFs. We construct several flavors of homomorphic signature schemes using HTDFs as a black-box. As the most basic flavor, we construct a signature scheme in the standard model where the setup procedure knows some bound N on the size of the dataset that will be signed and the size of the public parameters can depend on N. The public parameters $\mathsf{prms} = (v_1, \ldots , v_N)$ consist of N random outputs of the HTDF. Each user chooses a pubic/secret key pair (pk, sk) for an HTDF, which also serves as the key pair for the signature scheme. To sign some data $x = (x_1, \ldots , x_N) \in \{0,1\}^N$ the user simply finds inputs u_i such that $f_{pk,x_i}(u_i) = v_i$ by using sk to invert v_i. We think of u_i as a signature that ties x_i to its position i.

Given x, the signatures $u_1, \ldots, u_N$, and a function $g : \{0,1\}^N \to \{0,1\}$, anybody can homomorphically compute a signature $u^*_{g,y} := \mathsf{HTDF.Eval}^{in}(g, (x_1, u_1), \ldots, (x_N, u_N))$ which certifies $y = g(x_1, \ldots, x_N)$ as the output of the computation g. To verify the tuple $(g, y, u^*_{g,y})$, the verifier will compute $v^* := \mathsf{HTDF.Eval}^{out}(g, v_1, \ldots, v_N)$ and checks $f_{pk,y}(u^*_{g,y}) \overset{?}{=} v^*$. Notice that verification procedure only depends on the public parameters but does not know the data x. We can show that this basic scheme already satisfies selective security, where we assume that dataset $x_1, \ldots, x_N$ is chosen by the attacker before seeing the public parameters. In the reduction, instead of choosing v_i randomly, we choose a random input u_i and compute $v_i := f_{pk,x_i}(u_i)$ using the data x_i that the attacker wants signed. This makes it easy for the reduction to generate signatures for x_i. Furthermore, for any function g, the reduction can compute the honest signature $u = \mathsf{HTDF.Eval}^{in}(g, (x_1, u_1), \ldots, (x_N, u_N))$ which certifies the output $y = g(x_1, \ldots, x_N)$. If an attacker produces a forged signature u' that certifies $y' \neq y$ then $f_{pk,y}(u) = f_{pk,y'}(u')$ and therefore (u, u') breaks the claw-free security of the HTDF.

Upgrading Functionality and Security. We show how to generically start with a basic homomorphic signature scheme as above and convert into more powerful variants of homomorphic signatures with improved functionality, efficiency, and security.

Firstly, we note that since the public parameters $\mathsf{prms} = (v_1, \ldots, v_N)$ of our basic scheme are uniformly random values, we can easily compress them in the random oracle model to get a scheme with short public parameters. In particular, the public parameters of the new scheme only consist of a short random string r and we can derive the values $v_i = H(r, i)$ using a random oracle H. We can also translate this random-oracle scheme into a standard-model assumption on the hash function H which is simple-to-state and falsifiable, but nevertheless non-standard. This gives us a tradeoff between efficiency and assumptions.

Next, we give a generic transformation from a homomorphic signature scheme with selective security to a scheme with full adaptive security. Our transformation works in both the standard model and the random oracle model. Starting from a selectively secure leveled FHS scheme, we obtain a fully secure leveled FHS scheme.[2]

Lastly, following Boneh and Freeman [BF11a], we can extend the functionality of homomorphic signatures to allow the user to sign multiple different datasets under different *labels* τ (e.g., τ can correspond to a "file name"), where verifier must simply know the label of the dataset on which the computation was supposed to be performed. We show a generic transformation from a basic signature that only works for a single dataset into one that supports multiple datasets. Furthermore, this transformation gives us efficient amortized verification of a computation over multiple datasets.

Constructing HTDFs. We now briefly describe how to construct HTDFs based on the SIS problem. We rely on the homomorphic techniques developed by Gentry et al. [GSW13]

[2]Although Catalano et al. [CFW14] provide a similar transformation, it works only for bounded degree polynomial functions, and does not generalize to leveled FHS.

and by Boneh et al. [BGG+14] in the context of fully homomorphic encryption and attribute-based encryption.

The SIS problem states that, for a random matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ it should be hard to come up with a "short" non-zero vector $\mathbf{u} \in \mathbb{Z}_q^m$, such that $\mathbf{A} \cdot \mathbf{u} = \mathbf{0}$. However, there is a way to generate $\mathbf{A}$ along with a trapdoor td that makes this easy and, more generally, for any matrix $\mathbf{V} \in \mathbb{Z}_q^{n\times m}$, the trapdoor can be used to sample a "short" matrix $\mathbf{U} \in \mathbb{Z}_q^{m\times m}$ such that $\mathbf{AU} = \mathbf{V}$. There is also a public matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ with some special structure (not random) for which everyone can efficiently compute a "short" matrix $\mathbf{G}^{-1}(\mathbf{V})$ such that $\mathbf{G}\mathbf{G}^{-1}(\mathbf{V}) = \mathbf{V}$.[3]

Our HTDF consists of choosing $pk = \mathbf{A}$ together with trapdoor $sk = \mathsf{td}$ as above. We define $f_{pk,x}(\mathbf{U}) \overset{\text{def}}{=} \mathbf{AU} + x \cdot \mathbf{G}$, but we restrict the domain to "short" matrices $\mathbf{U}$. We show that finding a claw consisting of "short" matrices $\mathbf{U}_0, \mathbf{U}_1$ such that $f_{pk,0}(\mathbf{U}_0) = f_{pk,1}(\mathbf{U}_1) \Rightarrow \mathbf{A}(\mathbf{U}_0 - \mathbf{U}_1) = \mathbf{G}$ implies breaking the SIS problem. Next, we show how to perform homomorphic operations on this HTDF.

Homomorphic Operations. Let $\mathbf{U}_1, \mathbf{U}_2 \in \mathbb{Z}_q^{m\times m}$ be "short" matrices and

$$\mathbf{V}_1 = f_{pk,x_1}(\mathbf{U}_1) = \mathbf{AU}_1 + x_1 \cdot \mathbf{G} \ \ , \ \ \mathbf{V}_2 = f_{pk,x_2}(\mathbf{U}_2) = \mathbf{AU}_2 + x_2 \cdot \mathbf{G}$$

Addition. Firstly, it is very easy to perform homomorphic addition (over $\mathbb{Z}_q$). We can simply set: $\mathbf{V}^* = \mathbf{V}_1 + \mathbf{V}_2$ and $\mathbf{U}^* = \mathbf{U}_1 + \mathbf{U}_2$. This ensures:

$$\mathbf{V}^* = (\mathbf{AU}_1 + x_1 \cdot \mathbf{G}) + (\mathbf{AU}_2 + x_2 \cdot \mathbf{G}) = \mathbf{AU}^* + (x_1 + x_2)\mathbf{G} = f_{pk,x_1+x_2}(\mathbf{U}^*).$$

Multiplication. Homomorphic multiplication (over $\mathbb{Z}_q$) consists of setting $\mathbf{V}^* := \mathbf{V}_2\mathbf{G}^{-1}(\mathbf{V}_1)$ and $\mathbf{U}^* := x_2\mathbf{U}_1 + \mathbf{U}_2\mathbf{G}^{-1}(\mathbf{V}_1)$. This gives:

$$\begin{aligned}\mathbf{V}^* &= \mathbf{V}_2\mathbf{G}^{-1}(\mathbf{V}_1) = (\mathbf{AU}_2 + x_2\mathbf{G})\mathbf{G}^{-1}(\mathbf{V}_1) = \mathbf{AU}_2\mathbf{G}^{-1}(\mathbf{V}_1) + x_2(\mathbf{AU}_1 + x_1\mathbf{G}) \\ &= \mathbf{AU}^* + x_1x_2\mathbf{G} = f_{pk,x_1\cdot x_2}(\mathbf{U}^*)\end{aligned}$$

We define the *noise level* of a matrix $\mathbf{U}$ to be the maximal size (in absolute value) of any entry in the matrix. The noise level grows as we perform homomorphic operations. Intuitively, if the inputs to the operation have noise-level β then homomorphic addition just doubles the noise level to 2β, while multiplication of "small" values $x_1, x_2 \in \{0, 1\}$ multiplies the noise level to at most $(m+1)\beta$. Therefore, when evaluating a boolean circuit of depth d the noise level can grow to as much as $\beta(m+1)^d$. We pause to note that the noise growth is a crucial difference between our scheme and that of Boneh and Freeman [BF11a], where multiplication raises the noise level from β to β^2, meaning that evaluating a circuit of depth d could raise the noise level to as high as β^{2^d}. Since the modulus must satisfy $q \gg \beta(m+1)^d$ for security,

[3] Note that we are abusing notation and $\mathbf{G}^{-1}$ is not a matrix but rather a function - for any $\mathbf{V}$ there are many choices of $\mathbf{U}$ such that $\mathbf{GU} = \mathbf{V}$, and $\mathbf{G}^{-1}(\mathbf{V})$ deterministically outputs a particular short matrix from this set. For those familiar with [GSW13], multiplication by $\mathbf{G}$ corresponds to $\mathsf{PowersOf2}()$ and $\mathbf{G}^{-1}()$ corresponds to $\mathsf{BitDecomp}()$.

the level of homomorphism d must be fixed ahead of time, during the setup of the scheme. The overall efficiency degrades *polynomially* with d.

6.1.3 Chapter Organization

In Section 6.2 we present the syntax of our new homomorphic trapdoor functions and instantiate it from lattices. In Section 6.3, we present our construction for single data-set setting. In Section 6.4, we present a general transformation from single to multiple data-sets setting. In Section 6.5, we show how to achieve context-hiding property for our homomorphic signatures scheme. In Section 6.6, we present a connection of homomorphic trapdoor functions and fully-homomorphic encryption. In Section 6.7 we summarize and present some open problems.

6.2 Homomorphic Trapdoor Functions

A homomorphic trapdoor function allows us to take values $\{u_i, x_i, v_i = f_{pk,x_i}(u_i)\}_{i\in[N]}$ and create an input u^* (depending on u_i, x_i) and an output v^* (depending only on v_i) such that $f_{pk,g(x_1,\ldots,x_N)}(u^*) = v^*$. We now give a formal definition.

6.2.1 Definition

A *homomorphic trapdoor function (HTDF)* consists of the following five polynomial-time algorithms ($\mathsf{HTDF.KeyGen}$, f, Inv, $\mathsf{HTDF.Eval}^{in}$, $\mathsf{HTDF.Eval}^{out}$) with syntax:

- $(pk, sk) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda)$: a key generation procedure. The security parameter sec defines the *index space* $\mathcal{X}$, the *input space* $\mathcal{U}$, and the *output space* $\mathcal{V}$ and some efficiently samplable *input distribution* $D_{\mathcal{U}}$ over $\mathcal{U}$. We require that membership in the sets $\mathcal{U}, \mathcal{V}, \mathcal{X}$ can be efficiently tested and that one can efficiently sample uniformly at random from $\mathcal{V}$.

- $f_{pk,x} : \mathcal{U} \to \mathcal{V}$: a deterministic function indexed by $x \in \mathcal{X}$ and pk.

- $\mathsf{Inv}_{sk,x} : \mathcal{V} \to \mathcal{U}$: a probabilistic inverter indexed by $x \in \mathcal{X}$ and sk.

- $u^* = \mathsf{HTDF.Eval}^{in}(g, (x_1, u_1), \ldots, (x_\ell, u_\ell))$, $v^* = \mathsf{HTDF.Eval}^{out}(g, v_1, \ldots, v_\ell)$ are deterministic input/output homomorphic evaluation algorithms. The algorithms take as input some function $g : \mathcal{X}^\ell \to \mathcal{X}$ and values $x_i \in \mathcal{X}$, $u_i \in \mathcal{U}$, $v_i \in \mathcal{V}$. The outputs are $u^* \in \mathcal{U}$ and $v^* \in \mathcal{V}$.[4]

Note that we do not require $f_{pk,x}(\cdot)$ to be an injective function. Indeed, it will not be in our construction.

[4]More precisely, g is a *function description* in some specified format. In our case, this will always be either a boolean or an arithmetic circuit. For simplicity we often say "function g" but refer to a specific representation of the function.

Correctness of Homomorphic Evaluation. Let $(pk, sk) \in \mathsf{HTDF.KeyGen}(1^\lambda)$,[5] $x_1, \ldots, x_\ell \in \mathcal{X}$, $g : \mathcal{X}^\ell \to \mathcal{X}$ and $y := g(x_1, \ldots, x_\ell)$. Let $u_1, \ldots, u_\ell \in \mathcal{U}$ and set $v_i := f_{pk,x_i}(u_i)$ for $i \in [\ell]$. Let $u^* := \mathsf{HTDF.Eval}^{in}(g, (x_1, u_1), \ldots, (x_\ell, u_\ell))$, $v^* := \mathsf{HTDF.Eval}^{out}(g, v_1, \ldots, v_\ell)$. Then we require that $u^* \in \mathcal{U}$ and $f_{pk,y}(u^*) = v^*$.

Relaxation: In a *leveled* fully homomorphic scheme, each input $u_i \in \mathcal{U}$ will have some associated "noise-level" $\beta_i \in \mathbb{R}$. The initial samples from the input-distribution $D_{\mathcal{U}}$ have some "small" noise-level β_{init}. The noise-level β^* of the homomorphically computed input u^* depends on the noise-levels β_i of the inputs u_i, the function g and the indices x_i. If the noise level β^* of u^* exceeds some threshold $\beta^* > \beta_{max}$, then the above correctness need not hold. This will limit the type of functions that can be evaluated. A function g is *admissible* on the values $x_1, \ldots, x_\ell$ if, whenever the inputs u_i have noise-levels $\beta_i \le \beta_{init}$, then $u^* := \mathsf{HTDF.Eval}^{in}(g, (x_1, u_1), \ldots, (x_\ell, u_\ell))$ will have noise-level $\beta^* \le \beta_{max}$.

Distributional Equivalence of Inversion. We require the following statistical indistinguishability:

$$(pk, sk, x, u, v) \overset{\text{stat}}{\approx} (pk, sk, x, u', v')$$

where $(pk, sk) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda)$, $x \in \mathcal{X}$ can be an arbitrary random variable that depends on (pk, sk), $u \leftarrow D_{\mathcal{U}}$, $v := f_{pk,x}(u)$, $v' \leftarrow \mathcal{V}$, $u' \leftarrow \mathsf{Inv}_{sk,x}(v')$.

HTDF Security. We now define the security of HTDFs. Perhaps the most natural security requirement would be *one-wayness*, meaning that for a random $v \leftarrow \mathcal{V}$ and any $x \in \mathcal{X}$ it should be hard to find a pre-image $u \in \mathcal{U}$ such that $f_{pk,x}(u) = v$. Instead, we will require a stronger property which is similar to *claw-freeness*. In particular, it should be difficult to find $u, u' \in \mathcal{U}$ and $x \ne x' \in \mathcal{X}$ such that $f_{pk,x}(u) = f_{pk,x'}(u')$. Formally, we require that for any PPT attacker $\mathcal{A}$ we have:

$$\Pr\left[\begin{array}{c} f_{pk,x}(u) = f_{pk,x'}(u') \\ u, u' \in \mathcal{U}, \quad x, x' \in \mathcal{X} \quad , x \ne x' \end{array} \;\middle|\; \begin{array}{c} (pk, sk) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda) \\ (u, u', x, x') \leftarrow \mathcal{A}(1^\lambda, pk) \end{array}\right] \le \operatorname{negl}(\lambda).$$

6.2.2 Construction: Basic Algorithms and Security

We begin by describing the basic HTDF algorithms for key-generation, computing the function $f_{pk,x}$, and inverting it using sk. We prove the security of the scheme. Then, in 6.2.3 we show how to perform homomorphic operations.

Parameters. Our scheme will be defined by a flexible parameter $d = d(\lambda) = \operatorname{poly}(\lambda)$ which roughly determines the level of homomorphism. We choose parameters:

$$n \ , \ m \ , \ q \ , \ \beta_{SIS} \ , \ \beta_{max} \ , \ \beta_{init}$$

[5]Recall, we use this as shorthand for "(pk, sk) in the support of $\mathsf{HTDF.KeyGen}(1^\lambda)$".

depending on sec and d. We do so by setting $\beta_{max} := 2^{\omega(\log\lambda)d}$, $\beta_{SIS} := 2^{\omega(\log\lambda)}\beta_{\max}$. Then choose an integer $n = \text{poly}(\lambda)$ and a prime $q = 2^{\text{poly}(\lambda)} > \beta_{SIS}$ as small as possible so that the $\mathsf{SIS}(n, m, q, \beta_{SIS})$ assumption holds for all $m = \text{poly}(\lambda)$. Finally, let $m^* = m^*(n,q) := O(n \log q), \beta_{sam} := O(n\sqrt{\log q})$ be the parameters required by the trapdoor algorithms as in Lemma 2.4.1, and set $m = \max\{m^*, n\log q + \omega(\log\lambda)\} = \text{poly}(\lambda)$ and $\beta_{init} := \beta_{sam} = \text{poly}(\lambda)$. Note that $n, m, \log q$ all depend (polynomially) on λ, d.

Construction of HTDF. Let the algorithms $\mathsf{TrapSamp}$, SamPre, Sam, and the matrix $\mathbf{G}$ be as defined in Lemma 2.4.1.

- Define the domains $\mathcal{X} = \mathbb{Z}_q$ and $\mathcal{V} = \mathbb{Z}_q^{n\times m}$. Let $\mathcal{U} = \{\mathbf{U} \in \mathbb{Z}_q^{m\times m} \ : \ ||\mathbf{U}||_\infty \le \beta_{max}\}$. We define the distribution $\mathbf{U} \leftarrow D_{\mathcal{U}}$ to sample $\mathbf{U} \leftarrow \mathsf{Sam}(1^m, 1^m, q)$ as in Lemma 2.4.1, so that $||\mathbf{U}||_\infty \le \beta_{init}$.
- $(pk, sk) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda)$: Select $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$. Set $pk := \mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $sk = \mathsf{td}$.
- Define $f_{pk,x}(\mathbf{U}) \overset{\text{def}}{=} \mathbf{A}\cdot\mathbf{U} + x\cdot\mathbf{G}$. Note that, although the function f is well-defined on all of $\mathbb{Z}_q^{m\times m}$, we restrict the legal domain of f to the subset $\mathcal{U} \subseteq \mathbb{Z}_q^{m\times m}$.
- Define $\mathbf{U} \leftarrow \mathsf{Inv}_{sk,x}(\mathbf{V})$ to output $\mathbf{U} \leftarrow \mathsf{SamPre}(\mathbf{A}, \mathbf{V} - x\cdot\mathbf{G}, \mathsf{td})$.

We define the *noise-level* β of a value $\mathbf{U} \in \mathcal{U}$ as $\beta = ||\mathbf{U}||_\infty$. We note that all efficiency aspects of the scheme (run-time of procedures, sizes of keys/inputs/outputs, etc.) depend polynomially on sec and on the flexible parameter d.

Distributional Equivalence of Inversion. Let $(pk = \mathbf{A}, sk = \mathsf{td}) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda)$, and let $x \in \mathcal{X}$ be an arbitrary random variable that depends on (pk, sk). Let $\mathbf{U} \leftarrow D_{\mathcal{U}}$, $\mathbf{V} = \mathbf{AU} + x\cdot\mathbf{G} = f_{pk,x}(\mathbf{U})$, $\mathbf{V}' \leftarrow \mathcal{V}$, $\mathbf{U}' \leftarrow \{\mathsf{Inv}_{sk,x}(\mathbf{V}') = \mathsf{SamPre}(\mathbf{A}, \mathbf{V}' - x\cdot\mathbf{G}, \mathsf{td})\}$. Then we need to show:

$$(pk = \mathbf{A}, sk = \mathsf{td}, x, \mathbf{U}, \mathbf{V} = \mathbf{AU} + x\mathbf{G}) \overset{\text{stat}}{\approx} (pk = \mathbf{A}, sk = \mathsf{td}, x, \mathbf{U}', \mathbf{V}') \tag{6.1}$$

Lemma 2.4.1, part (2) tells us that:

$$(\mathbf{A}, \mathsf{td}, \mathbf{U}, \mathbf{AU}) \overset{\text{stat}}{\approx} (\mathbf{A}, \mathsf{td}, \mathbf{U}', \mathbf{V}' + x\cdot\mathbf{G}) \tag{6.2}$$

by noticing that $(\mathbf{V}' - x\cdot\mathbf{G})$ is just uniformly random. Equation 6.1 follows from 6.2 by applying the same function to both sides: append a sample x from the correct correlated distribution given $(\mathbf{A}, \mathsf{td})$ and subtract $x\cdot\mathbf{G}$ from the last component.

HTDF Security. We now prove the security of our HTDF construction under the SIS assumption.

Theorem 6.2.1. *Assuming the* $\mathsf{SIS}(n, m, q, \beta_{SIS})$*-assumption holds for the described parameter choices, the given scheme satisfies HTDF security.*

Proof. Assume that $\mathcal{A}$ is some PPT attacker that wins the HTDF security game for the above scheme with non-negligible probability. Let us modify the HTDF game so that, instead of choosing $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$ and setting $pk := \mathbf{A}$ and $sk = \mathsf{td}$, we just choose $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$ uniformly at random. Notice that $sk = \mathsf{td}$ is never used anywhere in the original HTDF game. Therefore, this modification is statistically indistinguishable by the security of $\mathsf{TrapSamp}$ (see Lemma 2.4.1, part (2)). In particular, the probability of $\mathcal{A}$ winning the modified game remains non-negligible.

We now show that an attacker who wins the modified HTDF game can be used to solve the SIS problem. The reduction uses the challenge matrix $\mathbf{A}$ of the SIS problem as the public key $pk = \mathbf{A}$ and runs the attacker $\mathcal{A}$. Assume the attacker $\mathcal{A}$ wins the modified HTDF game with the values $\mathbf{U}, \mathbf{U}' \in \mathcal{U}$ and $x \neq x' \in \mathcal{X}$ such that $f_{pk,x}(\mathbf{U}) = f_{pk,x'}(\mathbf{U}')$. Let $\mathbf{U}^* := \mathbf{U}' - \mathbf{U}$ and $x^* = (x - x')$. Then:

$$f_{pk,x}(\mathbf{U}) = \mathbf{A}\mathbf{U} + x\mathbf{G} = \mathbf{A}\mathbf{U}' + x'\mathbf{G} = f_{pk,x'}(\mathbf{U}') \quad \Rightarrow \quad \mathbf{A}\mathbf{U}^* = x^*\mathbf{G} \tag{6.3}$$

Moreover, since $\mathbf{U}, \mathbf{U}' \in \mathcal{U}$, we have $||\mathbf{U}||_\infty, ||\mathbf{U}'||_\infty \leq \beta_{max}$ and therefore $||\mathbf{U}^*||_\infty \leq 2\beta_{max}$. Moreover, since $x \neq x'$, we have $x^* \neq 0$.

We now show that knowledge of a "small" $\mathbf{U}^*$ and some $x^* \neq 0$ satisfying the right hand side of equation 6.3 can be used to find a solution to the SIS problem. Sample $\mathbf{r} \leftarrow \{0,1\}^m$, set $\mathbf{z} := \mathbf{A}\mathbf{r}$ and compute $\mathbf{r}' = \mathbf{G}^{-1}(\mathbf{z}/x^*)$ so that $\mathbf{r}' \in \{0,1\}^m$ and $x^*\mathbf{G}\mathbf{r}' = \mathbf{z}$. Then

$$\mathbf{A}(\mathbf{U}^*\mathbf{r}' - \mathbf{r}) = (\mathbf{A}\mathbf{U}^*)\mathbf{r}' - \mathbf{A}\mathbf{r} = x^*\mathbf{G}\mathbf{r}' - \mathbf{A}\mathbf{r} = \mathbf{z} - \mathbf{z} = \mathbf{0}.$$

Therefore, letting $\mathbf{u} := \mathbf{U}^*\mathbf{r}' - \mathbf{r}$, we have $\mathbf{A}\mathbf{u} = \mathbf{0}$ and $||\mathbf{u}||_\infty \leq (2m+1)\beta_{max} \leq \beta_{SIS}$. It remains to show that $\mathbf{u} \neq \mathbf{0}$, or equivalently, that $\mathbf{r} \neq \mathbf{U}^*\mathbf{r}'$. We use an entropy argument to show that this holds with overwhelming probability over the random choice of $\mathbf{r}$, even if we fix some worst-case choice of $\mathbf{A}, \mathbf{U}^*, x^*$. Notice that $\mathbf{r} \leftarrow \{0,1\}^m$ is chosen uniformly at random, but $\mathbf{r}'$ depends on $\mathbf{z} = \mathbf{A}\mathbf{r}$. Nevertheless $\mathbf{z}$ is too small to reveal much information about $\mathbf{r}$ and therefore cannot be used to predict $\mathbf{r}$. In particular

$$\mathbf{H}_\infty(\mathbf{r} \mid \mathbf{r}') \geq \mathbf{H}_\infty(\mathbf{r} \mid \mathbf{A}\mathbf{r}) \geq m - n\log q = \omega(\log \lambda)$$

where the first inequality follows since $\mathbf{r}'$ is chosen deterministically based on $\mathbf{z} = \mathbf{A}\mathbf{r}$, and the second inequality follows from Lemma 2.2.1. Therefore, $\Pr[\mathbf{r} = \mathbf{U}^* \cdot \mathbf{r}'] \leq 2^{m - n\log q} \leq \mathrm{negl}(\lambda)$. So, with overwhelming probability, whenever $\mathcal{A}$ wins the modified HTDF game, the reduction finds a valid solution to the $\mathsf{SIS}(n, m, q, \beta_{SIS})$-problem. This concludes the proof. □

6.2.3 Construction: Homomorphic Evaluation and Noise Growth

We now define the algorithms $\mathsf{HTDF.Eval}^{in}$, $\mathsf{HTDF.Eval}^{out}$ with the syntax

$$\mathbf{U}^* := \mathsf{HTDF.Eval}^{in}(g, (x_1, \mathbf{U}_1), \ldots, (x_\ell, \mathbf{U}_\ell)) \quad , \quad \mathbf{V}^* := \mathsf{HTDF.Eval}^{out}(g, \mathbf{V}_1, \ldots, \mathbf{V}_\ell).$$

Our approach closely follows the techniques of [GSW13, BGG+14]. As a basic building block, we consider homomorphic evaluation for certain base functions g which we think of as basic gates in an arithmetic circuit: *addition*, *multiplication*, *addition-with-constant* and *multiplication-by-constant*. These functions are complete and can be composed to evaluate an arbitrary aithmetic circuit. Let the matrices $\mathbf{U}_i$ have noise-levels bounded by β_i.

- Let $g(x_1, x_2) = x_1 + x_2$ be an *addition* gate. The algorithms $\mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out}$ respectively compute:

  $$\mathbf{U}^* := \mathbf{U}_1 + \mathbf{U}_2 \quad , \quad \mathbf{V}^* := \mathbf{V}_1 + \mathbf{V}_2.$$

  The matrix $\mathbf{U}^*$ has noise level $\beta^* \leq \beta_1 + \beta_2$. We remark that, in this case, the algorithm $\mathsf{HTDF.Eval}^{in}$ ignores the values x_1, x_2.

- Let $g(x_1, x_2) = x_1 \cdot x_2$ be a *multiplication* gate. Let $\mathbf{R} = \mathbf{G}^{-1}(\mathbf{V}_1)$ so that $\mathbf{R} \in \{0,1\}^{m \times m}$ and $\mathbf{G}\mathbf{R} = -\mathbf{V}_1$. The algorithms $\mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out}$ respectively compute:

  $$\mathbf{U}^* := x_2 \cdot \mathbf{U}_1 + \mathbf{U}_2\mathbf{G}^{-1}(\mathbf{V}_1) \quad , \quad \mathbf{V}^* := \mathbf{V}_2 \cdot \mathbf{G}^{-1}(\mathbf{V}_1).$$

  The matrix $\mathbf{U}^*$ has noise level $\beta^* \leq |x_2|\beta_1 + m\beta_2$. Note that the noise growth is asymmetric and the order of x_1, x_2 matters. To keep the noise level low, we require that $|x_2|$ is small.

- Let $g(x) = x + a$ be *addition-with-constant* gate, for the constant $a \in \mathbb{Z}_q$. The algorithms $\mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out}$ respectively compute:

  $$\mathbf{U}^* := \mathbf{U}_1 \quad , \quad \mathbf{V}^* := \mathbf{V}_1 + a \cdot \mathbf{G}.$$

  It's easy to see that the noise-level $\beta^* = \beta_1$ stays the same.

- Let $g(x) = a \cdot x$ be a *multiplication-by-constant* gate for the constant $a \in \mathbb{Z}_q$. We give two alternative methods that homomorphically compute g with different noise growth. In the first method, the algorithms $\mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out}$ respectively compute:

  $$\mathbf{U}^* := a \cdot \mathbf{U}_1 \quad , \quad \mathbf{V}^* := a \cdot \mathbf{V}_1.$$

  The noise level is $\beta^* = |a|\beta_1$, and therefore this method requires that a is small. In the second method, the algorithms $\mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out}$ respectively compute:

  $$\mathbf{U}^* := \mathbf{U} \cdot \mathbf{G}^{-1}(a \cdot \mathbf{G}) \quad , \quad \mathbf{V}^* := \mathbf{V} \cdot \mathbf{G}^{-1}(a \cdot \mathbf{G}).$$

  The noise level is $\beta^* \leq m \cdot \beta_1$, and is therefore independent of the size of a.

It is a simple exercise to check that, whenever the inputs $\mathbf{U}_i, \mathbf{V}_i$ satisfy $\mathbf{V}_i = f_{pk,x_i}(\mathbf{U}_i)$ then the above homomorphic evaluation procedures ensure that $f_{pk,g(x_1,\ldots,x_\ell)}(\mathbf{U}^*) = \mathbf{V}^*$. The above gate operations can be composed to compute any function g expressed as an arithmetic circuit. Therefore, the only limitation is the growth of the noise-level. In particular, if the noise-level of $\mathbf{U}^*$ is $\beta^* \geq \beta_{max}$ then $\mathbf{U}^* \notin \mathcal{U}$ is not a valid input.

Noise Growth and Bounded-Depth Circuits. The noise growth of the above homomorphic operations is fairly complex to describe in its full generality since it depends on the (size of) the inputs x_i, the order in which operations are performed etc. However, we can give bounds on the noise growth for the case of boolean circuits composed of NAND gates, and certain restricted arithmetic circuits.

Let g be a *boolean circuit of depth d composed of NAND gates* over inputs $x_i \in \{0, 1\}$. For $x_1, x_2 \in \{0, 1\}$ we can define an arithmetic-gate $\mathsf{NAND}(x_1, x_2) \overset{\text{def}}{=} 1 - x_1 \cdot x_2$. If U_1, U_2 have noise-levels $\leq \beta$, then $\mathbf{U}^* := \mathsf{HTDF.Eval}^{in}(\mathsf{NAND}, (x_1, \mathbf{U}_1), (x_2, \mathbf{U}_2))$ will have a noise-level $\beta^* \leq (m + 1)\beta$. Therefore if we compute $\mathbf{U}^* := \mathsf{HTDF.Eval}^{in}(g, (x_1, \mathbf{U}_1), \ldots, (x_\ell, \mathbf{U}_\ell))$ and the inputs $\mathbf{U}_i$ have noise-levels β_{init}, then the noise-level of $\mathbf{U}^*$ will be $\beta^* \leq \beta_{init} \cdot (m+1)^d \leq 2^{O(\log \lambda) \cdot d} \leq \beta_{max}$. This show that, with the parameters we chose, any depth-d boolean circuit g is admissible over any choice of boolean indices $x_i \in \{0, 1\}$.

More generally, let g be an *arithmetic circuit of depth d* consisting of fan-in-t addition gates, fan-in-2 multiplication gates, addition-with-constant, and multiplication-by-constant gates. Moreover, assume that for each fan-in-2 multiplication gate we are guaranteed that at least one input x_b is of size $|x_b| \leq p$, where $p = \text{poly}(\lambda), t = \text{poly}(\lambda)$ are some fixed polynomials in the security parameter. Evaluating each such gate increases the noise level by a multiplicative factor of at most $\max\{t, (p + m)\} = \text{poly}(\lambda)$. Therefore, if inputs $\mathbf{U}_i$ to g have noise-levels β_{init}, then the noise-level of $\mathbf{U}^* := \mathsf{HTDF.Eval}^{in}(g, (x_1, \mathbf{U}_1), \ldots, (x_\ell, \mathbf{U}_\ell))$ is bounded by $\beta_{init} \cdot \max\{t, (p + m)\}^d \leq 2^{O(\log \lambda) \cdot d} \leq \beta_{max}$. This shows that any such computation is admissible.

We mention that both of the above analyses are overly restrictive/pessimistic and we may be able to compute some function with lower noise growth than suggested above.

6.3 Fully Homomorphic Signatures (Single Dataset)

Roadmap. We now show how to construct fully homomorphic signatures from HTDFs as a black box. We do so in several stages.

We begin by defining and constructing homomorphic signatures that can only be used to sign a single dataset. We also initially only consider selective security, where the data to be signed is chosen by an attacker prior to seeing the public parameters of the scheme. In Section 6.3.2 we show how to construct such schemes in the standard model, albeit with large public parameters whose size exceeds the maximal size of the dataset to be signed. The public parameters are just uniformly random and therefore, in the random oracle model, we can easily compress them to get a scheme with short public parameters. We also show that the latter scheme can be proven secure in the standard model under a simple-to-state and falsifiable (but non-standard) assumptions on hash functions.

In Section 6.3.4 we then show a generic transformation that combines a homomorphic signature scheme with selective security and an HTDF to get a homomorphic signature scheme with full security. Finally, in 6.4 we define multi-data signatures where the signer can sign many different datasets under different labels. We give a generic transformation from single-data homomorphic signatures to multi-data ones. Both of these transformations work in the standard model and preserve the efficiency of the underlying scheme. (In Section 6.4.3, we also give an alternate transformation which yields a simpler construction of a multi-data scheme with full security in the RO model.) Lastly, in 6.5 we show how to make the signature schemes context hiding.

6.3.1 Definition

A *single-data homomorphic signature* scheme consists of poly-time algorithms ($\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}$, Verify, $\mathsf{Process}, \mathsf{Eval}$) with the following syntax.

- $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$: Gets the security parameter sec and a data-size bound N. Generates public parameters prms. The security parameter also defines the message space $\mathcal{X}$.
- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$: Gets the security parameter sec. Generates a verification/secret keys pk, sk.
- $(\sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}(x_1, \ldots, x_N)$: Signs some data $(x_1, \ldots, x_N) \in \mathcal{X}^N$.
- $\sigma^* \leftarrow \mathsf{Eval}_{\mathsf{prms}}(g, ((x_1, \sigma_1), \ldots, (x_\ell, \sigma_\ell)))$: Homomorphically computes a signature σ^*.
- $\alpha_g \leftarrow \mathsf{Process}_{\mathsf{prms}}(g)$: Homomorphically computes a "public-key" α_g for the function g from the public parameters.
- $\mathsf{Verify}_{pk}(\alpha_g, y, \sigma)$: Verifies that y is indeed the output of g by checking the signature σ against α_g. We use these algorithms to implicitly define the "combined verification procedure":
  $\mathsf{Verify}^*_{pk}(g, y, \sigma) : \{$ Compute $\alpha_g \leftarrow \mathsf{Process}_{\mathsf{prms}}(g)$ and output $\mathsf{Verify}_{pk}(\alpha_g, y, \sigma)\}$.

We can think of $\mathsf{Process}, \mathsf{Verify}$ as a component of the combined verification procedure Verify^*, but it will be useful to define them separately. In particular, we will think of the $\mathsf{Process}$ algorithm as "pre-processing" a function g. The computational complexity of this step can depend on the circuit size of g but it can be performed *offline* prior to seeing the signature σ or even the verification key pk. The public-key α_g for the function g can be small and the "online verification" procedure $\mathsf{Verify}_{pk}(\alpha_g, y, \sigma)$ can be fast, with size/time independent of g.

Signing Correctness. Let $\mathsf{id}_i : \mathcal{X}^N \to \mathcal{X}$ be a canonical description of the function $\mathsf{id}_i(x_1, \ldots, x_N) \stackrel{\text{def}}{=} x_i$ (i.e., a circuit consisting of a single wire taking the i'th input to the output.) We require that any $\mathsf{prms} \in \mathsf{PrmsGen}(1^\lambda, 1^N)$, any $(pk, sk) \in \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$, any $(x_1, \ldots, x_N) \in \mathcal{X}^N$ and any $(\sigma_1, \ldots, \sigma_N) \in \mathsf{Sign}_{sk}(x_1, \ldots, x_N)$ must satisfy $\mathsf{Verify}^*_{pk}(\mathsf{id}_i, x_i, \sigma_i) = \text{accept}$. In other words, σ_i certifies x_i as the i'th data item.

Evaluation Correctness. We require that for any $\mathsf{prms} \in \mathsf{PrmsGen}(1^\lambda, 1^N)$, any $(pk, sk) \in \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$, any $(x_1, \dots, x_N) \in \mathcal{X}^N$ and any $(\sigma_1, \dots, \sigma_N) \in \mathsf{Sign}_{sk}(x_1, \dots, x_N)$ and any $g : \mathcal{X}^N \to \mathcal{X}$, we have:

$$\mathsf{Verify}^*_{pk}(g, g(x_1, \dots, x_N), \sigma^*) = \text{accept}. \tag{6.4}$$

where $\sigma^* \leftarrow \mathsf{Eval}_{\mathsf{prms}}(g, ((x_1, \sigma_1), \dots, (x_N, \sigma_N))$. Moreover, we require correctness for *composed evaluation* of several different functions. For any $h_1, \dots, h_\ell$ with $h_i : \mathcal{X}^N \to \mathcal{X}$ and any $g : \mathcal{X}^\ell \to \mathcal{X}$ define the composition $(g \circ \bar{h}) : \mathcal{X}^N \to \mathcal{X}$ by $(g \circ \bar{h})(\bar{x}) = g(h_1(\bar{x}), \dots, h_\ell(\bar{x}))$. We require that for any $(x_1, \dots, x_\ell) \in \mathcal{X}^\ell$ and any $(\sigma_1, \dots, \sigma_\ell)$:

$$\begin{array}{c} \{\ \mathsf{Verify}^*_{pk}(h_i, x_i, \sigma_i) = \text{accept}\ \}_{i \in [\ell]} \\ \sigma^* := \mathsf{Eval}_{\mathsf{prms}}(g, (x_1, \sigma_1), \dots, (x_\ell, \sigma_\ell)) \end{array} \quad \Rightarrow \quad \mathsf{Verify}^*_{pk}((g \circ \bar{h}), g(x_1, \dots, x_\ell), \sigma^*) = \text{accept}. \tag{6.5}$$

In other words, if the signatures σ_i certify x_i as the output of h_i, then σ^* certifies $g(x_1, \dots, x_\ell)$ as the output of $g \circ \bar{h}$. Notice that 6.4 follows from 6.5 and the correctness of signing by setting $h_i \overset{\text{def}}{=} \mathsf{id}_i$.

Relaxing Correctness for Leveled Schemes. In a *leveled* fully homomorphic scheme, each signature σ_i will have some associated "noise-level" β_i. The initial signatures produced by $(\sigma_1, \dots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}(x_1, \dots, x_N)$ will have a "small" noise-level β_{init}. The noise-level β^* of the homomorphically computed signature $\sigma^* := \mathsf{Eval}_{\mathsf{prms}}(g, ((x_1, \sigma_1), \dots, (x_\ell, \sigma_\ell)))$ depends on the noise-levels β_i of the signatures σ_i, the function g and the messages x_i. If the noise level β^* of σ^* exceeds some threshold $\beta^* > \beta_{max}$, then the above correctness requirements need not hold. This will limit the type of functions that can be evaluated. A function g is *admissible* on the values $x_1, \dots, x_\ell$ if, whenever the signatures σ_i have noise-levels $\beta_i \le \beta_{init}$, then σ^* will have noise-level $\beta^* \le \beta_{max}$.

Security Game. We define the security of homomorphic signatures via the following game between an attacker $\mathcal{A}$ and a challenger:

- The challenger samples $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$ and $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$ and gives prms, pk to the adversary.
- The attacker $\mathcal{A}(1^\lambda)$ chooses data $(x_1, \dots, x_N) \in \mathcal{X}^*$ and sends it to the challenger.
- The challenger computes $(\sigma_1, \dots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}(x_1, \dots, x_N)$ and gives the signatures $(\sigma_1, \dots, \sigma_N)$ to $\mathcal{A}$.
- The attacker $\mathcal{A}$ chooses a function $g : \mathcal{X}^N \to \mathcal{X}$ and values y', σ'. Let $y := g(x_1, \dots, x_N)$. The attacker wins if all of the following hold: (1) g is admissible on the values $x_1, \dots, x_N$, (2) $y' \neq y$, and (3) $\mathsf{Verify}^*_{pk}(g, y', \sigma') = \text{accept}$.

We say a homomorphic signature scheme is *fully secure* if for all PPT $\mathcal{A}$, we have $\Pr[\mathcal{A} \text{ wins }] \le \text{negl}(\lambda)$ in the above game.

Remarks. We point out some extensions and relaxations of the definition that we also consider in this work.

- **Selective Security.** We will also consider a *selective security* game for single-data homomorphic signatures, where the attacker chooses the data $x_1, \ldots, x_N$ to be signed before seeing prms and pk. This is a natural security notion for the typical use-case where the user samples prms, pk, sk and signs the data in one step and therefore the data will not depend on prms, pk. We first show our basic construction satisfying *selective security.* We then show a generic transformation from a *selectively secure* scheme to a scheme satisfying *full security.*

- **Adaptive Individual Data Item Queries.** It is possible to extend the syntax of homomorphic signature schemes to also allow the user to sign different data items x_i individually with respect to their position i, rather than having to specify the entire dataset vector $(x_1, \ldots, x_N)$ all at at once. It is easy to see that our construction in Section 6.3.2 allows this. In this case, we can also extend our definition of full adaptive security to allow an adversary to query for signatures of individual data items x_i adaptively, after seeing the signatures of other items. It is easy to see that our transformation from selective to fully adaptive security in Section 6.3.4 achieves this notion of security (with minimal syntactic changes to the proof). A similar extension can also be added to the setting of multiple-data sets as defined in 6.4. In this case, the user would be able to sign an individual data-item x_i with respect to position i of a dataset with label τ. Again it is easy to see that our construction in 6.4 allows for this and achieves full adaptive security in this setting.

- **Verification and Admissible Functions.** We note that, under the above definition, security only holds when verifying a function g which is admissible on the signed values $x_1, \ldots, x_N$, but the verifier does not know these values. Therefore, we require some convention on the types of values x_i that the signer will sign and the type of functions g that the verifier is willing to verify to ensure that the function is admissible on the signed values. For example, our eventual construction ensures that if g is a boolean circuit of depth $\leq d$ then it is admissible on all boolean inputs with $x_i \in \{0, 1\} \subseteq \mathcal{X}$. Therefore, by convention, we can restrict the signer to only sign values $x_i \in \{0, 1\}$ and the verifier to only verify functions g that are boolean circuits of depth $\leq d$. Other combinations (e.g., $x_i \in \mathbb{Z}_q$ and g is an affine function) are also possible and therefore we leave this decision to the users of the scheme rather than its specification.

6.3.2 Basic Construction

Let $\mathcal{F} = (\mathsf{HTDF.KeyGen}, f, \mathsf{Inv}, \mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out})$ be an HTDF with index-space $\mathcal{X}$, input space $\mathcal{U}$, output space $\mathcal{V}$ and an input distribution $D_{\mathcal{U}}$. We construct a signature scheme $\mathcal{S} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Process}, \mathsf{Eval})$ with message space $\mathcal{X}$ as follows.

- $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$: Choose $v_1, \ldots, v_N$ by sampling $v_i \leftarrow \mathcal{V}$. Output $\mathsf{prms} = (v_1, \ldots, v_N)$.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$: Choose $(pk', sk') \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda)$ and set $pk = pk'$, $sk = (\mathsf{prms}, sk')$.
- $(\sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}(x_1, \ldots, x_N)$: Sample $u_i \leftarrow \mathsf{Inv}_{sk', x_i}(v_i)$ and set $\sigma_i := u_i$ for $i \in [N]$.
- $\sigma^* = \mathsf{Eval}_{pk}(g, (x_1, \sigma_1), \ldots, (x_\ell, \sigma_\ell))$: Run $\mathsf{HTDF.Eval}^{in}_{pk'}$ procedure of the HTDF.
- $\alpha_g \leftarrow \mathsf{Process}_{\mathsf{prms}}(g)$: Compute $\alpha_g := \mathsf{HTDF.Eval}^{out}_{pk'}(g, v_1, \ldots, v_N)$.
- $\mathsf{Verify}_{pk}(\alpha_g, y, \sigma)$: If $f_{pk', y}(\sigma) = \alpha_g$ accept, else reject.

Remarks. (I) We can think of $\mathsf{prms} = (v_1, \ldots, v_N)$ as public parameters that can be fixed for all users of the scheme. Each user's individual public/secret key then only consists of the small values pk', sk'. (II) Although we describe the signing procedure as signing the values $x_1, \ldots, x_N$ in one shot, it's easy to see that we can also sign the values x_i completely independently (e.g., at different times) without needing to keep any state beyond knowing the index i by setting $\sigma_i \leftarrow \mathsf{Inv}_{sk', x_i}(v_i)$. (III) We note that if the function g only "touches" a small subset of the inputs $i \in [N]$ then the pre-processing step $\mathsf{Process}_{\mathsf{prms}}(g)$ only needs to read the corresponding values v_i from the public parameters. The run-time of this step can therefore be sub-linear in N and only depends on the size of the circuit g (ignoring unused input wires). (IV) The efficiency of the scheme is inherited from that of the HTDF. Note that, although the pre-processing step $\mathsf{Process}_{\mathsf{prms}}(g)$ requires running $\mathsf{HTDF.Eval}$, the online verification step can be much more efficient. For our HTDF construction, it will only depend on the size of σ, α_g which only scale with the depth but not the size of the circuit g.

Correctness and Security. It's easy to see that correctness of signing and correctness of (leveled) homomorphic evaluation for the signature scheme $\mathcal{S}$ follow from the correctness properties of the underlying (leveled) HTDF $\mathcal{F}$. In a leveled scheme, the noise-level of signatures $\sigma_i = u_i$ is just defined as its noise-level of the HTDF input u_i. The initial noise-level β_{init}, the maximal noise level β_{max}, and the set of admissible functions is the same in $\mathcal{S}$ and in $\mathcal{F}$. We are left to prove security.

Theorem 6.3.1. *Assuming $\mathcal{F}$ satisfies HTDF security, the signature scheme $\mathcal{S}$ satisfies single-data security of homomorphic signatures.*

Proof. Assume that an adversary $\mathcal{A}$ has a non-negligible advantage in the single-data security game with the scheme $\mathcal{S}$. In the game, the attacker $\mathcal{A}$ selects some data $x_1, \ldots, x_N \in \mathcal{X}$ and gets back $\mathsf{prms} = (v_1, \ldots, v_N)$, pk' and $\sigma_1, \ldots, \sigma_N$, where $(pk', sk') \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda)$, $v_i \leftarrow \mathcal{V}$ and $\sigma_i = u_i \leftarrow \mathsf{Inv}_{sk', x_i}(v_i)$. Let us modify the game by choosing $u_i \leftarrow D_{\mathcal{U}}$ and setting $v_i := f_{pk', x_i}(u_i)$. This change is statistically indistinguishable by the "Distributional Equivalence of Inversion" property of the HTDF.[6] Therefore $\mathcal{A}$ wins the modified games with

[6]Technically, this requires N hybrid arguments where we switch how each u_i, v_i is sampled one-by-one. In each hybrid, we rely on the fact that indistinguishability holds even given sk' to sample the rest of the values u_j, v_j.

non-negligible probability.

We now give a polynomial-time reduction that takes any attacker $\mathcal{A}$ having a non-negligible advantage in the above modified game, and use it to break HTDF security of $\mathcal{F}$ with the same advantage. The reduction gets a challenge public key pk' and chooses the values u_i, v_i as in the modified game (without knowing sk') and gives these values to $\mathcal{A}$. Assume the attacker $\mathcal{A}$ wins the modified game by choosing some admissible function $g : \mathcal{X}^N \to \mathcal{X}$ on $x_1, \ldots, x_N$ and some values $y', \sigma' = u'$. Let $y := g(x_1, \ldots, x_N)$, $\alpha_g := \mathsf{HTDF.Eval}^{out}_{pk'}(g, v_1, \ldots, v_N)$, $u := \mathsf{HTDF.Eval}^{in}_{pk'}(g, (x_1, \sigma_1), \ldots, (x_N, \sigma_N))$. Then, since the signature σ' verifies, we have $f_{pk',y'}(u') = \alpha_g$. On the other hand, since g is an admissible function, the correctness of homomorphic evaluation ensures that $f_{pk',y}(u) = \alpha_g$. Therefore, the values $u, u' \in \mathcal{U}$ and $y \neq y' \in \mathcal{X}$ satisfy $f_{pk',y}(u) = f_{pk',y'}(u')$, allowing the reduction to break HTDF security whenever $\mathcal{A}$ wins the modified game. $\square$

6.3.3 A Scheme with Short Public Parameters

We can adapt the above construction to get a scheme with short public parameters in the random oracle model. Instead of choosing $\mathsf{prms} = (v_1, \ldots, v_N)$, with $v_i \leftarrow \mathcal{V}$ taken uniformly at random from the output space of the HTDF, we can set $\mathsf{prms} = r$ for some small $r \leftarrow \{0,1\}^\lambda$ and implicitly define $v_i = H(r, i)$ where $H : \{0,1\}^* \to \mathcal{V}$ is a hash function modeled as a random oracle. The rest of the algorithms remain unchanged. It is easy to see that the same security proof as above goes through for this scheme in the random-oracle mode.

Moreover, we can define a standard-model assumption on the hash function H under which we can prove the above scheme secure. The assumption is falsifiable and simple-to-state, but it is not a standard assumption. It ties together the security of the hash function H with that of the underlying HTDF $\mathcal{F}$.

Definition 6.3.1. *Let $\mathcal{F} = (\mathsf{HTDF.KeyGen}, f, \mathsf{Inv}, \mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out})$ be an HTDF with index-space $\mathcal{X}$, input space $\mathcal{U}$, output space $\mathcal{V}$ and an input distribution $D_{\mathcal{U}}$. Let $H : \{0,1\}^* \to \mathcal{V}$ be a hash function. We say that H is* inversion unhelpful *for $\mathcal{F}$ if for any PPT adversary $\mathcal{A}$ the probability of $\mathcal{A}(1^\lambda)$ winning the following game is negligible in λ:*

- *Adversary $\mathcal{A}$ chooses values $x_1, \ldots, x_N$ with $x_i \in \mathcal{X}$.*
- *Challenger chooses $(pk, sk) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda)$, $r \leftarrow \{0,1\}^\lambda$. For $i = 1, \ldots, N$, it computes $v_i = H(r, i)$, $u_i \leftarrow \mathsf{Inv}_{sk,x_i}(v_i)$. It gives $(pk, r, u_1, \ldots, u_N)$ to $\mathcal{A}$.*
- *Adversary $\mathcal{A}$ outputs $u, u' \in \mathcal{U}$ and $x \neq x' \in \mathcal{X}$ and wins if $f_{pk,x}(u) = f_{pk,x'}(u')$.*

Essentially, the above says that H is *inversion unhelpful* for $\mathcal{F}$ is seeing the inverses $u_i \leftarrow \mathsf{Inv}_{sk,x_i}(H(r,i))$ for x_i chosen adversarially but non-adaptively and r random, does not help the attacker in coming up with a claw for $\mathcal{F}$.

It is easy to see that the random-oracle construction outlined above is secure as long as the function H is *inversion unhelpful* for $\mathcal{F}$. The proof follows the proof of security of our basic construction in Section 6.3.2, with the only difference that the reduction sets the values $v_i = H(r, i)$ and the signatures $\sigma_i = u_i$ by getting r and $\{u_i\}$ them from the challenger in the above inversion-unhelpful security game.

It is also easy to see that if H is modeled as random oracle, then it is inversion-unhelpful for any HTDF $\mathcal{F}$. This is because we can "program" the outputs $H(r, i)$ to values $v_i = f_{pk,x_i}(u_i)$ for $u_i \leftarrow D_{\mathcal{U}}$ and then invert v_i to u_i without knowing the secret key sk of the HTDF. Therefore, seeing such inverses u_i cannot help the adversary in finding a claw.

We note that the above inversion-unhelpful assumption is simpler to state than simply assuming the resulting signature scheme is secure. In particular, the assumption does not depend on any homomorphic properties, the adversary does not get to choose a function of his choice, the challenger does not have to perform any homomorphic evaluations etc. Also, having such an assumption sets some contrast between the above homomorphic signature scheme with short parameters in the Random-Oracle model and a generic construction of homomorphic signatures based on SNARKs in the Random-Oracle model (see introduction). For the latter, we either need to rely on SNARK security, which is not a falsifiable assumption, or we could instead simply assume that the full signature scheme construction in the random-oracle model is secure but, since the construction of SNARKs is complex and involves heavy PCP machinery, this assumptions would be far from simple-to-state.

6.3.4 From Selective Security to Full Security

We now show how to construct a fully secure homomorphic signature scheme from any selectively secure scheme and an HTDF. On a high level, the transformation is similar to the use of *chameleon hashing* to go from security against selective chosen-message queries (e.g., the signing queries are all made ahead of time) to adaptive chosen-message security [KR00]. However, to make this work with homomorphic signatures, we would need the chameleon hash to also be homomorphic. Fortunately, HTDFs can be thought of as providing exactly such primitive.

In more detail, to sign some data $x_1, \ldots, x_N$ under the new signature scheme, we first choose random values $v_1, \ldots, v_N$ from the output of the HTDF, then we sign the values v_i under the selectively secure scheme to get signatures $\overline{\sigma}_i$ and finally we compute $u_i \leftarrow \mathsf{Inv}_{sk,x_i}(v_i)$ and give out the signature $\sigma_i = (v_i, u_i, \overline{\sigma}_i)$. To homomorphically evaluate some function g over such signatures we (1) run the input/output homomorphic computation the HTDF to compute values u^*, v^* so that u^* is an "opening" of v^* to the message $g(x_1, \ldots, x_N)$ and (2) we run the homomorphic evaluation of the signature scheme to compute a signature $\overline{\sigma}^*$ which certifies that v^* was computed correctly.

Security comes from the fact that values v_i signed under the selectively secure signature scheme are chosen uniformly at random and therefore non-adaptively. By the selective security of the underlying signature, this shows that an attacker cannot give the "wrong" v^* in his forgery. On the other hand, by the claw-free security of the HTDF, the attacker also cannot open the "right" v^* to the "wrong" message as this would produce a claw on the HTDF.

Construction. Let $\mathcal{F} = (\mathsf{HTDF.KeyGen}, f, \mathsf{Inv}, \mathsf{HTDF.Eval}^{in}, \mathsf{HTDF.Eval}^{out})$ be an HTDF with index-space $\mathcal{X}$, input space $\mathcal{U}$, output space $\mathcal{V}$ and an input distribution $D_{\mathcal{U}}$. Let $\mathcal{S}' = (\mathsf{PrmsGen}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}', \mathsf{Process}', \mathsf{Eval}')$ be a selectively secure homomorphic signature scheme. Without loss of generality and for simplicity of exposition, we will assume that the message space of $\mathcal{S}'$ is the same as the output space $\mathcal{V}$ of the HTDF (we can always represent the values $v \in \mathcal{V}$ as bits and sign them bit-by-bit if this is not the case). For a function $g : \mathcal{X}^\ell \to \mathcal{X}$ we define a corresponding function $g' : \mathcal{V}^\ell \to \mathcal{V}$ as $g'(v_1, \ldots, v_\ell) = \mathsf{HTDF.Eval}^{out}(g, v_1, \ldots, v_\ell)$.

- $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$: Use the selectively secure signature scheme $\mathsf{prms} \leftarrow \mathsf{PrmsGen}'(1^\lambda, 1^N)$.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$: Choose
$$(pk_h, sk_h) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda) \quad , \quad (pk', sk') \leftarrow \mathsf{KeyGen}'(1^\lambda, \mathsf{prms}).$$
Set $pk = (pk_h, pk'), sk = (sk_h, sk')$.

- $(\sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}_{sk'}(x_1, \ldots, x_N)$:
  - For each $i \in [N]$: sample $v_i \leftarrow \mathcal{V}$, $u_i \leftarrow \mathsf{Inv}_{sk_h, x_i}(v_i)$.
  - Compute $(\bar{\sigma}_1, \ldots, \bar{\sigma}_N) \leftarrow \mathsf{Sign}'_{sk'}(v_1, \ldots, v_N)$.
  - Set $\sigma_i = (v_i, u_i, \overline{\sigma}_i)$.

- $\sigma^* = \mathsf{Eval}_{pk}(g, (x_1, \sigma_1), \ldots, (x_\ell, \sigma_\ell))$: Parse $\sigma_i = (v_i, u_i, \overline{\sigma}_i)$.
  - Compute $v^* := \mathsf{HTDF.Eval}^{out}_{pk_h}(g, v_1, \ldots, v_\ell)$, $u^* := \mathsf{HTDF.Eval}^{in}_{pk_h}(g, (x_1, u_1), \ldots, (x_\ell, u_\ell))$.
  - Compute $\overline{\sigma}^* = \mathsf{Eval}'_{pk'}(g', (v_1, \overline{\sigma}_1), \ldots, (v_\ell, \overline{\sigma}_\ell))$.
  - Output $\sigma^* = (v^*, u^*, \overline{\sigma}^*)$,

- $\alpha_g \leftarrow \mathsf{Process}_{\mathsf{prms}}(g)$: Compute $\alpha_g := \mathsf{Process}'_{\mathsf{prms}}(g')$.

- $\mathsf{Verify}_{pk}(\alpha_g, y, \sigma^*)$: Parse $\sigma^* = (v^*, u^*, \overline{\sigma}^*)$. Verify that $f_{pk_h, y}(u^*) = v^*$ and $\mathsf{Verify}'_{pk'}(\alpha_g, v^*, \overline{\sigma}^*) = $ accept: if both conditions hold then accept, else reject.

Correctness. The correctness of the signature scheme follows readily from the correctness of $\mathcal{S}'$ and $\mathcal{F}$. A function g is admissible on values $x_1, \ldots, x_N$ under the scheme $\mathcal{S}$ if it is admissible under the HTDF $\mathcal{F}$ and if the corresponding function g' is also admissible over all values $(v_1, \ldots, v_n) \in \mathcal{V}^N$ under the selectively-secure signature scheme $\mathcal{S}'$.

Theorem 6.3.2. *If $\mathcal{F}$ is a secure (leveled) HTDF and $\mathcal{S}'$ is a selectively secure (leveled) homomorphic signature scheme, then the above construction of $\mathcal{S}$ is a fully secure (leveled) homomorphic signature scheme.*

Proof. Let $\mathcal{A}$ be some adversary in the adaptive signature security game against the scheme $\mathcal{S}$. Let (g, y', σ') denote the adversary's forgery at the end of the game, and parse $\sigma' = (v', u', \overline{\sigma}')$. Let $x_1, \ldots, x_N$ denote the messages chosen by the adversary in the game and $v_1, \ldots, v_N$ be the values contained in the signatures that it gets back. Let E_1 be the event that $\mathcal{A}$ wins the game *and* $v' = \mathsf{HTDF.Eval}^{out}_{pk_h}(g, v_1, \ldots, v_N)$. Let E_2 be the even that $\mathcal{A}$ wins the game $v' \neq \mathsf{HTDF.Eval}^{out}_{pk_h}(g, v_1, \ldots, v_N)$. The probability that $\mathcal{A}$ wins the game is $\Pr[E_1 \vee E_2] \leq \Pr[E_1] + \Pr[E_1]$.

Firstly, we show that $\Pr[E_1]$ is negligible. We do this via a reduction breaking HTDF security of $\mathcal{F}$ with probability $\Pr[E_1]$. The reduction gets an HTDF public key pk_h. It chooses $(\mathsf{prms}, pk', sk')$ for the selectively-secure signature scheme on its own and simulates the signature game for $\mathcal{A}$ with one modification: to answer the signing query, instead of choose $v_i \leftarrow \mathcal{V}$, $u_i \leftarrow \mathsf{Inv}_{sk_h, x_i}(v_i)$ it chooses $u_i \leftarrow D_{\mathcal{U}}$ and $v_i = f_{pk_h, x_i}(u_i)$. This change is statistically indistinguishable by the "Distributional Equivalence of Inversion" property of the HTDF. Finally, assume that $\mathcal{A}$ outputs a forgery causing E_1 to occur. Let $y = g(x_1, \ldots, x_N)$ and let $u = \mathsf{HTDF.Eval}^{in}_{pk_h}((x_1, u_1), \ldots, (x_N, u_N))$. Then $y \neq y'$ and $f_{pk,y}(u) = f_{pk,y'}(u')$. Therefore the tuple (u, u', y, y') allows the reduction to break HTDF security.

Secondly, we show that $\Pr[E_2]$ is negligible. We do this via reduction breaking the selective security of $\mathcal{S}'$ with $\Pr[E_2]$. The reduction starts by (non-adaptively) choosing random messages $v_1, \ldots, v_N$ with $v_i \leftarrow \mathcal{V}$ and giving them to its challenger. It gets back prms, pk' and $\overline{\sigma}_1, \ldots, \overline{\sigma}_N$. The reduction chooses its own keys (pk_h, sk_h) for the HTDF and gives $(\mathsf{prms}, (pk_h, pk'))$ to $\mathcal{A}$. The adversary $\mathcal{A}$ replies with messages $(x_1, \ldots, x_N)$ and the reduction computes $u_i \leftarrow \mathsf{Inv}_{sk_h, x_i}(v_i)$ and gives back the values $\sigma_i = (v_i, u_i, \overline{\sigma}_i)$ to $\mathcal{A}$. Finally, assume that $\mathcal{A}$ outputs a forgery causing E_1 to occur. Then $(g', v', \overline{\sigma}')$ is a forgery against $\mathcal{S}'$ since $v' \neq g'(v_1, \ldots, v_N)$.

Combining the above, this shows that the probability that $\mathcal{A}$ wins the adaptive signature security game against $\mathcal{S}$ is negligible, and the theorem follows.

□

6.4 Multi-Data Homomorphic Signatures

We now define and construct *multi-data* homomorphic signatures. In such a scheme, the signer can sign many different datasets of arbitrary size. Each dataset is tied to some labels τ_i (e.g., the name of the dataset) and the verifier is assumed to know the label of the dataset over which he wishes to verify computation. In 6.4.2, we show how to construct multi-data homomorphic signatures starting from a single dataset scheme. In Section 6.4.3, we show another transformation which enjoys efficiency improvements and supports signing of unbounded datasets starting from single dataset scheme with short public parameters (such as our construction in the random oracle model 6.3.3).

6.4.1 Definition

A *multi-data homomorphic signature* consists of the algorithms ($\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Process}, \mathsf{Eval}$) with the following syntax.

- $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$: Gets the security parameter sec and a data-size bound N. Generates public parameters prms.
- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$: produces a public verification key pk and a secret signing key sk.
- $(\sigma_\tau, \sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}((x_1, \ldots, x_N), \tau)$: Signs some data $\bar{x} \in \mathcal{X}^*$ under a label $\tau \in \{0,1\}^*$.
- $\sigma^* = \mathsf{Eval}_{\mathsf{prms}}(g, \sigma_\tau, (x_1, \sigma_1), \ldots, (x_\ell, \sigma_\ell))$: Homomorphically computes the signature σ^*.
- $\alpha_g \leftarrow \mathsf{Process}_{\mathsf{prms}}(g)$: Produces a "public-key" α_g for the function g.
- $\mathsf{Verify}_{pk}(\alpha_g, y, \tau, (\sigma_\tau, \sigma^*))$: Verifies that $y \in \mathcal{X}$ is indeed the output of the function g over the data signed with label τ. We define the "combined verification procedure": $\mathsf{Verify}^*_{pk}(g, y, \tau, \sigma_\tau, \sigma^*)$: { Compute $\alpha_g \leftarrow \mathsf{Process}_{\mathsf{prms}}(g)$ and output $\mathsf{Verify}_{pk}(\alpha_g, y, \tau, (\sigma_\tau, \sigma^*))$}.

Correctness. The correctness requirements are analogous to those of the single-data definition. We right away define correctness of evaluation to allow for *composed evaluation.*

Correctness of Signing. We require that any $\mathsf{prms} \in \mathsf{PrmsGen}(1^\lambda, 1^N)$, $(pk, sk) \in \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$, any $(x_1, \ldots, x_N) \in \mathcal{X}^N$, any $\tau \in \{0,1\}^*$ and any $(\sigma_\tau, \sigma_1, \ldots, \sigma_N) \in \mathsf{Sign}_{sk}(x_1, \ldots, x_N, \tau)$ must satisfy $\mathsf{Verify}^*_{pk}(\mathsf{id}_i, x_i, \tau, (\sigma_\tau, \sigma_i)) = \text{accept}$. In other words, (σ_τ, σ_i) certifies x_i as the i'th data item of the data with label τ.

Correctness of Evaluation. For any circuits $h_1, \ldots, h_\ell$ with $h_i : \mathcal{X}^N \to \mathcal{X}$ and any circuit $g : \mathcal{X}^\ell \to \mathcal{X}$, any $(x_1, \ldots, x_\ell) \in \mathcal{X}^\ell$, any $\tau \in \{0,1\}^*$ and any $\sigma_\tau, (\sigma_1, \ldots, \sigma_\ell)$:

$$\left\{ \begin{array}{l} \{\mathsf{Verify}_{pk}(h_i, x_i, \tau, (\sigma_\tau, \sigma_i)) = \text{accept}\}_{i \in [\ell]} \\ \sigma^* := \mathsf{Eval}_{pk}(g, \sigma_\tau, (x_1, \sigma_1), \ldots, (x_\ell, \sigma_\ell)) \end{array} \right\} \Rightarrow \mathsf{Verify}^*_{pk}((g \circ \bar{h}), g(x_1, \ldots, x_\ell), \tau, (\sigma_\tau, \sigma^*)) = \text{accept}.$$

In other words, if the signatures (σ_τ, σ_i) certify x_i as the outputs of functions h_i over the data labeled with τ, then (σ_τ, σ^*) certifies $g(x_1, \ldots, x_\ell)$ as the output of $g \circ \bar{h}$ over the data labeled with τ.

Multi-Data Security. We define the security via the following game between an attacker $\mathcal{A}$ and a challenger:

- The challenger samples $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$, $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$ and gives prms, pk to the attacker $\mathcal{A}$

- Signing Queries: The attacker $\mathcal{A}$ can ask an arbitrary number of signing queries. In each query j, the attacker chooses a fresh tag $\tau_j \in \{0,1\}^*$ which was never queried previously and a message $(x_{j,1}, \ldots, x_{j,N_j}) \in \mathcal{X}^*$. The challenger responds with

$$(\sigma_{\tau_j}, \sigma_{j,1}, \ldots, \sigma_{j,N_j}) \leftarrow \mathsf{Sign}_{sk}((x_{j,1}, \ldots, x_{j,N_j}), \tau_j).$$

- The attacker $\mathcal{A}$ chooses a circuit $g : \mathcal{X}^{N'} \to \mathcal{X}$ values $\tau, y', (\sigma'_\tau, \sigma')$. The attacker wins if $\mathsf{Verify}^*_{pk}(g, \tau, y', (\sigma'_\tau, \sigma')) =$ accept and either:
  - *Type I forgery:* $\tau \neq \tau_j$ for any j, or $\tau = \tau_j$ for some j but $N' \neq N_j$. (i.e., No signing query with label τ was ever made or there is a mismatch between the size of the data signed under label τ and the arity of the function g.)
  - *Type II forgery:* $\tau = \tau_j$ for some j with corresponding message $x_{j,1}, \ldots, x_{j,N'}$ such that (a) g is admissible on $x_{j,1}, \ldots, x_{j,N'}$, and (b) $y' \neq g(x_{j,1}, \ldots, x_{j,N'})$.

We require that for all PPT $\mathcal{A}$, we have $\Pr[\mathcal{A} \text{ wins }] \leq \text{negl}(\lambda)$ in the above game.

6.4.2 From Single-Data to Multi-Data, Construction 1

We now describe our transformation from a single-data homomorphic signature scheme to a multi-data scheme. We sample the public parameters of the single-data homomorphic signature scheme once and for all, then for each signing query we sample a pair of public/secret keys of the homomorphic scheme (using the same public parameters). The dataset information along with the public key are signed using the regular homomorphic signature scheme. The dataset itself is signed using the sampled secret key. To verify the authenticity, it is sufficient to verify the dataset information with the public key, and authenticate the output of the function with respect to this public key.

Let $\mathcal{S}' = (\mathsf{PrmsGen}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}', \mathsf{Process}', \mathsf{Eval}')$ be a fully secure one-dataset homomorphic signature scheme. Let $\mathcal{S}^{nh} = (\mathsf{NH.KeyGen}, \mathsf{NH.Sign}, \mathsf{NH.Verify})$ be any standard ($\underline{\text{n}}$ot $\underline{\text{h}}$omomorphic) signature scheme. We construct a multi-data homomorphic signature scheme $\mathcal{S} = (\mathsf{PrmsGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Process}, \mathsf{Eval})$ with message space $\mathcal{X}$ as follows.

- $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$: Sample and output parameters of the single-data homomorphic signature scheme: $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$.

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$: Choose $(pk_1, sk_1) \leftarrow \mathsf{NH.KeyGen}(1^\lambda, \mathsf{prms})$. Samp set $pk = pk_1$, $sk = (sk_1, \mathsf{prms})$.

- $(\sigma_\tau, \sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}((x_1, \ldots, x_N), \tau)$:
  - Sample secret and public keys of the single-data homomorphic signature scheme: $(pk_2, sk_2) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$.

  - Sign the dataset size, the tag and the public key of the single-data homomorphic signature scheme using non-homomorphic scheme: $\rho \leftarrow \mathsf{NH.Sign}_{sk_1}((pk_2, \tau, N))$. Set $\sigma_\tau = (pk_2, \tau, N, \rho)$.
  - Sign the dataset using the single-data homomorphic scheme: $(\sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}'_{sk_2}(x_1, \ldots, x_N)$.
  - Output $(\sigma_\tau, \sigma_1, \ldots, \sigma_N)$.

- $\sigma^* = \mathsf{Eval}_{pk}(g, \sigma_\tau, \sigma(x_1, \sigma_1), \ldots, (x_\ell, \sigma_\ell))$: Parse $\sigma_\tau = (pk_2, \tau, N, \rho)$. Apply the single-data evaluation algorithm. $\sigma^* = \mathsf{Eval}_{\mathsf{prms}}(g, (x_1, \sigma_1), \ldots, (x_N, \sigma_N))$.

- $\alpha_g \leftarrow \mathsf{Process}_{\mathsf{prms}}(g)$: Output $\alpha_g \leftarrow \mathsf{Process}'_{\mathsf{prms}}(g)$.

- $\mathsf{Verify}_{pk}(\alpha_g, y, \tau, (\sigma_\tau, \sigma^*))$: Parse $\sigma_\tau = (\mathsf{prms}, pk_2, \tau, N, \rho)$ and accept if and only if the following two conditions are satisfied:

  1. Verify the parameters of the single-data homomorphic scheme's public key and the dataset:
     $\mathsf{NH.Verify}_{pk_1}((pk_2, \tau, N), \rho) =$ accept, and
  2. Verify the homomorphically computed signature: $\mathsf{Verify}'_{pk_2}(\alpha_g, y, \sigma^*) =$ accept.

Correctness. Correctness of the scheme follows from the correctness of the regular signature scheme and single-data homomorphic scheme.

Security. The security follows from two main observations: first, no adversary is able to fake the parameters or the public key of the single-data homomorphic signature due to security of the standard signature scheme. Given that, no adversary is able to fake the result of the the computation due to the security of the single-data homomorphic signature scheme. In particular, we first switch to Game 1, where the adversary looses if it is able to make a forgery on some of the dataset information signed under the regular scheme: (pk_2, τ, N). Now, if there is an adversary able to win Game 1, then we can convert it to an adversary that breaks the security of the homomorphic scheme. The adversary takes (prms, pk_2) as a part of the challenge. Guesses an index j^* and sets $pk_{2,j^*} = pk_2$. It then asks the challenger to sign the data $(x_1, \ldots, x_N)$ at query j^* and signs all other datasets by itself by sampling a pair of public/secret keys. A forgery of type II can then be used to break the security of single-data homomorphic scheme.

Theorem 6.4.1. *Assume $\mathcal{S}'$ is a fully secure single-data homomorphic signature scheme and $\mathcal{S}^{nh}$ is a regular signature scheme. Then, $\mathcal{S}$ is a fully secure many-dataset homomorphic signature scheme.*

Proof. We prove the security via a series of indistinguishable games.

- Let **Game 0** be the multi-data security game.

- Let **Game 1** be the modified version of **Game 1**, except the attacker loses the game if it outputs a non-homomorphic forgery. That is, a forgery of the form $(g, y, \tau, (\sigma_\tau = (pk_2, \tau, N', \rho), \sigma^*)$, where:

    1. $\tau \neq \tau_j$ for any j, or
    2. $\tau = \tau_j$ for some j but $N' \neq N_j$ or
    3. $\tau = \tau_j$ for some j, $N' = N_j$, but $pk_2 \neq pk_{2,j}$, where $pk_{2,j}$ is the public key used for single-data signature scheme.

  That is, if the parameters of the dataset are invalid, the adversary losses the game. Note that this is the superset of type I forgeries. Clearly, if there exists a winning adversary in **Game 1**, then we can break the security of the regular signature scheme, since we obtain a valid signature ρ of (pk_2, τ, N) which was never signed before.

Now, assume there exists an adversary $\mathcal{A}$ that wins in **Game 1**. Then, it must be able to come up with a type II forgery. Hence, we can construct an adversary $\mathcal{A}'$ that breaks the security of the single-data homomorphic signature scheme. $\mathcal{A}'$ receives parameters prms, pk_2 as the challenge, it then generates parameters of the regular signature scheme $(pk_1, sk_1) \leftarrow \mathsf{NH.KeyGen}(1^\lambda)$ and forwards $\mathsf{prms}, pk = pk_1$ to $\mathcal{A}$. It also chooses an index j^* of the dataset on which it guesses the type II will be made and sets $pk_{2,j^*} = pk_2$. When $\mathcal{A}$ asks to sign a dataset $j = j^*$ with items $(x_1, \ldots, x_N)$, $\mathcal{A}'$ forwards it to the challenger to obtain the signatures $(\sigma_1, \ldots, \sigma_N)$. It signs (pk_{2,j^*}, τ, N) to obtain σ_τ using sk_1 and forwards $(\sigma_\tau, \sigma_1, \ldots, \sigma_N)$ to $\mathcal{A}$. All other datasets $j \neq j^*$ it signs honestly by generating the public/secret keys of the single-data homomorphic signature scheme. Finally, suppose $\mathcal{A}$ outputs $(g, \tau, y, (\sigma_\tau = (pk_2, \tau, N', \rho), \sigma^*)$ of type II forgery. Then, assuming $\mathcal{A}'$ guessed the dataset correctly, we know that $\tau = \tau_{j^*}$, $N = N_{j^*}$ and $pk_2 = pk_{2,j^*}$, $\mathsf{Verify}^*_{pk_2}(g, y, \sigma^*) = \text{accept}$ but $g \neq g(x_1, \ldots, x_N)$. Hence, $\mathcal{A}'$ can output (g, y, σ^*) to break the security of single-data homomorphic signature scheme. This shows that if the regular signature scheme is secure and the single-data homomorphic signature scheme is secure, then $\mathcal{S}$ is also secure. □

6.4.3 From Single-Data to Multi-Data, Construction 2

We describe another generic transformation from single-data homomorphic signature scheme with short public parameters prms (independent on the data size; realized by our construction in 6.3.3) to multi-data scheme. We point out that for this transformation it is sufficient to start with a selectively secure single data scheme leading efficiency improvements. However, the resulting construction does not work well for verifiable outsourcing since the verification algorithm runs in time proportional to the run-time of the function.

Let $\mathcal{S}' = (\mathsf{PrmsGen}', \mathsf{KeyGen}', \mathsf{Sign}', \mathsf{Verify}', \mathsf{Process}', \mathsf{Eval}')$ be a selectively secure homomorphic signature scheme. Let $\mathcal{S}^{nh} = (\mathsf{NH.KeyGen}, \mathsf{NH.Sign}, \mathsf{NH.Verify})$ be any standard (<u>n</u>ot <u>h</u>omomorphic) signature scheme. We construct a multi-data homomorphic signature scheme

$\mathcal{S} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify}, \mathsf{Eval})$ with message space $\mathcal{X}$ as follows.[7]

- $(pk, sk) \leftarrow \mathsf{KeyGen}(1^\lambda)$: Choose $(pk_1, sk_1) \leftarrow \mathsf{NH.KeyGen}(1^\lambda)$ set $pk = pk_1$, $sk = sk_1$.
- $(\sigma_\tau, \sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}((x_1, \ldots, x_N), \tau)$:
  - Sample parameters and keys of the single-data homomorphic signature scheme: $\mathsf{prms} \leftarrow \mathsf{PrmsGen}(1^\lambda, 1^N)$, $(pk_2, sk_2) \leftarrow \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$.
  - Sign the dataset size, the tag and the public parameters of the single-data homomorphic signature scheme using non-homomorphic scheme: $\rho \leftarrow \mathsf{NH.Sign}_{sk_1}((\mathsf{prms}, pk_2, \tau, N))$. Set $\sigma_\tau = (\mathsf{prms}, pk_2, \tau, N, \rho)$.
  - Sign the dataset using the single-data homomorphic scheme: $(\sigma_1, \ldots, \sigma_N) \leftarrow \mathsf{Sign}'_{sk_2}(x_1, \ldots, x_N)$.
  - Output $(\sigma_\tau, \sigma_1, \ldots, \sigma_N)$.
- $\sigma^* = \mathsf{Eval}_{pk}(g, \sigma_\tau, \sigma(x_1, \sigma_1), \ldots, (x_\ell, \sigma_\ell))$: Parse $\sigma_\tau = (\mathsf{prms}, pk_2, \tau, N, \rho)$. Apply the single-data evaluation algorithm. $\sigma^* = \mathsf{Eval}_{\mathsf{prms}}(g, (x_1, \sigma_1), \ldots, (x_N, \sigma_N))$.
- $\mathsf{Verify}_{pk}(g, y, \tau, (\sigma_\tau, \sigma^*))$: Parse $\sigma_\tau = (\mathsf{prms}, pk_2, \tau, N, \rho)$ and accept if and only if the following two conditions are satisfied:
  1. Verify the parameters of the single-data homomorphic scheme and the dataset: $\mathsf{NH.Verify}_{pk_1}((\mathsf{prms}, pk_2, \tau, N), \rho) =$ accept, and
  2. Verify the homomorphically computed signature: $\mathsf{Verify}'_{pk_2}(g, \mathsf{Process}_{\mathsf{prms}}(g), y, \sigma^*) =$ accept.

Remarks. We note that that there is no a-prior bound on the size of the datasets in this construction. However, since σ_τ includes the description of the public parameters of the single-data homomorphic scheme, these parameters must be small (as in our construction in the Random Oracle model).

Correctness. Correctness of the scheme follows readily from the correctness of the regular signature scheme and the single-data homomorphic signature scheme.

Security. On the high level, security follows from the fact that by the security property of the standard signature scheme, the adversary cannot modify the data size or the public parameters of the single-data signature scheme. And, given the correct parameters of the single-data signature scheme, we can efficiently verify the result of the computation by verifying the homomorphically computed signature.

[7]Note that we do not define a separate $\mathsf{PrmsGen}$ or $\mathsf{Process}$ algorithms, since this construction does not support efficient verification with preprocessing.

Theorem 6.4.2. *Assume $\mathcal{S}'$ is a selectively secure single-data homomorphic signature scheme and $\mathcal{S}^{nh}$ is a regular signature scheme. Then, $\mathcal{S}$ is a fully secure many-dataset homomorphic signature scheme.*

Proof. We prove the security via a series of indistinguishable games.

- Let **Game 0** be the multi-data security game.
- Let **Game 1** be the modified version of **Game 1**, except the attacker loses the game if it outputs a non-homomorphic forgery. That is, a forgery of the form $(g, y, \tau, (\sigma_\tau = (\mathsf{prms}, pk_2, \tau, N', \rho), \sigma^*)$, where:

  1. $\tau \neq \tau_j$ for any j, or
  2. $\tau = \tau_j$ for some j but $N' \neq N_j$ or
  3. $\tau = \tau_j$ for some j, $N' = N_j$, but $\mathsf{prms} \neq \mathsf{prms}_j$ or $pk_2 \neq pk_{2,j}$, where $\mathsf{prms}_j, pk_{2,j}$ are the parameters used for single-data signature scheme.

  That is, if the parameters of the dataset are invalid, the adversary losses the game. Note that this is the superset of type I forgeries. Clearly, if there exists a winning adversary in **Game 1**, then we can break the security of the regular signature scheme, since we obtain a valid signature ρ of $(\mathsf{prms}, pk_2, \tau, N')$ which was never signed before.

Now, assume there exists an adversary $\mathcal{A}$ that wins in **Game 1**. Then, it must be able to come up with a type II forgery. Hence, we can construct an adversary $\mathcal{A}'$ that breaks the security of the single-data homomorphic signature scheme. $\mathcal{A}'$ generates parameters of the regular signature scheme $(pk_1, sk_1) \leftarrow \mathsf{NH.KeyGen}(1^\lambda)$ and forwards $pk = pk_1$ to $\mathcal{A}$. It also chooses an index j^* of the dataset on which it guesses the type II will be made. When $\mathcal{A}$ asks to sign a dataset $j = j^*$ with items $(x_1, \ldots, x_N)$, $\mathcal{A}'$ forwards it to the single-data challenger to obtain the signatures $(\sigma_1, \ldots, \sigma_N)$ along with the public parameters $(\mathsf{prms}_{j^*}, pk_{2,j^*})$. It signs $(\mathsf{prms}_{j^*}, pk_{2,j^*}, \tau, N)$ to obtain σ_τ using sk_1 and forwards $(\sigma_\tau, \sigma_1, \ldots, \sigma_N)$ to $\mathcal{A}$. All other datasets $j \neq j^*$ it signs honestly by generating the parameters of the single-data homomorphic signature scheme. Finally, suppose $\mathcal{A}$ outputs $(g, \tau, y, (\sigma_\tau = (\mathsf{prms}, pk_2, \tau, N', \rho), \sigma^*)$ of type II forgery. Then, assuming $\mathcal{A}'$ guessed the dataset correctly, we know that $\tau = \tau_{j^*}, N = N_{j^*}, \mathsf{prms} = \mathsf{prms}_{j^*}$ and $pk_2 = pk_{2,j^*}$, $\mathsf{Verify}'_{pk_2}(g, \mathsf{Process}_{\mathsf{prms}}(g), y, \sigma^*) =$ accept but $g \neq g(x_1, \ldots, x_N)$. Hence, $\mathcal{A}'$ can output (g, y, σ^*) to break the security of single-data homomorphic signature scheme. This shows that if the regular signature scheme is secure and the single-data homomorphic signature scheme is secure, then $\mathcal{S}$ is also secure. □

6.5 Context-Hiding Security

In many applications, we may also want to guarantee that a signature which certifies y as the output of some computation g over Alice's data should not reveal anything about the underlying data beyond the output of the computation. We will show how to achieve context-hiding by taking our original schemes which produces some signature σ (that is not context hiding) and applying some procedure $\widetilde{\sigma} \leftarrow \mathsf{Hide}_{pk,y}(\sigma)$ which makes the signature context hiding. The "hiding" signature $\widetilde{\sigma}$ can be simulated given only g, y no matter which original signature σ was used to create it. One additional advantage of this procedure is that it also compresses the size of the signature from $m^2 \log q$ bits needed to represent σ to $O(m \log q)$ bits needed to represent $\widetilde{\sigma}$. However, once the hiding procedure is applied, the signatures no longer support additional homomorphic operations on them.

Context-Hiding Security for Signatures. We give a simulation-based notion of security, requiring that a context-hiding signature $\widetilde{\sigma}$ can be simulated given knowledge of only the computation g and the output y, but without any other knowledge of Alice's data. The simulation remains indistinguishable even given the underlying data, the underlying signatures, and even the public/secret key of the scheme. In other words, the derived signature does not reveal anything beyond the output of the computation even to an attacker that may have some partial information on the underlying values.

Definition 6.5.1. *A single-data homomorphic signature supports* context hiding *if there exist additional PPT procedures* $\widetilde{\sigma} \leftarrow \mathsf{Hide}_{pk,y}(\sigma)$ *and* $\mathsf{HVerify}_{pk}(\alpha, y, \sigma)$ *such that:*

- *Correctness: For any* $\mathsf{prms} \in \mathsf{PrmsGen}(1^\lambda, 1^N), (pk, sk) \in \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$ *and any* α, y, σ *such that* $\mathsf{Verify}_{pk}(\alpha, y, \sigma) = \text{accept}$, *for any* $\widetilde{\sigma} \in \mathsf{Hide}_{pk,y}(\sigma)$ *we have* $\mathsf{HVerify}_{pk}(\alpha, y, \widetilde{\sigma}) = \text{accept}$.

- *Unforgeability: Single-data signature security holds when we replace the* Verify *procedure by* HVerify *in the security game.*

- *Context-Hiding Security: There is a simulator* Sim *such that, for any* fixed *(worst-case) choice of* $\mathsf{prms} \in \mathsf{PrmsGen}(1^\lambda, 1^N), (pk, sk) \in \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$ *and any* α, y, σ *such that* $\mathsf{Verify}_{pk}(\alpha, y, \sigma) = \text{accept}$ *we have:* $\mathsf{Hide}_{pk,y}(\sigma) \approx \mathsf{Sim}(sk, \alpha, y)$ *where the randomness is only over the random coins of the simulator and the* Hide *procedure.*[8] *We say that such schemes are* statistically context hiding *if the above indistinguishability holds statistically.*

The case of multi-data signatures is defined analogously.

Definition 6.5.2. *A multi-data homomorphic signature supports* context hiding *if there exist additional PPT procedures* $\widetilde{\sigma} \leftarrow \mathsf{Hide}_{pk,x}(\sigma)$, $\mathsf{HVerify}_{pk}(g, \mathsf{Process}(g), y, \tau, (\sigma_\tau, \sigma))$ *such that:*

- *Correctness: For any* $\mathsf{prms} \in \mathsf{PrmsGen}(1^\lambda, 1^N), (pk, sk) \in \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$ *and any* $\alpha, y, \sigma_\tau, \sigma$ *such that* $\mathsf{Verify}_{pk}(\alpha, y, \tau, (\sigma_\tau, \sigma)) = \text{accept}$, *for any* $\widetilde{\sigma} \in \mathsf{Hide}_{pk,y}(\sigma)$ *we have*

$$\mathsf{HVerify}_{pk}(\alpha, y, \tau, (\sigma_\tau, \widetilde{\sigma})) = \text{accept}$$

[8]Since $pk, sk, \alpha, y, \sigma$ are fixed, indistinguishability holds even if these values are known to the distinguisher.

- *Unforgeability: Multi-data signature security holds when we replace the* Verify *procedure by* $\mathsf{HVerify}$ *in the security game.*

- *Context-Hiding Security: Firstly, in the procedure* $(\sigma_\tau, \sigma_1, \dots, \sigma_N) \leftarrow \mathsf{Sign}_{sk}(x_1, \dots, x_N, \tau)$, *we require that* σ_τ *can only depend on* (sk, N, τ) *but not on the data* $\{x_i\}$. *Secondly, we require that there is a simulator* Sim *such that, for any* fixed *(worst-case) choice of* $\mathsf{prms} \in \mathsf{PrmsGen}(1^\lambda, 1^N), (pk, sk) \in \mathsf{KeyGen}(1^\lambda, \mathsf{prms})$ *and any* $\alpha, y, \sigma, \sigma_\tau$ *such that* $\mathsf{Verify}_{pk}(\alpha, y, \tau, (\sigma_\tau, \sigma)) = \text{accept}$ *we have:*

$$\mathsf{Hide}_{pk,y}(\sigma) \approx \mathsf{Sim}(sk, \alpha, y, \tau, \sigma_\tau)$$

  where the randomness is only over the random coins of the simulator and the Hide *procedure. We say that such schemes are* statistically context hiding *if the above indistinguishability holds statistically.*

Context-Hiding Security for HTDF. We also define a *context hiding HTDF* as an augmentation of standard HTDFs. We will build context-hiding signatures by relying on context-hiding HTDFs.

Definition 6.5.3. *A context-hiding HTDF comes with two additional algorithms* $\tilde{u} \leftarrow \mathsf{HTDF.Hide}_{pk,x}(u)$ *and* $\mathsf{HTDF.Verify}_{pk}(\tilde{u}, x, v)$ *satisfying:*

- Correctness: *For any* $(pk, sk) \in \mathsf{KeyGen}(1^\lambda)$ *any* $u \in \mathcal{U}$, *any* $x \in \mathcal{X}$ *and any* $\tilde{u} \in \mathsf{HTDF.Hide}_{pk,x}(u)$ *we have* $\mathsf{HTDF.Verify}_{pk}(\tilde{u}, x, f_{pk,x}(u)) = \text{accept}$.

- Claw-freeness on hidden inputs: *We augment standard HTDF security with the following requirement. For all PPT* $\mathcal{A}$ *we require:*

$$\Pr\left[\begin{array}{c} \mathsf{HTDF.Verify}_{pk}(\tilde{u}', x', f_{pk,x}(u)) = \text{accept} \\ u \in \mathcal{U}, x, x' \in \mathcal{X} \quad , x \neq x' \end{array} \middle| \begin{array}{c} (pk, sk) \leftarrow \mathsf{HTDF.KeyGen}(1^\lambda) \\ (u, \tilde{u}', x, x') \leftarrow \mathcal{A}(1^\lambda, pk) \end{array}\right] \leq \text{negl}(\lambda).$$

  In other words, if an attacker know u *such that* $f_{pk,x}(u) = v$ *then he cannot also produce* $\tilde{u}'$ *such that* $\mathsf{HTDF.Verify}_{pk}(\tilde{u}', x', v) = \text{accept}$ *when* $x' \neq x$. [9]

- Context Hiding: *There is a simulator* $\mathsf{HTDF.Sim}$ *such that for all choices of* $(pk, sk) \in \mathsf{HTDF.KeyGen}(1^\lambda)$, $u \in \mathcal{U}$ *and* $x \in \mathcal{X}$ *the following distributions are indistinguishable:*

$$\mathsf{HTDF.Hide}_{pk,x}(u) \approx \mathsf{HTDF.Sim}(sk, x, f_{pk,x}(u)).$$

  We say that such schemes are statistically context hiding *if the above indistinguishability holds statistically.*

[9] This implies standard HTDF security since any attacker that finds $x \neq x', u, u'$ such that $f_{pk,x}(u) = f_{pk,x'}(u')$ can also apply $\tilde{u}' \leftarrow \mathsf{Hide}_{pk,x'}(u')$ to break claw-freeness on hidden inputs.

From Context-Hiding HTDFs to Signatures. We can easily modify the signature schemes constructed in 6.3 (single-data) and 6.4 (multi-data) in the natural way to make them context hiding by using a context-hiding HTDF. In particular, the procedure Hide of the signature scheme is defined to be the same as that of the underlying HTDF. The procedure $\mathsf{HVerify}_{pk}(\alpha, y, \widetilde{\sigma})$ of the signature scheme (resp. $\mathsf{HVerify}_{pk}(\alpha, y, \tau, (\sigma_\tau, \widetilde{\sigma}))$ for a multi-data scheme) are defined the same ways as the original Verify procedures of the signature, *except* that, instead of checking $f_{pk,y}(\widetilde{\sigma}) = \alpha$ we now check $\mathsf{HTDF.Verify}_{pk}(\widetilde{\sigma}, y, \alpha) = \text{accept}$. It is easy to check that this modification satisfies the given correctness and security requirements as outlined below.

Unforgeability with the modified verification procedure $\mathsf{HVerify}$ follows from the "claw-freeness on hidden inputs" property of the HTDF. This follows from the proof of Theorem 6.3.1. In both proofs, the reduction knows one value $u \in \mathcal{U}$ such that $f_{pk,y}(u) = v^*$ and a signature forgery allows it to come up with $\tilde{u}$ such that $\mathsf{HTDF.Verify}_{pk}(\widetilde{\sigma}, y', v^*) = \text{accept}$ for $y' \neq y$.

Context-Hiding security of the signature scheme follows from that of the HTDF. We define the signature simulator $\mathsf{Sim}(sk, g, y, [\tau, \sigma_\tau])$ to compute the value $v^* = \mathsf{Eval}^{in}_{pk}(g, v_1, \ldots, v_N)$ as is done by the verification procedure of the signature schemes. It then output $\widetilde{\sigma} \leftarrow \mathsf{HTDF.Sim}(sk, y, v^*)$. The indistinguishability of the signature simulator follows form that of the HTDF simulator.

General Construction via NIZKs. Before we give our main construction of context-hiding HTDF and therefore context-hiding signatures, we mention that it is possible to solve this problem generically using *non-interactive zero knowledge (ZK) proof of knowledge (PoK) NIZK-PoKs.* In particular, we can make any HTDF context-hiding by setting $\tilde{u} \leftarrow \mathsf{HTDF.Hide}_{pk,x}(u)$ to be a NIZK-PoK with the statement v and witness u for the relation $f_{pk,x}(u) = v$. The $\mathsf{HTDF.Verify}$ procedure would simply verify the proof $\tilde{u}$. Claw-freeness follows from the PoK property and context-hiding follows from ZK.[10] However, this approach requires an additional assumption (existence of NIZK-PoK) which is not known to follow from SIS. Therefore, we now proceed to construct context-hiding HTDFs directly.

6.5.1 Construction of Context-Hiding HTDF

Lattice Preliminaries. Before giving our construction of context-hiding HTDFS, we start by recalling some additional useful tools from lattice-based cryptography (abstracting as much as possible). Let $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$ and let $\mathbf{H} = [\mathbf{A} \mid \mathbf{B}] \in \mathbb{Z}_q^{n \times 2m}$. We will rely on the existence of two algorithms $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ which both take $\mathbf{z} \in \mathbb{Z}_q^n$ and manage to output some "short" vector $\mathbf{r} \in \mathbb{Z}_q^{2m}$ such that $\mathbf{H} \cdot \mathbf{r} = \mathbf{z}$. The algorithm $\mathsf{SampleLeft}$ does so by knowing some trapdoor td for the matrix $\mathbf{A}$. The algorithm $\mathsf{SampleRight}$ does so by knowing some "short" matrix $\mathbf{U}$ such that $\mathbf{B} = \mathbf{AU} + y\mathbf{G}$ for some $y \neq 0$. Nevertheless, the outputs of $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ are statistically indistinguishable.

[10]The syntactic definition would need to be modified slightly to include a common reference string (CRS).

(See [CHKP12, ABB10a, MP12, BGG$^+$14] for details on the following lemma; our exposition follows [BGG$^+$14] with additional abstraction.)

Lemma 6.5.1. *Using the notation of Lemma 2.4.1, let $n, q \geq 2$, $m \geq m^*(n,q)$ and β be parameters. Then there exist polynomial time algorithms* $\mathsf{SampleLeft}, \mathsf{SampleRight}$ *and some polynomial $p_{extra}(n, m, \log q)$ such that for $\beta' := \beta \cdot p_{extra}(n, m, \log q)$ the following holds: For any choice of $(\mathbf{A}, \mathsf{td}) \in \mathsf{TrapSamp}(1^n, 1^m, q)$, any $\mathbf{z} \in \mathbb{Z}_q^n$ and any $\mathbf{U} \in \mathbb{Z}_q^{m \times m}$ with $||\mathbf{U}||_\infty \leq \beta$ and any $y \in \mathbb{Z}_q$ with $y \neq 0$ let $\mathbf{H} = [\mathbf{A} \mid \mathbf{AU} + y\mathbf{G}]$, where $\mathbf{G}$ is the matrix from part (3) of Lemma 2.4.1. Then:*

- *For any $\mathbf{r}_0 \in \mathsf{SampleLeft}(\mathbf{H}, \mathsf{td}, \mathbf{z})$, $\mathbf{r}_1 \in \mathsf{SampleRight}(\mathbf{H}, \mathbf{U}, \mathbf{z})$ and for each $b \in \{0, 1\}$ we have $\mathbf{r}_b \in \mathbb{Z}_q^{2m}$, $||\mathbf{r}_b||_\infty \leq \beta'$ and $\mathbf{H} \cdot \mathbf{r}_b = \mathbf{z}$.*
- *For $\mathbf{r}_0 \leftarrow \mathsf{SampleLeft}(\mathbf{H}, \mathsf{td}, \mathbf{z})$ and $\mathbf{r}_1 \leftarrow \mathsf{SampleRight}(\mathbf{H}, \mathbf{U}, \mathbf{z})$ we have $\mathbf{r}_0 \approx \mathbf{r}_1$ are statistically indistinguishable (the statistical distance is negligible in n).*

HTDF with Context Hiding. We augment our construction of HTDFs from 6.2 to add context-hiding security. Firstly, we make the following modifications to the underlying HTDF construction.

- We restrict the index space $\mathcal{X}$ to just bits $\mathcal{X} = \{0, 1\} \subseteq \mathbb{Z}_q$ (rather than $\mathcal{X} = \mathbb{Z}_q$ as previously). We also modify the parameters and set $\beta_{SIS} = 2^{\omega(\log \lambda)}(\beta_{max})^2$ to be larger than before (which impacts how q, n are chosen to maintain security).
- We augment the public-key to $pk = (\mathbf{A}, \mathbf{z})$ by appending $\mathbf{z} \in \mathbb{Z}_q^n$ which is chosen by selecting a random $\mathbf{r} \leftarrow \{0, 1\}^m$ and setting $\mathbf{z} = \mathbf{A} \cdot \mathbf{r}$ (and discarding $\mathbf{r}$).[11] Let $p_{extra}(n, m, \log q) = \text{poly}(\lambda)$ be the polynomial form Lemma 6.5.1 and define $\tilde{\beta}_{max} = \beta_{max} \cdot p_{extra}(n, m, \log q)$.

In addition, we add the following procedures for context-hiding security.

- $\tilde{\mathbf{u}} \leftarrow \mathsf{HTDF.Hide}_{pk,x}(\mathbf{U})$: Let $\mathbf{V} = f_{pk,x}(\mathbf{U}) = \mathbf{AU} + x\mathbf{G}$. Set
$$\mathbf{H} := [\mathbf{A} \mid \mathbf{V} + (x-1)\mathbf{G}] = [\mathbf{A} \mid \mathbf{AU} + (2x-1)\mathbf{G}].$$
Note that $(2x - 1) \in \{-1, 1\} \neq 0$. Output $\tilde{\mathbf{u}} \leftarrow \mathsf{SampleRight}(\mathbf{H}, \mathbf{U}, \mathbf{z})$. Note that $\mathbf{H} \cdot \tilde{\mathbf{u}} = \mathbf{z}$ and $||\tilde{\mathbf{u}}||_\infty \leq \tilde{\beta}_{max}$.
- $\mathsf{HTDF.Verify}_{pk}(\tilde{\mathbf{u}}, x, \mathbf{V})$: Compute $\mathbf{H} := [\mathbf{A} \mid \mathbf{V} + (x-1)\mathbf{G}]$. Check $||\tilde{\mathbf{u}}||_\infty \leq \tilde{\beta}_{max}$ and $\mathbf{H} \cdot \tilde{\mathbf{u}} = \mathbf{z}$. If so accept, else reject.
- For context-hiding security, we define $\mathsf{HTDF.Sim}(sk = \mathsf{td}, x, \mathbf{V})$ which computes $\mathbf{H} := [\mathbf{A} \mid \mathbf{V} + (x-1)\mathbf{G}]$ and outputs $\tilde{\mathbf{u}} \leftarrow \mathsf{SampleLeft}(\mathbf{H}, \mathsf{td}, \mathbf{z})$.

[11] We note that, using the leftover-hash-lemma, we can show that this is statistically close to choosing $\mathbf{z} \leftarrow \mathbb{Z}_q^n$ at random. However, we will not need to rely on this fact.

Theorem 6.5.2. *The above scheme is* statistically context-hiding. *It satisfies* claw-freeness on hidden inputs *under the* $\mathsf{SIS}(n, m, q, \beta_{SIS})$ *assumption.*

Proof. It's easy to check that correctness holds. Statistical context-hiding security follows directly from Lemma 6.5.1. We are left to show claw-freeness on hidden inputs.

The proof of security closely follows that of Theorem 6.2.1. Assume that $\mathcal{A}$ is a PPT attacker that breaks this security property of the scheme. As a first step, we modify the game so that, instead of sampling $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapSamp}(1^n, 1^m, q)$ and setting $pk := \mathbf{A}$ and $sk = \mathsf{td}$, we just choose $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$ uniformly at random. This modification is statistically indistinguishable by the security of $\mathsf{TrapSamp}$ (see Lemma 2.4.1, part (2)). In particular, the probability of $\mathcal{A}$ winning the modified game remains non-negligible.

We now show that an attacker who wins the above-modified game can be used to solve the SIS problem. The reduction gets a challenge matrix $\mathbf{A}$ of the SIS problem and chooses $\mathbf{r} \leftarrow \{0,1\}^m$ and sets $\mathbf{z} = \mathbf{A} \cdot \mathbf{r}$. It gives the public key $pk = (\mathbf{A}, \mathbf{z})$ to the attacker $\mathcal{A}$. The attacker wins if he comes up with bits $x \neq x' \in \{0,1\}$ and values $\mathbf{U}, \tilde{\mathbf{u}}'$ such that $||\mathbf{U}||_\infty \leq \beta_{max}$, $||\tilde{\mathbf{u}}'||_\infty \leq \tilde{\beta}_{max}$, and $\mathbf{H}\cdot\tilde{\mathbf{u}}' = \mathbf{z}$ where $\mathbf{H}$ is defined by setting $\mathbf{V} := f_{pk,x}(\mathbf{U}) = \mathbf{A}\mathbf{U} + x\mathbf{G}$ and

$$\mathbf{H} := [\mathbf{A} \mid \mathbf{V} + (x'-1)\mathbf{G}] = [\mathbf{A} \mid \mathbf{A}\mathbf{U} + (x + x' - 1)\mathbf{G}] = [\mathbf{A} \mid \mathbf{A}\mathbf{U}]$$

where the last equality follows since $x \neq x' \;\; \Rightarrow \;\; x + x' = 1$. Let's write $\tilde{\mathbf{u}}' = (\mathbf{r}_1', \mathbf{r}_2')$ where $\mathbf{r}_1', \mathbf{r}_2' \in \mathbb{Z}_q^m$ are the first and last m components of $\tilde{\mathbf{u}}'$ respectively. Then:

$$\mathbf{H} \cdot \tilde{\mathbf{u}}' = \mathbf{z} \;\;\Rightarrow\;\; \mathbf{A}\mathbf{r}_1 + (\mathbf{A}\mathbf{U})\mathbf{r}_2 = \mathbf{A}\mathbf{r} \;\;\Rightarrow\;\; \mathbf{A}(\mathbf{U}\mathbf{r}_2 + \mathbf{r}_1 - \mathbf{r}) = \mathbf{0}$$

Furthermore

$$||(\mathbf{U}\mathbf{r}_2 + \mathbf{r}_1 - \mathbf{r})||_\infty \leq m\beta_{max}\tilde{\beta}_{max} + \tilde{\beta}_{max} + 1 \leq \mathrm{poly}(\lambda)(\beta_{max})^2 \leq \beta_{SIS}.$$

Therefore, it remains to show that $(\mathbf{U}\mathbf{r}_2 + \mathbf{r}_1 - \mathbf{r}) \neq \mathbf{0}$. We use the same argument as in the proof of Theorem 6.2.1: the randomness $\mathbf{r}$ is independent of $\mathbf{U}, \mathbf{r}_1, \mathbf{r}_2$ when conditioned on $\mathbf{z}$. Since $\mathbf{z}$ is short, $\mathbf{r}$ still has $m - n\log q = \omega(\log\lambda)$ bits of conditional entropy left and therefore $\Pr[\mathbf{U}\mathbf{r}_2 + \mathbf{r}_1 = \mathbf{r}] \leq \mathrm{negl}(\lambda)$. This concludes the proof. □

6.6 HTDFs and Fully Homomorphic Encryption

We briefly and informally sketch an interesting conceptual connection between fully homomorphic encryption and signatures. We show that both of these primitives can be constructed from a type of HTDFs: signatures correspond to *equivocable* HTDFs, whereas encryption corresponds to *extractable* HTDFs.

For equivocable HTDFs, which was the notion we defined in this paper, we wanted the output $v = f_{pk,x}(u)$ over a random u to statistically hide x and to have an "equivocation trapdoor" sk that allows us to open any v to any index x, by providing a value u such that $f_{pk,x}(u) = v$. For an adversary without this trapdoor, each output v would be computationally binding to x.

For an extractable HTDF, we would instead want $v = f_{pk,x}(u)$ to be statistically binding to x and to have an "extraction trapdoor" sk that allows us to extract/decrypt the value x from v. For an adversary without this trapdoor, the output v would computationally hide x.

In 6.2, we gave a construction of equivocable HTDFs with security based on the SIS problem. We now show that, by modifying the key generation procedure of our construction, we can easily convert it to an extractable HTDF. In fact, there is a single HTDF constriction with two *indistinguishable* modes of choosing the public key pk: in one mode, the resulting HTDF is equivocable with an equivocation trapdoor sk and in the other more the resulting HTDF is extractable with extraction trapdoor sk. The indistinguishability of these two modes follows from the learning with errors (LWE) assumption.

Recall that in our construction of HTDFs we set $pk = \mathbf{A}$ where $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ is (statistically close to) a uniformly random matrix. This corresponds to the equivocation mode. For the extractable mode we choose a matrix $\mathbf{A}' \leftarrow \mathbb{Z}_q^{(n-1)\times m}$ and a secret $\mathbf{s}' \leftarrow \mathbb{Z}_q^{n-1}$. We set

$$\mathbf{A} = \begin{pmatrix} \mathbf{A}' \\ \mathbf{s}'\mathbf{A}' + \mathbf{e} \end{pmatrix}$$

where $\mathbf{e}$ is some appropriately sampled short "noise vector". This corresponds to a *learning with errors* (LWE) instance with secret $\mathbf{s}'$, and therefore $\mathbf{A}$ chosen as above is computationally indistinguishable from a uniformly random. Let $\mathbf{s} = (-\mathbf{s}', 1) \in \mathbb{Z}_q^n$ so that $\mathbf{sA} = \mathbf{e}$. We set $pk = \mathbf{A}$ and $sk = \mathbf{s}$ to be the public key and secret extraction key of the HTDF. Otherwise, the construction $f_{pk,x}(\mathbf{U}) = \mathbf{AU} + x\mathbf{G}$ and the homomorphic operations are performed the exact same way as in 6.2.2 and 6.2.3. Assume $\mathbf{V} = f_{pk,x}(\mathbf{U}) = \mathbf{AU} + x\mathbf{G}$ where $\mathbf{U}$ is short. Let $\mathbf{z} = (0, \dots, 0, r) \in \mathbb{Z}_q^n$ where r is a scalar of "medium size". We can then compute

$$\mathbf{s} \cdot \mathbf{V} \cdot \mathbf{G}^{-1}(\mathbf{z}) = \mathbf{e} \cdot \mathbf{U} \cdot \mathbf{G}^{-1}(\mathbf{z}) + x \cdot \langle \mathbf{s}, \mathbf{z} \rangle = x \cdot r + e'$$

where e' is short. As long as the parameters are chosen so that $|x \cdot r + e'| < q/2$ so there is no wrap-around, and $|e'| < |r|$, the above allows us to extract x.

With the above HTDF in extraction mode, if we think of $\mathsf{Enc}_{pk}(x; u) = f_{pk,x}(u)$ as a public-key encryption scheme with randomness u, and of the Eval^{out} procedure as a homomorphic evaluation on ciphertexts, then this scheme corresponds to the FHE scheme of Gentry, Sahai and Waters [GSW13].

6.7 Conclusions and Open Problems

In this work, we construct the first leveled fully homomorphic signature schemes. It remains an open problem to get rid of the *leveled* aspect and ideally come up with a signature scheme where there is no a priori bound on the depth of the circuits that can be evaluated and the signature size stays fixed. It also remains an open problem to come up with a (leveled) fully homomorphic signature scheme with short public parameters under a standard assumption without random oracles.

Bibliography

[AB09] Shweta Agrawal and Dan Boneh. Homomorphic MACs: MAC-based integrity for network coding. In *ACNS*, pages 292–305, 2009.

[ABB10a] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[ABB10b] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Lattice basis delegation in fixed dimension and shorter ciphertext hierarchical ibe. In *CRYPTO*, pages 98–115, 2010.

[ABC$^+$07] Giuseppe Ateniese, Randal C. Burns, Reza Curtmola, Zachary N. J. Peterson Joseph Herring, Lea Kissner, and Dawn Song. Provable data possession at untrusted stores. In *CCS*, pages 598–609, 2007.

[ABC$^+$12] Jae Hyun Ahn, Dan Boneh, Jan Camenisch, Susan Hohenberger, Abhi Shelat, and Brent Waters. Computing on authenticated data. In *TCC*, pages 1–20, 2012.

[ABSV14] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. Cryptology ePrint Archive: Report 2014/917, 2014.

[ABV$^+$12] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or, fuzzy IBE) from lattices. In *PKC*, pages 280–297, 2012.

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. In *ACM Trans. Inf. Syst. Secur.*, 2006.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.

[AGHS13] Shweta Agrawal, Craig Gentry, Shai Halevi, and Amit Sahai. Discrete gaussian leftover hash lemma over infinite domains. In *ASIACRYPT*, pages 97–116. 2013.

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In *CRYPTO*, pages 500–518, 2013.

[AIK10] Benny Applebaum, Yuval Ishai, and Eyal Kushilevitz. From secrecy to soundness: Efficient verification via secure computation. In *ICALP*, pages 152–163, 2010.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. Cryptology ePrint Archive, Report 2015/173, 2015.

[Ajt96] Miklós Ajtai. Generating hard instances of lattice problems (extended abstract). In *STOC*, pages 99–108, 1996.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.

[AKK09] Giuseppe Ateniese, Seny Kamara, and Jonathan Katz. Proofs of storage from homomorphic identification protocols. In *ASIACRYPT*, pages 319–333, 2009.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[AL11] Nuttapong Attrapadung and Benoît Libert. Homomorphic network coding signatures in the standard model. In *PKC*, pages 17–34, 2011.

[AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, pages 75–86, 2009.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *CRYPTO*, pages 297–314, 2014.

[APG$^+$11] Joseph A. Akinyele, Matthew W. Pagano, Matthew D. Green, Christoph U. Lehmann, Zachary N.J. Peterson, and Aviel D. Rubin. Securing electronic medical records using attribute-based encryption on mobile devices. In *SPSM*, pages 75–86, 2011.

[AR13] Divesh Aggarwal and Oded Regev. A note on discrete gaussian combinations of lattice vectors. *CoRR*, abs/1308.2405, 2013.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.

[BCCT12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge and back again. In *ITCS*, pages 326–349, 2012.

[BCCT13] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for snarks and proof-carrying data. In *STOC*, pages 111–120, 2013.

[BCI$^+$13] Nir Bitansky, Alessandro Chiesa, Yuval Ishai, Rafail Ostrovsky, and Omer Paneth. Succinct non-interactive arguments via linear interactive proofs. In *TCC*, pages 315–333, 2013.

[BCPR14] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the existence of extractable one-way functions. In *STOC*, pages 505–514, 2014.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BF11a] Dan Boneh and David Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168, 2011.

[BF11b] Dan Boneh and David Freeman. Linearly homomorphic signatures over binary fields and new tools for lattice-based signatures. In *PKC*, pages 1–16, 2011.

[BFKW09] Dan Boneh, David Freeman, Jonathan Katz, and Brent Waters. Signing a linear subspace: Signature schemes for network coding. In *PKC*, pages 68–87, 2009.

[BFR13] Michael Backes, Dario Fiore, and Raphael M. Reischuk. Verifiable delegation of computation on outsourced data. In *CCS*, pages 863–874, 2013.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, pages 1–18, 2001.

[BGK$^+$14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In *EUROCRYPT*, pages 221–238, 2014.

[BGV11] Siavosh Benabbas, Rosario Gennaro, and Yevgeniy Vahlis. Verifiable delegation of computation over large datasets. In *CRYPTO*, pages 111–131, 2011.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, pages 258–275, 2005.

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *CCS*, pages 784–796, 2012.

[Boy13a] Xavier Boyen. Attribute-based functional encryption on lattices. In *TCC*, pages 122–142, 2013.

[Boy13b] Xavier Boyen. Attribute-based functional encryption on lattices. In *TCC*, pages 122–142, 2013.

[BR14a] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25, 2014.

[BR14b] Zvika Brakerski and GuyN. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In *TCC*, pages 1–25. 2014.

[BS03] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. *Contemporary Mathematics*, 324:71–90, 2003.

[BSCG$^+$13] Eli Ben-Sasson, Alessandro Chiesa, Daniel Genkin, Eran Tromer, and Madars Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*, pages 90–108, 2013.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BSW12] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: A new vision for public-key cryptography. *Commun. ACM*, 55(11):56–64, November 2012.

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Fully homomorphic encryption from ring-LWE and security for key dependent messages. In *CRYPTO*, pages 505–524, 2011.

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In *ITCS*, pages 1–12, 2014.

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In *Cryptology ePrint Archive, Report 2015/163*, 2015.

[BW07] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, pages 480–499. 2014.

[CF13] Dario Catalano and Dario Fiore. Practical homomorphic macs for arithmetic circuits. In *EUROCRYPT*, pages 336–352, 2013.

[CFGN14] Dario Catalano, Dario Fiore, Rosario Gennaro, and Luca Nizzardo. Generalizing homomorphic macs for arithmetic circuits. In *PKC*, pages 538–555, 2014.

[CFW12] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Efficient network coding signatures in the standard model. In *PKC*, pages 680–696, 2012.

[CFW14] Dario Catalano, Dario Fiore, and Bogdan Warinschi. Homomorphic signatures with efficient verification for polynomial functions. In *CRYPTO*, pages 371–389, 2014.

[CHKP12] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *J. Cryptology*, 25(4):601–639, 2012.

[CHL$^+$15] JungHee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien StehlÃľ. Cryptanalysis of the multilinear map over the integers. In *EUROCRYPT*, pages 3–12. 2015.

[CKLR11] Kai-Min Chung, Yael Tauman Kalai, Feng-Hao Liu, and Ran Raz. Memory delegation. In *CRYPTO*, pages 151–168, 2011.

[CKP10] Kai-Min Chung, Yael Kalai, and Salil P. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[CKV10] Kai-Min Chung, Yael Kalai, and Salil P. Vadhan. Improved delegation of computation using fully homomorphic encryption. In *CRYPTO*, pages 483–501, 2010.

[CLT13] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In *CRYPTO*, pages 476–493. 2013.

[CLT14] Jean-Sebastien Coron, Tancrede Lepoint, and Mehdi Tibouchi. Cryptanalysis of two candidate fixes of multilinear maps over the integers. Cryptology ePrint Archive, Report 2014/975, 2014.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[CW79] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143–154, 1979.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DVW09] Yevgeniy Dodis, Salil P. Vadhan, and Daniel Wichs. Proofs of retrievability via hardness amplification. In *TCC*, pages 109–127, 2009.

[Fre12] David Mandell Freeman. Improved security for linearly homomorphic signatures: A generic framework. In *PKC*, pages 697–714, 2012.

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

[GGH$^+$13b] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, pages 40–49, 2013.

[GGH$^+$13c] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, pages 479–499, 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In *TCC*, pages 498–527, 2015.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure functional encryption without obfuscation. Cryptology ePrint Archive, Report 2014/666, 2014.

[GGP10a] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GGP10b] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, pages 626–645, 2013.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476, 2013.

[GHW11] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *SEC*, pages 34–34, 2011.

[GKKR10] Rosario Gennaro, Jonathan Katz, Hugo Krawczyk, and Tal Rabin. Secure network coding over the integers. In *PKC*, pages 142–160, 2010.

[GKN08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GKP$^+$13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO*, pages 536–553, 2013.

[GKP$^+$13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GLSW14] Craig Gentry, Allison B. Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. *IACR Cryptology ePrint Archive*, 2014:309, 2014.

[GMW15] Romain Gay, Pierrick Méaux, and Hoeteck Wee. Predicate encryption for multi-dimensional range queries from lattices. In *PKC*, 2015.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*, pages 75–92, 2013.

[GV14] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. Cryptology ePrint Archive, Report 2014/819, 2014.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *STOC*, pages 99–108, 2011.

[GW13] Rosario Gennaro and Daniel Wichs. Fully homomorphic message authenticators. In *ASIACRYPT*, pages 301–320, 2013.

[HILL99] Johan Hastad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM J. Comput.*, 28:1364–1396, March 1999.

[HJ15] Yupu Hu and Huiwen Jia. Cryptanalysis of ggh map. Cryptology ePrint Archive, Report 2015/301, 2015.

[HRSV11] Susan Hohenberger, Guy N. Rothblum, Abhi Shelat, and Vinod Vaikuntanathan. Securely obfuscating re-encryption. *J. Cryptology*, 24(4):694–719, 2011.

[IW14] Yuval Ishai and Hoeteck Wee. Partial garbling schemes and their applications. In *ICALP*, pages 650–662, 2014.

[JMSW02] Robert Johnson, David Molnar, Dawn Xiaodong Song, and David Wagner. Homomorphic signature schemes. In *RSA*, pages 244–262, 2002.

[jou04] A one round protocol for tripartite diffieâĂŞhellman. *Journal of Cryptology*, 17(4):263–276, 2004.

[Kil88] Joe Kilian. Founding crytpography on oblivious transfer. In *STOC*, pages 20–31, 1988.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS*, 2000.

[KRR13] Yael Tauman Kalai, Ran Raz, and Ron Rothblum. How to delegate computations: The power of no-signaling proofs. *ECCC*, 20:183, 2013.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LOS$^+$10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In *EUROCRYPT*, pages 35–54, 2013.

[LSS14] Adeline Langlois, Damien Stehlé, and Ron Steinfeld. Gghlite: More efficient multilinear maps from ideal lattices. In *EUROCRYPT*, pages 239–256, 2014.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC*, pages 455–479, 2010.

[LW12] Allison B. Lewko and Brent Waters. New proof methods for attribute-based encryption: Achieving full security through selective techniques. In *CRYPTO*, pages 180–198, 2012.

[LYZ$^+$13] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and secure sharing of personal health records in cloud computing using attribute-based encryption. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):131–143, 2013.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. 2002.

[Mic94] Silvio Micali. CS proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[Mic00] Silvio Micali. Computationally sound proofs. *SIAM J. Comput.*, 30(4):1253–1298, 2000.

[Mic04] Daniele Micciancio. Almost perfect lattices, the covering radius problem, and applications to ajtai's connection factor. *SIAM J. Comput.*, 34(1):118–169, 2004.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.

[MP13] Daniele Micciancio and Chris Peikert. Hardness of sis and lwe with small parameters. In *CRYPTO*, pages 21–39, 2013.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In *STOC*, pages 351–358, 2010.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[OT12a] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *EUROCRYPT*, pages 591–608, 2012.

[OT12b] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. In *ASIACRYPT*, pages 349–366, 2012.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly practical verifiable computation. In *IEEE Security and Privacy*, pages 238–252, 2013.

[PPM$^+$14] John P. Papanis, Stavros I. Papapanagiotou, Aziz S. Mousas, Georgios V. Lioudakis, Dimitra I. Kaklamani, and Iakovos S. Venieris. On the use of attribute-based encryption for multimedia content protection over information-centric networks. *Transactions on Emerging Telecommunications Technologies*, 25(4):422–435, 2014.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[PRW14] Omkant Pandey, Kim Ramchen, and Brent Waters. Relaxed two-to-one recoding schemes. In *Security and Cryptography for Networks*, pages 57–76. 2014.

[PST13] Charalampos Papamanthou, Elaine Shi, and Roberto Tamassia. Signatures of correct computation. In *TCC*, pages 222–242, 2013.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In *CRYPTO*, pages 500–517, 2014.

[PTT10] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal authenticated data structures with multilinear forms. In *Pairing*, pages 246–264. 2010.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[Rot13] Ron D. Rothblum. On the circular security of bit-encryption. In *TCC*, pages 579–598. 2013.

[RS09] Markus RÃijckert and Dominique SchrÃűder. Aggregate and verifiably encrypted signatures from multilinear maps without random oracles. In *ISA*, pages 750–759. 2009.

[RS10] Alon Rosen and Gil Segev. Chosen-ciphertext security via correlated products. *SIAM J. Comput.*, 39(7):3058–3088, 2010.

[SBC$^+$07] Elaine Shi, John Bethencourt, Hubert T.-H. Chan, Dawn Xiaodong Song, and Adrian Perrig. Multi-Dimensional Range Query over Encrypted Data. In *SP*, pages 350–364, 2007.

[SBW08] Hovav Shacham and title = "Compact Proofs of Retrievability Brent Waters". In *ASIACRYPT*, pages 90–107, 2008.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[SOK00] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *SCIS*, 2000.

[SS10a] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *CCS*, pages 463–472, 2010.

[SS10b] Damien Stehlé and Ron Steinfeld. Faster fully homomorphic encryption. In *ASIACRYPT*, pages 377–394, 2010.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

[Wat12] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, pages 218–235, 2012.

[WC81] Mark N. Wegman and J.Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265–279, 1981.