ABSTRACT

Title of Dissertation: FRONTIERS IN LATTICE CRYPTOGRAPHY AND PROGRAM OBFUSCATION

Daniel Apon, Doctor of Philosophy, 2017

Dissertation directed by: Professor Jonathan Katz
Department of Computer Science

In this dissertation, we explore the frontiers of theory of cryptography along two lines. In the first direction, we explore Lattice Cryptography, which is the primary sub-area of post-quantum cryptographic research.

Our first contribution is the construction of a *deniable* attribute-based encryption scheme from lattices. A deniable encryption scheme is secure against not only eavesdropping attacks as required by semantic security, but also stronger coercion attacks performed after the fact. An attribute-based encryption scheme allows "fine-grained" access to ciphertexts, allowing for a decryption access policy to be embedded in ciphertexts and keys. We achieve both properties simultaneously for the first time from lattices.

Our second contribution is the construction of a digital signature scheme that enjoys both short signatures and a *completely tight* security reduction from lattices. As a matter of independent interest, we give an improved method of randomized inversion of the $\mathbf{G}$ gadget matrix, which reduces the noise growth rate in homomorphic evaluations performed in a large number of lattice-based cryptographic schemes, without incurring the high cost of sampling discrete Gaussians.

In the second direction, we explore Cryptographic Program Obfuscation. A program obfuscator is a type of cryptographic software compiler that outputs executable code with the guarantee that "whatever can be hidden about the internal workings of program code, is hidden." Indeed, program obfuscation can be viewed as a "universal and cryptographically-complete" tool.

Our third contribution is the first, full-scale implementation of secure program obfuscation in software. Our toolchain takes code written in a C-like programming language, specialized for cryptography, and produces secure, obfuscated software.

Our fourth contribution is a new cryptanalytic attack against a variety of "early" program obfuscation candidates. We provide a general, efficiently-testable property for any two branching programs, called *partial inequivalence,* which we show is sufficient for launching an "annihilation attack" against several obfuscation candidates based on Garg-Gentry-Halevi multilinear maps.

FRONTIERS IN LATTICE CRYPTOGRAPHY AND PROGRAM OBFUSCATION

by

Daniel Apon

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2017

Examining Committee:
Professor Jonathan Katz, Chair
Professor Andrew Childs
Professor Nikhil Chopra, Dean's Representative
Professor Dana Dachman-Soled
Professor William Gasarch

Acknowledgements

Dear Jon, they said I have to be succinct. Thank you for all the talks, and all the wisdom you shared. There is no spoon; just read more books, listen to more music! I will always enjoy our conversations together.

To Bill – Sokath! His eyes uncovered! – Thank you. *Thank. You.* Without your help, I wouldn't have had the opportunity to continue my education in such an amazing place.

This dissertation is funded in part by a generous grant from the Apon Foundation and Viewers Like You. Mom, Dad: I love you very much!

Finally, I would like to thank all of my other, many co-authors, without whom I would have never made quite this journey nor learned so much:

Jacob Alperin-Sheriff, David W. Archer, Dan Boneh, Brent Carmer, Chongwon Cho, Seung Geol Choi, Nico Döttling, Karim ElDefrawy, Sanjam Garg, Yan Huang, Leo Fan, Adam Foltzer, Kevin Lawler, Kevin Lewi, Wing-Ning Li, Feng-Hao Liu, Alex J. Malozemoff, Pratyay Mukherjee, Mariana Raykova, Daniel S. Roche, Elaine Shi, Aishwarya Thiruvengadam, Daniel Wagner, and Arkady Yerukhimovich.

Table of Contents

List of Figures

Chapter 1

Introduction

In this dissertation, we study two frontiers in modern cryptography: lattice-based cryptography and cryptographic program obfuscation. In the first direction, lattice cryptography (based on Regev's Learning with Errors and various, related problems) is the primary subarea of *post-quantum* cryptographic research. Much of the interest in lattices originates with Gentry's 2009 breakthrough lattice-based construction of Fully Homomorphic Encryption, which is a type of encryption scheme that permits computation on encrypted data without the need to decrypt first. The development of homomorphic computation techniques over lattices has led to a number of highly expressive cryptosystems that were previously unachievable – though, unfortunately, the resulting schemes tend to be quite inefficient in practice.

Our two contributions to lattice cryptography further the expressiveness of such cryptosystems on the one hand, and address some of the practical deficiencies of lattices on the other hand. Indeed as we describe in more detail in the sequel, in Chapter 3 we construct the first *deniable* Attribute-Based Encryption scheme from LWE (and the first provably quantum-secure such system); that is, an ABE that is resistant to so-called *coercion attacks* after the fact. Further, in Chapter 4 we design the first lattice-based digital signature scheme that simultaneously enjoys short signatures and a totally tight security reduction. These properties are inherently valuable (perhaps, even necessary) in any real-world instantiation

of digital signatures whose security is based on the computational intractability of lattice problems.

In the second direction, we study cryptographic program obfuscation. A program obfuscator is a type of cryptographic software compiler that takes as input a program P and outputs an obfuscated program $\mathcal{O}(P)$. The compiled program $\mathcal{O}(P)$ should compute the same function as P, but the *description of the code* of $\mathcal{O}(P)$ should somehow "provably hide the internal workings of the computation." The notion of program obfuscation is quite strong, theoretically speaking. Indeed, it is fair to say that *indistinguishability obfuscation* (for all polynomial-size circuits) is "Crypto-complete" – in the sense that cryptographers currently believe that essentially any realizable cryptographic system can be achieved given a secure program obfuscator (plus other, simpler tools such as any one-way function).

The current status of obfuscation constructions, and their security, is somewhat complex. In a breakthrough result in 2013, Garg, Gentry, and Halevi gave the first *candidate* construction of cryptographic multilinear maps. Shortly thereafter, Garg, Gentry, Halevi, Raykova, Sahai, and Waters used these multilinear maps to propose the first *candidate* construction of indistinguishability obfuscation. The difference between a regular cryptosystem and a candidate cryptosystem, is that candidate systems do not have a traditional proof of security (rather, they simply *appear* to be reasonable, by community opinion and by lack of concrete attacks against them). Ultimately, this has led to an ongoing process where a scheme is proposed, then an attack is found, then the scheme is patched, and the cycle continues.

That said, our two contributions to program obfuscation are as follows. In Chapter 5, we report on the *first* full-scale, software implementation of an entire program obfuscation toolchain. Our compiler takes as input program code (in a C-like language specialized for cryptographic computation, called Cryptol) and produces an obfuscated version of that code. And finally, in Chapter 6 we give a new, efficient, cryptanalytic attack against a large class of "early" obfuscation candidates. This last work is part of the latest iteration of "attack" in the candidate-attack-patch cycle for obfuscation candidates mentioned above.

In what follows, we briefly introduce each of these four contributions in further detail.

1.1 Overview of Contributions

1.1.1 Chapter 3 – Deniable Encryption

Deniable encryption (Canetti et al. CRYPTO '97) is an intriguing primitive that provides a security guarantee against not only eavesdropping attacks as required by semantic security, but also stronger coercion attacks performed after the fact. The concept of deniability has later demonstrated useful and powerful in many other contexts, such as leakage resilience, adaptive security of protocols, and security against selective opening attacks. Despite its conceptual usefulness, our understanding of how to construct deniable primitives under standard assumptions is restricted.

In particular from standard lattice assumptions, i.e. Learning with Errors (LWE), we have only flexibly and non-negligible advantage deniable public-key encryption schemes, whereas with the much stronger assumption of indistinguishable obfuscation, we can obtain at least fully sender-deniable PKE and computation. How to achieve deniability for other more advanced encryption schemes under standard assumptions remains an interesting open question.

In Chapter 3, we construct a flexibly bi-deniable Attribute-Based Encryption (ABE) scheme for all polynomial-size Branching Programs from LWE. Our techniques involve new ways of manipulating Gaussian noise that may be of independent interest, and lead to a significantly sharper analysis of noise growth in Dual Regev type encryption schemes. We hope these ideas give insight into achieving deniability and related properties for further, advanced cryptographic systems from lattice assumptions.

1.1.2 Chapter 4 – Short Signatures with Tight Security

The Boyen-Li signature scheme (Asiacrypt'16) is a major theoretical breakthrough. Via a clever homomorphic evaluation of a pseudorandom function over their verification key, they achieve a reduction loss in security linear in the underlying security parameter and entirely independent of the number of message queries made, while still maintaining short signatures (consisting of a single short lattice vector). All previous schemes with such an independent reduction loss in security required a linear number of such lattice vectors, and even in the classical world, the only schemes achieving short signatures relied on non-standard assumptions.

In Chapter 4, we improve on their result, providing a verification key smaller by a linear factor, a significantly tighter reduction with only a constant loss, and signing and verification algorithms that could plausibly run in about 1 second. Our main idea is to change the scheme in a manner that allows us to replace the pseudorandom function evaluation with an evaluation of a much more efficient weak pseudorandom function. As a matter of independent interest, we give an improved method of randomized inversion of the $\mathbf{G}$ gadget matrix, which reduces the noise growth rate in homomorphic evaluations performed in a large number of lattice-based cryptographic schemes, without incurring the high cost of sampling discrete Gaussians.

1.1.3 Chapter 5 – 5Gen: A Framework for Prototyping Obfuscation

Secure multilinear maps (mmaps) have been shown to have remarkable applications in cryptography, such as multi-input functional encryption (MIFE) and program obfuscation. To date, there has been little evaluation of the performance of these applications.

In Chapter 5, we initiate a systematic study of mmap-based constructions. We build a general framework, called `5Gen`, to experiment with these applications. At the top layer we develop a compiler that takes in a high-level program and produces an optimized matrix branching program needed for the applications we consider. Next, we optimize and

experiment with several MIFE and obfuscation constructions and evaluate their performance. The `5Gen` framework is modular and can easily accommodate new mmap constructions as well as new MIFE and obfuscation constructions, as well as being an open-source tool that can be used by other research groups to experiment with a variety of mmap-based constructions.

1.1.4 Chapter 6 – Cryptanalysis of Indistinguishability Obfuscators

Annihilation attacks, introduced in the work of Miles, Sahai, and Zhandry (CRYPTO 2016), are a class of polynomial-time attacks against several candidate indistinguishability obfuscation ($i\mathcal{O}$) schemes, built from Garg, Gentry, and Halevi (EUROCRYPT 2013) multilinear maps.

In Chapter 6, we provide a *general* efficiently-testable property for two single-input branching programs, called *partial inequivalence,* which we show is sufficient for our variant of annihilation attacks on several obfuscation constructions based on GGH13 multilinear maps. We also give examples of pairs of natural NC^1 circuits, which – when processed via Barrington's Theorem – yield pairs of branching programs that are partially inequivalent. As a consequence we are also able to show examples of "bootstrapping circuits," used to obtain obfuscations for all circuits (given an obfuscator for NC^1 circuits), in certain settings also yield partially inequivalent branching programs. Prior to our work, no attacks on any obfuscation constructions for these settings were known.

Chapter 2

Technical Background and Preliminaries

In this chapter, we introduce notation and also define the various cryptographic constructions. We remark that Chapter 6 of this work (describing a new cryptanalytic attack) uses more detailed and specialized notation and preliminaries, and so we defer the relevant background for that part of this work until that chapter.

Notations. Let PPT denote probabilistic polynomial time. We use λ to represent the security parameter, where "λ-bit security" means that security should hold up to 2^λ *clock cycles*. We assume that all of our procedures run *efficiently*, or more formally, in PPT with respect to the security parameter λ.

We use bold uppercase letters to denote matrices, and bold lowercase letters to denote vectors, where vectors are by default column vectors throughout the work. For an integer $n > 0$, we use $[n]$ to denote the set of integers $\{1, \ldots, n\}$. We let $|\boldsymbol{t}|$ denote the number of bits in a string or vector $\boldsymbol{t}$. We denote the i-th bit value of a string $\boldsymbol{s}$ by $\boldsymbol{s}[i]$. We use $[\cdot|\cdot]$ to denote the concatenation of vectors or matrices, and $||\cdot||$ to denote the norm of vectors or matrices respectively. We use the ℓ_2 norm for all vectors unless explicitly stated otherwise. We use $\otimes$ to denote the Kronecker product of two matrices.

2.1 Non-Cryptographic Preliminaries

2.1.1 Branching Programs

We recall the definition of branching program in [BV14, GV15]. A width-w branching program BP of length L with input space $\{0,1\}^\ell$ and s states (represented by $[s]$) is a sequence of L tuples of form $(\mathsf{var}(t), \sigma_{t,0}, \sigma_{t,1})$, where

- $\sigma_{t,0}$ and $\sigma_{t,1}$ are injective functions from $[s]$ to itself.
- $\mathsf{var} : [L] \to [\ell]$ is a function that associates the t-th tuple $\sigma_{t,0}, \sigma_{t,1}$ with the input bit $x_{\mathsf{var}(t)}$.

The branching program BP on input $\boldsymbol{x} = (x_1, ..., x_\ell)$ computes its outputs as follows. At step t, we denote the state of the computation by $\eta_t \in [s]$. The initial state is set to be $\eta_0 = 1$. In general, η_t can be computed recursively as

$$\eta_t = \sigma_{t,x_{\mathsf{var}(t)}}(\eta_{t-1})$$

Finally, after L steps, the output of the computation $\mathsf{BP}(\boldsymbol{x}) = 1$ if $\eta_L = 1$ and 0 otherwise. As mentioned in [BV14], we represent states with bits rather than numbers to bound the noise growth. In particular, we represent the state $\eta_t \in [s]$ by a unit vector $\boldsymbol{v}_t \in \{0,1\}^s$. The idea is that $\boldsymbol{v}_t = 1$ if and only if $\sigma_{t,x_{\mathsf{var}(t)}}(\eta_{t-1}) = i$. Note that we can also write the above expression as $\boldsymbol{v}_t[i] = 1$ if and only if either:

- $\boldsymbol{v}_{t-1}[\sigma_{t,0}^{-1}(i)] = 1$ and $x_{\mathsf{var}(t)} = 0$.
- $\boldsymbol{v}_{t-1}[\sigma_{t,1}^{-1}(i)] = 1$ and $x_{\mathsf{var}(t)} = 1$.

This later form will be useful since we can rewrite the above conditions in the following formula. For $t \in [L]$ and $i \in [s]$,

$$\begin{aligned}\boldsymbol{v}_t[i] :&= \boldsymbol{v}[\sigma_{t,0}^{-1}(i)](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\sigma_{t,1}^{-1}(i)] \cdot x_{\mathsf{var}(t)} \\ &= \boldsymbol{v}_{t-1}[\gamma_{t,i,0}](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\gamma_{t,i,1}] \cdot x_{\mathsf{var}(t)}\end{aligned}$$

where we set $\gamma_{t,i,0} = \sigma_{t,i}^{-1}(i)$ and $\gamma_{t,i,1} = \sigma_{t,i}^{-1}(i)$, and $\gamma_{t,i,0}, \gamma_{t,i,1}$ can be publicly computed from the description of the branching program. Hence, $\{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[s]}\}$ is also a valid representation of a branching program BP.

For clarity of representation, we will deal with width-5 permutation branching program, which is shown to be equivalent to the circuit class NC^1 [Bar89]. Hence, we have $s = w = 5$, and the functions σ_0, σ_1 are permutations on $[5]$.

2.1.2 Matrix Branching Programs

A *matrix branching program* (MBP) of length n on length-ℓ, base-d inputs is a collection of variable-dimension matrices $\mathbf{B}_{i,j}$ for $i \in [n]$ and $j \in \{0, \ldots, d-1\}$, a "final matrix" $\mathbf{P}$, and an "input mapper" function $\mathsf{inp} : [\ell] \to [n]$. We require that, for each $i \in [2, n]$ and $j \in \{0, \ldots, d-1\}$, the number of columns of $\mathbf{B}_{i-1,j}$ is equal to the number of rows of $\mathbf{B}_{i,j}$, so that the product of these matrices is well-defined. The evaluation of an MBP on input $x \in \{0, \ldots, d-1\}^\ell$ is defined as

$$\mathsf{MBP}(x) = \begin{cases} 1, & \text{if } \prod_{i=1}^{n} \mathbf{B}_{i,x_{\mathsf{inp}(i)}} = \mathbf{P}, \\ 0, & \text{otherwise.} \end{cases}$$

We note that numerous generalizations and extra properties [BLR+15, SZ14] of MBPs have been explored in the literature– however, we will defer details beyond the above, simplified definition of MBPs until the final chapter on cryptanalysis.

2.2 General Cryptographic Definitions

2.2.1 Digital Signatures

A *signature scheme* for message space $\mathcal{M}$ consists of three algorithms $\Sigma = (\mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$ with details as follows:

- $\mathsf{Setup}(1^\lambda)$: Given security parameter λ, the setup algorithm outputs signing key sk and verification key vk.
- $\mathsf{Sign}(\mathsf{sk}, \mu \in \mathcal{M})$: Given secret key sk and message $\mu \in \mathcal{M}$, the signing algorithm outputs a signature σ for the message.
- $\mathsf{Verify}(\mathsf{vk}, \mu, \sigma)$: Given verification key vk, a message μ and a signature σ, the verification algorithm outputs 1 (accept) or 0 (reject).

Definition 2.2.1 (Correctness)**.** *We say a signature scheme Σ is correct, if for any message $\mu \in \mathcal{M}$ and any $(\mathsf{sk}, \mathsf{vk}) \leftarrow \mathsf{Setup}(1^\lambda)$, we have*

$$\mathbf{Pr}[\mathsf{Verify}(\mathsf{vk}, \mu, \mathsf{Sign}(\mathsf{sk}, \mu)) = 1] = 1$$

We now recall the standard security definitions for digital signature schemes. Existential unforgeability under adaptive chosen-message attack, or eu-acma, is as follows: For any PPT adversary $\mathcal{A}$, we consider the experiment $\mathbf{Expt}^{\mathsf{sig}}_{\mathcal{A}}(1^\lambda)$:

1. **Setup**: A challenger runs the $\mathsf{Setup}(1^\lambda)$ algorithm, and sends the verification key vk to the adversary.
2. **Query Phase**: Proceeding adaptively, the adversary $\mathcal{A}$ queries a sequence of messages $(\mu_1, ..., \mu_m)$. On the i-th query, the challenger runs $\sigma_i \leftarrow \mathsf{Sign}(\mathsf{sk}, \mu_i)$, and sends the result σ_i to $\mathcal{A}$.

3. **Forgery**: Once adversary $\mathcal{A}$ decides that Query Phase is over, it outputs a message/signature pair (μ^*, σ^*), where message μ^* is not queried before.

We define the advantage of adversary $\mathcal{A}$ in attacking an IBE scheme Π as

$$\mathbf{Adv}^{\mathsf{sig}}_{\mathcal{A}}(1^\lambda) = \Pr[\mathsf{Verify}(\mathsf{vk}, \mu^*, \sigma^*) = 1]$$

where the probability is over the randomness of the challenger and adversary.

Definition 2.2.2 (Unforgeability)**.** *We say a signature scheme Σ is exisentially unforgeable under adaptive chosen-message attack (eu-acma secure), if for all* PPT *adversaries $\mathcal{A}$, we have*

$$\mathbf{Adv}^{\mathsf{sig}}_{\mathcal{A}}(1^\lambda) \leq \mathsf{negl}(\lambda)$$

2.2.2 Chameleon Hashing

We give the definition of chameleon hash functions due to Ducas and Micciancio [DM14], (but looking forward) specialized to the lattice setting.

A *chameleon hash function family* is a set of three algorithms $CH = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Hash}^{-1})$ along with an efficiently computable input distribution $\mathcal{X}_n$, $\mathcal{Y}_n$ for each integer n, where $\perp \notin \mathcal{Y}_n$. Except with negligible probability over the choice of $(ek, td) \leftarrow \mathsf{Gen}(1^n)$, the security requirements are:

Uniformity: For a fixed message μ, evaluation key ek, trapdoor td, $(x \leftarrow \mathcal{X}_n, y \leftarrow \mathsf{Hash}_{ek}(\mu, x))$ should be distributed within negligible statistical distance of $(x \leftarrow \mathsf{Hash}^{-1}_{td}(\mu, y), y \leftarrow \mathcal{Y}_n)$

Collision Resistance: Given only access to ek and public parameters, it should be hard for any PPT algorithm $\mathcal{A}$ to output $(\mu, r) \neq (\mu', r')$ such that $\mathsf{Hash}_{ek}(\mu, r) = \mathsf{Hash}_{ek}(\mu', r') \neq \perp$.

We also note that the Ducas and Micciancio paper provides an explicit ring-based construction that is highly efficient, and that this construction has a very straightforward adaptation to general lattices and module lattices [LS15].

2.2.3 Fully Homomorphic Encryption

We recall the definition of (leveled) fully homomorphic encryption in the following. A (leveled) FHE is a tuple of algorithms $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ described as follows:

- $\mathsf{Setup}(1^\lambda, 1^d)$: Given input the security parameter λ and maximum supported depth d, the setup algorithm outputs secret key sk and public key pk.
- $\mathsf{Enc}(\mathsf{pk}, \mu)$: On input pk and a message μ, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Eval}(\mathsf{pk}, \mathcal{C}, (\mathsf{ct}_1, ..., \mathsf{ct}_\ell))$: On input a boolean circuit $\mathcal{C}$ of depth $\leq d$ along with ℓ ciphertexts $(\mathsf{ct}_1, ..., \mathsf{ct}_\ell)$, the evaluation algorithm outputs an evaluated ciphertext ct.
- $\mathsf{Dec}(\mathsf{sk}, \mathsf{ct})$: On input some ciphertext ct and a secret key sk, the decryption algorithm outputs a message μ.

Definition 2.2.3 ((leveled) FHE)**.** *We call a scheme $\Pi = (\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ described above a (leveled) FHE scheme, if it satisfies:*

Correctness: *Let $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$ and $\mathsf{ct}_i \leftarrow \mathsf{Enc}(\mathsf{pk}, \mu_i)$, for $i \in [\ell]$. Let $\mathcal{C}$ be any boolean circuit of depth $\leq d$ and $\mathsf{ct} \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathcal{C}, (\mathsf{ct}_1, ..., \mathsf{ct}_\ell))$. Then we have $\mathsf{Dec}(\mathsf{ct}, \mathsf{sk}) = \mathcal{C}(\mu_1, ..., \mu_\ell)$.*

Semantic security: *For any polynomial $d = d(\lambda)$ and any two messages μ_0, μ_1, the following distributions are computationally indistinguishable*

$$(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, \mu_0)) \approx (\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, \mu_1))$$

where $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^d)$.

Compactness: *The size of the evaluated ciphertext, i.e.* $\mathsf{ct} \leftarrow \mathsf{Eval}(\mathsf{pk}, \mathcal{C}, (\mathsf{ct}_1, ..., \mathsf{ct}_\ell))$*, should be independent of circuit* $\mathcal{C}$ *and* ℓ*, but can depend on* λ *and* d*.*

2.2.4 Multilinear Maps

Boneh and Silverberg [BS02] first proposed the concept of *multilinear maps* (mmaps), but it was only in 2013 that Garg, Gentry, and Halevi [GGH13a] introduced the first plausible construction of an mmap. Since then, mmaps have been shown to be powerful tools in solving numerous problems in cryptography.

A *multilinear map* [BS02, GGH13a] (or *graded encoding scheme*) is a primitive for producing randomized encodings of plaintexts that may be publicly added, multiplied, and zero-tested but otherwise "do not reveal any information." Encodings are associated with a "level" that restricts the types of operations that may be performed on that encoding. More formally, a *degree-*κ *multilinear map* is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Encode}, \mathsf{Add}, \mathsf{Mult}, \mathsf{ZeroTest})$ where:

- Setup takes as input the security parameter, and outputs a private parameter sp and a public parameter pp that, in particular, specifies a ring R.
- Encode takes sp, an element $x \in R$, and a level $S \subseteq [\kappa]$, and outputs a *level-*S *encoding of* x denoted $[\![x]\!]_S$.
- Add takes pp and two encodings $[\![x]\!]_S$, $[\![y]\!]_S$ at the *same level* S, and outputs an encoding $[\![x + y]\!]_S$.
- Mult takes pp and two encodings $[\![x]\!]_{S_1}$, $[\![y]\!]_{S_2}$ for *disjoint* levels S_1, S_2, and outputs an encoding $[\![x \cdot y]\!]_{S_1 \cup S_2}$.
- ZeroTest takes pp and an encoding $[\![x]\!]_U$ for $U = [\kappa]$. It outputs 1 if and only if $x = 0$.

Informally, an mmap is secure if the only information that an attacker can figure out from the encodings of random elements is exactly the information that can be obtained from running Add, Mult, and ZeroTest, and no more. (We defer further, formal definitions to the final chapter on cryptanalysis, since we do not directly rely on them until then.)

2.2.5 Indistinguishability Obfuscation

A uniform PPT machine $i\mathcal{O}$ is called an *indistinguishability obfuscator* [GGH+13b] for a circuit class $\{\mathcal{C}_\lambda\}$ if the following conditions are satisfied:

- For all security parameters $\lambda \in \mathbb{N}$, for all $C \in \mathcal{C}_\lambda$, for all inputs x, we have that

$$\Pr[C'(x) = C(x) : C' \leftarrow i\mathcal{O}(\lambda, C)] = 1$$

- For any (not necessarily uniform) PPT distinguisher D, there exists a negligible function α such that the following holds: For all security parameters $\lambda \in \mathbb{N}$, for all pairs of circuits $C_0, C_1 \in \mathcal{C}_\lambda$, we have that if $C_0(x) = C_1(x)$ for all inputs x, then

$$\left| \Pr\left[D(i\mathcal{O}(\lambda, C_0)) = 1\right] - \Pr\left[D(i\mathcal{O}(\lambda, C_1)) = 1\right] \right| \leq \alpha(\lambda)$$

2.3 Lattice Cryptography Background

2.3.1 Lattices

A full-rank m-dimensional integer lattice $\Lambda \subset \mathbb{Z}^m$ is a discrete additive subgroup whose linear span is $\mathbb{R}^m$. The basis of Λ is a linearly independent set of vectors whose linear combinations are exactly Λ. Every integer lattice is generated as the $\mathbb{Z}$-linear combination of linearly independent vectors $\mathbf{B} = \{\boldsymbol{b}_1, ..., \boldsymbol{b}_m\} \subset \mathbb{Z}^m$. For a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, we define

the "q-ary" integer lattices:

$$\Lambda_q^{\perp} = \{\boldsymbol{e} \in \mathbb{Z}^m | \mathbf{A}\boldsymbol{e} = \mathbf{0} \bmod q\}, \qquad \Lambda_q^{\mathbf{u}} = \{\boldsymbol{e} \in \mathbb{Z}^m | \mathbf{A}\boldsymbol{e} = \boldsymbol{u} \bmod q\}$$

It is obvious that $\Lambda_q^{\boldsymbol{u}}$ is a coset of $\Lambda_q^{\perp}$.

For an integer $q \geq 2$, we use $\mathbb{Z}_q$ to denote the ring of integers modulo q, and somewhat abuse notation by also using $\mathbb{Z}_q$ to explicitly represent the integers in $(-q/2, q/2]$. We define $|x| \in \mathbb{Z}_q$ by taking the absolute value of the representative in this range. In the context of integer lattices in order to aid analysis, we also use the notation $\mathbb{Z}_1$ (implicitly parameterized by some "bit-precision" q) to denote the set of rational representatives of $\mathbb{Z}_q$ in the range $(-1/2, 1/2]$. A conceptually useful view here is that $\mathbb{Z}_1$ is a discrete approximation (depending of q) of the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$.

2.3.2 Discrete Gaussian Distributions

Let Λ be a discrete subset of $\mathbb{Z}^m$. For any vector $\boldsymbol{c} \in \mathbb{R}^m$, and any positive parameter $\sigma \in \mathbb{R}$, let $\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \exp(-\pi||\boldsymbol{x} - \boldsymbol{c}||^2/\sigma^2)$ be the Gaussian function on $\mathbb{R}^m$ with center $\boldsymbol{c}$ and parameter σ. Next, we let $\rho_{\sigma,\boldsymbol{c}}(\Lambda) = \sum_{\boldsymbol{x}\in\Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})$ be the discrete integral of $\rho_{\sigma,\boldsymbol{x}}$ over Λ, and let $\mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{y}) := \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}{\rho_{\sigma,\boldsymbol{c}}(\Lambda)}$. We abbreviate this as $\mathcal{D}_{\Lambda,\sigma}$ when $\boldsymbol{c} = \mathbf{0}$.

Multi-Dimensional Discrete Gaussians. We will also use the generalized multi-dimensional (or m-variate) Discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}_1^m,\mathbf{Q}}$, which denotes sampling a $\mathbb{Z}_1$-valued m-vector with *covariance matrix* $\mathbf{Q} \in \mathbb{Z}_1^{m\times m}$. In order to sample from the distribution $\mathcal{D}_{\mathbb{Z}_1^m,\mathbf{Q}}$, proceed as follows:

- Sample $\boldsymbol{t}' = (t'_1, ..., t'_m) \in \mathbb{R}^m$ independently as $t'_i \leftarrow \mathcal{D}_1$ for $i \in [m]$.
- Find the Cholesky decomposition $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$.
- Output the vector $\boldsymbol{t} := \mathbf{L}\boldsymbol{t}'$ as the sample $\boldsymbol{t} \leftarrow \mathcal{D}_{\mathbb{Z}_1^m,\mathbf{Q}}$.

Recall that the Cholesky decomposition takes as input any positive-definite matrix $\mathbf{Q} \in \mathbb{R}^{m \times m}$ and outputs a lower triangular matrix $\mathbf{L}$ so that $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$. Further, we recall the fact that the sum of two m-variate Gaussians with means μ_1, μ_2 and variances $\mathbf{Q}_1, \mathbf{Q}_2$ is an m-variate Gaussian with mean $\mu_1 + \mu_2$ and variance $\mathbf{Q}_1 + \mathbf{Q}_2$.

Next we show a lemma that is useful when manipulating m-variate Gaussians via the Cholesky decomposition.

Lemma 2.3.1. *Let $\mathbf{I}_{m\times m}$ be the m-by-m identity matrix, $\mathbf{R} \in \mathbb{R}^{m\times m}$, and $\mathbf{Q} \stackrel{\text{def}}{=} a^2\mathbf{I}_{m\times m} - b^2\mathbf{R}^T\mathbf{R}$ for positive numbers a, b such that $a > b||\mathbf{R}^T||$. Then $\mathbf{Q}$ is positive definite.*

Proof. To show that $\mathbf{Q}$ is positive definite, we need to show that for any column vector $\boldsymbol{x}$ of dimension m, we have $\boldsymbol{x}^T \cdot \mathbf{Q} \cdot \boldsymbol{x} > 0$. We prove this by unfolding the matrix $\mathbf{Q}$:

$$\begin{aligned}
\boldsymbol{x}^T \cdot \mathbf{Q} \cdot \boldsymbol{x} &= \boldsymbol{x}^T \cdot (a^2\mathbf{I}_{m\times m} - b^2\mathbf{R}^T\mathbf{R}) \cdot \boldsymbol{x} \\
&= a^2\boldsymbol{x}^T\mathbf{I}_{m\times m}\boldsymbol{x} - b^2\boldsymbol{x}^T\mathbf{R}^T\mathbf{R}\boldsymbol{x} \\
&= a^2||\boldsymbol{x}^T||^2 - b^2||\boldsymbol{x}^T\mathbf{R}^T||^2 \\
&> b^2||\boldsymbol{x}^T||^2 \cdot ||\mathbf{R}^T||^2 - b^2||\boldsymbol{x}^T\mathbf{R}^T||^2.
\end{aligned}$$

Since $||\boldsymbol{x}^T|| \cdot ||\mathbf{R}^T|| \geq ||\boldsymbol{x}^T\mathbf{R}^T||$, we can conclude $\boldsymbol{x}^T \cdot \mathbf{Q} \cdot \boldsymbol{x} > 0$. □

2.3.3 Randomness Extraction and the Matrix Norm

Let S^m denote the set of vectors in $\mathbb{R}^m$ whose length is 1. Then the norm of a matrix $\mathbf{R} \in \mathbb{R}^{m\times m}$ is defined to be $\mathsf{sup}_{\boldsymbol{x}\in S^m}||\mathbf{R}\boldsymbol{x}||$. Then we have the following lemma, which bounds the norm for some specified distributions.

Lemma 2.3.2 ([ABB10])**.** *With respect to the norm defined above, we have the following bounds:*

- *Let $\mathbf{R} \in \{-1, 1\}^{m\times m}$ be chosen at random, then we have $\mathbf{Pr}[||\mathbf{R}|| > 12\sqrt{2m}] < e^{-2m}$.*

- *Let* $\mathbf{R}$ *be sampled from* $\mathcal{D}_{\mathbb{Z}^{m\times m},\sigma}$*, then we have* $\mathbf{Pr}[||\mathbf{R}|| > \sigma\sqrt{m}] < e^{-2m}$.

We will use the following lemma to argue the indistinghishability of two different distributions, which is a generalization of the leftover hash lemma proposed by Dodis et al. [DRS04].

Lemma 2.3.3 ([ABB10])**.** *Suppose that* $m > (n+1)\log q + \omega(\log n)$*. Let* $\mathbf{R} \in \{-1, 1\}^{m\times k}$ *be chosen uniformly at random for some polynomial* $k = k(n)$*. Let* $\mathbf{A}, \mathbf{B}$ *be matrix chosen randomly from* $\mathbb{Z}_q^{n\times m}, \mathbb{Z}_q^{n\times k}$ *respectively. Then, for all vectors* $\boldsymbol{w} \in \mathbb{Z}^m$*, the two following distributions are statistically close:*

$$(\mathbf{A}, \mathbf{AR}, \mathbf{R}^{\mathcal{T}}\boldsymbol{w}) \approx (\mathbf{A}, \mathbf{B}, \mathbf{R}^{\mathcal{T}}\boldsymbol{w})$$

2.3.4 Hard Lattice Problems

Learning With Errors. The LWE problem was introduced by Regev [Reg05], who showed that solving it *on the average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*.

Definition 2.3.4 (LWE)**.** *For an integer* $q = q(n) \geq 2$*, and an error distribution* $\chi = \chi(n)$ *over* $\mathbb{Z}_q$*, the learning with errors problem* $\mathsf{LWE}_{n,m,q,\chi}$ *is to distinguish between the following pairs of distributions:*

$$\{\mathbf{A}, \boldsymbol{b} = \mathbf{A}^T\boldsymbol{s} + \boldsymbol{e}\} \text{ and } \{\mathbf{A}, \boldsymbol{u}\}$$

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

Regev [Reg05] showed that for $q > \sqrt{m}/\beta'$, an efficient algorithm for $\mathsf{LWE}_{n,m,q,\chi}$ for $\chi = \mathcal{D}_{\beta' q}$ $\left(\text{and } \beta' q \geq \sqrt{n}\omega(\log(n))\right)$ implies an efficient quantum algorithm for approximating the SIVP and GapSVP problems, to within $\tilde{O}(n/\beta')$ approximation factors in the worst case. (Improved reductions exist, but we will not need them explicitly in this work.)

Short Integer Solution. The SIS problem was first suggested to be hard on average by Ajtai [Ajt99] and then formalized by Micciancio and Regev [MR04].

Definition 2.3.5 (SIS)**.** *For any $n \in \mathbb{Z}$, and any functions $m = m(n), q = q(n), \beta = \beta(n)$, the average-case Short Integer Solution problem ($\mathsf{SIS}_{q,n,m,\beta}$) is: Given an integer q, a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ chosen uniformly at random and a real $\beta \in \mathbb{R}$, find a non-zero integer vector $\boldsymbol{z} \in \mathbb{Z}^m \setminus \{\mathbf{0}\}$, such that $\mathbf{A}\boldsymbol{z} = 0 \mod q$ and $||\boldsymbol{z}|| \leq \beta$.*

Micciancio and Regev [MR04] showed that solving the average-case $\mathsf{SIS}_{q,n,m,\beta}$ problem for certain parameters is as hard as approximating the Shortest Independent Vector Problem in the worst case to within certain $\gamma = \beta \cdot \tilde{O}(\sqrt{n})$ factors.

For $q \geq \beta\sqrt{n}\omega(\sqrt{\log n})$, it has been shown that solving the SIS problem with non-negligible success probability over the random choice of $\mathbf{A}$ is at least as hard as probabilistically approximating the classic Shortest Independent Vectors Problem (SIVP) on n-dimensional lattices to within $\tilde{O}(\beta\sqrt{n})$ factors in the *worst case*. [Ajt04, MR07, GPV08]. Analogous worst-case reductions exist for the more general case of module lattices, where $\mathbf{A} \in R_q^{d \times m}$ for some arbitrary ring of integers R algebraic number field K [LS15], which essentially includes the above results as a special case.

2.3.5 Trapdoors and Sampling Algorithms

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

Lemma 2.3.6 ([GPV08, AP10])**.** *Let q, n, m be positive integers with $q \geq 2$ and sufficiently large $m = \Omega(n \log q)$. There exists a* PPT *algorithm $\mathsf{TrapGen}(q, n, m)$ that with overwhelming probability outputs a pair $(\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{T}_\mathbf{A} \in \mathbb{Z}^{m \times m})$ such that $\mathbf{A}$ is statistically close to uniform in $\mathbb{Z}_q^{n \times m}$ and $\mathbf{T}_\mathbf{A}$ is a basis for $\Lambda_q^\perp(\mathbf{A})$ satisfying*

$$||\mathbf{T}_\mathbf{A}|| \leq O(n \log q) \quad \textit{and} \quad ||\widetilde{\mathbf{T}_\mathbf{A}}|| \leq O(\sqrt{n \log q})$$

except with $\mathsf{negl}(n)$ probability.

Lemma 2.3.7 ([GPV08, CHKP10, ABB10]). *Let $q > 2, m > n$ and $s > ||\mathbf{T_A}|| \cdot w(\sqrt{\log m + m_1})$. There are two algorithms as follows:*

- *There is an efficient algorithm* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \boldsymbol{u}, s)$*: It takes in* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$*, a short basis* $\mathbf{T_A}$ *for lattice* $\Lambda_q^{\perp}(\mathbf{A})$*, a matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times m_1}$*, a vector* $\boldsymbol{u} \in \mathbb{Z}_q^n$ *and a Gaussian parameter* s*, then outputs a vector* $\mathbf{r} \in \mathbb{Z}_q^{m+m_1}$ *such that* $\boldsymbol{r} \in \Lambda_q^{\boldsymbol{u}}(\mathbf{F})$*, where* $\mathbf{F} := (\mathbf{A}|\mathbf{B})$*, and is statistical close to* $D_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$.

- *There is an efficient algorithm* $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \boldsymbol{u}, s)$*: It takes in* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}, \mathbf{R} \in \mathbb{Z}_q^{m \times n}$*, a matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times n}$*, a short basis* $\mathbf{T_B}$ *for lattice* $\Lambda_q^{\perp}(\mathbf{B})$*, a vector* $\boldsymbol{u} \in \mathbb{Z}_q^n$ *and a Gaussian parameter* s*, then outputs a vector* $\boldsymbol{r} \in \mathbb{Z}_q^{m+n}$ *such that* $\boldsymbol{r} \in \Lambda_q^{\mathbf{u}}(\mathbf{F})$*, where* $\mathbf{F} := (\mathbf{A}|\mathbf{AR} + \mathbf{B})$*, and is statistical close to* $D_{\Lambda_q^{\boldsymbol{u}}(\mathbf{F}),s}$.

- *There is a deterministic polynomial time algorithm* $\mathsf{Invert}(\mathbf{A}, \mathbf{T_A}, \boldsymbol{b})$ *that, given any* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ *with its trapdoor* $\mathbf{T_A} \in \mathbb{Z}_q^{m \times m}$ *such that* $||\mathbf{T}|| \cdot w(\sqrt{\log n}) \leq 1/\beta$ *for some* $\beta > 0$*, and* $\boldsymbol{b} = \boldsymbol{s}^T\mathbf{A} + \boldsymbol{x}$ *for arbitrary* $\boldsymbol{s} \in \mathbb{Z}_q^n$ *and random* $\boldsymbol{x} \leftarrow D_\beta^m$*, outputs* $\boldsymbol{x}$ *with overwhelming probability.*

- *There is a* PPT *algorithm* $\mathsf{SampleD}(\mathbf{T}, \boldsymbol{c}, s)$ *that, given arbitrary* $\boldsymbol{c} \in \mathbb{R}^m$ *and* $r \geq ||\widetilde{T}|| \cdot w(\log n)$*, generates a sample from* $\mathcal{D}_{\Lambda+c,r}$ *(up to* $\mathsf{negl}(n)$ *statistical distance).*

2.3.6 The "Gadget" Matrix G

We recall the gadget matrix $\mathbf{G}$ defined by Micciancio and Peikert [MP12]. We focus on the case that the modulus $q = 2^k$ for ease of analysis.

We define

$$\boldsymbol{g}^t = [1, 2, 2^2, \ldots, 2^{k-1}] \in \mathbb{Z}_q^{1 \times k}$$

Then we have that

$$\mathbf{G} = \mathbf{I}_n \otimes \boldsymbol{g}^t \in \mathbb{Z}_q^{n \times nk}$$

While $\mathbf{G}$ is used by Miccancio and Peikert to sample discrete Gaussians, in this work we only use it for computing a simpler distribution (see Section 4.3 for details).

2.3.7 Subgaussian Random Variables

To analyze our distribution in Section 4.3, we make use of the notion of *subgaussian* random variables. (For further details and full proofs, see [Ver12].) A random vector $\boldsymbol{x}$ is subgaussian with parameter $r > 0$ if for all $t \in \mathbb{R}$ and all (fixed) real unit vectors $\boldsymbol{u}$, its (scaled) moment-generating function satisfies $\mathbb{E}[\exp(\langle \boldsymbol{u}, \boldsymbol{x} \rangle)] \leq \exp(Cr^2t^2)$ for an absolute constant C (for our application, we may take $C = 1$). By a Markov argument, for all $t \geq 0$, we have

$$\Pr[||\boldsymbol{x}|| \geq t] \leq 2\exp(-t^2/r^2). \tag{2.1}$$

Any B-bounded centered random vector $\boldsymbol{x}$ (i.e., $\mathbb{E}[\boldsymbol{x}] = \mathbf{0}$ and $|X| \leq B$ always) is subgaussian with parameter B.

We recall the following additional properties of subgaussian vectors $\boldsymbol{x}$ [Ver12].

Homogeneity: If $\boldsymbol{x}$ is subgaussian with parameter r, then $c \cdot \boldsymbol{x}$ is subgaussian with parameter $c \cdot r$ for any constant $c \geq 0$.

Pythagorean additivity: if $\boldsymbol{x}_1$ is subgaussian with parameter r_1, and $\boldsymbol{x}_2$ is subgaussian with parameter r_2 conditioned on *any* value of $\boldsymbol{x}_1$ (in particular, if $\boldsymbol{x}_2$ is subgaussian with parameter r_2 and independent of $\boldsymbol{x}_1$), then $\boldsymbol{x}_1 + \boldsymbol{x}_2$ is subgaussian with parameter $\sqrt{r_1^2 + r_2^2}$.

Euclidean Norm : Let $\boldsymbol{x} \in \mathbb{R}^n$ be a random vector with independent coordinates that are subgaussian with parameter r. Then for some (small) universal constant $0 < C$, we have $\Pr[||\boldsymbol{x}||_2 > C \cdot r\sqrt{n}] \leq 2^{-\Omega(n)}$

2.3.8 Weak Pseudorandom Functions and Learning with Rounding

Here we give a basic definition of weak pseudorandom functions [DN02].

More formally, a weak pseudorandom function family (outputting a single bit) $\mathsf{W{-}PRF} : \{0,1\}^\lambda \times \{0,1\}^m \to \{0,1\}$ is considered secure if no probabilistic polynomial-time adversary can distinguish a member of the family $f_{\boldsymbol{k}} : \{0,1\}^m \to \{0,1\}, f_{\boldsymbol{k}} := \mathsf{W{-}PRF}(\boldsymbol{k}, \cdot)$ (where $\boldsymbol{k} \leftarrow \{0,1\}^\lambda$ uniformly at random) from a truly random function with advantage greater than $\mathsf{negl}(\lambda)$, given that it can observe

$$(x_1, f_{\boldsymbol{k}}(x_1)), \ldots, (x_m, f_{\boldsymbol{k}}(x_m))$$

for any $m \in \mathsf{poly}(\lambda)$, where each $x_1, \ldots, x_m$ is sampled uniformly at random from $\{0,1\}^m$.

A concrete candidate weak pseudorandom function family is the learning with rounding function family ($\mathsf{LWR}_{n,Q,p}$) [BPR12]. Functions in the family are indexed by a secret key $\boldsymbol{s} \in \mathbb{Z}_Q^n$. For a given secret key $\boldsymbol{s}$, the function is defined as

$$\mathsf{LWR}_{n,Q,p}(\boldsymbol{a}) = \lfloor \tfrac{p}{Q} \langle \boldsymbol{a}, \boldsymbol{s} \rangle \rceil,$$

where $\lfloor \cdot \rceil$ denotes rounding to the nearest integer.

LWR has been shown to be a weak pseudorandom function under the better-known LWE assumption with discrete Gaussian noise terms (and hence on worst-case shortest vector problems on lattices) in a number of different results [BPR12, AKPW13, ASA16, BGM+16, BLL+15]. While all of these results require the ratio Q/p to grow with the number of samples revealed (meaning that hardness for non-a-priori bounded $m = \mathsf{poly}(\lambda)$ requires assuming the shortest vector problem is hard to approximate to within a superpolynomial ratio), there is a reduction [BGM+16] with a sample loss ratio of Q/p in security from LWE with bounded uniform error to LWR. This latter result strongly suggests that LWR remains a weak pseudorandom function for any $Q/p = \Omega(\sqrt{n})$, and that the weaker reductions from

LWE with Gaussian error are likely all artifacts of the proof techniques used.

2.3.9 Puncturable Homomorphic Trapdoor Functions

We recall Alperin-Sheriff's definition of Puncturable Homomorphic Trapdoor Functions (PHTDFs).

$pk \leftarrow \mathsf{Gen}(1^\lambda)$ takes as input a security parameter λ, which for a concrete instantiation implictly defines parameters for a ring $\mathcal{T}$ representing a tag space, a trapdoor space $\mathcal{R}$, a tagged function space $\mathcal{A}$, an index space $\mathcal{X}$, an input space $\mathcal{U}$ and an output space $\mathcal{V}$, and then generates the public key for the PHTDF. $\mathcal{R}$ and $\mathcal{U}$ are associated with parameterized efficiently sampleable distributions $D_{\mathcal{R},\beta}, D_{\mathcal{U},s}$, with the distribution details depending on the instantiation.

$(a, r) \leftarrow \mathsf{GenTrap}(pk, t)$ generates a *trapdoor* $r \leftarrow D_{\mathcal{R}}$ for the (pk, a), with t the tag *associated with* a, r. We need the distribution of a to be statistically close to uniform over $\mathcal{A}$.

$t \leftarrow \mathsf{Tag}(pk, a, r)$ is an auxiliary function which outputs the tag t associated with a and r is a trapdoor for (pk, a).

$f_{pk,a,x} : \mathcal{U} \to \mathcal{V}$ is a deterministic function indexed by $pk, x \in \mathcal{X}, a \in \mathcal{A}$.

$\mathsf{Invert}_{r,pk,a,x,s} : \mathcal{V} \to \mathcal{U}$ is a trapdoor-inverter indexed by $x \in \mathcal{X}$, $r \in \mathcal{R}$ and $pk, a \in \mathcal{A}$. If $f_{pk,a,x}$ is not injective, then Invert is a probabilistic function, and the parameter $s \in \mathbb{R}$ relates to the noise level $\mathsf{Prop}(u)$ of the inverse u output by Invert; in particular, we want $\mathsf{Prop}(u) \leq s$. We require that $\mathsf{Prop}(r) = \beta$ should be small enough to allow Invert to invert with parameter s when the tag t associated with a, r is *invertible* over the ring $\mathcal{T}$. If t is not invertible, then the trapdoor is considered *punctured*, and Invert outputs $\perp$.

$r^* \leftarrow \mathsf{Eval}^{td}_{pk}(g, \{(a_i, r_i)\}_{i\in[\kappa]}, \boldsymbol{y})$, $a^* \leftarrow \mathsf{Eval}^{func}_{pk}(g, \{a_i\}_{i\in[\kappa]}, \boldsymbol{y})$ are deterministic trapdoor/function homomorphic evaluation algorithms, respectively. The algorithms take as input some function $g : \mathcal{T}^\kappa \times \mathcal{T}^w \to \mathcal{T}$, a vector $\boldsymbol{y} \in \mathcal{T}^w$, as well as functions $a_i \in \mathcal{A}$ with associated trapdoors $r_i \in \mathcal{R}$. The outputs are $r^* \in \mathcal{R}$ and $a^* \in \mathcal{A}$.

Correctness. Let $pk \leftarrow \mathsf{Gen}(1^\lambda), \hat{T} = \{\hat{t}_i \in \mathcal{T}\}_{i\in[\kappa]}, T = \{t_j \in \mathcal{T}\}_{j\in[\ell]}, (r_i, a_i) \leftarrow \mathsf{GenTrap}(pk, \hat{t}_i)$. Let $g : \mathcal{T}^\kappa \times \mathcal{T}^\ell \to \mathcal{T}$ and let $t^* := g(\hat{T}, T)$. We require that for all

$$r^* \leftarrow \mathsf{Eval}^{td}_{pk}(g, \{(a_i, r_i)\}_{i\in[\kappa]}, T), \qquad a^* \leftarrow \mathsf{Eval}^{func}_{pk}(g, \{(a_i)\}_{i\in[\kappa]}, T),$$

we have that r^* is (indeed) a trapdoor for (pk, a^*), and that t^* is the tag associated with a^*.

This can be relaxed to a leveled notion similar to a leveled FHE; we omit the details.

Definition 2.3.8. *We call a function g* admissible with parameter *s on the set of tags $\hat{T} := \{t_i\}_{i\in[\kappa]}$, if whenever the initial trapdoors r_i have noise levels $\beta_i \le \beta_{init} = \omega(\sqrt{\log n})$, r^* will have noise level $\beta^* \le s/\omega(\sqrt{\log n})$ with overwhelming probability.*

Security Properties. The following should hold for $pk \leftarrow \mathsf{Gen}(1^\lambda)$, trapdoor and function pair (r, a) with an invertible tag t:

$$(pk, r, a, x, u, v) \overset{s}{\approx} (pk, r, a, x, u', v')$$

where $x \in \mathcal{X}$ is arbitrary, $u \leftarrow D_{\mathcal{U},s}$, $v := f_{pk,a,x}(u)$, $v' \leftarrow \mathcal{V}$ and $u' \leftarrow \mathsf{Invert}_{r,pk,a,x,s}(v')$.

The security game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ is parameterized by a security parameter λ, as well as a function $g : \mathcal{T}^\kappa \times \mathcal{T}^w \to \mathcal{T}$ such that g is admissible with some parameter s on some subset of tags $\mathcal{S} \subseteq \mathcal{T}^\kappa$

1. $\mathcal{C}$ runs $pk \leftarrow \mathsf{Gen}(1^\lambda)$ and then computes $(a_i, r_i) \leftarrow \mathsf{GenTrap}(pk, \boldsymbol{t})$ for each $i \in [\kappa]$. $\mathcal{A}$ is given pk and $\{a_i\}$.

2. $\mathcal{A}$ may make (a polynomial number of) inversion queries, sending some $v \in \mathcal{V}$, $x \in \mathcal{X}$ and some $\boldsymbol{y} \in \mathcal{T}^w$ such that $g(\boldsymbol{s}, \boldsymbol{y})$ is *invertible*. Then $\mathcal{C}$ computes $r' \leftarrow \mathsf{Eval}^{td}_{pk}(g, \{(a_i, r_i)\}, \boldsymbol{y})$ as well as $a' \leftarrow \mathsf{Eval}^{func}_{pk}(g, \{a_i\}, \boldsymbol{y})$, samples $u \leftarrow \mathsf{Invert}_{r', pk, a', x, s}(v)$, and returns u to $\mathcal{A}$.

3. $\mathcal{A}(1^\lambda)$ outputs tag sets $\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)} \in \mathcal{T}^w$ which satisfy $g(\boldsymbol{t}, \boldsymbol{y}) = g(\boldsymbol{t}, \boldsymbol{y}') = 0$, as well as $u^{(1)}, u^{(2)}, x^{(1)}) \neq x^{(2)}$, and wins if

$$f_{pk, a^{(1)}, x^{(1)}}(u^{(1)}) = f_{pk, a^{(2)}, x^{(2)}}(u^{(2)}),$$

where $\mathsf{Prop}(u^{(1)}), \mathsf{Prop}(u^{(2)}), \mathsf{Prop}(x^{(1)}), \mathsf{Prop}(x^{(2)}) \leq s$ and $a^{(b)} \leftarrow \mathsf{Eval}^{func}_{pk}(g, \{(a_i)\}, \boldsymbol{y}^{(b)})$ for $b \in \{1, 2\}$.

We say the PHTDF satisfies $(\epsilon = \epsilon(\lambda), t = t(\lambda), g, \mathcal{S})$-collision resistance when punctured (CRP) security if every PPT adversary taking at most time t has success probability at most ϵ in this game.

Concrete Instantiation. The instantiation in [Alp15] is written in terms of general lattices, but as mentioned in that work, can easily be instantiated over rings or modules [LS15]. We briefly recall the relevant results on security, leaving tag instantiation descriptions to later in the work.

Theorem 2.3.9 ([Alp15]). *Let g be admissible with parameter s. If there exists an adversary $\mathcal{A}$ breaking $CRP_{\epsilon,t,g,\mathcal{S}}$ security of the PHTDF, then there exists $\mathcal{A}'$ running in time t that solves $\mathcal{SIS}_{n,q,\beta}$ with advantage $\epsilon - \mathsf{negl}(\lambda)$ for $\beta = O(s^2\sqrt{n \log q})$.*

We will also need to recall the growth rate of the $\mathsf{Prop}(r)$ for the trapdoors used in this instantiation as a result of homomorphic operations. In particular, we have

Homomorphic Addition of a_1 and a_2 with trapdoors r_1, r_2 induces a new trapdoor r^* with

$\mathsf{Prop}(r^*) = \mathsf{Prop}(r_1) + \mathsf{Prop}(r_2)$

Homomorphic Multiplication of a_1, a_2 with trapdoor r_1, r_2, tags t_1, t_2 induces trapdoor r^* with

$$\mathsf{Prop}(r^*) = \mathsf{Prop}(r_1)\mathbf{G}^{-1}(a_2) + \mathsf{Prop}(t_1)\mathsf{Prop}(r_2)$$

A key trick with homomorphic multiplication is to chain them together in a left-associative manner, causing a quasi-additive growth in the trapdoors [BV14, ASP14].

Chapter 3

Deniable Attribute Based Encryption for Branching Programs from LWE

3.1 Introduction

Deniable encryption, introduced by Canetti et al. [CDNO97] at CRYPTO 1997, is an intriguing primitive that allows Alice to privately communicate with Bob in a way that resists not only eavesdropping attacks as required by semantic security, but also stronger *coercion attacks* performed after the fact. An eavesdropper Eve stages a cocercion attack by additionally approaching Alice (or Bob, or both) *after* a ciphertext is transmitted and demanding to see all secret information: the plaintext, the random coins used by Alice for encryption, and any private keys held by Bob (or Alice) related to the ciphertext. In particular, Eve can use this information to "fully unroll" the *exact transcript* of some deterministic decryption procedure purportedly computed by Bob, as well as verify that the exact coins and decrypted plaintext in fact produce the coerced ciphertext. A secure deniable encryption scheme should maintain privacy of the sensitive data originally communicated between Alice and Bob under the coerced ciphertext (instead substituting a benign yet *convincing* plaintext in the view of Eve), even in the face of such a revealing attack and even if Alice

and Bob may not interact during the coercion phase.

Historically, deniable encryption schemes have been challenging to construct. Under standard assumptions, Canetti et al. [CDNO97] constructed a sender-deniable[1] PKE where the distinguishing advantage between real and fake openings is an inverse polynomial depending on the public key size. But it was not until 2011 that O'Neill, Peikert, and Waters [OPW11] proposed the first constructions of bi-deniable PKE with *negligible* deniability distinguishing advantage: from simulatable PKE generically, as well as from Learning with Errors (LWE [Reg05]) directly.

Concurrently, Bendlin et al. [BNNO11] showed an inherent limitation: any non-interactive public-key encryption scheme may be receiver-deniable (resp. bi-deniable) only with *non-negligible* $\Omega(1/\mathsf{size}(\mathsf{pk}))$ distinguishing advantage in the deniability experiment. Indeed, O'Neill et al. [OPW11] bypass the impossibility result of [BNNO11] by working in the so-called *flexible*[2] model of deniability. In the flexible of deniability, private keys sk are distributed by a central key authority. In the event that Bob is coerced to reveal a key sk that decrypts chosen ciphertext ct^*, the key authority distributes a *faking key* fk to Bob, which Bob can use to generate a fake key sk^* (designed to behave identically to sk except on ciphertext ct^*). If this step is allowed, then O'Neill et al. demonstrate that for their constructions, Eve has at most negligible advantage in distinguishing whether Bob revealed an honest sk or fake sk^*.

A major breakthrough in deniable encryption arrived with the work of Sahai and Waters [SW14], who proposed the first sender-deniable PKE with negligible distinguishing advantage from indistinguishability obfuscation ($i\mathcal{O}$) for P/poly [GGH+13b]. The concept of deniability has been demonstrated useful in the contexts of leakage resilience [DLZ15], adaptive security for protocols, and as well as deniable computation (or algorithms) [CGP15,

[1] We differentiate between sender-, receiver-, and bi-deniable schemes. A bi-deniable scheme is both sender- and receiver-deniable.

[2] We borrow the name "flexible" from Boneh, Lewi, and Wu [BLW15] as the original term "multi-distributional" of O'Neill et al. [OPW11] is used to define a slightly different security property in the recent work by De Caro et al. [CIO16] than we achieve here.

DKR15, GP15]. In addition to coercion resistance, a bi-deniable encryption scheme is a non-committing encryption scheme [CFGN96], as well as a scheme secure under selective opening (SOA) attacks [BHY09], which are of independent theoretical interest.

Very recently, De Caro, Iovino, and O'Neill [CIO16] gave various constructions of deniable *functional* encryption. First, they show a generic transformation of any IND-secure FE scheme for circuits into a flexibly receiver-deniable FE for circuits. Second, they give a direct construction of receiver-deniable FE for Boolean formulae from bilinear maps. Further, in the stronger *multi-distributional* model of deniable functional encryption – where there are special "deniable" set-up and encryption algorithms in addition to the plain ones, and where under coercion, it may non-interactively be made to seem as only the normal algorithms were used – De Caro et al. [CIO16] construct receiver-deniable FE for circuits under the additional (powerful) assumption of different-inputs obfuscation ($di\mathcal{O}$).

De Caro et al. [CIO16] also show (loosely speaking) that any receiver-deniable FE implies SIM-secure FE for the same functionality. Following [CIO16], we also emphasize that deniability for functional encryption is a **strictly stronger** property than SIM security, since fixed coerced ciphertexts must decrypt correctly and benignly *in the real world*. Finally, we mention that in concurrent work, Apon, Fan, and Liu, in an unpublished work [AFL15], construct flexibly bi-deniable inner product encryption from standard *lattice* assumptions. This work generalizes and thus subsumes the prior results of [AFL15].

Despite the apparent theoretical utility in understanding the extent to which cryptographic constructions are deniable, our current knowledge of constructing such schemes from standard lattice assumptions is still limited. From LWE, we have only flexible and non-negligible advantage deniable encryption schemes (or IPE from [AFL15]), whereas with the much more powerful assumption of indistinguishability obfuscation ($i\mathcal{O}$), we can obtain at least fully-secure sender-deniable PKE and computation [CGP15, DKR15, GP15], or as mentioned above even a multi-distributional receiver-deniable FE for all circuits from the even stronger assumption of $di\mathcal{O}$.

3.1.1 Our Contributions

In this work, we further narrow this gap by investigating a richer primitive – attribute-based encryption (ABE) [GPSW06, BGG+14, GV15] – *without* the use of obfuscation as a black box primitive. We hope that the techniques developed in this work can further shed light on deniability for even richer schemes such as functional encryption [BSW11, GGH+13b, BGG+14, GVW15a] under standard assumptions.

- Our main contribution is the construction of a flexibly bi-deniable ABE for poly-sized branching programs (which can compute $\mathsf{NC1}$ via Barrington's theorem [Bar89]) from the standard Learning with Errors assumption [Reg05].

Theorem 3.1.1. *Under the standard* LWE *assumption, there is a flexibly bi-deniable attribute-based encryption scheme for all poly-size branching programs.*

Recall that in an attribute-based encryption (ABE) scheme for a family of functions $\mathcal{F} : \mathcal{X} \to \mathcal{Y}$, every secret key sk_f is associated with a predicate $f \in \mathcal{F}$, and every ciphertext $\mathsf{ct}_{\boldsymbol{x}}$ is associated with an attribute $\boldsymbol{x} \in \mathcal{X}$. A ciphertext $\mathsf{ct}_{\boldsymbol{x}}$ can be decrypted by a given secret key sk_f to its payload message m only when $f(x) = 0 \in \mathcal{Y}$. Informally, the typical security notion for an ABE scheme is *collusion resistance*, which means no collection of keys can provide information on a ciphertext's message, if the individual keys are not authorized to decrypt the ciphertext in the first place. Intuitively, a bi-deniable ABE must provide both collusion and coercion resistance.

Other contributions of this work can be summarized as:

- A new form of the Extended Learning with Errors (eLWE) assumption [OPW11, AP12, BLP+13], which is convenient in the context of Dual Regev type ABE/FE schemes that apply the Leftover Hash Lemma [DRS04] in their security proofs.

- An explicit, tightened noise growth analysis for lattice-based ABE for branching programs. Prior work used the loose l_∞ norm to give a rough upper bound, which is

technically insufficient to achieve deniability using our proof techniques. (We require matching upper *and lower* bounds on post-evaluation noise sizes.)

The eLWE assumption above is roughly the standard LWE assumption, but where the distinguisher also receives "hints" on the LWE sample's noise vector $\boldsymbol{e}$ in the form of inner products, i.e. distributions $\left\{\mathbf{A}, \boldsymbol{b} = \mathbf{A}^T\boldsymbol{s} + \boldsymbol{e}, \boldsymbol{z}, \langle \boldsymbol{z}, \boldsymbol{e}\rangle\right\}$ where (intuitively) $\boldsymbol{z}$ is a decryption key in the real system (which are denoted $\boldsymbol{r}$ elsewhere). Our contribution here is a new reduction from the standard LWE assumption to our correlated variant of extended-LWE, eLWE$^+$, where the adversary requests arbitrary correlations (expressed as a matrix $\mathbf{R}$) between the hints, in the case of a prime poly-size modulus with noise-less hints. We show this by extending the LWE to eLWE reduction of Alperin-Sheriff and Peikert [AP12] to our setting.

3.1.2 Our Approach

At a high level, our work begins with the ABE for branching programs of Gorbunov and Vinayagamurthy [GV15]. We will augment the basic ABE-BP = (Setup, Keygen, Enc, Dec) with an additional suite of algorithms (DenSetup, DenEnc, SendFake, RecFake) to form our flexibly bi-deniable ABE-BP. Doing so requires careful attention to the setting of parameters, as we explain in the sequel.

We remark now that – due to reasons related to the delicateness of our parameter setting – the ABE scheme of [GV15] is *particularly suited* to being made bi-deniable, as compared to similar schemes such as the ABE for arithmetic circuits of Boneh et al. [BGG$^+$14]. We will explain this in what follows as well.

Intuition for Our New Deniability Mechanism. As in the work of O'Neill et al. [OPW11], our approach to bi-deniability relies primarily on a curious property of Dual Regev type [GPV08] secret keys: by correctness of any such scheme, each key $\boldsymbol{r}$ is guaranteed to behave as intended for some $1 - \mathsf{negl}(n)$ fraction of the possible random coins used to encrypt, but

system parameters may be set so that each key is also guaranteed to be *faulty* (i.e. fail to decrypt) on some $\mathsf{negl}(n)$ fraction of the possible encryption randomness. More concretely, each secret key vector $\boldsymbol{r}$ in lattice-based schemes is sampled from an m-dimensional Gaussian distribution, as is the error term $\boldsymbol{e}$ (for LWE public key $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$). For every fixed $\boldsymbol{r}$, with overwhelming probability over the choice of $\boldsymbol{e}$, the vectors $\boldsymbol{r}, \boldsymbol{e} \in \mathbb{Z}_q^m$ will point in highly uncorrelated directions in m-space. However, if the vector $\boldsymbol{r}$ and $\boldsymbol{e}$ happen to point in similar directions, the error magnitude will be (loosely) *squared* during decryption.

Our scheme is based around the idea that a receiver, coerced on honest key-ciphertext pair $(\boldsymbol{r}, \mathsf{ct}^*)$, can use the key authority's *faking key* fk to learn the precise error vector $\boldsymbol{e}^*$ used to construct ct^*. Given $\boldsymbol{e}^*, \boldsymbol{r}$, and fk, the receiver re-samples a fresh secret key $\boldsymbol{r}^*$ that is functionally-equivalent to the honest key $\boldsymbol{r}$, except that $\boldsymbol{r}^*$ is strongly correlated with the vector $\boldsymbol{e}^*$ in ct^*. When the coercer then attempts to decrypt the challenge ciphertext ct^* using $\boldsymbol{r}^*$, the magnitude of decryption error will artificially grow and cause the decryption to output the value we want to deny to. Yet, when the coercer attempts to decrypt any other independently-sampled ciphertext ct, decryption will succeed with overwhelming probability under $\boldsymbol{r}^*$ if it would have under $\boldsymbol{r}$.

We emphasize that to properly show coercion resistance (when extending this intuition to the case of Dual Regev ABE instead of Dual Regev PKE), this behavior of $\boldsymbol{r}^*$ should hold *even when* ct *and* ct^* *embed the same attribute* $\boldsymbol{x}$. (Indeed, the majority of our effort is devoted to ensuring this simple geometric intuition allows a valid instantiation of the denying algorithms $(\mathsf{DenSetup}, \mathsf{DenEnc}, \mathsf{SendFake}, \mathsf{RecFake})$ without "damaging" the basic operation of $(\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ in the underlying ABE scheme.)

Then, given the ability to "artificially blow-up" the decryption procedure of a specific key on a ciphertext-by-ciphertext basis, we can employ an idea originally due to Canetti et al. [CDNO97] of *translucent sets*, but generalized to the setting of ABE instead of PKE, to construct our new, flexibly bi-deniable ABE-BP scheme out of the framework provided by the "plain" SIM-secure ABE-BP scheme of [GV15].

Highlights of the Gorbunov-Vinayagamurthy Scheme. In the ABE for (width 5) branching programs of [GV15], bits a are "LWE-encoded" by the vector

$$\psi_{\mathbf{A},\boldsymbol{s},a} = \boldsymbol{s}^T(\mathbf{A} + a \cdot \mathbf{G}) + \boldsymbol{e} \in \mathbb{Z}_q^m$$

where $\mathbf{G}$ is the gadget matrix [MP12].

The ciphertext ct encrypting message μ under BP-input $\boldsymbol{x}$ is given by

$$\mathsf{ct} = (\psi_0, \psi^c, \{\psi_i\}_{i\in[\ell]}, \{\psi_{0,i}\}_{i\in[5]}, c),$$

and is composed of a Dual Regev ct-pair of vectors (ψ_0, c) encrypting the ciphertext's message μ, an encoding ψ^c representing the (freshly randomized) encoding of the constant 1, five encodings $\{\psi_{0,i}\}_{i\in[5]}$ representing a (freshly randomized) encoding of the initial state of a width-5, length-ℓ branching program BP, and ℓ encodings $\{\psi_i\}_{i\in[\ell]}$ – one for each step of the branching program's evaluation, storing a constant-sized *permutation matrix* associated with the i-th level of BP. Note that each "LWE encoding" ψ is performed under a distinct public key matrix $\mathbf{A}$, $\mathbf{A}^c$, $\{\mathbf{A}_i\}$, or $\{\mathbf{A}_{0,i}\}$ respectively.

The (key-homomorphic) evaluation procedure takes as input a ciphertext sequence $\mathsf{ct} = (\psi_0, \psi^c, \{\psi_i\}, \{\psi_{0,i}\}, c)$ and the public key $\mathsf{pk} = (\mathbf{A}, \mathbf{A}^c, \{\mathbf{A}_i\}, \{\mathbf{A}_{0,i}\})$, as well as the cleartext branching program description BP and the BP-input $\boldsymbol{x}$. It produces the evaluated public key $\mathbf{V}_{\mathsf{BP}}$ and the evaluated encoding $\psi_{\mathsf{BP}(\boldsymbol{x})}$. Given a short secret key *vector* $\boldsymbol{r} \in \mathbb{Z}^{2m}$ matching (some public coset $\boldsymbol{u}$ of) the lattice generated by $[\mathbf{A}|\mathbf{V}_{\mathsf{BP}}] \in \mathbb{Z}^{n\times 2m}$, the encoding *vector* $\psi_{\mathsf{BP}(\boldsymbol{x})}$ (whose Dual Regev encoding-components (ψ_0, c) also match coset $\boldsymbol{u}$) can be decrypted to the message μ if and only if $\mathsf{BP}(\boldsymbol{x}) = \mathsf{accept} = \mathbf{0}$.

On the Necessity of Exact Noise Control. In order to push the intuition for our deniability mechanism through for an ABE of the above form, we must overcome a number of technical hurdles.

The major challenge is an implicit technical requirement to *very tightly control* the precise noise magnitude of evaluated ciphertexts. In previous functional (and homomorphic) encryption schemes from lattices, the emphasis is placed on upper bounding evaluated noise terms, to ensure that they do not grow too large and cause decryption to fail. Moreover, security (typically) holds for any ciphertext noise level at or above the starting ciphertexts' noises. In short, noise growth during evaluation is nearly always undesirable.

As with previous schemes, we too must upper bound the noise growth of evaluated ciphertexts in order to ensure basic correctness of our ABE. But unlike previous schemes, we must take the step of also (carefully) *lower bounding* the noise growth during the branching program evaluation (which technically motivates deviating from the l_∞ norm of prior analyses). This is due to the fact, highlighted above, that producing directional alignment between a key and error term can at most *square* the noise present during decryption. Since coercion resistance requires that it must always be possible to deny any ciphertext originally intended for any honest key, it must be that, with overwhelming probability, every honest key and every honest ciphertext produce evaluated error that is no less than the square root of the maximum noise threshold tolerated.

In a little more detail – as we will later demonstrate in Section 3.3 – in dimension m there is precisely an expected $\mathsf{poly}(m)$ *gap* in magnitude between the inner products of **(i)** two relatively *orthogonal* key/error vectors $\boldsymbol{r}, \boldsymbol{e}_{\mathsf{BP}(\boldsymbol{x})}$, and **(ii)** two highly *correlated* key/error vectors $\boldsymbol{r}^*, \boldsymbol{e}_{\mathsf{BP}(\boldsymbol{x})}$. The ability to deny is based around our ability to design $\boldsymbol{r}^*$ that are statistically indistinguishable from $\boldsymbol{r}$ in the attacker's view, but where $\boldsymbol{r}^*$ "punctures out" decryptions of ciphertexts with error vectors pointing in the *direction* of $\boldsymbol{e}_{\mathsf{BP}(\boldsymbol{x})}$ in m-space (error-vector directions are unique to each honest ct with overwhelming probability).

Crucially, this approach **generically forces** the use of a *polynomial-sized modulus* q in the scheme.[3] In particular, when error vectors $\boldsymbol{e}$ may (potentially) grow to be some

[3] One consequence of a poly-size modulus requirement is that the fully key-homomorphic scheme of Boneh et al. [BGG+14], taken verbatim, can only be denied for up to $\mathsf{NC0}$ functions using our approach. Past this, attempts to produce fake keys in an identical manner to this work may be detected by a statistical test under coercion.

superpolynomial magnitude in the dimension m of the public/secret keys, we totally lose any efficiently testable notion of "error vector orientation in m-space" for the purposes of Dual Regev type decryption.

Further, in order to "correctly trace and distinguish" different orientations throughout the computation of an arbitrary branching program BP, we are required to make careful use of *multi-dimensional Gaussian* distributions. These are sampled using covariance matrices $\mathbf{Q} \in \mathbb{Z}^{m \times m}$ that allow us to succinctly describe the underlying, geometric *randomized rotation* action on error vector *orientations* in m-space with each arithmetic operation of the BP evaluation in the overall ABE-BP scheme. (We use the geometrically-inspired term "rotation matrix" to describe our low-norm matrices $\mathbf{R}$ for this reason.)

An additional subtlety in our new noise analysis is that we require the individual multiplications of the ct evaluation procedure to have *independently sampled* error vectors in each operand-encoding – and thus be "independently oriented" – in order for the overall analysis to go through correctly. (While there could in principle be some way around this technical obstacle in the analysis, we were unable to find one.) This appears to a priori exclude a straightforward denying procedure for *all circuits* [BGG+14], where a gate's input wires' preceding sub-circuits may have cross-wires between them. But it naturally permits denying *branching program computations*, where at the i-th time-step, an i-th *independently generated* ct-component is merged into an accumulated BP state, as with [GV15].

Finally, we mention that an inherent limitation in the techniques of Apon et al. [AFL15], used to construct (the weaker notion of) flexibly bi-deniable inner product encryption from LWE, is bypassed in the current work at the cost of supporting only BP computations of an a-priori bounded length ℓ. Namely, it was the case in [AFL15] that the *length* ℓ of the attribute vector $\boldsymbol{w}$ had to be "traded off" against the dimension m of the public/secret keys. We suppress the details, other than to point out that this issue can be resolved by artificially boosting the magnitude of the low-norm matrices used to generate error terms in fresh ciphertexts from $\{-1, 1\}$ up to $\{-\Theta(m\ell), \Theta(m\ell)\}$-valued matrices. This, of course,

requires knowing the length ℓ of the branching program up front. (Intuitively, this technical change as compared to [AFL15] allows for a sharp *inductive lower bound* on the *minimum* noise growth across all possible function-input pairs that might be evaluated in a given instance of our bi-deniable ABE-BP scheme.)

3.1.3 Future Directions

The next, most natural question is whether bi-deniable functional encryption can be built out of similar techniques (from only LWE), perhaps by leveraging our bi-deniable ABE for $\mathsf{NC1}$ computations as a building block. We briefly sketch one possible approach and the obstacles encountered. Recall that Goldwasser et al. [GKP+13] show to transform the combination of **(i)** any ABE for a circuit family $\mathcal{C}$, **(ii)** fully homomorphic encryption, and **(iii)** a randomized encoding scheme (such as Yao's garbled circuits) into a 1-key (resp. *bounded collusion*) SIM-secure functional encryption scheme for $\mathcal{C}$.

If we instantiate the Goldwasser et al. transformation with our deniable ABE, we get a functional encryption scheme for $\mathsf{NC1}$. We can then boost functional encryption for shallow circuits to functional encryption for all circuits using the "trojan method" of Ananth et al. [ABSV15]. As it turns out, it is easy to directly prove *flexible receiver-deniability* of the final scheme, independently of but matching the generic results of De Caro et al. [CIO16] for receiver-deniable FE.

Unfortunately, we do not know how to prove (even, flexible) *sender-deniability* of this final scheme. Roughly speaking, the problem is that each ciphertext's attribute in such a scheme contains an FHE ciphertext $\mathsf{ct}_{\mathsf{FHE}}$ for its attribute, and this attribute leaks to the attacker (resp. cocercer) on decryptions that succeed. In particular, there is nothing stopping the coercer from demanding that the sender also provide randomness r_S that *opens the attribute's FHE ciphertext*.

We speculate that a possible way around this obstacle would be to use an *adaptively-secure* homomorphic encryption scheme for $\mathsf{NC1}$ computations. Note that adaptively-

secure FHE is known to be impossible for circuits with $\omega(\log(n))$ depth due to a counting argument lower bound by Katz, Thiruvengadam, and Zhou [KTZ13], but this leaves open the possibility of an $\mathsf{NC1}$-homomorphic encryption scheme with the necessary properties to re-obtain (flexible) sender deniability for lattice-based FE. We leave this as an intriguing open problem for future work.

3.2 New Definitions and Tools

In this section, we first describe our new notion of flexibly bi-deniable ABE, which is a natural generalization of the flexibly bi-deniable PKE of [OPW11]. Then we define the notion of a flexibly attribute-based bi-translucent set (AB-BTS), which generalizes the idea of bi-translucent set (BTS) in the work [OPW11]. Using a similar argument as in the work [OPW11], we can show that an AB-BTS suffices to construct bi-deniable ABE. In the last part of this section, we define a new assumption called Extended LWE Plus, and show its hardness by giving a reduction from the standard LWE problem.

3.2.1 Flexibly Bi-Deniable ABE: Syntax and Deniability Definition

A flexibly bi-deniable key-policy attribute based encryption for a class of Boolean circuits $\mathcal{C} : \{0,1\}^{\ell} \to \{0,1\}$ consists a tuple of PPT algorithms

$$\Pi = (\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{DenSetup}, \mathsf{DenEnc}, \mathsf{SendFake}, \mathsf{RecFake}).$$

We describe them in detail as follows:

$\mathsf{Setup}(1^{\lambda})$: On input the security parameter λ, the setup algorithm outputs public parameters pp and master secret key msk.

$\mathsf{Keygen}(\mathsf{msk}, f)$: On input the master secret key msk and a function $f \in \mathcal{C}$, it outputs a secret key sk_f.

$\mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}, \mu; r_S)$: On input the public parameter pp, an attribute/message pair $(\boldsymbol{x}, \mu)$ and randomness r_S, it outputs a ciphertext $c_{\boldsymbol{x}}$.

$\mathsf{Dec}(\mathsf{sk}_f, c_{\boldsymbol{x}})$: On input the secret key sk_f and a ciphertext $c_{\boldsymbol{x}}$, it outputs the corresponding plaintext μ if $f(\boldsymbol{x}) = 0$; otherwise, it outputs $\perp$.

$\mathsf{DenSetup}(1^\lambda)$: On input the security parameter λ, the deniable setup algorithm outputs pubic parameters pp, master secret key msk and faking key fk.

$\mathsf{DenEnc}(\mathsf{pp}, \boldsymbol{x}, \mu; r_S)$: On input the public parameter pp, an attribute/message pair $(\boldsymbol{x}, \mu)$ and randomness r_S, it outputs a ciphertext $c_{\boldsymbol{x}}$.

$\mathsf{SendFake}(\mathsf{pp}, r_S, \mu, \mu')$: On input public parameters pp, original random coins r_S, message μ of DenEnc and desired message μ', it outputs a faked random coin r'_S.

$\mathsf{RecFake}(\mathsf{pp}, \mathsf{fk}, c_{\boldsymbol{x}}, f, \mu')$: On input public parameters pp, faking key fk, a ciphertext $c_{\boldsymbol{x}}$, a function $f \in \mathcal{C}$, and desired message μ', the receiver faking algorithm outputs a faked secret key sk'_f.

Correctness. We say the flexibly bi-deniable ABE scheme described above is correct, if for any $(\mathsf{msk}, \mathsf{pp}) \leftarrow \mathsf{S}(1^\lambda)$, where $\mathsf{S} \in \{\mathsf{Setup}, \mathsf{DenSetup}\}$, any message μ, function $f \in \mathcal{C}$, and any attribute vector $\boldsymbol{x}$ where $f(\boldsymbol{x}) = 0$, we have $\mathsf{Dec}(\mathsf{sk}_f, c_{\boldsymbol{x}}) = \mu$, where $\mathsf{sk}_f \leftarrow \mathsf{Keygen}(\mathsf{msk}, f)$ and $c_{\boldsymbol{x}} \leftarrow \mathsf{E}(\mathsf{pp}, \boldsymbol{x}, \mu; r_S)$ where $\mathsf{E} \in (\mathsf{Enc}, \mathsf{DenEnc})$.

Bi-deniability definition. Let μ, μ' be two arbitrary messages, not necessarily different. We propose the bi-deniability definition by describing real experiment $\mathbf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mu,\mu'}(1^\lambda)$ and faking experiment $\mathbf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mu,\mu'}(1^\lambda)$ regarding adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ below:

1. $(\boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
2. $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$
3. $c'_{\boldsymbol{x}^*} \leftarrow \mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}^*, \mu; r_S)$
4. $(f^*, \mathsf{st}_2) \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{pp}, \mathsf{st}_1, c_{\boldsymbol{x}^*})$
5. $\mathsf{sk}_{f^*} \leftarrow \mathsf{Keygen}(\mathsf{msk}, f^*)$
6. $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{sk}_{f^*}, c, \mathsf{st}_2, r_S)$
7. Output $b \in \{0, 1\}$

(a) $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{Real}}(1^\lambda)$

1. $(\boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
2. $(\mathsf{pp}, \mathsf{msk}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$
3. $c'_{\boldsymbol{x}^*} \leftarrow \mathsf{DenEnc}(\mathsf{pp}, \boldsymbol{x}^*, \mu'; r_S)$
4. $(f^*, \mathsf{st}_2) \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{pp}, \mathsf{st}_1, c'_{\boldsymbol{x}^*})$
5. $r'_S \leftarrow \mathsf{SendFake}(\mathsf{pp}, \mu, \mu', r_S)$
6. $\mathsf{sk}_{f^*} \leftarrow \mathsf{RecFake}(\mathsf{pp}, \mathsf{fk}, c'_{\boldsymbol{x}}, \boldsymbol{v}^*, \mu')$
7. $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{sk}_{f^*}, c, \mathsf{st}_2, r'_S)$
8. Output $b \in \{0, 1\}$

(b) $\mathbf{Expt}_{\mathcal{A}}^{\mathsf{Fake}}(1^\lambda)$

Figure 3.2.1: Security experiments for bi-deniable ABE

where $\mathsf{KG}(\mathsf{msk}, \boldsymbol{w}^*, \cdot)$ returns a secret key $\mathsf{sk}_{\boldsymbol{v}} \leftarrow \mathsf{Keygen}(\mathsf{msk}, \boldsymbol{v})$ if $\langle \boldsymbol{v}, \boldsymbol{w}^* \rangle \neq 0$ and $\perp$ otherwise.

Definition 3.2.1 (Flexibly Bi-Deniable ABE)**.** *An ABE scheme Π is bi-deniable if for any two messages μ, μ', any probabilistic polynomial-time adversaries $\mathcal{A}$ where $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$, there is a negligible function $\mathsf{negl}(\lambda)$ such that*

$$\mathbf{Adv}^{\Pi}_{\mathcal{A},\mu,\mu'}(1^\lambda) = |\mathbf{Pr}[\mathbf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mu,\mu'}(1^\lambda) = 1] - \mathbf{Pr}[\mathbf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mu,\mu'}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

3.2.2 Attribute Based Bitranslucent Set Scheme

In this section, we define the notion of a *Attribute Based Bitranslucent Set* (AB-BTS), which is an extension of bitranslucent sets (BTS) as defined by O'Neill et al. in [OPW11]. Our new notion permits a more fine-grained degree of access control, where pseudorandom samples and secret keys are associated with attributes $\boldsymbol{x}$, and the testing algorithm can successfully distinguish a pseudorandom sample from a truly random one if and only if the attribute of the sample is accepted under a given secret key's policy f – i.e. when $f(\boldsymbol{x}) = \mathbf{0}$. This concept is reminiscent of *attribute-based encryption* (ABE), and in fact, we will show in the sequel how to construct a flexibly bi-deniable ABE from an AB-BTS. This is analogous to

the construction of a flexibly bi-deniable PKE from O'Neill et al.'s BTS. We present the formal definition below.

Let $\mathcal{F}$ be some family of functions. An attribute based bitranslucent set (AB-BTS) scheme for $\mathcal{F}$ consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$: On input the security parameter, the normal setup algorithm outputs a public parameter pp and master secret key msk.

$\mathsf{DenSetup}(1^\lambda)$: On input the security parameter, the deniable setup algorithm outputs a public parameter pp, master secret key msk and faking key fk.

$\mathsf{Keygen}(\mathsf{msk}, f)$: On input the master secret key msk and a function $f \in \mathcal{F}$, the key generation algorithm outputs a secret key sk_f.

P- and U-samplers $\mathsf{SampleP}(\mathsf{pp}, \boldsymbol{x}; r_S)$ and $\mathsf{SampleU}(\mathsf{pp}, \boldsymbol{x}; r_S)$ output some $\boldsymbol{c}$.

$\mathsf{TestP}(\mathsf{sk}_f, \boldsymbol{c}_{\boldsymbol{x}})$: On input a secret key sk_f and a ciphertext $\boldsymbol{c}_{\boldsymbol{x}}$, the P-tester algorithm outputs 1 (accepts) or 0 (rejects).

$\mathsf{FakeSCoins}(\mathsf{pp}, r_S)$: On input a public parameters pp and randomness r_S, the sender-faker algorithm outputs randomness r_S^*.

$\mathsf{FakeRCoins}(\mathsf{pp}, \mathsf{fk}, \boldsymbol{c}_{\boldsymbol{x}}, f)$: On input a public parameters pp, the faking key fk, a ciphertext $\boldsymbol{c}_{\boldsymbol{x}}$ and a function $f \in \mathcal{F}$, the receiver-faker algorithm outputs a faked secret key sk'_f.

Definition 3.2.2 (AB-BTS)**.** *We say a scheme* $\Pi = (\mathsf{Setup}, \mathsf{DenSetup}, \mathsf{Keygen}, \mathsf{SampleP}, \mathsf{SampleU}, \mathsf{TestP}, \mathsf{FakeSCoins}, \mathsf{FakeRCoins})$ *is an AB-BTS scheme for a function family* $\mathcal{F}$ *if it satisfies:*

1. *(Correctness.) The following experiments accept or respectively reject with overwhelming probability over the randomness.*

- *Let* $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $f \in \mathcal{F}$, $\mathsf{sk}_f \leftarrow \mathsf{Keygen}(\mathsf{msk}, f)$. *If* $f(\boldsymbol{x}) = 0$ *and* $\boldsymbol{c}_{\boldsymbol{x}} \leftarrow \mathsf{SampleP}(\mathsf{pp}, \boldsymbol{x}; r_S)$, *then* $\mathsf{TestP}(\mathsf{sk}_f, \boldsymbol{c}_{\boldsymbol{x}}) = 1$*; otherwise,* $\mathsf{TestP}(\mathsf{sk}_f, \boldsymbol{c}_{\boldsymbol{x}}) = 0$.
- *Let* $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$, $f \in \mathcal{F}$, $\mathsf{sk}_f \leftarrow \mathsf{Keygen}(\mathsf{msk}, f)$, $\boldsymbol{c} \leftarrow \mathsf{SampleU}(\mathsf{pp}; r_S)$. *Then* $\mathsf{TestP}(\mathsf{sk}_f, \boldsymbol{c}) = 0$.

2. *(Indistinguishable public parameters.) The public parameters* pp *generated by the two setup algorithms* $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda)$ *and* $(\mathsf{pp}, \mathsf{msk}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$ *should be indistinguishable.*

3. *(Selective bi-deniability.) Let* $\mathcal{F}$ *be a family of functions. We define the following two experiments: the real experiment* $\mathbf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mathcal{F}}(1^\lambda)$ *and the faking experiment* $\mathbf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mathcal{F}}(1^\lambda)$ *regarding an adversary* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$ *below:*

   where $\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)$ *returns a secret key* $\mathsf{sk}_f \leftarrow \mathsf{Keygen}(\mathsf{msk}, f)$ *if* $f \in \mathcal{F}$ *and* $f(\boldsymbol{x}^*) \neq 0$*; it returns* $\perp$ *otherwise. We also require that* $f^* \in \mathcal{F}$.

(a)
(a) $(f^*, \boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
(b) $(\mathsf{pp}, \mathsf{msk}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$
(c) $\boldsymbol{c} \leftarrow \mathsf{SampleU}(\mathsf{pp}; r_S)$
(d) $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{pp}, \mathsf{st}_1, \boldsymbol{c})$
(e) $\mathsf{sk}_{f^*} \leftarrow \mathsf{Keygen}(\mathsf{msk}, f^*)$
(f) $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{sk}_{f^*}, c, \mathsf{st}_2, r_S)$
(g) Output $b \in \{0, 1\}$

(a) $\mathbf{Expt}^{\mathsf{Real}}_{\mathcal{A}}(1^\lambda)$

(b)
(a) $(f^*, \boldsymbol{x}^*, \mathsf{st}_1) \leftarrow \mathcal{A}_1(\lambda)$
(b) $(\mathsf{pp}, \mathsf{msk}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$
(c) $\boldsymbol{c} \leftarrow \mathsf{SampleP}(\mathsf{pp}, \boldsymbol{x}^*; r_S)$
(d) $\mathsf{st}_2 \leftarrow \mathcal{A}_2^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{pp}, \mathsf{st}_1, \boldsymbol{c})$
(e) $r'_S \leftarrow \mathsf{FakeSCoins}(\mathsf{pp}, r_S)$
(f) $\mathsf{sk}_{f^*} \leftarrow \mathsf{FakeRCoins}(\mathsf{pp}, \mathsf{fk}, \boldsymbol{c}, f^*)$
(g) $b \leftarrow \mathcal{A}_3^{\mathsf{KG}(\mathsf{msk}, \boldsymbol{x}^*, \cdot)}(\mathsf{sk}_{f^*}, c, \mathsf{st}_2, r'_S)$
(h) Output $b \in \{0, 1\}$

(b) $\mathbf{Expt}^{\mathsf{Fake}}_{\mathcal{A}}(1^\lambda)$

Figure 3.2.2: Security experiments for AB-BTS

We say the scheme is selectively bi-deniable for $\mathcal{F}$*, if for any probabilistic polynomial-time adversaries* $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2, \mathcal{A}_3)$*, there is a negligible function* $\mathsf{negl}(\lambda)$ *such that*

$$\mathbf{Adv}^{\Pi}_{\mathcal{A}}(1^\lambda) = |\mathbf{Pr}[\mathbf{Expt}^{\mathsf{Real}}_{\mathcal{A},\mathcal{F}}(1^\lambda) = 1] - \mathbf{Pr}[\mathbf{Expt}^{\mathsf{Fake}}_{\mathcal{A},\mathcal{F}}(1^\lambda) = 1]| \leq \mathsf{negl}(\lambda)$$

Remark 3.2.3. *Correctness for the faking algorithms is implied by the bi-deniability property. In particular, with overwhelming probability over the overall randomness, the following holds: let* $(\mathsf{pp}, \mathsf{msk}, \mathsf{fk}) \leftarrow \mathsf{DenSetup}(1^\lambda)$*,* $f \in \mathcal{F}$*,* $\mathsf{sk}_f \leftarrow \mathsf{Keygen}(\mathsf{msk}, f)$*,* x *be a string and* $\boldsymbol{c_x} \leftarrow \mathsf{SampleP}(\mathsf{pp}, x; r_S)$*, then*

- $\mathsf{SampleU}(\mathsf{pp}; \mathsf{FakeSCoins}(\mathsf{pp}, r_S)) = \boldsymbol{c_x}$*,*
- $\mathsf{TestP}(\mathsf{FakeRCoins}(\mathsf{pp}, \mathsf{fk}, \boldsymbol{c_x}, f), \boldsymbol{c_x}) = 0$
- *For any other* x'*, let* $\boldsymbol{c}' \leftarrow \mathsf{SampleP}(\mathsf{pp}, x'; r'_S)$*, then (with overwhelming probability) we have*

$$\mathsf{TestP}\left(\mathsf{FakeRCoins}(\mathsf{pp}, \mathsf{fk}, \boldsymbol{c_x}, f), \boldsymbol{c}'\right) = \mathsf{TestP}(\mathsf{sk}_f, c').$$

It is not hard to see that if one of these does not hold, then one can easily distinguish the real experiment from the faking experiment.

Remark 3.2.4. *Canetti et al. [CDNO97] gave a simple encoding technique to construct a sender-deniable encryption scheme from a translucent set. O'Neill, Peikert, and Waters [OPW11] used a similar method to construct a flexibly bi-deniable encryption from a bi-translucent set scheme. Here we further observe that the same method as well allows us to construct a flexibly bi-deniable ABE scheme from bi-deniable AB-BTS. We present the construction in Section 3.3.4.*

3.2.3 Extended LWE and Our New Variant

O'Neill et al. [OPW11] introduced the Extended LWE problem, which allows a "hint" on the error vector $\boldsymbol{x}$ to leak in form of a noisy inner product. They observe a trivial "blurring" argument shows that LWE reduces to eLWE when the hint-noise βq is superpolynomially larger than the magnitude of samples from χ, and also allows for unboundedly many *independent* hint vectors $\langle \boldsymbol{z}, \boldsymbol{x}_i \rangle$ while retaining LWE-hardness.

Definition 3.2.5 (Extended LWE)**.** *For an integer $q = q(n) \geq 2$, and an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the extended learning with errors problem $\mathsf{eLWE}_{n,m,q,\chi,\beta}$ is to distinguish between the following pairs of distributions:*

$$\{\mathbf{A}, \boldsymbol{b} = \mathbf{A}^T\boldsymbol{s} + \boldsymbol{e}, \boldsymbol{z}, \langle \boldsymbol{z}, \boldsymbol{b} - \boldsymbol{e}\rangle + e'\} \text{ and } \{\mathbf{A}, \boldsymbol{u}, \boldsymbol{z}, \langle \boldsymbol{z}, \boldsymbol{u} - \boldsymbol{x}\rangle + e'\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$, $\boldsymbol{e}, \boldsymbol{z} \xleftarrow{\$} \chi^m$ and $e' \xleftarrow{\$} \mathcal{D}_{\beta q}$.

Further, Alperin-Sheriff and Peikert [AP12] show that LWE reduces to eLWE with a polynomial modulus and no hint-noise (i.e. $\beta = 0$), even in the case of a bounded number of *independent* hints.

We introduce the following new form of extended-LWE, called eLWE^+, which considers leaking a pair of *correlated hints* on the same noise vector. Our security proof of the AB-BTS construction relies on this new assumption.

Definition 3.2.6 (Extended LWE Plus)**.** *For integer $q = q(n) \geq 2, m = m(n)$, an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, and a matrix $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$, the extended learning with errors problem $\mathsf{eLWE}^+_{n,m,q,\chi,\beta,\mathbf{R}}$ is to distinguish between the following pairs of distributions:*

$$\{\mathbf{A}, \boldsymbol{b} = \mathbf{A}^T\boldsymbol{s} + \boldsymbol{e}, \boldsymbol{z}_0, \boldsymbol{z}_1, \langle \boldsymbol{z}_0, \boldsymbol{b} - \boldsymbol{e}\rangle + e, \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{b} - \boldsymbol{e}\rangle + e'\} \text{ and}$$

$$\{\mathbf{A}, \boldsymbol{u}, \boldsymbol{z}_0, \boldsymbol{z}_1, \langle \boldsymbol{z}_0, \boldsymbol{u} - \boldsymbol{e}\rangle + e, \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{u} - \boldsymbol{e}\rangle + e'\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$, $\boldsymbol{e}, \boldsymbol{z}_0, \boldsymbol{z}_1 \xleftarrow{\$} \chi^m$ and $e, e' \xleftarrow{\$} \mathcal{D}_{\beta q}$.

Hardness of extended-LWE$^+$. A simple observation, following prior work, is that when χ is $\mathsf{poly}(n)$-bounded and the hint noise βq (and thus, modulus q) is superpolynomial in n, then $\mathsf{LWE}_{n,m,q,\chi}$ trivially reduces to $\mathsf{eLWE}^+_{n,m,q,\chi,\beta,\mathbf{R}}$ for every $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$ so that $\mathbf{R}\mathbf{z}_1$ has $\mathsf{poly}(n)$-bounded norm. This is because, for any $r = \omega(\sqrt{\log n}), c \in \mathbb{Z}$, the statistical distance between $\mathcal{D}_{\mathbb{Z},r}$ and $c + \mathcal{D}_{\mathbb{Z},r}$ is at most $O(|c|/r)$.

However, our cryptosystem will require a polynomial-size modulus q. So, we next consider the case of *prime* modulus q of $\mathsf{poly}(n)$ size and no noise on the hints (i.e. $\beta = 0$). Following [AP12][4], it will be convenient to swap to the "knapsack" form of LWE, which is: given $\mathbf{H} \leftarrow \mathbb{Z}_q^{(m-n)\times m}$ and $\boldsymbol{c} \in \mathbb{Z}_q^{m-n}$, where either $\mathbf{c} = \mathbf{H}\boldsymbol{e}$ for $\boldsymbol{e} \leftarrow \chi^m$ or $\mathbf{c}$ uniformly random and independent of $\mathbf{H}$, determine which is the case (with non-negligible advantage). The "extended-plus" form of the knapsack problem also reveals a pair of hints $(\boldsymbol{z}_0, \boldsymbol{z}_1, \langle \boldsymbol{z}_0, \boldsymbol{e}\rangle, \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}\rangle)$. Note the equivalence between LWE and knapsack-LWE is proven in [MM11] for $m \geq n + \omega(\log n)$.

Theorem 3.2.7. *For $m \geq n + \omega(\log n)$, for every prime $q = \mathsf{poly}(n)$, for every $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$, and for every $\beta \geq 0$, $\mathbf{Adv}_{\mathcal{B}^{\mathcal{A}}}^{\mathsf{LWE}_{n,m,q,\chi}}(1^\lambda) \geq (1/q^2)\mathbf{Adv}_{\mathcal{A}}^{\mathsf{eLWE}^+_{n,m,q,\chi,\beta,\mathbf{R}}}(1^\lambda)$.*

Proof. We construct an LWE to eLWE^+ reduction $\mathcal{B}$ as follows. $\mathcal{B}$ receives a knapsack-LWE instance $\mathbf{H} \in \mathbb{Z}_q^{(m-n)\times m}, \boldsymbol{c} \in \mathbb{Z}_q^{m-n}$. It samples $\boldsymbol{e}', \boldsymbol{z}_0, \boldsymbol{z}_1 \leftarrow \chi^m$ and uniform $\boldsymbol{v}_0, \boldsymbol{v}_1 \leftarrow \mathbb{Z}_q^{m-n}$. It chooses any $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$, then sets

$$\mathbf{H}' := \mathbf{H} - \boldsymbol{v}_0\boldsymbol{z}_0^T - \boldsymbol{v}_1\left(\mathbf{R}\boldsymbol{z}_1\right)^T \in \mathbb{Z}_q^{(m-n)\times m},$$
$$\boldsymbol{c}' := \boldsymbol{c} - \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e}'\rangle - \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}'\rangle \in \mathbb{Z}_q^{m-n}.$$

It sends $(\mathbf{H}', \boldsymbol{c}', \mathbf{z}_0, \mathbf{z}_1, \langle \boldsymbol{z}_0, \boldsymbol{e}'\rangle, \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}'\rangle)$ to the knapsack-eLWE^+ adversary $\mathcal{A}$, and outputs what $\mathcal{A}$ outputs.

Notice that when $\mathbf{H}, \boldsymbol{c}$ are independent and uniform, so are $\mathbf{H}', \mathbf{c}'$, in which case $\mathcal{B}$'s simulation is perfect.

Now, consider the case when $\mathbf{H}, \mathbf{c}$ are drawn from the knapsack-LWE distribution, with $\mathbf{c} = \mathbf{H}\boldsymbol{x}$ for $\boldsymbol{e} \leftarrow \chi^m$. In this case, $\mathbf{H}'$ is uniformly random over the choice of $\mathbf{H}$, and we

[4] We note that a higher quality reduction from LWE to eLWE is given in [BLP+13] in the case of binary secret keys. However for our cryptosystem, it will be more convenient to have secret key coordinates in $\mathbb{Z}_q$, so we extend the reduction of [AP12] to eLWE^+ instead.

have

$$\begin{aligned}\boldsymbol{c}' &= \mathbf{H}\boldsymbol{x} - \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e}' \rangle - \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}' \rangle \\ &= \left(\mathbf{H}' + \boldsymbol{v}_0 \boldsymbol{z}_0^T + \boldsymbol{v}_1 \left(\mathbf{R}\boldsymbol{z}_1\right)^T\right) \boldsymbol{e} - \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e}' \rangle - \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}' \rangle \\ &= \mathbf{H}'\boldsymbol{e} + \boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e} - \boldsymbol{e}' \rangle + \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} - \boldsymbol{e}' \rangle.\end{aligned}$$

Define the event $E = [E_0 \wedge E_1]$ as

$$\begin{aligned}E_0 &\stackrel{\text{def}}{=} \left[\langle \boldsymbol{z}_0, \boldsymbol{e} \rangle = \langle \boldsymbol{z}_0, \boldsymbol{e}' \rangle\right], \\ E_1 &\stackrel{\text{def}}{=} \left[\langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} \rangle = \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}' \rangle\right].\end{aligned}$$

If event E occurs, then the reduction $\mathcal{B}$ perfectly simulates a pseudorandom instance of knapsack-eLWE$^+$ to $\mathcal{A}$, as then $\boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e} - \boldsymbol{e}' \rangle + \boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} - \boldsymbol{e}' \rangle$ vanishes, leaving $\boldsymbol{c}' = \mathbf{H}'\boldsymbol{e}$ for $\mathbf{H}' \leftarrow \mathbb{Z}_q^{(m-n)\times m}$ and $\boldsymbol{e} \leftarrow \chi^m$ as required. Otherwise since q is prime, the reduction $\mathcal{B}$ (incorrectly) simulates an independent and uniform instance of knapsack-eLWE$^+$ to $\mathcal{A}$, as then either one of $\boldsymbol{v}_0 \cdot \langle \boldsymbol{z}_0, \boldsymbol{e} - \boldsymbol{e}' \rangle$ or $\boldsymbol{v}_1 \cdot \langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e} - \boldsymbol{e}' \rangle$ does not vanish, implying that $\boldsymbol{c}'$ is uniform in $\mathbb{Z}_q^{m-n}$ over the choice of $\boldsymbol{v}_0$ (resp. $\boldsymbol{v}_1$) alone, independent of the choices of $\mathbf{H}'$ and $\boldsymbol{x}$.

It remains to analyze the probability that event E occurs. Because $\boldsymbol{e}$ and $\boldsymbol{e}'$ are i.i.d., we may define the random variable $\mathcal{Z}_0$ that takes values $\langle \boldsymbol{z}_0, \boldsymbol{e}^* \rangle \in \mathbb{Z}_q$ and the random variable $\mathcal{Z}_1$ that takes values $\langle \mathbf{R}\boldsymbol{z}_1, \boldsymbol{e}^* \rangle \in \mathbb{Z}_q$ jointly over choice of $\boldsymbol{e}^* \leftarrow \chi^m$, and analyze their collision probabilities independently. Since the collision probability of *any* random variable $\mathcal{Z}$ is at least $1/|\mathsf{Supp}(\mathcal{Z})|$, we have that $\Pr[E] \geq \min CP[\mathcal{Z}_0] \cdot \min CP[\mathcal{Z}_1] = 1/q^2 = 1/\mathsf{poly}(n)$, and the theorem follows. $\square$

3.3 Flexibly Bi-Deniable Attribute-Based Encryption (ABE) for Branching Programs

In this section, we present our flexibly bi-deniable ABE for bounded-length Branching Program. We organize our approach into the following three steps: **(1)** first, we recall the encoding scheme proposed in the SIM-secure ABE-BP of [GV15]; **(2)** Then, we present our flexibly bi-deniable attribute bi-translucent set (AB-BTS) scheme, as was defined in Definition 3.2.2. Our AB-BTS construction uses the ideas of Gorbunov and Vinayagamurthy [GV15], with essential modifications that allow us to tightly upper and lower bound evaluated noise terms. As discussed in the Introduction, this tighter analysis plays a key role in proving bi-deniability. **(3)** Finally, we show how to obtain the desired bi-deniable ABE scheme from our AB-BTS. As pointed out by Canetti et al. [CDNO97] and O'Neill et al. [OPW11], a bitranslucent set scheme implies flexibly bi-deniable PKE. We observe that the same idea generalizes to the case of an AB-BTS scheme and flexibly bi-deniable ABE in a straightforward manner.

3.3.1 Encoding Schemes for Branching Programs

Basic Homomorphic Encoding. Before proceeding to the public key evaluation algorithm, we first described basic homomorphic addition and multiplication over public keys and encoded ciphertexts based on the techniques in [GSW13, AP14, BGG+14].

Definition 3.3.1 (LWE Encoding)**.** *For any matrix* $\mathbf{A} \leftarrow \mathbb{Z}_q^{n\times m}$*, we define an LWE encoding of a bit* $a \in \{0,1\}$ *with respect to a public key* $\mathbf{A}$ *and randomness* $\boldsymbol{s} \leftarrow \mathbb{Z}_q^n$ *as*

$$\psi_{\mathbf{A},\boldsymbol{s},a} = \boldsymbol{s}^T(\mathbf{A} + a \cdot \mathbf{G}) + \boldsymbol{e} \in \mathbb{Z}_q^m$$

for error vector $\boldsymbol{e} \leftarrow \chi^m$ *and the gadget matrix* $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$.

In our construction, all LWE encodings will be encoded using the same LWE secret $\boldsymbol{s}$,

thus for simplicity, we will simply refer to such an encoding as $\psi_{\mathbf{A},a}$.

For homomorphic addition, the addition algorithm takes as input two encodings $\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a}$, and outputs the sum of them. Let $\mathbf{A}^+ = \mathbf{A} + \mathbf{A}'$ and $a^+ = a + a'$

$$\mathsf{Add}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'}) = \psi_{\mathbf{A},a} + \psi_{\mathbf{A}',a'} = \psi_{\mathbf{A}^+,a^+}$$

For homomorphic multiplication, the multiplication algorithm takes as input two encodings $\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a}$, and outputs an encoding $\psi_{\mathbf{A}^\times,a^\times}$, where $\mathbf{A}^\times = -\mathbf{A}\mathbf{G}^{-1}(\mathbf{A}')$ and $a^\times = aa'$.

$$\mathsf{Mult}(\psi_{\mathbf{A},a}, \psi_{\mathbf{A}',a'}) = -\psi \cdot \mathbf{G}^{-1}(\mathbf{A}') + a \cdot \psi' = \psi_{\mathbf{A}^\times,a^\times}$$

Public Key Evaluation Algorithm. Following the notation in [GV15], we define a public evaluation algorithm $\mathsf{Eval}_{\mathsf{pk}}$. The algorithm takes as input a description of the branching program BP, a collection of public keys $\{\mathbf{A}_i\}_{i\in[\ell]}$ (one for each attribute bit x_i), a collection of public keys $\mathbf{V}_{0,i}$ for initial state vector and an auxiliary matrix $\mathbf{A}^c$, and outputs an evaluated public key corresponding to the branching program BP.

$$\mathbf{V}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$$

where the auxiliary matrix $\mathbf{A}^c$ are used to encoded constant 1 for each input wire. We also define matrix $\mathbf{A}'_i = \mathbf{A}^c - \mathbf{A}_i$ as a public key used to encode $1 - x_i$. By the definition of branching programs, the output $\mathbf{V}_{\mathsf{BP}} \in \mathbb{Z}_q^{n\times m}$ is the homomorphically generated public key $\mathbf{V}_{L,1}$ at position 1 of the state vector for the L-th step of the branching program evaluation.

Recall that in the definition of branching programs, BP is represented by the tuple $\{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[5]}\}$ for $t \in [L]$, and the initial state vector is set to be $\boldsymbol{v}_0 = (1, 0, 0, 0, 0)$. Further, for $t \in [L]$, the computation is performed as $\boldsymbol{v}_t[i] = \boldsymbol{v}_{t-1}[\gamma_{t,i,0}](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\gamma_{t,i,1}] \cdot x_{\mathsf{var}(t)}$. It is important for the security proof (among other reasons) that the evaluated state vector in each step is independent of the attribute vector.

Encoding Evaluation Algorithm. We define an encoding evaluation algorithm $\mathsf{Eval}_{\mathsf{ct}}$ that takes as input the description of a branching program BP, an attribute vector $\boldsymbol{x}$, a set of encodings for the attribute $\{\mathbf{A}_i, \psi_i := \psi_{\mathbf{A}_i, x_i}\}_{i\in[\ell]}$, encodings of the initial state vector $\{\mathbf{V}_{0,i}, \psi_{0,i} := \psi_{\mathbf{V}_{0,i}\boldsymbol{v}_0[i]}\}_{i\in[5]}$ and an encoding of a constant 1, i.e. $\psi^c := \psi_{\mathbf{A}^c,1}$. The algorithm $\mathsf{Eval}_{\mathsf{ct}}$ outputs an encoding of the result $y := \mathsf{BP}(\boldsymbol{x})$ with respect to the homomorphically derived public key $\mathbf{V}_{\mathsf{BP}} := \mathbf{V}_{L,1}$

$$\psi_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \psi_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \psi_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c, \psi^c\})$$

As mentioned above, in branching program computation, for $t \in [L]$, we have for all $i \in [5]$

$$\boldsymbol{v}_t[i] = \boldsymbol{v}_{t-1}[\gamma_{t,i,0}](1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_{t-1}[\gamma_{t,i,1}] \cdot x_{\mathsf{var}(t)}$$

The evaluation algorithm proceeds inductively to update the encoding of the state vector for each step of the branching program. Next, we need to instantiate this inductive computation using the homomorphic operations described above, i.e. $\mathsf{Add}, \mathsf{Mult}$. Following the notation used in [GV15], we define $\psi'_i := \psi_{\mathbf{A}'_i,(1-x_i)} = \boldsymbol{s}^T(\mathbf{A}'_i + (1-x_i)\mathbf{G}) + \boldsymbol{e}'_i$, where $\mathbf{A}'_i = \mathbf{A}^c - \mathbf{A}_i$, to denote the encoding of $1 - x_i$. This encoding can be computed using $\mathsf{Add}(\psi_{\mathbf{A}^c_i,1}, -\psi_{\mathbf{A}_i,x_i})$. Then assuming at time $t-1 \in [L]$ we hold encodings of the state vector $\{\psi_{\mathbf{V}_{t-1},i,\boldsymbol{v}_{t-1}[i]}\}_{i\in[5]}$. For $i \in [5]$, we compute the encodings of new state values as

$$\psi_{i,t} = \mathsf{Add}(\mathsf{Mult}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Mult}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1}))$$

where $\gamma_0 := \gamma_{t,i,0}$ and $\gamma_1 := \gamma_{t,i,1}$. We omit the correctness proof of the encoding here, which is presented in [GV15].

Simulated Public Key Evaluation Algorithm. The simulation strategy was first developed in [BGG+14], and then adapted by Gorbunov and Vinayagamurthy [GV15] in branching program scenario. In particular, set $\mathbf{A}_i = \mathbf{A}_i\mathbf{R}_i - x_i\mathbf{G}$ for some shared public key

matrix $\mathbf{A}$ and low norm matrix $\mathbf{R}_i$. Similarly, the state public keys $\mathbf{A}_{t,i} = \mathbf{A}\mathbf{R}_{t,i} - \boldsymbol{v}_t[i]\mathbf{G}$, and matrices $\mathbf{A}^c = \mathbf{A}\mathbf{R}^c - \mathbf{G}$. The evaluation algorithm $\mathsf{Eval}_{\mathsf{Sim}}$ takes as input the description of branching program BP, the attribute vector $\boldsymbol{x}$, collections of low norm matrices $\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c$ corresponding to input public key, initial state vector and complement matrices respectively, and a shared matrix $\mathbf{A}$. It outputs a homomorphically derived low norm matrix $\mathbf{R}_{\mathsf{BP}}$:

$$\mathbf{R}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c, \mathbf{A})$$

In particular, let $\mathbf{R}'_i = \mathbf{R}^c_i - \mathbf{R}_i$ for $i \in [\ell]$. We derive the low-norm matrices $\mathbf{R}_{t,i}$ for $i \in [5]$ as

1. Let $\gamma_0 := \gamma_{t,i,0}$ and $\gamma_1 := \gamma_{t,i,1}$.

2. Compute

$$\begin{aligned}\mathbf{R}_{t,i} = (-\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{R}_{t-1,\gamma_0}) \\ + (-\mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) + x_{\mathsf{var}(t)} \cdot \mathbf{R}_{t-1,\gamma_1})\end{aligned} \tag{3.1}$$

We let $\mathbf{R}_{L,1}$ be the matrix obtained at L-th step corresponding to state value 1 by the above algorithm. The correctness requires the norm of $\mathbf{R}_{\mathsf{BP}}$ remains small and the matrix $\mathbf{V}_{\mathsf{BP}}$ output by $\mathsf{Eval}_{\mathsf{pk}}$ satisfies $\mathbf{V}_{\mathsf{BP}} = \mathbf{A}\mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\boldsymbol{x})\mathbf{G}$. We refer to the counterpart in [GV15] for the detailed correctness proof.

In order to achieve correctness and deniability, it is important for us to both lower and upper bound the norm of $||\mathbf{R}_{\mathsf{BP}}||$. Here we apply the triangular inequality of the norm and obtain the following lemma:

Lemma 3.3.2. *Let $\mathbf{R}_{i,j}$'s be the matrices defined as above. Then for every $t \in [\ell], i \in [5]$ and every error vector $\boldsymbol{e} \in \mathbb{Z}_q^m$, we have $||\boldsymbol{e}^T \cdot \mathbf{R}_{t-1,j}|| - \Theta(m^{1.5}) \cdot ||\boldsymbol{e}|| \leq ||\boldsymbol{e}^T \cdot \mathbf{R}_{t,i}|| \leq ||\boldsymbol{e}|| \cdot ||\mathbf{R}_{t-1,j}|| + \Theta(m^{1.5}) \cdot ||\boldsymbol{e}||$, where $j = \gamma_{x_{\mathsf{var}(t)}}$.*

Proof. Recall the matrix $\mathbf{R}_{i,j}$ is computed as

$$\mathbf{R}_{t,i} = (-\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + (1 - x_{\mathsf{var}(t)}) \cdot \mathbf{R}_{t-1,\gamma_0}) + (-\mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) + x_{\mathsf{var}(t)} \cdot \mathbf{R}_{t-1,\gamma_1})$$

where $x_{\mathsf{var}(t)} \in \{0,1\}$. Without loss of generality, we assume $x_{\mathsf{var}(t)} = 1$, thus we obtain

$$\mathbf{R}_{t,i} = -\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + \mathbf{R}_{t-1,\gamma_1} - \mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1})$$

Since $\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) \in \{0,1\}^{m \times m}$, we know $||\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1})|| \leq m$. Since matrices $\mathbf{R}_{\mathsf{var}(t)}, \mathbf{R}'_{\mathsf{var}(t)}$ were chosen uniformly at random in $\in \{-1,1\}^{m \times m}$, we know that their norm is bounded by $\Theta(\sqrt{m})$ with high probability by Lemma 2.3.2. Therefore, we can bound the norm of term $||\mathbf{R}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + \mathbf{R}_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1})|| \leq \Theta(m^{1.5})$. By applying the triangular inequality, it holds for every $t \in [\ell], i \in [5]$ and vector $\boldsymbol{e} \in \mathbb{Z}_q^m$,

$$||\boldsymbol{e}^T \cdot \mathbf{R}_{t-1,j}|| - \Theta(m^{1.5})||\boldsymbol{e}|| \leq ||\boldsymbol{e}^T \cdot \mathbf{R}_{t,i}|| \leq ||\boldsymbol{e}|| \cdot ||\mathbf{R}_{t-1,j}|| + \Theta(m^{1.5})||\boldsymbol{e}||$$

where $j = \gamma_{x_{\mathsf{var}(t)}}$. □

By applying the above lemma inductively on the equation (3.1) of computing matrix $\mathbf{R}_{\mathsf{BP}}$ for input length ℓ, we can obtain the following theorem:

Theorem 3.3.3. *Let* BP *be a length* ℓ *branching program, and* $\mathbf{R}_{\mathsf{BP}}$ *be the matrix as defined above. Then we have* $||\boldsymbol{e}^T \cdot \mathbf{R}_{0,j}|| - 2m^{1.5}\ell||\boldsymbol{e}|| \leq ||\boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}|| \leq ||\boldsymbol{e}^T|| \cdot ||\mathbf{R}_{0,j}|| + 2m^{1.5}\ell||\boldsymbol{e}||$ *for some* $j \in [5]$.

Proof. Applying Lemma 3.3.2 inductively on input length ℓ, we have

$$||\boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}|| \geq ||\boldsymbol{e}^T \cdot \mathbf{R}_{\ell-1,j}|| - 2m^{1.5}||\boldsymbol{e}|| \geq \cdots \geq ||\boldsymbol{e}^T \cdot \mathbf{R}_{0,j}|| - 2m^{1.5}\ell||\boldsymbol{e}||$$

We can obtain the upper bound of $||\boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}||$ using similar computation. □

Lemma 3.3.4. *Let* $\mathbf{R}$ *is an* $m \times m$ *be a matrix chosen at random from* $\{-1,1\}^{m\times m}$*, and* $\boldsymbol{u} = (u_1, ..., u_m) \in \mathbb{R}^m$ *be a vector chosen according to the* m *dimensional Gaussian with width* α*. Then we have*

$$\Pr\left[||\boldsymbol{u}^T\mathbf{R}||^2 \in \Theta(m^2\alpha^2)\right] > 1 - \mathsf{negl}(m).$$

Proof. We know with overwhelming probability over the choice of $\boldsymbol{u}$, all of its entries have absolute value less than $B = \alpha\omega(\log m)$. Also, we know that with overwhelming probability, we have $||\boldsymbol{u}||^2 = \Theta(m\alpha^2)$. We call a sample typical if it satisfies these two conditions. Note that it is without loss of generality to just consider the typical samples, from a simple union bound argument.

Then we consider a fixed typical choice of vector $\boldsymbol{u} = (u_1, ..., u_m) \in \mathbb{R}^m$. We write the inner product of $\boldsymbol{u}^T \cdot \boldsymbol{r}$ where $\boldsymbol{r} = (r_1, \ldots, r_m)$ is sampled uniformly from $\{-1,1\}^m$. We observe that $\mathbb{E}\left[||\boldsymbol{u}^T \cdot \boldsymbol{r}||^2\right] = \mathbb{E}\left[\sum_{i=1}^m r_i^2u_i^2 + \sum_{i<j\leq m} r_ir_ju_iu_j\right] = \sum_{i=1}^m u_i^2 = ||\boldsymbol{u}||^2$. This is because each r_i, r_j are independent and have mean 0.

Now, for such a fixed $\boldsymbol{u}$ we denote random variables $X_1, \ldots, X_m$ be i.i.d. samples of $\boldsymbol{r}^T\boldsymbol{u}$. It is not hard to see that

- $||\boldsymbol{u}^T\mathbf{R}||^2 = X_1^2 + X_2^2 + \cdots + X_m^2$, (one can view X_i as the i-th entry of $\boldsymbol{u}^T\mathbf{R}$),
- $\mathbb{E}\left[||\boldsymbol{u}^T\mathbf{R}||^2\right] = m||\boldsymbol{u}||^2$.

Next we claim that for each i, $X_i^2 \leq mB^2\omega(\log m)$ with overwhelming probability. By Hoeffding's inequality, we have

$$\Pr\left[\left|\sum_{j\in[m]} r_ju_j\right| > t\right] < 2e^{-\frac{2t^2}{m\cdot 4B^2}}.$$

This is because each $r_ju_j \in [-B, B]$. (Recall that we consider a fixed $\boldsymbol{u}$ for the typical case). By setting $t = \sqrt{m}B\omega(\log m)$, we have $\Pr[|X_i| > t] < \mathsf{negl}(m)$. Thus $X_i^2 \leq mB^2\omega(\log m)$ with overwhelming probability. So we can consider truncated versions of

X_i^2's, where we cut out the large samples. This will only induce a negligible statistical distance, and change the expectation by a negligible amount. For simplicity of presentation, we still use the notation X_i^2's in the following arguments, but the reader should keep in mind that they were truncated.

Next again we apply Hoeffding's inequality to the X_i^2's to obtain

$$\Pr\left[\left|||\boldsymbol{u}^T\mathbf{R}||^2 - m||\boldsymbol{u}||^2\right| > t'\right] < 2e^{-\frac{2t'^2}{\sum_{i=1}^{m}(mB^2\omega(\log m))^2}} = 2e^{-\frac{2t'^2}{m^3B^4\omega(\log m)}}.$$

By taking $t' = m||\boldsymbol{u}||^2/2$, we have

$$\Pr\left[\left|||\boldsymbol{u}^T\mathbf{R}||^2 - m||\boldsymbol{u}||^2\right| > t'\right] < 2e^{-\frac{||\boldsymbol{u}||^4}{2mB^4\omega(\log m)}}.$$

Since $\boldsymbol{u}$ is typical, we know that $||\boldsymbol{u}||^2 = \Theta(m\alpha^2)$. Also recall that $B = \alpha\omega(\log m)$. So we have

$$\Pr\left[||\boldsymbol{u}^T\mathbf{R}||^2 \in \Theta(m^2\alpha^2)\right] > 1 - 2e^{-\frac{m}{\omega(\log m)}} = 1 - \mathsf{negl}(m).$$

This completes the proof. □

3.3.2 Construction of Flexibly Bi-Deniable ABE for Branching Programs

In this part, we present our flexibly bi-deniable AB-BTS scheme for bounded-length Branching Programs. We use a semantically-secure public key encryption $\Pi = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ with message space $\mathcal{M}_\Pi = \mathbb{Z}_q^{m\times m}$ and ciphertext space $\mathcal{C}_\Pi$. For a family of branching programs of length bounded by L and input space $\{0,1\}^\ell$, the description of BiDenAB-BTS = (Setup, DenSetup, Keygen, SampleP, SampleU, TestP, FakeRCoins, FakeSCoins) are as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^\ell)$: On input the security parameter λ, the length of the branching program

L and length of the attribute vector ℓ,

1. Set the LWE dimension be $n = n(\lambda)$, modulus $q = q(n, L)$. Choose Gaussian distribution parameter $s = s(n)$. Let $\mathsf{params} = (n, q, m, s)$.
2. Sample one random matrix associated with its trapdoor as

$$(\mathbf{A}, \mathbf{T}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(q, n, m)$$

3. Choose $\ell + 6$ random matrices $\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c$ from $\mathbb{Z}_q^{n\times m}$.
4. Choose a random vector $\boldsymbol{u} \in \mathbb{Z}_q^n$.
5. Compute a public/secret key pair $(\mathsf{pk}', \mathsf{sk}')$ for a semantically secure public key encryption $(\mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{Gen}'(1^\lambda)$
6. Output the public parameter pp and master secret key msk as

$$\mathsf{pp} = (\mathsf{params}, \mathbf{A}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c, \boldsymbol{u}, \mathsf{pk}'), \quad \mathsf{msk} = (\mathbf{T}_{\mathbf{A}}, \mathsf{sk}')$$

- $\mathsf{DenSetup}(1^\lambda, 1^L, 1^\ell)$: On input the security parameter λ, the length of branching program L and length of attribute vector ℓ, the deniable setup algorithm runs the same computation as setup algorithm, and outputs

$$\mathsf{pp} = (\mathsf{params}, \mathbf{A}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c, \boldsymbol{u}, \mathsf{pk}'), \quad \mathsf{msk} = (\mathbf{T}_{\mathbf{A}}, \mathsf{sk}') \quad \mathsf{fk} = (\mathbf{T}_{\mathbf{A}}, \mathsf{sk}')$$

- $\mathsf{Keygen}(\mathsf{msk}, \mathsf{BP})$: On input the master secret key msk and the description of a branching program BP, $\mathsf{BP} = (\boldsymbol{v}_0, \{\mathsf{var}(t), \{\gamma_{t,i,0}, \gamma_{t,i,1}\}_{i\in[5]}\}_{t\in[L]})$.
  1. Homomorphically compute a public matrix with respect to the branching program BP: $\mathbf{V}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$.

2. Sample a low norm vector $\boldsymbol{r}_{\mathsf{BP}} \in \mathbb{Z}_q^{2m}$, using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, (\mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \boldsymbol{u}, sq)$$

such that $\boldsymbol{r}_{\mathsf{BP}}^T \cdot [\mathbf{A}|\mathbf{V}_{\mathsf{BP}} + \mathbf{G}] = \boldsymbol{u}$.

3. Output the secret key $\mathsf{sk}_{\mathsf{BP}}$ for branching program as $\mathsf{sk}_{\mathsf{BP}} = (\boldsymbol{r}_{\mathsf{BP}}, \mathsf{BP})$.

- $\mathsf{SampleP}(\mathsf{pp}, \boldsymbol{x})$: On input public parameters pp and attribute $\boldsymbol{x}$,

1. Choose an LWE secret $\boldsymbol{s} \in \mathbb{Z}_q^n$ uniformly at random.
2. Choose noise vector $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}_q^m, \alpha}$, and compute $\psi_0 = \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}$.
3. Choose one random matrices $\mathbf{R}^c \leftarrow \{-1, 1\}^{m \times m}$, and let $\boldsymbol{e}^c = \boldsymbol{e}^T\mathbf{R}^c$. Compute an encoding of constant 1: $\psi^c = \boldsymbol{s}^T(\mathbf{A}^c + \mathbf{G}) + \boldsymbol{e}^c$.
4. Encode each bit $i \in [\ell]$ of the attribute vector:
   (a) Choose a random matrix $\mathbf{R}_i \leftarrow \{-1, 1\}^{m \times m}$, and let $\boldsymbol{e}_i = \boldsymbol{e}^T\mathbf{R}_i$.
   (b) Compute $\psi_i = \boldsymbol{s}^T(\mathbf{A}_i + x_i\mathbf{G}) + \boldsymbol{e}_i$.
5. Encode the initial state vector $\boldsymbol{v}_0 = (1, 0, 0, 0, 0)$, for $i \in [5]$
   (a) Choose a random matrix $\mathbf{R}'_{0,i} \leftarrow \{-1, 1\}^{m \times m}$, and let $\mathbf{R}_{0,i} = \eta\mathbf{R}'_{0,i}, \boldsymbol{e}_{0,i} = \boldsymbol{e}^T\mathbf{R}_{0,i}$, where the noise scaling parameter η is set in Section 3.3.3.
   (b) Compute $\psi_{0,i} = \boldsymbol{s}^T(\mathbf{A}_i + \boldsymbol{v}_0[i]\mathbf{G}) + \boldsymbol{e}_{0,i}$.
6. Compute $c = \boldsymbol{s}^T\boldsymbol{u} + e$, where $e \leftarrow \mathcal{D}_{\mathbb{Z}_q, s}$
7. Use PKE to encrypt randomly chosen matrices $\mathbf{R}^c, \{\mathbf{R}_i\}_{i\in[\ell]}$ and $\{\mathbf{R}_{0,i}\}_{i\in[5]}$:

$$\mathbf{T}_i \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{R}_i), \mathbf{T}^c \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{R}^c), \mathbf{T}_{0,i} \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{R}_{0,i})$$

8. Output the ciphertext

$$\mathsf{ct}_{\boldsymbol{x}} = (\boldsymbol{x}, \psi_0, \{\psi_i\}_{i\in[\ell]}, \psi^c, \{\psi_{0,i}\}_{i\in[5]}, c, \{\mathbf{T}_i\}_{i\in[\ell]}, \mathbf{T}^c, \{\mathbf{T}_{0,i}\}_{i\in[5]})$$

- $\mathsf{SampleU}(\mathsf{pp}, \boldsymbol{x})$: Output a uniformly random vector $\mathsf{ct} \in \mathbb{Z}_q^m \times \mathbb{Z}_q^{\ell m} \times \mathbb{Z}_q^{\ell m} \times \mathbb{Z}_q^{5m} \times \mathbb{Z}_q \times \mathcal{C}_\Pi^\ell \times \mathcal{C}_\Pi \times \mathcal{C}_\Pi^5$.

- $\mathsf{TestP}(\mathsf{sk}_{\mathsf{BP}}, \mathsf{ct}_{\boldsymbol{x}})$: On input the secret key $\mathsf{sk}_{\mathsf{BP}}$ for a branching program BP and a ciphertext associated with attribute $\boldsymbol{x}$, if $\mathsf{BP}(\boldsymbol{x}) = 0$, output $\perp$, otherwise,

  1. Homomorphically compute the evaluated ciphertext of result $\mathsf{BP}(\boldsymbol{x})$

  $$\psi_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \psi_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \psi_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c, \psi_i^c\}_{i\in[\ell]})$$

  2. Then compute $\phi = [\psi_0|\psi_{\mathsf{BP}}]^T \cdot \boldsymbol{r}_{\mathsf{BP}}$. Accept $\mathsf{ct}_{\boldsymbol{x}}$ as a P-sample if $|c - \phi| < 1/4$, otherwise reject.

- $\mathsf{FakeSCoins}(r_S)$: Simply output the P-sample $\mathbf{c}$ as the randomness r_S^* that would cause $\mathsf{SampleU}$ to output $\boldsymbol{c}_{\boldsymbol{x}}$.

- $\mathsf{FakeRCoins}(\mathsf{pp}, \mathsf{fk}, \mathsf{ct}_{\boldsymbol{x}}, \mathsf{BP})$: On input the public parameters pp, the faking key fk, a ciphertext $\mathsf{ct}_{\boldsymbol{x}}$ and description of a branching program BP

  1. If $\mathsf{BP}(\boldsymbol{x}) \neq 0$, then output $\mathsf{sk}_f \leftarrow \mathsf{Keygen}(\mathsf{fk}, \mathsf{BP})$.

  2. Otherwise, parse ciphertext $\mathsf{ct}_{\boldsymbol{x}}$ as

  $$\mathsf{ct}_{\boldsymbol{x}} = (\boldsymbol{x}, \psi_0, \{\psi_i\}_{i\in[\ell]}, \psi^c, \{\psi_{0,i}\}_{i\in[5]}, c, \{\mathbf{T}_i\}_{i\in[\ell]}, \mathbf{T}^c, \{\mathbf{T}_{0,i}\}_{i\in[5]})$$

  Compute $\boldsymbol{e} \leftarrow \mathsf{Invert}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \psi_0)$. Then decrypt $(\{\mathbf{T}_i\}_{i\in[\ell]}, \mathbf{T}^c, \{\mathbf{T}_{0,i}\}_{i\in[5]})$ respectively using $\mathsf{Dec}(\mathsf{sk}', \cdot)$ to obtain $\{\mathbf{R}_i\}_{i\in[\ell]}, \mathbf{R}^c, \{\mathbf{R}_{0,i}\}_{i\in[5]}$. Compute evaluated error

  $$\boldsymbol{e}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \boldsymbol{e}^T\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \boldsymbol{e}^T\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c, \boldsymbol{e}^T\mathbf{R}^c\})$$

  such that $\boldsymbol{e}_{\mathsf{BP}} = \boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}$.

3. Homomorphically compute a public matrix with respect to the branching program BP: $\mathbf{V}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c\}_{i\in[\ell]})$. Then sample a properly distributed secret key $\boldsymbol{r}_{\mathsf{BP}} \in \mathbb{Z}_q^{2m}$, using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, (\mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \boldsymbol{u}, s)$$

4. Sample correlation vector $\mathbf{y}_0 \leftarrow \mathcal{D}_{\mathbb{Z}_q^m, \beta^2 q^2 \mathbf{I}_{m\times m}}$. Then sample correlation coefficient $\mu \leftarrow \mathcal{D}_\gamma$, and set vector $\boldsymbol{y}_1 = (\mu \boldsymbol{e}_{\mathsf{BP}} + \mathcal{D}_{\mathbf{Z}^m, \mathbf{Q}})q$, where

$$\mathbf{Q} = \beta^2 \mathbf{I}_{m\times m} - \gamma^2 \alpha^2 \mathbf{R}_{\mathsf{BP}}^T \mathbf{R}_{\mathsf{BP}} \tag{3.2}$$

5. Let $\boldsymbol{y} = (\boldsymbol{y}_0 | \boldsymbol{y}_1)$, then sample and output the faked secret key $\mathsf{sk}^*_{\mathsf{BP}} = \mathbf{r}^*_{\mathsf{BP}}$ as $\mathbf{r}^*_{\mathsf{BP}} \leftarrow \boldsymbol{y} + \mathcal{D}_{\Lambda + \boldsymbol{r}_{\mathsf{BP}} - \boldsymbol{y}, \sqrt{s^2 - \beta^2}}$, using $\mathsf{SampleD}(\mathsf{ExtBasis}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, \mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \boldsymbol{r}_{\mathsf{BP}} - \boldsymbol{y}, \sqrt{s^2 - \beta^2})$, where $\Lambda = \Lambda^\perp([\mathbf{A}|\mathbf{V}_{\mathsf{BP}} + \mathbf{G}])$.

The SampleP algorithm is similar to the ABE ciphertexts in the work [GV15], except that we add another scaling factor η to the rotation matrices $\mathbf{R}_{0,i}$'s. This allows us to both upper and lower bound the noise growth, which is essential to achieve bi-deniability. As we discussed in the introduction, the FakeRCoins embeds the evaluated noise into the secret key, so that it will change the decrypted value of the targeted ciphertext, but not others. Next we present the theorem we achieve and a high level ideas of the proof. We describe the intuition of our proof as follows.

Overview of Our Security Proof. At a high level, our security proof begins at the Fake experiment (cf. Definition 3.2.1 for a formal description), where first a ciphertext ct^* and its associated noise terms $\boldsymbol{e}^*$ are sampled, then a fake key $\boldsymbol{r}^*$ is generated that "artificially" fails to decrypt any ciphertext with noise vector (oriented close to) $\boldsymbol{e}^*$. In the end, we will arrive at the Real experiment, where an honest key $\boldsymbol{r}$ is generated that "genuinely" fails to decrypt the honestly generated, coerced ciphertext ct^*. (Multi-ct coercion security follows

by a standard hybrid argument that repeatedly modifies respective $\boldsymbol{r}^*$ to $\boldsymbol{r}$ for each coerced ct^* in order.) In order to transition from Fake to Real, we move through a sequence of computationally- or statistically-indistinguishable hybrid experiments.

The first set of intermediate experiments (represented by H_1 and H_2 in our formal proof) embeds the attribute $\boldsymbol{x}$ of the challenge ciphertext ct^* in the public parameters, in a similar fashion to the beginning of every SIM-secure proof of lattice-based ABE. Indistinguishability follows via the Leftover Hash Lemma [DRS04]. (Note that the additional hybrid in our proof is used to ensure that the random rotation matrices $\mathbf{R}$ employed by the LHL for public key embedding of $\boldsymbol{x}$ are the *exact same* matrices $\mathbf{R}$ as used to generate the noise terms of the coerced ct^*, and uses the security of any semantically-secure PKE for computational indistinguishability.)

The next set of intermediate experiments (given by $\mathsf{H}_3, \mathsf{H}_4$, and H_5 in our formal proof) perform the "main, new work" of our security proof. Specifically, they "swap the order" of the generation of the pk matrices $\{\mathbf{A}\}$, the public coset $\boldsymbol{u}$ (in the public parameters and in the coerced ciphertext), and the error vector(s) $\boldsymbol{e}$ in the coerced ciphertext components. (An additional hybrid is used to toggle the order of a "correlation vector" $\boldsymbol{y}$ – a random, planted vector used to allow for a more modular analysis of these steps.) In each case, we give a *statistical* argument that the adversary's view in adjacent hybrids is indistinguishable or identical, using elementary properties of multi-dimensional Gaussians.

In the next step (given by H_6), we apply the eLWE^+ assumption to (roughly) change every component of the coerced ciphertext ct^* to uniform – except for the final c^* component used to blind the message μ.

In the final step (given by H_7), we transition to the Real experiment by changing the c^* component to uniform (in the presence of Dual Regev decryption under honest $\boldsymbol{z}$), using our sharper noise analysis as described above to show statistical indistinguishability of the final decryption output of $\boldsymbol{z}$ on ct^*.

Theorem 3.3.5. *Assuming the hardness of extended-*$\mathsf{LWE}_{q,\beta'}$*, the above algorithms form a*

secure attribute-based bitranslucent set schemem, as in Definition 3.2.2.

Lemma 3.3.6. *For parameters set in Section 3.3.3, the AB-BTS defined above satisfies the correctness property in Definition 3.2.2.*

Proof. As we mentioned in Remark 3.2.3, the correctness of faking algorithms is implied by the bi-deniability property. Therefore, we only need to prove the correctness of normal decryption algorithm. For branching program BP and input $\boldsymbol{x}$, such that $\mathsf{BP}(\boldsymbol{x}) = 1$, we compute $\psi_{t,i}$ for $t \in [\ell]$ as

$$\begin{aligned}
\psi_{t,i} &= \mathsf{Add}(\mathsf{Mult}(\psi'_{\mathsf{var}(t)}, \psi_{t-1,\gamma_0}), \mathsf{Mult}(\psi_{\mathsf{var}(t)}, \psi_{t-1,\gamma_1})) \\
&= \mathsf{Add}\Big([\boldsymbol{s}^T(-\mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) + (\boldsymbol{v}_t[\gamma_0] \cdot (1 - x_{\mathsf{var}(t)})) \cdot \mathbf{G}) + \boldsymbol{e}_1], \\
&\qquad ([\boldsymbol{s}^T(-\mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) + (\boldsymbol{v}_t[\gamma_1] \cdot x_{\mathsf{var}(t)}) \cdot \mathbf{G}) + \boldsymbol{e}_2] \Big) \\
&= \boldsymbol{s}^T \Big[\underbrace{\big(-\mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_0}) - \mathbf{A}'_{\mathsf{var}(t)}\mathbf{G}^{-1}(\mathbf{V}_{t-1,\gamma_1}) \big)}_{\mathbf{V}_{t,i}} \\
&\qquad + \underbrace{\big(\boldsymbol{v}_t[\gamma_0] \cdot (1 - x_{\mathsf{var}(t)}) + \boldsymbol{v}_t[\gamma_1] \cdot x_{\mathsf{var}(t)}\big)}_{\boldsymbol{v}_t[i]} \cdot \mathbf{G} \Big] + \boldsymbol{e}_{t,i}
\end{aligned}$$

At the end of the ciphertext evaluation, since $\mathsf{BP}(\boldsymbol{x}) = 1$, we can obtain $\psi_{\mathsf{BP}} = \boldsymbol{s}^T(\mathbf{V}_{\mathsf{BP}} + \mathbf{G}) + \boldsymbol{e}_{\mathsf{BP}}$, where $\boldsymbol{e}_{\mathsf{BP}} = \boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}}$. Recall that the secret key $\mathsf{sk} = \boldsymbol{r}_{\mathsf{BP}}$ satisfying $[\mathbf{A}|\mathbf{V}_{\mathsf{BP}} + \mathbf{G}] \cdot \boldsymbol{r}_{\mathsf{BP}} = \boldsymbol{u}$. Then for $c - [\psi_0|\psi_{\mathsf{BP}}] \cdot \boldsymbol{r}_{\mathsf{BP}}$, it holds that

$$c - [\psi_0|\psi_{\mathsf{BP}}]^T \cdot \boldsymbol{r}_{\mathsf{BP}} = e - \boldsymbol{e}^T\mathbf{R}_{\mathsf{BP}} \cdot \boldsymbol{r}_{\mathsf{BP}}$$

Now we need to compute a bound for the final noise term. By applying Theorem 3.3.3, we obtain that

$$||\boldsymbol{e}^T|| \cdot ||\mathbf{R}_{\mathsf{BP}}|| + 2m^{1.5}\ell||e|| \leq (2m^{1.5}\ell + \eta\sqrt{m})||\boldsymbol{e}|| \leq \alpha\sqrt{m}(2m^{1.5}\ell + \eta\sqrt{m}) \cdot sq\sqrt{m} \leq \frac{1}{4}$$

So by setting the parameters appropriately, as in Section 3.3.3, we have that

$$|c - [\psi_0|\psi_{\mathsf{BP}}]^T \cdot \boldsymbol{r}_{\mathsf{BP}}| \leq 1/4$$

and the lemma follows. □

Lemma 3.3.7. *Assuming the hardness of extended-*$\mathsf{LWE}_{q,\beta'}$*, the AB-BTS scheme described above is bi-deniable as defined in Definition 3.2.2.*

Proof. First, we notice that because $\mathsf{SampleU}$ simply outputs its random coins as a uniformly random ct, we can use ct itself as the coins.

We prove the bi-deniability property by a sequence of hybrids H_i with details as follows:

Hybrid H_0: Hybrid H_0 is the same as the view of adversary $\mathcal{A}$ in the right-hand faking experiment in the definition of bi-deniability. We use the fact that algorithm Invert successfully recovers $\boldsymbol{e}$ from ct with overwhelming probability over all randomness in the experiment.

Hybrid H_1: In hybrid H_2, we switch the encryptions of matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ in the ciphertext to encryptions of zero.

Recall that in hybrid H_0, we encrypt the randomness matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ using semantically secure PKE Π, i.e.

$$\mathbf{T}_i \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{R}_i), \quad \mathbf{T}^c \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{R}^c), \quad \mathbf{T}_{0,i} \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{R}_{0,i})$$

In hybrid H_1, we just set

$$\mathbf{T}_i \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{0}), \quad \mathbf{T}^c \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{0}), \quad \mathbf{T}_{0,i} \leftarrow \mathsf{Enc}'(\mathsf{pk}', \mathbf{0})$$

to be encryptions of $\mathbf{0} \in \mathbb{Z}^{m\times m}$ to replace encryptions of matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$.

Hybrid H_2: In hybrid H_2, we embed random matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ and challenge attribute $\boldsymbol{x}^*$ in the public parameters pp.

Recall that in hybrid H_1 the matrices $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$ are sampled at random. In hybrid H_2, we slightly change how these matrices are generated. Let $\boldsymbol{x}^* = (x_1^*, ..., x_\ell^*)$ be the challenge attribute that the adversary $\mathcal{A}$ intends to attack. We sample matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}'_{0,i}\}_{i\in[5]}, \mathbf{R}^c)$ uniformly random from $\{-1, 1\}^{m\times m}$ and set $\mathbf{R}_{0,i} = \eta\mathbf{R}'_{0,i}$, which would be used both in the generation of public parameters and challenge ciphertext. We set $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$ respectively as

$$\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i^*\mathbf{G}, \quad \mathbf{V}_{0,i} = \mathbf{A}\mathbf{R}_{0,i} - \boldsymbol{v}_0[i]\mathbf{G}, \quad \mathbf{A}^c = \mathbf{A}\mathbf{R}^c - \mathbf{G}$$

where $\boldsymbol{v}_0 = [1, 0, 0, 0, 0]$. The rest of the hybrid remains unchanged.

Hybrid H_3: In hybrid H_3, we change the generation of matrix $\mathbf{A}$ and vector $\boldsymbol{u}$ in public parameters pp.

Let $\mathbf{A}$ be a random matrix in $\mathbb{Z}_q^{n\times m}$. The construction of matrices $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$ remains the same, as in hybrid H_2. Sample error vectors $\boldsymbol{e}$ that would be used in algorithm $\mathsf{SampleP}$ later. Then compute the error vector

$$\boldsymbol{e}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \boldsymbol{e}^T\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \boldsymbol{e}^T\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c, \boldsymbol{e}^T\mathbf{R}^c\})$$

and choose a correlation coefficient $\mu \leftarrow \mathcal{D}_\gamma$, and set vector $\boldsymbol{y}_1 = (\mu\boldsymbol{e}_{\mathsf{BP}^*} + \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}})q$, where

$$\mathbf{Q} = \beta^2\mathbf{I}_{m\times m} - \gamma^2\alpha^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*}$$

Then let $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1)$, where $\boldsymbol{y}_0 \leftarrow \mathcal{D}_{\mathbf{Z}_q^m,\beta^2q^2\mathbf{I}_{m\times m}}$. Sample vector $\mathbf{r}_{\mathsf{BP}^*} \leftarrow \boldsymbol{y}+\mathcal{D}_{\mathbb{Z}^{2m}-\boldsymbol{y},(s^2-\beta^2)q^2\mathbf{I}_{2m\times 2m}}$, and compute matrix

$$\mathbf{V}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \mathbf{A}^c)$$

Set vector $\boldsymbol{u}$ in public parameters pp as $\mathbf{u} = [\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}] \cdot \mathbf{r}_{\mathsf{BP}^*}$. Since $\mathbf{A}$ is a random matrix without trapdoor $\mathbf{T_A}$ to answer key queries, we will use trapdoor $\mathbf{T_G}$ to answer queries as follows. Consider a secret key query for branching program BP such that $\mathsf{BP}(\boldsymbol{x}^*) = 0$. To respond, we do the following computations:

1. First, we compute

$$\mathbf{R}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \mathbf{R}^c, \mathbf{A})$$

   to obtain a low-norm matrix $\mathbf{R}_{\mathsf{BP}} \in \mathbb{Z}_q^{m\times m}$ satisfying $\mathbf{A}\mathbf{R}_{\mathsf{BP}} - \mathsf{BP}(\boldsymbol{x}^*)\mathbf{G} = \mathbf{V}_{\mathsf{BP}}$.

2. Then, we sample $\boldsymbol{r}_{\mathsf{BP}}$ using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_{\mathsf{BP}}, \mathbf{T_G}, \boldsymbol{u}, sq)$$

   such that

$$\boldsymbol{r}_{\mathsf{BP}}^T \cdot [\mathbf{A}|\mathbf{V}_{\mathsf{BP}} + \mathbf{G}] = \boldsymbol{u}$$

   By Lemma 2.3.7, vector $\boldsymbol{r}_{\mathsf{BP}}$ is distributed as required.

The computation of answering P-sampler query, $\mathsf{SampleP}$ is the same as hybrid H_1 with error vectors $\boldsymbol{e}$, For faking receiver coins, $\mathsf{FakeRCoins}$, simply output the vector $\boldsymbol{r}_{\mathsf{BP}^*}$ pre-sampled in the generation of vector $\boldsymbol{u}$ before.

Hybrid H_4: In hybrid H_4, we change the generation order of vector $\boldsymbol{y}$ and error vector $\boldsymbol{e}$. First sample vector $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},\beta^2q^2\mathbf{I}_{2m\times 2m}}$ and compute $\boldsymbol{r}_{\mathsf{BP}^*}$ from $\boldsymbol{y}$ as in previous hybrid. Next, we compute error term $\boldsymbol{e}$ as $\boldsymbol{e} = \nu\boldsymbol{y}_1^T\mathbf{R}_{\mathsf{BP}^*}/q + \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'}$, where $\nu \leftarrow \mathcal{D}_\tau, \tau = \gamma\alpha^2/\beta^2$, and $\mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'}$ is sampled as $\mathbf{L}'\mathcal{D}_{\mathbf{Z}_1^m,\mathbf{I}_{m\times m}}$ for

$$\mathbf{Q}' = \mathbf{L}'\mathbf{L}'^T = \alpha^2\mathbf{I} - \tau^2\beta^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*} \tag{3.3}$$

Additionally, we modify the challenge ciphertext to be

$$\psi_0^* = \boldsymbol{s}^T\mathbf{A}/q + \boldsymbol{e}, \quad \psi_i^* = \psi_0^{*T}\mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T}\mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T}\mathbf{R}^c/q$$

and $c^* = \boldsymbol{s}^T\boldsymbol{u} + \mathcal{D}_{\mathbf{Z}^m,\alpha\mathbf{I}_{m\times m}}$.

Hybrid H_5: In hybrid H_5, we change the generation order of secret key $\boldsymbol{r}_{\mathsf{BP}^*}$ and vector $\boldsymbol{y}$. We first sample matrix $\boldsymbol{r}_{\mathsf{BP}^*}$ from discrete Gaussian distribution $\mathcal{D}_{\mathbb{Z}^{2m},s^2q^2\mathbf{I}_{2m\times 2m}}$, and set vector $\boldsymbol{u}$ in public parameters pp to be $\boldsymbol{u} = [\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}]\cdot\boldsymbol{r}_{\mathsf{BP}^*}$, where

$$\mathbf{V}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathsf{BP}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c\}_{i\in[\ell]})$$

Then set $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) = \boldsymbol{r}_{\mathsf{BP}^*}/2 + \mathcal{D}_{\mathbb{Z}^{2m},(\beta^2-s^2/4)q^2\mathbf{I}_{2m\times 2m}}$. The remainder of the hybrid remains roughly the same. In particular, the challenge ciphertext ct^* is generated in the same manner as Hybrid H_4. We break the noise term $\boldsymbol{e}$ into two terms $\boldsymbol{e} = \boldsymbol{e}_0^{(1)} + \boldsymbol{e}_0^{(2)} + \nu\boldsymbol{y}_1^T\mathbf{R}_{\mathsf{BP}^*}/q$, where $\boldsymbol{e}_0^{(1)} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\beta'\mathbf{I}_{m\times m}}, \boldsymbol{e}_0^{(2)} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'-\beta'^2\mathbf{I}_{m\times m}}$ and $\beta' = \alpha/2$.

Hybrid H_6: In hybrid H_6, we change how the challenge ciphertext is generated by using the Extended-LWE$^+$ instance.

First sample uniformly random vector $\boldsymbol{b} \in \mathbb{Z}^m$ and set the challenge ciphertext as

$$\psi_0^* = \boldsymbol{b}/q + \boldsymbol{e}_0^{(2)}, \quad \psi_i^* = \psi_0^{*T}\mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T}\mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T}\mathbf{R}^c/q$$

and $c^* = \boldsymbol{r}_{\mathsf{BP}^*}^T[\mathbf{I}_{m\times m}|\mathbf{R}_{\mathsf{BP}^*}](\boldsymbol{b}/q - \boldsymbol{e}_0^{(1)}) + \mathcal{D}_{\mathbb{Z}^m,\alpha\mathbf{I}_{m\times m}}$.

Hybrid H_7: In hybrid H_7, we change the challenge ciphertext to be uniformly random.

In algorithm $\mathsf{SampleP}$, sample uniformly random vectors $\mathsf{ct} \in \mathbb{Z}_q^m \times \mathbb{Z}_q^{\ell m} \times \mathbb{Z}_q^m \times \mathbb{Z}_q^{5m} \times \mathbb{Z}_q$ and outputs ct.

Claim 3.3.8. *Assuming the semantic security of PKE* $\Pi = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$*, hybrid* H_0

and H_1 *are computationally indistinguishable.*

Proof. Observe there is only one difference between hybrids H_0 and H_1 occurs in the challenge ciphertext, i.e. the encryption (under PKE Π) of the random matrices $\mathbf{S}_i$ are replaced by encryption of 0. If a PPT adversary $\mathcal{A}$ distinguishes between the H_0-encryptions of $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}^c\})$ and the H_1-encryptions of $\mathbf{0}$ with non-negligible probability, then we can construct an efficient reduction $\mathcal{B}$ that uses $\mathcal{A}$ to break the semantic security of PKE Π with similar probability. □

Claim 3.3.9. *Hybrids* H_1 *and* H_2 *are statistically indistinguishable.*

Proof. Observe the only difference between hybrids H_1 and H_2 is the generation of matrices

$$(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}_i^c\}_{i\in[\ell]})$$

The random matrices $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}_i^c\}_{i\in[\ell]})$ are used in the generation of public parameters pp:

$$\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i^*\mathbf{G}, \quad \mathbf{V}_{0,i} = \mathbf{A}\mathbf{R}_{0,i} - \boldsymbol{v}_0[i]\mathbf{G}, \quad \mathbf{A}^c = \mathbf{A}\mathbf{R}^c - \mathbf{G}$$

and the construction of errors in challenge ciphertext

$$\boldsymbol{e}_i = \boldsymbol{e}^T\mathbf{R}_i, \quad \boldsymbol{e}^c = \boldsymbol{e}^T\mathbf{R}^c, \quad \boldsymbol{e}_{0,i} = \boldsymbol{e}^T\mathbf{R}_{0,i}$$

Then by Leftover Hash Lemma 2.3.3, the following two distributions are statistically indistinguishable

$$(\mathbf{A}, \{\mathbf{A}\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{A}\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}\mathbf{R}^c\}, \widetilde{\boldsymbol{e}}) \approx (\mathbf{A}, \{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c\}, \widetilde{\boldsymbol{e}})$$

where $\widetilde{\boldsymbol{e}} = (\{\boldsymbol{e}_i\}_{i\in[\ell]}, \{\boldsymbol{e}_{0,i}\}_{i\in[5]}, \{\boldsymbol{e}^c\})$. Hence, hybrid H_0 and H_1 are statistically indistinguishable. □

Claim 3.3.10. *Hybrids* H_2 *and* H_3 *are statistically indistinguishable.*

Proof. Observe there are three differences between hybrid H_2 and H_3: The generation of matrix $\mathbf{A}$ and vector $\boldsymbol{u}$ in pp, challenge secret key $\mathsf{sk}_{\mathsf{BP}^*}$ and the computation methods to answer secret key queries. By the property of algorithm $\mathsf{TrapGen}(q, n, m)$ in Lemma 2.3.6, the distribution of matrix $\mathbf{A}$ in hybrid H_2 is statistically close to uniform distribution, from which matrix $\mathbf{A}$ in hybrid H_3 is sampled.

For secret key queries regarding branching program BP, in hybrid H_2, we sample vector $\boldsymbol{r}_{\mathsf{BP}}$, using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{T}_{\mathbf{A}}, (\mathbf{V}_{\mathsf{BP}} + \mathbf{G}), \boldsymbol{u}, s)$$

While in hybrid H_3, we sample vector $\boldsymbol{r}_{\mathsf{BP}}$, using

$$\boldsymbol{r}_{\mathsf{BP}} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{G}, \mathbf{R}_{\mathsf{BP}}, \mathbf{T}_{\mathbf{G}}, \boldsymbol{u}, sq)$$

By setting the parameters appropriately as specified in Section 3.3.3, and the properties of algorithms SampleLeft and SampleRight in Lemma 2.3.7, the answers to secret key queries are statistically close.

By Leftover Hash Lemma 2.3.3, the distribution $([\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}], [\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}] \cdot \boldsymbol{r}_{\mathsf{BP}^*})$ and $([\mathbf{A}|\mathbf{V}_{\mathsf{BP}^*}], \boldsymbol{u})$ are statistically close. Hence, hybrid H_2 and H_3 are statistically indistinguishable. □

Claim 3.3.11. *Hybrids* H_3 *and* H_4 *are statistically indistinguishable.*

Proof. The only difference between the two experiments is in the choice of $\boldsymbol{y}$ and $\boldsymbol{e}$, specifically, the choice of the $\boldsymbol{y}_1$ component of $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1)$. We will show that the joint distribution of $(\boldsymbol{e}, \boldsymbol{y}_1)$ is identically distributed in these two hybrids:

In hybrid H_3, $\boldsymbol{y}_1$ is set as $\boldsymbol{y}_1 = (\mu \boldsymbol{e}_{\mathsf{BP}^*} + \mathcal{D}_{\mathbb{Z}^m, \mathbf{Q}})q$, where $\mathbf{Q} = \beta^2 \mathbf{I}_{m \times m} - \gamma^2 \alpha^2 \mathbf{R}_{\mathsf{BP}^*}^T \mathbf{R}_{\mathsf{BP}^*}$

with $\boldsymbol{e} \leftarrow \mathcal{D}_{\mathbb{Z}^m,\alpha^2\mathbf{I}_{m\times m}}$ and

$$\boldsymbol{e}_{\mathsf{BP}^*} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{A}_i, \boldsymbol{e}^T\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}, \boldsymbol{e}^T\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c, \boldsymbol{e}^T\mathbf{R}^c\})$$

Therefore, in hybrid H_3, we may write the joint distribution of $(\boldsymbol{e}, \boldsymbol{y}_1)$ as $\mathbf{T}_1 \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}}$, where $\mathbf{T}_1 \stackrel{\text{def}}{=} \begin{pmatrix} \alpha\mathbf{I}_{m\times m} & \mathbf{0}_{m\times m} \\ \gamma\alpha q\mathbf{R}_{\mathsf{BP}^*}^T & q\mathbf{L} \end{pmatrix}$ for $\mathbf{Q} = \mathbf{L}\mathbf{L}^T \in \mathbb{Z}^{m\times m}$ via the Cholesky decomposition due to Lemma 2.3.1.

In hybrid H_4, vector $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1)$ is sampled as $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},\beta^2q^2\mathbf{I}_{2m\times 2m}}$. Then $\boldsymbol{e}$ is computed as $\boldsymbol{e} = \nu\boldsymbol{y}_1^T\mathbf{R}_{\mathsf{BP}^*}/q + \mathcal{D}_{\mathbb{Z}^m,\mathbf{Q}'}$, where $\nu \leftarrow \mathcal{D}_\tau, \tau = \gamma\alpha^2/\beta^2$, and $\mathbf{Q}' = \alpha^2\mathbf{I} - \tau^2\beta^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*}$. Then in hybrid H_4, we may write the joint distribution of $(\boldsymbol{e}, \boldsymbol{y}_1)$ as $\mathbf{T}_2 \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}}$, where $\mathbf{T}_2 \stackrel{\text{def}}{=} \begin{pmatrix} \mathbf{L}' & \tau\beta\mathbf{R}_{\mathsf{BP}^*} \\ \mathbf{0}_{m\times m} & \beta q\mathbf{I}_{m\times m} \end{pmatrix}$ for $\mathbf{Q}' = \mathbf{L}'\mathbf{L}'^T \in \mathbb{Z}^{m\times m}$ via the Cholesky decomposition due to Lemma 2.3.1.

We claim equality of the following systems of equations:

$$\begin{aligned}\mathbf{T}_1\mathbf{T}_1^T &= \begin{pmatrix} \alpha^2\mathbf{I}_{m\times m} & \gamma\alpha^2 q\mathbf{R}_{\mathsf{BP}^*} \\ \gamma\alpha^2 q\mathbf{R}_{\mathsf{BP}^*}^T & \gamma^2\alpha^2q^2\mathbf{R}_{\mathsf{BP}^*}^T\mathbf{R}_{\mathsf{BP}^*} + q^2\mathbf{L}\mathbf{L}^T \end{pmatrix} \\ &= \begin{pmatrix} \mathbf{L}'\mathbf{L}'^T + \tau^2\beta^2\mathbf{R}_{\mathsf{BP}^*}\mathbf{R}_{\mathsf{BP}^*}^T & \tau\beta^2 q\mathbf{R}_{\mathsf{BP}^*} \\ \tau\beta^2 q\mathbf{R}_{\mathsf{BP}^*}^T & \beta^2q^2\mathbf{I}_{m\times m} \end{pmatrix} = \mathbf{T}_2\mathbf{T}_2^T.\end{aligned}$$

This fact may be seen quadrant-wise by our choice of $\tau = \gamma\alpha^2/\beta^2$ and the settings of $\mathbf{Q} = \mathbf{L}\mathbf{L}^T$ and $\mathbf{Q}' = \mathbf{L}'\mathbf{L}'^T$ in Equations (3.2) and (3.3). It then follows that $(\mathbf{T}_2^{-1}\mathbf{T}_1)(\mathbf{T}_2^{-1}\mathbf{T}_1)^T = \mathbf{I}_{2m\times 2m}$, implying $\mathbf{T}_1 = \mathbf{T}_2\mathbf{Q}^*$ for some orthogonal matrix $\mathbf{Q}^*$. Because the spherical Gaussian $\mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}}$ is invariant under rigid transformations, we have $\mathbf{T}_1 \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}} = \mathbf{T}_2\mathbf{Q}^* \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}} = \mathbf{T}_2 \cdot \mathcal{D}_{\mathbb{Z}^{2m},\mathbf{I}_{2m\times 2m}}$, and the claim follows. □

Claim 3.3.12. *Hybrids* H_4 *and* H_5 *are statistically indistinguishable.*

Proof. Observe the main difference between hybrids H_4 and H_5 is the order of generation

of vectors $\boldsymbol{y}$ and $\boldsymbol{r}_{\mathsf{BP}^*}$: In hybrid H_4, we first sample $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},\beta^2 q^2 \mathbf{I}_{2m\times 2m}}$ and set $\boldsymbol{r}_{\mathsf{BP}^*} \leftarrow \boldsymbol{y} + \mathcal{D}_{\mathbb{Z}^{2m}-\boldsymbol{y},q^2(s^2-\beta^2)\mathbf{I}_{2m\times 2m}}$, while in hybrid H_5, we first sample $\boldsymbol{r}_{\mathsf{BP}^*} \leftarrow \mathcal{D}_{\mathbb{Z}^{2m},s^2 q^2 \mathbf{I}_{2m\times 2m}}$ and set $\boldsymbol{y} = (\boldsymbol{y}_0|\boldsymbol{y}_1) \leftarrow \boldsymbol{r}_{\mathsf{BP}^*}/2 + \mathcal{D}_{\mathbb{Z}^{2m},(\beta^2-s^2/4)q^2\mathbf{I}_{2m\times 2m}}$. By setting parameters appropriately as in Section 3.3.3, these two distributions are statistically close. □

Claim 3.3.13. *Assuming the hardness of extended-*$\mathsf{LWE}^+_{n,m,q,D_{\mathbb{Z}^m,\beta'},\mathbf{R}}$ *for any adversarially chosen distribution over matrices* $\mathbf{R} \in \mathbb{Z}_q^{m\times m}$*, then hybrids* H_5 *and* H_6 *are computationally indistinguishable.*

Proof. Suppose $\mathcal{A}$ has non-negligible advantage in distinguishing hybrid H_5 and H_6, then we use $\mathcal{A}$ to construct an extended-LWE$^+$ algorithm $\mathcal{B}$ as follows:

Invocation. $\mathcal{B}$ invokes adversary $\mathcal{A}$ to commit to a challenge attribute vector $\boldsymbol{x}^* = (x_1^*, ..., x_\ell^*)$ and challenge branching program BP^*. Then $\mathcal{B}$ generates $\mathbf{R}_{\mathsf{BP}^*}$ by first sampling $(\{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}^c\})$ as in the hybrid, and computes

$$\mathbf{R}_{\mathsf{BP}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}(\mathsf{BP}, \boldsymbol{x}, \{\mathbf{R}_i\}_{i\in[\ell]}, \{\mathbf{R}_{0,i}\}_{i\in[5]}, \{\mathbf{R}^c\}, \mathbf{A})$$

Then it receives an extended-LWE$^+$ instance for the matrix $\mathbf{R} = \mathbf{R}_{\mathsf{BP}^*}$ as follows:

$$\{\mathbf{A}, \boldsymbol{b} = \boldsymbol{s}^T\mathbf{A} + \boldsymbol{e}, \boldsymbol{z}_0, \boldsymbol{z}_1, \langle \boldsymbol{z}_0, \boldsymbol{b} - \boldsymbol{e}\rangle + e, \langle \boldsymbol{z}_1^T\mathbf{R}, \boldsymbol{b} - \boldsymbol{e}\rangle + e'\}$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^m$, $\boldsymbol{e}, \boldsymbol{z}_0, \boldsymbol{z}_1 \xleftarrow{\$} \chi^n$ and $e, e' \xleftarrow{\$} \chi$. Algorithm $\mathcal{B}$ aims to leverage adversary $\mathcal{A}$'s output to solve the extended-LWE$^+$ assumption.

Setup. $\mathcal{B}$ generates matrices $(\{\mathbf{A}_i\}_{i\in[\ell]}, \{\mathbf{V}_{0,i}\}_{i\in[5]}, \{\mathbf{A}^c\})$ as specified in hybrid H_1. Then, $\mathcal{B}$ sets challenge secret key $\mathsf{sk}_{\mathsf{BP}^*} = \boldsymbol{r}_{\mathsf{BP}^*} = (\boldsymbol{r}_0^*|\boldsymbol{r}_1^*) = (\boldsymbol{z}_0|\boldsymbol{z}_1)$ from extended-LWE$^+$ instance and computes vector $\boldsymbol{u}$ as in hybrid H_5.

Secret key queries. $\mathcal{B}$ answers adversary $\mathcal{A}$'s secret key queries as in hybrid H_2.

Challenge ciphertext. $\mathcal{B}$ answers adversary $\mathcal{A}$'s P-sample query by setting

$$\psi_0^* = \boldsymbol{b}/q + \boldsymbol{e}_0^{(2)} + \nu \boldsymbol{y}_1^T \mathbf{R}_{\mathsf{BP}^*}/q, \quad \psi_i^* = \psi_0^{*T} \mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T} \mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T} \mathbf{R}^c/q$$

and $c^* = \boldsymbol{r}_{\mathsf{BP}^*}^T [\mathbf{I}_{m\times m} | \mathbf{R}_{\mathsf{BP}^*}](\boldsymbol{b}/q - \boldsymbol{e}^{(1)}) + \mathcal{D}_{\mathbb{Z}^m, \alpha \mathbf{I}_{m\times m}}$.

Faking receiver coin query. $\mathcal{B}$ answers adversary $\mathcal{A}$'s faking receiver coin query by outputting the extended-LWE instance's vector $\mathsf{sk}_{\mathsf{BP}^*} = \boldsymbol{r}_{\mathsf{BP}^*}$.

Output. $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

We can rewrite the expression of $c^{*'}$ to be

$$\begin{aligned} c^{*'} &= ([\mathbf{A}^* | \mathbf{A}^* \mathbf{R}_{\mathsf{BP}^*}] \begin{pmatrix} \boldsymbol{z}_0 \\ \boldsymbol{z}_1 \end{pmatrix})^T \boldsymbol{s}/q + \mathcal{D}_{\mathbb{Z}_1, \alpha} \\ &= ((\boldsymbol{z}_0 | \boldsymbol{z}_1) \begin{pmatrix} \mathbf{A}^{*T} \\ \mathbf{R}_{\mathsf{BP}^*}^T \mathbf{A}^{*T} \end{pmatrix}) \boldsymbol{s}/q + D_{\mathbb{Z}_1, \alpha} = \boldsymbol{z}_0 \mathbf{A}^{*T} \boldsymbol{s}/q + \boldsymbol{z}_1 \mathbf{R}_{\mathsf{BP}^*}^T \mathbf{A}^{*T} \boldsymbol{s}/q + \mathcal{D}_{\mathbb{Z}_1, \alpha} \\ &= \langle \boldsymbol{z}_0, \boldsymbol{b}/q - \boldsymbol{e}^{(1)} \rangle + \langle \boldsymbol{z}_1^T \mathbf{R}_{\mathsf{BP}^*}, \boldsymbol{b}/q - \boldsymbol{e}^{(1)} \rangle + \mathcal{D}_{\mathbb{Z}_1, \alpha} \end{aligned}$$

We can see that if the eLWE^+ instance's vector $\boldsymbol{b}$ is pseudorandom, then the distribution simulated by $\mathcal{B}$ is exactly the same as H_5. If $\boldsymbol{b}$ is truly random and independent, then the distribution simulated by $\mathcal{B}$ is exactly the same as H_6. Therefore, if $\mathcal{A}$ can distinguish H_5 from H_6 with non-negligible probability, then $\mathcal{B}$ can break the $\mathsf{eLWE}^+_{n,m,q,\mathcal{D}_{(\alpha/2)q},\alpha',\mathbf{S}_{f^*}}$ problem for some $\alpha' \geq 0$ with non-negligible probability. □

Claim 3.3.14. *Hybrids* H_6 *and* H_7 *are statistically indistinguishable.*

Proof. Recall the only difference between hybrids H_6 and H_7 is the generation of challenge ciphertext. In hybrid H_7, we observe if ψ_0^* is chosen from uniform distribution, then by Leftover Hash Lemma 2.3.3, it holds

$$\psi_i^* = \psi_0^{*T} \mathbf{R}_i/q, \quad \psi_{0,i}^* = \psi_0^{*T} \mathbf{R}_{0,i}/q, \quad \psi^{*c} = \psi_0^{*T} \mathbf{R}^c/q$$

is also uniformly random (in their marginal distribution). Therefore, it remains to show that c^* is still uniformly random even conditioned on fixed samples of $(\psi_0^*, \{\psi_i^*\}_i, \{\psi_{0,i}^*\}_i, \{\psi^c\})$.

As calculated above, we can unfold the expression of c^* as

$$c^* = \langle \boldsymbol{z}_0, \boldsymbol{b}/q - \boldsymbol{x}^{(1)} \rangle + \langle \boldsymbol{z}_1^T \mathbf{R}_{\mathsf{BP}^*}, \boldsymbol{b}/q - \boldsymbol{x}^{(1)} \rangle + \mathcal{D}_{\mathbb{Z}_1,\alpha}$$

We note that $\boldsymbol{b}/q - \boldsymbol{x}^{(1)} = \psi_0^* - \boldsymbol{x}^{(1)} - \boldsymbol{x}^{(2)} - \nu \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{y}_1/q$, thus if we show that

$$\langle \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1, \nu \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{y}_1/q \rangle$$

is close to uniform distribution (modulo 1), then c^* will also be close to the uniform distribution (modulo 1), as c^* is masked by this uniformly random number. Recall in hybrids, we set $\boldsymbol{y}_1 = \boldsymbol{z}_1/2 +$ (shift), so it is sufficient to analyze

$$\langle \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1, \nu \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{y}_1/q \rangle = \nu \langle \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1, \mathbf{R}_{\mathsf{BP}^*} \boldsymbol{z}_1/q \rangle = \nu ||\mathbf{R}_{\mathsf{BP}^*}^* \boldsymbol{z}_1||^2/q$$

By applying Lemma 3.3.2 inductively on matrix $\mathbf{R}_{\mathsf{BP}^*}$, we can obtain that

$$||\mathbf{R}_{\mathsf{BP}^*}^* \boldsymbol{z}_1||^2/q \geq \frac{(||\mathbf{R}_{0,j} \boldsymbol{z}_1|| - \Theta(m^{1.5})\ell ||\boldsymbol{z}_1||)^2}{q}$$

where $\mathbf{R}_{0,j} \in \{-1, 1\}^{m \times m}$. Since vector $\boldsymbol{z}_1$ is sampled from Gaussian with width sq, so its two-norm is at least $\sqrt{m}(sq)$ with overwhelming probability. Then by Lemma 3.3.4, the distribution $\nu ||\mathbf{R}_{\mathsf{BP}^*}^* \boldsymbol{z}_1||^2/q$ is a Gaussian distribution with width at least

$$d = \tau \frac{(\eta sqm - \Theta(m^2\ell)sq)^2}{q} = \frac{\gamma\alpha^2(\eta sqm - \Theta(m^2\ell)sq)^2}{\beta^2 q}$$

We recall again that ν was sampled from a Gaussian with parameter $\tau = \gamma\alpha^2/\beta^2$. By our setting of parameters, we have $d/\omega(\log(n)) \geq 1$. A Gaussian with such width is statistically close to uniform in the domain $\mathbb{Z}_1$. This completes the proof. □

This completes the proof of Lemma 3.3.7. Further, Theorem 3.3.5 follows from Lemmas 3.3.6 and 3.3.7. A (flexibly) bi-deniable ABE from LWE then follows. □

3.3.3 Parameter Setting

The parameters in Table 3.3.1 are selected in order to satisfy the following constraints:

Parameters	Description	Setting
n, m	lattice dimension	$n = \lambda, m = n^2 \log n$
ℓ	length of input to branching program	$\ell = n$
q	modulus (resp. bit-precision)	smallest prime $\geq n^{1.5} m^{2.5} \omega(\log n)$
α	sampling error terms $\boldsymbol{e}, e$	$\frac{1}{n^{2.5} \log^3 n}$
β	sampling correlation vector $\boldsymbol{y}$	$\alpha/2$
γ	sampling correlation coefficient μ	$\frac{1}{n \log^{1.5} n}$
s	sampling secret key $\boldsymbol{r}$	$3\beta/2$
η	scaling parameter for $\mathbf{R}_{0,j}$	$\Theta(m\ell)$

Table 3.3.1: Parameter Description and Simple Example Setting

- To ensure correctness in Lemma 3.3.6, we have $\alpha s q m(\eta\sqrt{m} + 2m^{1.5}\ell) \leq 1/4$.
- To ensure deniability in Hybrid H_7, we have $d/\omega(\log(n)) > \frac{\gamma\alpha^2(\eta s q m - \Theta(m^2 \ell s q))^2}{\beta^2 q \omega(\log(n))} > 1$.
- To ensure large enough LWE noise, we need $\alpha \geq (\sqrt{n} \log^{1+\delta} n)/q$.
- To apply the leftover hash lemma, we need $m \geq 2n \log(q)$.
- To ensure that the matrix $\mathbf{Q}$ in FakeRCoins is positive definite, we have $\beta \geq \alpha\gamma\sqrt{\eta\sqrt{m} + 2m^{1.5}\ell}$; To ensure that the matrix $\mathbf{Q}'$ in the security proof is positive definite, we have $\alpha \geq \tau\beta\sqrt{\eta\sqrt{m} + 2m^{1.5}\ell}$. This constraint will also imply that in the security proof, both $\mathbf{Q}'$ and $\mathbf{Q}' - \beta'\mathbf{I}_{m\times m}$ are positive definite (note $\beta' = \alpha/2$).

- To ensure hybrids H_3 and H_5 are well-defined, we have $s > \beta$ and $\beta > s/2$. Let $s := (3/2)\beta$.

Regev [Reg05] showed that for $q > \sqrt{m}/\beta'$, an efficient algorithm for $\mathsf{LWE}_{n,m,q,\chi}$ for $\chi = \mathcal{D}_{\beta' q}$ $\left(\text{and } \beta' q \geq \sqrt{n}\omega(\log(n))\right)$ implies an efficient quantum algorithm for approximating the SIVP and GapSVP problems, to within $\tilde{O}(n/\beta')$ approximation factors in the worst case. Our example parameter setting yields a bi-deniable AB-BTS based on the (quantum) hardness of solving $\mathsf{SIVP}_{\tilde{O}(n^{9.5})}$, respectively $\mathsf{GapSVP}_{\tilde{O}(n^{9.5})}$. (We write this term to additionally absorb the $(1/q^2)$ loss from our LWE to eLWE^+ reduction.) We leave further optimizing the lattice problem approximation factor to future work.

3.3.4 From AB-BTS to Flexible Bi-Deniable ABE

We present the instantiation of a flexible bi-deniable ABE using our AB-BTS scheme described above. We let

$$\Sigma' = (\mathsf{Setup}', \mathsf{DenSetup}', \mathsf{Keygen}', \mathsf{SampleP}', \mathsf{SampleU}', \mathsf{TestP}', \mathsf{FakeRCoins}', \mathsf{FakeSCoins}')$$

be an AB-BTS scheme. Then the flexible bi-deniable ABE

$$\Sigma = (\mathsf{Setup}, \mathsf{DenSetup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{DenEnc}, \mathsf{Dec}, \mathsf{SendFake}, \mathsf{RecFake})$$

is:

- $\mathsf{Setup}(1^\lambda)$: Run algorithm $(\mathsf{pp}', \mathsf{msk}') \leftarrow \mathsf{Setup}'(1^\lambda)$ in AB-BTS and set $\mathsf{pp} = \mathsf{pp}', \mathsf{msk} = \mathsf{msk}'$.
- $\mathsf{DenSetup}(1^\lambda)$: Run algorithm $(\mathsf{pp}', \mathsf{msk}', \mathsf{fk}') \leftarrow \mathsf{DenSetup}'(1^\lambda)$ in AB-BTS and set $\mathsf{pp} = \mathsf{pp}', \mathsf{msk} = \mathsf{msk}', \mathsf{fk} = (\mathsf{fk}', \mathsf{msk}')$.
- $\mathsf{Keygen}(\mathsf{msk}, f)$: Run algorithm $\mathsf{sk}'_f \leftarrow \mathsf{Keygen}'(\mathsf{msk}, f)$ in AB-BTS and set $\mathsf{sk}_f = \mathsf{sk}'_f$.

- $\mathsf{Enc}(\mathsf{pp}, \boldsymbol{x}, \mu; (r_S^{(1)}, r_S^{(2)}))$: On input the message $\mu \in \{0,1\}$, if $\mu = 0$, then run $c_i \leftarrow \mathsf{SampleU}'(\mathsf{pp}, \boldsymbol{x}; r_S^{(i)})$ for $i = 1, 2$, otherwise, $\mu = 1$, run $c_1 \leftarrow \mathsf{SampleU}'(\mathsf{pp}, \boldsymbol{x}; r_S^{(1)})$ and $c_2 \leftarrow \mathsf{SampleP}'(\mathsf{pp}, \boldsymbol{x}; r_S^{(2)})$. Output $\mathsf{ct}_{\boldsymbol{x}} = (c_1, c_2)$.

- $\mathsf{DenEnc}(\mathsf{pp}, \boldsymbol{x}, \mu; (r_S^{(1)}, r_S^{(2)}))$: On input the message $\mu \in \{0,1\}$, then run $c_i \leftarrow \mathsf{SampleP}'(\mathsf{pp}, \boldsymbol{x}; r_S^{(i)})$ for $i = 1, 2$, otherwise, $\mu = 1$, run $c_1 \leftarrow \mathsf{SampleU}'(\mathsf{pp}, \boldsymbol{x}; r_S^{(1)})$ and $c_2 \leftarrow \mathsf{SampleP}'(\mathsf{pp}, \boldsymbol{x}; r_S^{(2)})$. Output $\mathsf{ct}_{\boldsymbol{x}} = (c_1, c_2)$.

- $\mathsf{Dec}(\mathsf{ct}_{\boldsymbol{x}}, \mathsf{sk}_f)$: If $f(\boldsymbol{x}) \neq 0$, then output $\perp$. Otherwise, parse $\mathsf{ct}_{\boldsymbol{x}} = (c_1, c_2)$ and run $b_i \leftarrow \mathsf{TestP}'(\mathsf{sk}_f, c_i)$ for $i = 1, 2$. Output 0 if the $b_1 = b_2$ and 1 if $b_1 \neq b_2$.

- $\mathsf{SendFake}(\mathsf{pp}, r_S, \mu, \mu')$: If $\mu = \mu'$, return r_S. If $(\mu, \mu') = (0, 1)$, then run $r_S^{*(2)} \leftarrow \mathsf{FakeSCoins}'(\mathsf{pp}, r_S^{(2)})$ and return $(r_S^{(1)}, r_S^{*(2)})$. Else if $(\mu, \mu') = (1, 0)$, run $r_S^{*(1)} \leftarrow \mathsf{FakeSCoins}'(\mathsf{pp}, r_S^{(1)})$ and return $(r_S^{*(1)}, r_S^{(2)})$.

- $\mathsf{RecFake}(\mathsf{pp}, \mathsf{fk}, \mathsf{ct}_{\boldsymbol{x}}, f, \mu')$: Parse $\mathsf{ct}_{\boldsymbol{x}} = (c_1, c_2)$ and use fk to decrypt the ciphertext $\mathsf{ct}_{\boldsymbol{x}}$ then obtain the plaintext μ. If $\mu = \mu'$, then run the honest key generation of the BTS scheme, i.e. $\mathsf{sk}'_f \leftarrow \mathsf{Keygen}'(\mathsf{msk}', f)$. Otherwise, run $\mathsf{sk}'_f \leftarrow \mathsf{FakeRCoins}'(\mathsf{pp}, \mathsf{fk}, c_{\mu+1}, f)$. Return sk'_f.

Similar to the work by Canetti et al. [CDNO97] and O'Neil et al. [OPW11], the following, desired theorem can be proven in a straightforward manner.

Theorem 3.3.15. *Assume that Σ' is a flexible bi-deniable AB-BTS, as in Definition 3.2.2. Then Σ is a flexibly bi-deniable ABE, as in Definition 3.2.1.*

Chapter 4

Weak is Better: Tightly Secure Short (Lattice) Signatures from Weak PRFs

4.1 Introduction

The Boyen-Li lattice-based signature scheme [BL16] from Asiacrypt 2016, is a theoretical breakthrough in terms of signature schemes with tight security reductions in the standard model.

Previous schemes achieving a reduction loss in security independent of the number of signing queries were either only proven secure in the random-oracle model [KW03, BLS01], required large signatures (of size at least quadratic in the security parameter) [BKKP15], or are insecure against quantum attacks [CW13, GHR99, BB08]. Their scheme manages to achieve short signatures (consisting of a single lattice vector in $\mathbb{Z}_q^m$) in the standard model with a loss in security depending (up to a constant factor) only on the loss from the security reduction of the pseudorandom function family (PRF) which underlies their scheme.

The starting point of their scheme is the Katz-Wang signature scheme, proven secure in the random oracle model [KW03]. Viewed at a high level, this scheme requires a pseudorandom function (PRF) PRF_k, a hash function H modeled as a random oracle, and

a trapdoor function f. The signer computes a signature σ by inverting the trapdoor at $H(\mathsf{PRF}_k(\mu)||\mu)$, and the verifier accepts if $f(\sigma) = H(b||\mu)$ for some $b \in \{0,1\}$. In the security proof, the random oracle is programmed so that it can only invert the trapdoor function f at $f^{-1}(H(\mathsf{PRF}_k(\mu)||\mu))$, while learning $f^{-1}((H(1-\mathsf{PRF}_k(\mu))||\mu))$ would allow it to solve some hard problem. If the PRF is secure, the latter case will happen with probability 1/2.

Boyen and Li notice that instead of using a random oracle, they can use the key-homomorphic trapdoor functions of Boneh et al. to encode the bits of the PRF key in the verification key [BGG+14]. To sign a message μ, they simply use the properties of the key-homomorphic trapdoor functions to homomorphically evaluate the PRF on μ, setting up the public key so they can only invert the trapdoor at $b = \mathsf{PRF}_{k,\mu}$. This enables them to gain tight security (with loss $1/2$) in the same manner as Katz and Wang, without having to resort to the random oracle heuristic.

Evaluation of non-trivial circuits over homomorphic trapdoor functions comes at a price. In particular, while the Boyen-Li scheme can be instantiated with AES or any other block cipher as the PRF, these appear to be less than ideal choices. The reason for this is because block ciphers such as AES appear to require relatively high circuit depth to evaluate homomorphically [GHS12b], which forces the scheme to be based on the hardness of approximating SIS to within a subexponentially large factor. Unlike SIS with polynomially large factors, this problem can be solved in subexponential time using the BKZ algorithm [SE94, CN11].

As an alternative to assuming the subexponential hardness of SIS, they recall that a function with a circuit in NC^1 can be evaluated using the standard lattice-based homomorphic trapdoor constructions with only polynomial growth in the size of the underlying trapdoor [BV14, GV15]. Those PRFs which are known to have an NC^1 circuit are mostly based on quantum insecure assumptions [NR99, NRR02].

The only potentially post-quantum PRF with an NC^1 circuit that we are aware of is the

ring-LWE (RLWE) based PRF of Banerjee et al. [BPR12], and indeed, this PRF is used by Boyen and Li for the quantum-secure instantiation of their scheme. The PRF of Banerjee et al., along with the quantum-insecure PRFs referenced above [1], all have a linear security loss in their reduction to the underlying hard problem on which they are based. As a result, Boyen and Li are able to instantiate their signature scheme with an overall loss in security linear in the underlying security parameter, but independent of the number of queries made by the adversary, which is a first for standard model lattice-based signature schemes with short signatures.

For completeness, we note that their paper also constructs a fully-secure IBE scheme enjoying very similar properties, where it enjoys some additional relative advantage to other such lattice-based schemes, as all such schemes require very large public keys (albeit not as large as theirs). However, our focus in this work is on the signature scheme only, as our main techniques do not transfer to the IBE setting.

4.1.1 Improving the Boyen-Li Scheme

Our main result is an improved version of the Boyen-Li signature scheme.

Better Parameters and Runtime. First, we improve greatly on the size of the public key and the hardness assumptions required. While our public key is still large (compared to some other lattice-based schemes), it is nevertheless a significant improvement over that of Boyen and Li. Although they never explicitly state the size of the public key in their work, it is clear that they need to encode the secret key for the PRF Banerjee et al. in the public key [BPR12], and furthermore, that at least one matrix is required in the public key for each bit of the PRF secret key in order to homomorphically evaluate the PRF using the ideas of Brakerski and Vaikuntanathan discussed above. The $nw \log^2 n = O(n^2 \log^2 n)$ value we give in Figure 4.1.1 then follows from the conditions required of the modulus of the PRF

[1] While they mention a DDH-based PRF construction by Jager that achieves a polylogarithmic loss in security, the paper containing it has since been withdrawn.

Scheme	Pub. Key $R_q^{1\times k}$ mat.	Signature R_q^k vec.	Reduct. loss	Assumption(s)
[CHKP12]	n	n	Q	$\mathsf{RSIS}(n^{3/2})$
[Boy10]	n	1	Q	$\mathsf{RSIS}(n^{7/2})$
[MP12]	n	1	Q	$\mathsf{RSIS}(n^{5/2})$
[BHJ+15]	1	d	$(Q^2/\epsilon)^c$	$\mathsf{RSIS}(n^{5/2})$
[DM14]	d	1	$(Q^2/\epsilon)^c$	$\mathsf{RSIS}(n^{7/2})$
[Alp15]	1	1	$(Q^2/\epsilon)^c$	$\mathsf{RSIS}(d^{2d}\cdot n^{11/2})$
[BL16]	$nw\log^2 n$	1	λ	$\mathsf{RSIS}(w^4n^{7/2}), \mathsf{RLWE}(wn^{w/2})$
This work	$n\log n$	1	1	$\mathsf{RSIS}(n^{7/2}), \mathsf{LWR}(n)$

To avoid clutter, we ignore constant parameters above. We write $\mathsf{SIS}(\cdot)$ to specify the SIS parameter, and $\mathsf{LWE}(\cdot), \mathsf{LWR}(\cdot)$ to specify the LWE noise ratio and LWR rounding ratio, respectively. For SIS parameters, we also ignore logarithmic parameters, i.e. we write the parameter in $\tilde{O}$ notation without the $\tilde{O}$. The comparison is in the ring setting because as written, some of these schemes are only realizable in the ring setting. For those schemes using confined guessing or variants thereof, d is a value satisfying $2Q^2/\epsilon < 2^{\lfloor c^d \rfloor}$ for a constant $c = 2$. For the Boyen-Li Scheme, w is a parameter representing the length of input messages, and $\delta > 1$ is such that the PRF of Banerjee et al. [BPR12] can be computed by an NC^1 circuit of depth $\delta \log w$.

Figure 4.1.1: Comparison to other standard model lattice-based signature schemes in the ring setting

scheme.

Furthermore, we contend that while our public key is indeed large compared to those in [DM14, Alp15], the reduction loss in security in those schemes is so great as to make their proofs of security somewhat vacuous, while our reduction loss in security is constant (6). For an adversary making 2^{40} queries and succeeding with advantage 2^{-40} (which would be considered a practical "break" of the scheme), the reduction would yield an attack on the underlying hard problems that succeeds with advantage only 2^{-280}, which is almost certainly too small to be considered practically meaningful. By contrast, an adversary making 2^{40} queries and succeeding with advantage 2^{-40} against our scheme would yield an attack against the underlying hard problems with advantage 2^{-43}, which is still quite meaningful. For the other known standard-model lattice-based signature schemes, our public key is bigger by at most a logarithmic factor.

We also gain significantly in terms of the size required for the modulus q (which itself

affects the size of the public key). They need to set the modulus to be at least $q = \omega(\lambda^{4(1+c)})$, where the given PRF can be evaluated in depth $d = c \log \lambda$, and it takes time $\Omega(\lambda^{2c+1})$ to evaluate the PRF homomorphically, even in the ring setting. The specific values of c for pseudorandom functions known to be in NC^1 [BPR12, BP14, NR04, DS15] are not explicitly investigated by Boyen and Li (or by the authors of the papers in which the PRFs appear). However, the paper by Banerjee et al describes three sequential steps of a multi-product of vectors, a discrete Fourier Transform, and a rounding step, each of which can be seen to require at least $\log n$ depth (since each step depends on all n input bits) [RT92], which suggests that c is at least 3.

Our scheme gains even further in terms of runtime, and we discuss this point further below in Section 4.1.2.

Tighter and Weaker Assumptions. In addition to a much smaller SIS parameter of $n^{7/2}$ versus what appears to be $O(n^{7/2}w^{12}) = O(\lambda^{31/2})$ in the Boyen-Li paper (based on the apparent depth of the underlying PRF), we avoid the need for relying on the very strong and somewhat questionably quantum-safe[2] assumptions like the hardness of RLWE for subexponentially large error ratios. Instead, our additional assumption is the hardness of the Learning with Rounding problem (LWR) over *general lattices* with a small (linear) rounding ratio. LWR over general lattices remains very plausibly quantum-hard for even for subexponential rounding ratios, and for the rounding ratio we require, we would expect any efficient (quantum) attack on LWR to be adaptable into an efficient attack on essentially all lattice-based cryptography; see Section 2.3.8 for some further justification.

[2]While no attacks on RLWE itself are known for subexponential large error ratios, Cramer, Ducas and Wesolowski recently gave a quantum polynomial-time algorithm for approximating the worst-case shortest vector on ideal lattices to within a factor of $\exp(\tilde{O}(\sqrt{n}))$ [CDW16], making the reduction from RLWE to worst-case ideal lattice problems essentially vacuous

4.1.2 Our Techniques

Our starting point is an investigation of the minimal security properties the function homomorphically evaluated in the Boyen-Li scheme must satisfy in order that the entire signature scheme be secure, i.e. existentially unforgeable against adaptive chosen-message attacks (eu-acma). We note that the adversary has the ability to choose which messages will be (homomorphically) evaluated by the function when computing the signature by simply asking the signing oracle to sign those messages. As a result, in order for the function's output to remain unpredictable by any means more successful than a random guess, the function must indeed be a strong pseudorandom function.

However, we recall that by hashing the message with a chameleon hash function [KR00] before signing, we can essentially eliminate an adversary's ability to choose which messages will be signed. In more detail, it has been shown [ST01] that a signature scheme that is existentially unforgeable against an adversary who can only observe signatures for (uniformly) *random* messages, can be turned into an euf-acma secure signature scheme by applying a chameleon hash function to a concatenation of the message with some auxiliary sampled randomness, signing the output of the chameleon hash function instead of directly signing the message, and including the auxiliary sampled randomness as part of the signature.

Weak Pseudorandom Functions. Once the ability of the adversary to choose messages has been eliminated and the adversary is limited to observing signature on messages sampled uniformly at random, we now see that the function being homomorphically evaluated need only be a *weak* pseudorandom function (W–PRFs) [DN02]. In contrast to strong pseudorandom functions, the output must only remain unpredictable against an adversary who can observe the output of the function on any polynomial number of messages chosen uniformly at *random*.

At a complexity theoretic level, there is very strong evidence that weak pseudoran-

dom functions are much simpler to compute than strong PRFs. In particular, Razborov and Rudich [RR94] have shown that any candidate strong pseudorandom function family computable in the complexity class $\mathsf{AC}^0(\mathsf{MOD}_2)$ of polynomial-size (in parameter λ) constant-depth circuit families with unbounded fan-in $\mathsf{AND}, \mathsf{OR}$ and MOD_2 gates can be only superpolynomially hard. In particular, they can be attacked by an adversary of size h with advantage at least $1/h$ for $h = O(\exp(\mathsf{poly}(\log n))$. By contrast, there exist candidate weak pseudorandom functions (that are plausibly exponentially hard) in this complexity class [ABG+14].

Instead of using the candidate weak pseudorandom functions in $\mathsf{AC}^0(\mathsf{MOD}_2)$, we instead opt for using the Learning With Rounding ($\mathsf{LWR}_{n,q,2}$) function family, for modulus $q = 2^\ell$ [BPR12]. Each function in the family is indexed by a secret $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, and on input $\mathbf{a} \in \mathbb{Z}_q^n$ outputs

$$f_{\mathbf{s}}(\mathbf{a}) = \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_2 = \lfloor \tfrac{2}{q} \langle \mathbf{a}, \mathbf{s} \rangle \rceil \bmod 2.$$

Under the assumption that $\mathsf{LWR}_{n,q,2}$'s output is indistinguishable from a truly random function (when evaluated on $\mathbf{a}$ sampled uniformly at random), known as the $decisional$-$\mathsf{LWR}_{n,q,2}$ assumption, we can immediately see that the LWR function is a pseudorandom function. It is also very easy to see that it is not a strong PRF, because one can learn the jth most significant bit of the ith coordinate of $\mathbf{s}$ by making a query on input $2^{j-1}\mathbf{e}_i$, where $\mathbf{e}_i$ is the ith vector of the standard basis.

Efficient Homomorphic Evaluation of LWR. The main reason we opt for using LWR as our weak PRF is that, viewed in the above manner, it is *identical* to the *decryption function* for most lattice-based encryption schemes [Reg09]. Homomorphic evaluation of the decryption function, of course, is better known in fully homomorphic encryption (FHE) contexts as bootstrapping, where it is a central operation necessary to allow unbounded homomorphic computation [Gen09]. A large body of work has focused on optimizing the evaluation of this function for various fully homomorphic encryption schemes [GHS12a,

AP13, OvdPS15, HS15], but for our purposes, we are particularly interested in those works focused on bootstrapping LWE ciphertexts using the Gentry-Sahai-Waters (GSW) [GSW13] encryption scheme [ASP14, HAO15, DM15, CGGI16].

This interests stems from the extreme similiarities between homomorphic evaluations over GSW ciphertexts and homomorphic evaluations over key-homomorphic trapdoor functions. In particular, as has been noted implicitly by Gorbunov and Vinayagamurthy, the cost of a given sequence of homomorphic operations in terms of error growth in GSW ciphertexts is essentially identical to the cost of that same sequence of operations in terms of trapdoor growth over key-homomorphic trapdoor functions [GV15].

As a result, algorithms using the GSW encryption scheme to boostrap LWE can in fact be used in our context to homomorphically evaluate the LWR function over key-homomorphic trapdoor functions. While FHE has generally become known for its inefficiency, and it is indeed very very far away from being useful for its main intended use, a single bootstrapping operation has recently moved much closer to efficiency. An implementation by Ducas and Miccancio [DM15] in the ring setting was notably able to implement bootstrapping for plausible security parameters in about 0.6 seconds. As homomorphic evaluation of the LWR function should be the most computationally intensive part of signing and verifying, this suggests plausibly being able to sign and verify in under a second.

We note that a more recent work by Chillotti et al. [CGGI16], to appear in Asiacrypt 2016, is faster than the work of Ducas and Micciancio for similar security parameters by a factor of 12, but does not, strictly speaking, perform its evaluation over full GSW ciphertexts, and hence we cannot reasonably extrapolate anything about the speed of our construction from it.

In addition to the more standard primitives of chameleon hash functions and weak pseudorandom functions (see below), we instantiate our signature scheme using Puncturable Homomorphic Trapdoor Functions (PHTDFs), which were introduced by Alperin-Sheriff [Alp15] as an extension of the Gorbunov et al. definition of homomorphic trapdoor

functions [GVW15b] and the Boneh et al. definition of key-homomorphic trapdoor functions [BGG+14]. PHTDFs give a formal way to encapsulate out repeatedly used properties in lattice-based signature schemes, and avoid having to repeat technical arguments in each new lattice-based signature schemes.

Reducing Trapdoor Growth Rates. In addition to the Boyen-Li scheme, key-homomorphic trapdoor functions [BGG+14] have been at the heart of wide variety of cryptographic constructions in recent years, including attributed-based encryption and predicate encryption schemes [BP14, GVW15a, GVW15b]. In the standard lattice-based construction, a key operation is sampling a matrix $\mathbf{X} \in \mathbb{Z}_q^{m \times m}$ with small norm such that $\mathbf{GX} = \mathbf{U} \pmod{q}$ for the so-called "gadget" matrix $\mathbf{G}$, which we denote $\mathbf{G}^{-1}(\mathbf{U})$.

In all existing schemes using key-homomorphic trapdoor functions, $\mathbf{G}^{-1}$ is evaluated via deterministic bit decomposition. While efficient (taking linear time) and fully parallelizable, it has downsides when trying to limit the noise growth (or trapdoor growth), as each column can have length as large as m. More concretely, when $\mathbf{G}^{-1}$ is evaluated in this deterministic manner, for a (fixed) vector $\mathbf{a} \in \mathbb{Z}^m$ and random vector $\mathbf{u} \in \mathbb{Z}_q^n$, we expect that $\|\langle \mathbf{a}, \mathbf{G}^{-1}(\mathbf{u}) \rangle\| \approx \frac{m}{2}\|\mathbf{a}\|$.

However, $\mathbf{G}^{-1}$ can also be evaluated in a randomized manner (sampled). Indeed, Miccancio and Peikert first defined and analyzed the gadget matrix $\mathbf{G}$ [MP12] with discrete Gaussian sampling applications in mind. Using Gaussian sampling instead of deterministic bit decomposition to evaluate $\mathbf{G}$ can reduce noise growth from m to $\sqrt{m}$ [ASP14]. This ultimately allows for basing the security of the given scheme on a harder problem and allows for smaller parameters for real security against known attacks.

For several reasons, this randomization technique has not been used in schemes utilizing key-homomorphic trapdoor functions. First, in all key-homomorphic trapdoor function-based schemes, the function evaluated on the master public and secret key by the key-generating authority when generating a given secret key (or signature) must correspond to

the function evaluated by a user holding that secret key (or signature) for a ciphertext (or a set of trapdoor functions) that the given secret key is authorized to decrypt (or that the given signature should be verifiable on). In particular, it necessary that both the key-generating authority and the user perform each evaluation of $\mathbf{G}^{-1}$ in the same manner. If a randomized version of $\mathbf{G}^{-1}$ is used, the bits used by they key-generating authority in evaluating $\mathbf{G}^{-1}$ will have to be transferred in some manner to the user holding the secret key. As many as n bits are required in order for a single sample to be within $2^{-\Omega(n)}$ distance of the actual discrete Gaussian distribution when using rejection sampling [GPV08]. One can also use tables to help with the computation (which is advisable because rejection sampling does not run in so-called "constant time"), but table lookups come with their own set of disadvantages [DG14].

We present a much simpler randomized method of sampling $\mathbf{G}^{-1}$ which is nearly as fast as bit decomposition and does not require the use of tables. We also discuss the manner in which the randomness is transferred to the user. In some schemes, one is "stuck" with using the trivial method of including the randomness in each signature as a separate element. While this ultimately requires a signature with more elements on a conceptual level, the reduction in the size of the modulus q and dimension n required for an equivalent level of security that the randomized sampling allows us will more than make up for this addition, resulting in less actual bits required per signature to achieve a given security level.

However, in schemes which already use a chameleon hash function to randomize the input messages, including but not limited to [Alp15, MP12, Boy10, DM14], we show that we do not need to include any additional elements in the signature. Instead, it is sufficient to include the randomness in the scheme's public key, and for the signing algorithm to "reuse" it on all signatures it issues, without more than a negligible loss in effectiveness in reducing the rate of noise growth.

4.1.3 Open Problems

While it provides massive improvements in the runtime of the signature scheme and improves the size of the parameters somewhat over the Boyen-Li Scheme, our scheme still requires a very large public key. The "holy grail" of achieving a standard-model signature scheme with a tight reduction to a plausible hardness assumption, while keeping both signatures and public keys small, remains open.

4.2 Improved Signature Scheme With Tight Security

In this section, we define our improved signature scheme, and prove security with a tight reduction. We also discuss how to homomorphically evaluate our weak PRF in an efficient manner.

To avoid the clunky notation that has characterized many lattice-based constructions (and indeed, many cryptographic constructions in general), we write the scheme generically in terms of the PHTDF cryptographic primitive, as well as in terms of a generic weak PRF and chameleon hash function, and then focus on the homomorphic evaluation details in Section 4.2.4.

4.2.1 Parameters

Here we discuss the basic parameters of the scheme and the requirements they need to satisfy. All parameters in the scheme are parameterized by an underlying security λ. We have that $\alpha, w = \theta(\lambda)$. We will have that $\mathcal{T} = \mathbb{Z}_q$. The form of the parameter $\kappa \geq 2$ depends on the concrete instantiation and evaluation of the PHTDF. We defer details to Section 4.2.4.

We need $\mathsf{W\text{–}PRF} : \mathcal{T}^{\kappa-2} \times \mathcal{T}^{w-1} \to \{0,1\}$ be an $(\epsilon_{\mathsf{W\text{–}PRF}}, t_{\mathsf{W\text{–}PRF}})$-secure weak pseudorandom function, where $\mathcal{T}^{\kappa-2}$ is the keyspace and $\mathcal{T}^{w-1}$ is the input space.

For convenience reasons, we also need the PRF to be trivial in the (negligibly likely) case that $\mathcal{T}^{k-2} = \mathbf{0}$; formally, we need $\mathsf{W\text{–}PRF}(\mathbf{0}, \cdot) = 0$ for all $\mathbf{y} \in \mathcal{T}^{w-1}$. This property is

satisfied naturally by LWR, and it and we can construct such a weak PRF from an arbitrary weak PRF W–PRF$'$ by setting

$$\mathsf{W\text{–}PRF}(\mathbf{s}, \mathbf{y}) := \mathsf{W\text{–}PRF}'(\mathbf{s}, \mathbf{y}) \oplus \mathsf{W\text{–}PRF}'(\mathbf{0}, \mathbf{y}).$$

We also need $\mathrm{CH} = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Hash}^{-1})$ to be a secure chameleon hash function family such that the input space $\mathcal{X}$ is efficiently sampleable and grows in size as a (polynomial) function inthe underlying security parameter λ and such that the output space $\mathcal{Y} = \mathcal{T}^{w-1}$. This can be achieved essentially without loss of generality, as long as there is an efficiently computable embedding of the output space $\mathcal{Y}$ into $\mathcal{T}^{w-1}$. We use $w - 1$ instead w because we will be appending a bit b to $\mathbf{y} \in \mathcal{T}^{w-1}$ as input to the function g that will be evaluated homorphically.

We define $g : \mathcal{T}^{\kappa} \times \mathcal{T}^{w} \to \mathcal{T}$ as

$$g((\mathbf{s}, z^{(0)}, z^{(1)}), (\mathbf{y}, b)) = (1 - z^{(b)}) - \mathsf{W\text{–}PRF}(\mathbf{s}, \mathbf{y}). \tag{4.1}$$

In the above equation, $\mathbf{y} \in \mathcal{T}^{w-1}$ will be the output of a chameleon hash function, and $\mathbf{s} \in \mathcal{T}^{\kappa-2}$ will be the secret key for the weak PRF.

Finally, we need

$$\mathrm{PHT} = (\mathrm{PHT.Gen}, \mathrm{PHT.GenTrap}, \mathrm{PHT}.f, \mathrm{PHT.Invert}, \mathrm{PHT.Eval}_{pk}^{td}, \mathrm{PHT.Eval}_{pk}^{func})$$

to be an $(\epsilon_{\mathrm{PHT}}, t_{\mathrm{PHT}}, g, \mathcal{S})$ CRP-secure PHTDF family, where $\mathcal{S} = \mathcal{T}^{\kappa-2}$.

4.2.2 Construction

We explicitly sample the tag encoded in the public key and add it to the signing key for use in signing. We also make the use of a chameleon hash function explicit, and we implicitly require the input space $\mathcal{X}$ for the chameleon hash function to satisfy shortness properties

(and be such that $|\mathcal{X}| = 2^{\Omega(\lambda)}$, which is indeed the case for the Ducas-Micciancio chameleon hash function family construction.

Encoding the PRF secret key. Often, including in the instantiation we detail below in Section 4.2.4, the way of encoding the weak PRF secret key $\mathbf{s}$ into the verification key will be something other than its "natural representation," and we abstract this out with the additional utility function $\mathsf{Encode} : \mathcal{T}^{\kappa} \to \mathcal{T}^{\ell}$.

$\mathsf{Gen}(1^{\lambda})$ On input security parameter λ,

1. Choose $v \leftarrow \mathcal{V}$, $\mathbf{s} \leftarrow \mathcal{T}^{\kappa-2}$. Set $z^{(0)} = 0, z^{(1)} = 1$.
   Compute $\tilde{\mathbf{z}} = \mathsf{Encode}(\mathbf{s}, z^{(0)}, z^{(1)}) \in \mathcal{T}^{\ell}$.
2. Compute $(ek, td) \leftarrow \mathsf{CH.Gen}(1^{\lambda})$, $pk \leftarrow \mathsf{PHT.Gen}(1^{\lambda})$
3. Compute $\{(a_i, r_i) \leftarrow \mathsf{PHT.GenTrap}(pk, \tilde{\mathbf{z}}_i)\}_{i \in [\ell]}$.
4. Output $vk = (pk, \mathbf{a} = \{a_i\}_{i \in [\ell]}, ek, v)$, $sk = (\mathbf{r} = \{r_i\}_{i \in [\ell]}, td, \mathbf{z})$

$\mathsf{Sign}(vk, sk, \mu)$ On input message $\mu \in \{0,1\}^{\alpha}$, secret key sk, verification key vk:

1. Sample the input randomness to the chameleon hash function $x \leftarrow \mathcal{X}$, compute the output $\mathbf{y} = \mathsf{CH.Hash}(x, \mu)$, and then evaluate the weak pseudorandom function at $\mathbf{y}$ to get $b = \mathsf{W\text{–}PRF}(\mathbf{s}, \mathbf{y})$.
2. Perform the homomorphic evaluation of g over the PHTDF to get

$$r^* \leftarrow \mathsf{Eval}_{pk}^{td}(g, (\mathbf{a}, \mathbf{r}), (\mathbf{y}, b)), \qquad a^* \leftarrow \mathsf{Eval}_{pk}^{func}(g, \mathbf{a}, (\mathbf{y}, b)).$$

3. Invert the trapdoor function to compute $u \leftarrow \mathsf{PHT.Invert}_{r^*, pk, a^*, x, s}(v)$. Output $\sigma = (u, x, b)$ as the signature.

$\mathsf{Ver}(vk, \mu, \sigma)$ On input message $\mu \in \{0,1\}^{\alpha}$, verification key vk, signature $\sigma = (u, x, b)$:

1. Compute $\mathbf{y} \leftarrow \mathsf{CH.Hash}(x, \mu)$.

2. Compute $a^* \leftarrow \mathsf{Eval}_{pk}^{func}(g, \mathbf{a}, (\mathbf{y}, b))$

3. Verify that the PHT.$\mathsf{Prop}(x)$

   PHT.$\mathsf{Prop}(u) \leq s$, and that PHT.$f_{pk,a^*,x}(u) = v$.

Correctness. Since $b = \mathsf{W\text{–}PRF}(s, \mathbf{y})$, we have that $g(s, (\mathbf{y}, b)) = 1-b-b = 1-2b = \pm 1$ whenever $b \in \{0, 1\}$. Since this will always be an invertible element of the ring $\mathcal{T} = R_q$, we have that PHT.Invert will invert successfully, and so verification will succeed with all but negligible probability.

4.2.3 Security

Security. We now prove that the signature scheme in Section 4.2.2 is secure.

Theorem 4.2.1. *Let $\mathcal{A}$ be a PPT adversary which breaks eu-acma security of the scheme in Section 4.2.2 in time t with advantage ϵ. Then there exists an $\mathcal{A}'$ which runs in time $t + \mathsf{poly}(\lambda)$ and breaks the security of one of the weak PRF, the chameleon hash function and the PHTDF with advantage at least $\epsilon/6$.*

Proof. We classify a successfully forging adversary into one of three mutually exclusive categories, and show that each category of adversary can be used for a successful attack on one of the cryptographic primitives underlying our scheme.

In more detail, our reduction then proceeds by guessing uniformly at random which of these six categories our adversary belongs to and running the attack below corresponding to that category of adversary. It is easy to see that if any adversary succeeds with probability ϵ in breaking our scheme, then we will succeed in breaking one of the three, underlying cryptographic primitives with advantage at least $(\epsilon - \mathsf{negl}(\lambda))/6$. This is because the probability that the adversary is successful *and* the simulator's guess is successful in each case is at least $(\epsilon - \mathsf{negl}(\lambda))/2$. (Concretely, we will present three cases, and the simulator may guess the correct sub-case matching the challenge bit b with probability 1/2 in each of these three cases.)

To complete our proof, we proceed to describe these categories of adversaries and how we use them to mount an "independent" attack on the underlying cryptographic primitives that succeeds except with probability negligible in the underlying security parameter λ.

Weak PRF. First, we consider the case that, conditioned on having forged successfully, the adversary outputs μ, $\sigma = (u, x, b)$ such that $\mathsf{W\text{–}PRF}(\mathbf{z}, (\mathsf{Hash}(x, \mu)) = b$. For this case, we use $\mathcal{A}$ to help us succeed in the underlying $\mathsf{W\text{–}PRF}$ security game against some challenger.

We change our behavior in the signature scheme's security game as follows:

1. Instead of encoding the challenger's actual $\mathsf{W\text{–}PRF}$ secret key (which we do not know), we set the secret key $\mathbf{z} = (\mathbf{s}, \mathbf{z}_{k-1}, \mathbf{z}_k)$ in our scheme to a dummy secret key of $(\mathbf{0}, 0, 0)$ in Gen, which will always allow us to sign regardless of the value of the $b = \mathsf{W\text{–}PRF}(\mathbf{s}, \mathbf{y})$.

   Since the a_i output by $\mathsf{PHT.GenTrap}$ are statistically close to uniform, independent of the $\mathbf{z}$ encoded in them, and everything else in the public key is completely independent of $\mathbf{z}$, this change is statistically indistinguishable.

2. Given a query for message μ, we send a query to the $\mathsf{W\text{–}PRF}$ challenger, receiving back a uniform random $\mathbf{y} \in \mathcal{T}^{w-1}$, $b = \mathsf{W\text{–}PRF}(\mathbf{s}, \mathbf{y})$ for some unknown $\mathbf{s}$. We then compute $x \leftarrow \mathsf{Hash}^{-1}(\mu, \mathbf{y})$ as usual. Since $\mathbf{y}$ is uniform, this is distributed within negligible statistical distance of the manner it is chosen actual signature scheme. Moreover, since by our condition on $\mathsf{W\text{–}PRF}$ we will have that $g(\mathbf{z}, (\mathsf{Hash}(x, \mu), b)) = 1 - z_{\kappa - b} - 0 = 1$, we can invert successfully and sign as usual.

As a result, our behavior in this game is statistically indistinguishable from that in the real game. At the end, if the adversary fails to output a valid signature, we choose a message μ^* and bit b^* uniformly at random and send them to the $\mathsf{W\text{–}PRF}$ challenger as our guess. Otherwise, if the adversary is successful and outputs a valid signature $\sigma = (u^*, x^*, b^*)$, we

send that μ^*, b^* to the challenger. Since the adversary will correctly predict b, we succeed in guessing the output of the W–PRF at μ^*.

Otherwise, a successfully forging $\mathcal{A}$ outputs μ^*, $\sigma^* = (u^*, x^*, b^*)$ such that $\mathsf{W\text{–}PRF}(\mathbf{z}, (\mathsf{Hash}(x^*, \mu^*)) \neq b^*$. In this case, we consider whether or not the forgery was achieved based on a hash collision.

Chameleon Hash Collision. We consider the case that $\mathcal{A}$ finds a hash collision by successfully outputting $\mu^*, \sigma = (u^*, x^*, b^*)$ such that $\mathsf{Hash}(x^*, \mu^*) = \mathsf{Hash}(x', \mu')$ for some x', μ' it already queried on, conditioned on a successful forgery and an unsuccessful weak PRF prediction. We now show how to use $\mathcal{A}$ to break the collision-resistance of the chameleon hash function family. We change our behavior in the security game as follows:

1. In Gen, we receive the evaluation key ek for the chameleon hash function from a challenger instead of generating it (along with a trapdoor) ourselves. The public key itself remains exactly the same (although we no longer have td in our secret key), so the adversary's view does not change.

2. In Step 1 of Sign, we instead sample $x \leftarrow \mathcal{X}$, and then compute $\mathbf{y} = \mathsf{Hash}(x, \mu)$. By the uniformity property of the collision-resistance hash function, this is within statistically negligible distance of the manner x and $\mathbf{y}$ are evaluated in the actual scheme.

Since our behavior is statistically indistinguishable from that in the real game, $\mathcal{A}$ will still output a forgery such that $\mathsf{Hash}(x^*, \mu^*) = \mathsf{Hash}(x', \mu')$ for some previously output x', μ'. We can then send x', x^*, μ', μ^*, succeeding in breaking the collision-resistance of the chameleon hash function.

PHTDF Collision When Punctured. Finally, if $\mathcal{A}$ has forged successfully, but has neither successfully predicting the weak PRF output nor output a successfull chameleon hash

collision, we can break collision-resistance when punctured (CRP)-security of the PHTDF. We change our behavior in the security game as follows:

1. During Gen, we choose a message μ' uniformly at random. Then, instead of choosing $v \leftarrow \mathcal{V}$, we invert the process, sampling $u' \leftarrow D_{\mathcal{U}}$, $x' \leftarrow \mathcal{X}$. We then let $\mathbf{y}' = \mathsf{CH.Hash}(x', \mu')$, $b = \mathsf{W\text{–}PRF}(\mathbf{s}, \mathbf{y}')$, and compute $a' \leftarrow \mathsf{Eval}_{pk}^{func}(g, \mathbf{a}, (\mathbf{y}', 1-b))$.

   Finally, we set $v \leftarrow f_{pk,a',x'}(u')$, and hold onto u' to use in forming our collision. By the statisticial distributional equivalence of inversion property of the PHTDF, this is statistically indistinguishable from having sampled $v \leftarrow \mathcal{V}$ as in the real security game. Moreover, we have that $g((\mathbf{s}, z^{(0)}, z^{(1)}), (\mathbf{y}', (1-b))) = 0$.

If the adversary outputs a successful forgery $(\mu^*, \sigma^* = (x^*, u^*, b^*))$, where $\mathsf{W\text{–}PRF}(\mathbf{s}, (\mathsf{Hash}(x^*, \mu^*)) = 1 - b$, then because $\mathcal{A}$ has no knowledge of the x' we sampled above, with all but probability $1/|\mathcal{X}| = \mathsf{negl}(\lambda)$, $x^* \neq x'$. If they are in fact not equal, we can output $u', x', \mathbf{y}', u^*, x^*, \mathbf{y}^* = \mathsf{Hash}(\mu^*, x^*)$ as our collision, breaking the CRP security of the underlying PHTDF except with probability negligible in λ. □

4.2.4 Efficient Evaluation of g

Here we discuss how to homomorphically evaluate the function

$$g(\mathbf{z}, (\mathbf{y}, b)) = (1 - z_{2-b}) - \mathsf{W\text{–}PRF}(\mathbf{z}, \mathbf{y})$$

using a cyclotomic ring-based instantiation of PHTDFs in a manner that is (somewhat) efficient in terms of both time and noise growth, for the specific case that $\mathsf{W\text{–}PRF}$ is the LWR function. This essentially boils down to efficiently evaluating the LWR function, as once $b = lwr_{k,Q,2}$ has been homomorphically computed, the remaining operations consist solely of additions. Our method of evaluation is very close to that of Ducas and Micciancio [DM15], but avoids any use of the general lattice setting for efficiency.

Lagrange Interpolation over R_q. To evaluate the rounding part of the $\mathsf{LWR}_{n,Q,2}$ function homomorphically, we will need to be able to evaluate a function over R_q (where $R = \mathcal{O}_m$) that sends ζ^i to 0 when $i \in [-Q/4, Q/4)$, and sends ζ^i to 1 otherwise.

To homomorphically evaluate $\mathsf{LWR}_{n,Q,2}$ (where $Q = 2^\ell = O(\lambda)$ for $\ell = \lceil \log_2(\lambda) \rceil$), it will be easiest to work directly over the 2^ℓth cyclotomic ring $R = \mathcal{O}[2^\ell]$; this restricts us to odd moduli q in the actual PHTDF instantiation over R_q. The actual evaluation process will then be as described in the work of Ducas and Micciancio [DM15], using $q = 3^\alpha$ for some α. We restate the relevant result here for LWR, recall again that evaluating LWR homomorphically over a PHTDF instantiated in the ring setting. is morally equivalent to evaluating the decryption function. Note that the conditions on parameter s assumes the use of the sampling ideas found Section 4.3 to minimize the size of parameter s, for modulus $q = 3^\alpha$.

Theorem 4.2.2 ([DM15]). *Let PHT be instantiated over R_q, where $R = \mathcal{O}[m]$, where q and m are coprime. Then $\mathsf{LWR}_{n,Q,2}$ is admissible with parameter $s = O(n^{1.5} \log Q)$, and can be evaluated with $O(n \log Q)$ homomorphic operations, with the secret key $\mathbf{z} \in \mathbb{Z}_Q^n$ encoded in $\kappa - 2 = n \log Q$ functions $a_1, \ldots, a_{\kappa-2}$ with associated tags $\tilde{z}_1, \ldots, \tilde{z}_{\kappa-2}$.*

We finally have the following corollary for a concrete instantiation using the PHTDF of Alperin-Sheriff (see Section 2.3.9).

Corollary 4.2.3 ([DM15]). *Let PHT be instantiated over R_q, where $R = \mathcal{O}[m]$, where q and m are coprime. Then the above signature scheme is secure as long as both of the following hold:*

- $\mathsf{LWR}_{n,Q,2}$ *is secure for* $n, Q = O(\lambda)$
- RSIS *is secure over* m *with parameter* $\beta = \tilde{O}(n^{7/2})$.

4.3 Reducing Trapdoor Growth

Here we give a sampling algorithm that is nearly as efficient as bit decomposition, has (slightly) *lower* rates of growth than samples generated as discrete Gaussians, and which requires exactly 2 random coins per element of the vector being sampled.

We later show how to reduce the total number of random coins needed per signature/encryption etc. to $O(\lambda)$.

4.3.1 Distribution Definition and Properties

Definition 4.3.1. *Let $q \geq 2 \in \mathbb{Z}$. Let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ such that linear combination of its columns generate all of $\mathbb{Z}_q^n$, and let $\mathbf{u} \in \mathbb{Z}_q^n$. We define $B_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{A}),s}$ as the distribution which outputs $\mathbf{z} \in \mathbb{Z}_q^m$, where each element of $\mathbf{z}$ is chosen to be some integer $x \in [-s, \ldots, +s]$ with probability $\frac{(s+1-|x|)}{(s+1)^2}$, conditioned on $\mathbf{Az} = \mathbf{u}$.*

For the remainder of Section 4.3, we focus only on $B_{\Lambda_u^{\perp}(\mathbf{g}^t),p-1}$, where $q = p^k$ and $\mathbf{g}^t$ is the "gadget" vector $\mathbf{g}^t = [1, p, \ldots, p^{k-1}] \in \mathbb{Z}_q^{1 \times k}$.

Lemma 4.3.2. *For all $u \in \mathbb{Z}_q$, $B_{\Lambda_u^{\perp}(\mathbf{g}^t),p-1}$ is a centered subgaussian with parameter $(p-1)\sqrt{k}$.*

Proof. Denote by $\mathbf{g}^{(j)}$ the first j vectors of $\mathbf{g}$. To show that the output $\mathbf{z} \leftarrow B_{\Lambda_u^{\perp}((\mathbf{g}^{(j)})^t),1}$ is centered, we proceed by induction on the first j vectors of $\mathbf{z}$, with the inductive assumption being that over the integers, $\langle \mathbf{g}^{(j)}, \mathbf{z} \rangle = u$ with probability $\max(0, \frac{p^j - |u|}{p^{2j}})$. It is not difficult to see that this inductive hypothesis immediately implies centering.

For $j = 1$, the condition is satisfied by the definition of the distribution. Now, we assume our hypothesis is true for all $k < j$. Let $\mathbf{x} = (\tilde{\mathbf{x}} \in \mathbb{Z}_q^{k-1}, x_k)$, and let Let $y = \langle \mathbf{g}^{(k)}, \mathbf{x} \rangle$ (over the integers), and let $y^* = \langle \mathbf{g}^{(k-1)}, \tilde{\mathbf{x}} \rangle$, so $y = y^* + x_k$. Let $u = y \bmod p^{k-1}$ be the unique coset representative of y in $\mathbb{Z}_{p^{k-1}}$, and let t be such that $p^{k-1}t + u = y$.

Then either $y^* = u$, $x^k = t$, or $y^* = u - \mathsf{sign}(u)p^{k-1}$ and $x^k = t + \mathsf{sign}(u)$. As a result,

the probability over the choice of $\mathbf{x}$ that $\langle \mathbf{g}^{(k)}, \mathbf{x} \rangle = y$ is

$$\begin{aligned}\rho &= \frac{(p^{k-1} - |u|)(p - |t|) + (p^{k-1} - |u - \mathsf{sign}(u)p^{k-1}|)(p - |t + \mathsf{sign}(u)|)}{p^{2k}} \\ &= \frac{p^k - u - p^{k-1}t}{p^{2k}} = \frac{p^k - y}{p^{2k}}.\end{aligned}$$

Thus, the inductive assumption is true for $j = k$, so the distribution is indeed centered conditioned on any u.

To see that it is subgaussian with parameter $(p-1)\sqrt{k}$, it is sufficient to note that for any $\mathbf{x}$ in the support of B, $\max_{\|\mathbf{u}\|=1} \langle \mathbf{u}, \mathbf{x} \rangle \leq (p-1)\sqrt{k}$. □

By the standard properties of subgaussian distributions and the form of the decomposition of $\mathbf{G}$, we immediately have the following corollary, which allows us to achieve noise growth rate $\tilde{O}((p-1)\sqrt{n})$ per multiply instead of $\tilde{O}((p-1)n)$ (assuming we can efficiently sample the distribution).

Corollary 4.3.3. *For all $u \in \mathbb{Z}_q$, $q = p^k$ $B_{\Lambda_{\mathbf{u}}^{\perp}(\mathbf{G}),(p-1)}$ is a centered subgaussian with parameter $(p-1)\sqrt{nk}$.*

The following lemma is necessary for showing how to actually sample $B_{\Lambda_u^{\perp}(\mathbf{G}),p-1}$ in practice.

Lemma 4.3.4. *For $\mathbf{x} \leftarrow B^k$, $\langle \mathbf{g}, \mathbf{x} \rangle$ is uniformly distributed modulo q.*

Proof. It is easy to see that the kth element of $\mathbf{x}$ randomizes the kth least significant mod-p digit of $\langle \mathbf{g}, \mathbf{x} \rangle + q/2$ modulo q, so $\langle \mathbf{g}, \mathbf{x} \rangle$ must be uniform modulo q. □

Sampling the Distribution. The distribution can be sampled with two different approaches, or a hybrid of the two, in the same manner used for sampling a discrete Gaussian distribution over $\Lambda_{\mathbf{u}}^{\perp}(\mathbf{G})$ as described in [MP12]. The advantage here is that instead of having to sample any discrete Gaussians in each coordinate with the various disadvantages

thereof, we can sample $B_{\mathbb{Z},p-1}$ in a very simple manner (and sample $B_{\mathbb{Z}^k,p-1}$ by sampling each coordinate independently according to $B_{\mathbb{Z},p-1}$).

Concretely, to generate a sample $B_{\mathbb{Z},p-1}$, we receive 2 (pseudo)random elements in $[0, 1, \ldots, p-1]$, viewed as a single element $r \in [0, 1, \ldots, p^2-1]$, and set the output x to be k with probability $|p-k|p^2$, mapping values in $[0, 1, \ldots, p^2-1]$ in the canonical manner. In particular,

$$x = \begin{cases} -k & \text{if } \frac{(p-(k)-1))(p-k)}{2} \leq r < \frac{(p-k)(p-k+1)}{2} \text{ and } r < \lceil p^2/2 \rceil \\ k & \text{if } \frac{(p-(k)-1))(p-k)}{2} \leq p^2 - 1 - r < \frac{(p-k)(p-k+1)}{2} \text{ and } r \geq \lceil p^2/2 \rceil \end{cases}$$

Given the above algorithm for sampling $B_{\mathbb{Z}^k,p-1}$, as stated, we have two options at the extremes of storage/parallelism trade-offs, as well as various hybrids between the two. The first is to precompute, sampling and storing many independent samples $\mathbf{x} \leftarrow B_{\mathbb{Z}^k,p-1}$. Assuming that values u for which samples of $B_{\Lambda_u^\perp(\mathbf{g}^t),p-1}$ are required are uniformly distributed modulo q (which will generally be the case in our desired application), and that we will require tq samples overall, by a coupon-collecting argument, we would have to store about $tq \log q$ samples to ensure we have enough on average, and tq^2 to ensure we have enough with overwhelming probability. For the second approach, we sample each coordinate one at a time in a randomized nearest-plane manner [GPV08, MP12]. Specifically, for $i = 1, \ldots, k$: choose $x_i \leftarrow B_{p\mathbb{Z}+u,s}$, and let $u \leftarrow (u - x_i)/p \in \mathbb{Z}$. This approach requires $\log q$ steps, but this is essentially no worse than bit decomposition because sampling B is so efficient. There is also a hybrid between the two approaches that involves choosing ℓ coordinates of $\mathbf{x}$ at a time for $1 < \ell < k$. For further details, see [MP12]. The key takeaway is that we can avoid the additional time and/or space costs of rejection sampling (which is potentially problematic in applications because it does not run in so-called "constant time") or storing tables of results that are required for sampling discrete Gaussians.

4.3.2 How to Inject Verifiable Randomness

Above, we showed how randomizing the computation of the homomorphic trapdoor functions can significantly reduce noise growth without significantly increasing computation time.

Here we discuss a semi-generic method of injecting verifiable randomness into the homomorphic computation itself, that can be applied in essentially all cryptographic schemes using key-homomorphic trapdoor functions. We also discuss a further optimized method of randomness injection that applies to $\mathbf{G}$-based signature schemes that use chameleon hash functions to randomize the messages being signed.

Generic Method

The trivial method of injecting verifiable randomness into the homomorphic computation is to simply include all of the random bits needed as part of the ciphertext or signature. While this approach may be acceptable for very simple functions, for more complex functions it will increase the size of the ciphertext or signature by an unacceptably large amount.

Instead, it is sufficient to simply include λ bits in each signature or ciphertext, where λ is the desired security parameter, and to stretch those bits to the necessary length with a pseudorandom number generator specified as a public parameter of the scheme in question.

Interestingly, a cryptographically secure pseudorandom number generator is *not* required for this purpose, as we do not need the generated bits to be computationally indistinguishable from random under *all* polynomial-time statistical tests. Instead, the role of the PRG here is similar to that of the PRF used by Hohenberger and Waters [HW09] in their short, stateless signature scheme, where they simply needed the output of the PRF to be randomly distributed with respect to several non-adversarial statistical tests.

Indeed, for our bounds from Section 4.3.1 to hold for a given pseudorandom number generator G, we simply need $B_{\Lambda_u^\perp(\mathbf{g}^t)}$ to remain a centered subgaussian with parameter $\sqrt{k}$ when the bits used in sampling B come from $G(\mathbf{x})$, where $\mathbf{x}$ is a seed chosen uniformly

at random. As a result, any heuristic pseudorandom number generator suitable for Monte Carlo simulations ought to be satisfactory.

4.3.3 Using Chameleon Hash Functions

While the method in Section 4.3.2 will already reduce the overall length of the outputs, it does have the downside of having to include λ bits of randomness in each individual signature or encryption. However, we can do better in signature schemes that already use chameleon hash functions to randomize the messages being signed. In this case, we show that it is sufficient to simply include an additional λ bits of randomness in the public key. We stress that the signatures remain stateless, as we reuse the same λ bits of verifiable randomness in the homomomorphic computation of each individual signature.

We now present a (semi)-formal transformation. While the conditions required of the transformation seem very specific, they do apply to several previous lattice-based signature schemes [DM14, Alp15] as well as to our scheme.

Let $\mathsf{SIG} = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ be a secure lattice-based signature scheme which applies a secure hash function $ch = (\mathsf{Gen}, \mathsf{Hash}, \mathsf{Hash}^{-1})$ to the messages and then deterministically evaluates a $\mathbf{G}$-based key-homomorphic trapdoor function requiring ℓ invocations of $\mathbf{G}^{-1}$ (meaning a total of $2\ell n^2 k^2$ random bits) in the signing and verification algorithms. We construct $\mathsf{SIG}' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ as follows:

$\mathsf{Gen}'(1^\lambda)$ Run Gen, sample λ random bits, choose some secure pseudorandom generator PRG and output the λ random bits $\mathbf{b}$ along with a description of the pseudorandom generator.

$\mathsf{Sign}'(\mu)$ Run Sign, but evaluate the $\mathbf{G}$-based key-homomorphic functions using the $2\ell n^2 k^2$ bits output by $\mathsf{PRG}(\mathbf{b})$. Output signature σ.

$\mathsf{Ver}'((\mu, \mathbf{SIG}))$ Run Ver, but evaluate the $\mathbf{G}$-based key-homomorphic functions using the $2\ell n^2 k^2$ bits output by $\mathsf{PRG}(\mathbf{b})$. Output the result of Ver.

For stating our result, it will be convenient to view the signature scheme as being instantiated by a PHTDF which evaluates some function g admissible with parameter s. All standard model lattice-based signature schemes that we are aware of which use $\mathbf{G}$-based trapdoors can be described in this manner.

Theorem 4.3.5 (Informal)**.** *If SIG is evaluates a function admissible with parameter s, then $SIG' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Ver}')$ evaluates a function admissible with parameter $\sqrt{s}$.*

Proof. That the scheme remains secure is straightforward. The only change is the manner in which $\mathbf{G}^{-1}$ is evaluated, and this depends only on the extra λ bits $\mathbf{b}$ in the public key and the pseudorandom generator PRG. In particular, the evaluation is performed entirely independently of the scheme's secret key. As a result, this change leaks no additional information, and so the scheme remains secure.

It remains to show that the function is admissible with parameter $\sqrt{s}$. If $\mathbf{G}^{-1}$ were evaluated with truly random bits, this would informally follow by noting the growth rate of $\mathbf{G}^{-1}$ in Corollary 4.3.3 versus the growth rate from bit decomposition.

First, consider an individual message query μ by the adversary. Since the scheme uses chameleon hash functions, the actual (hashed) messages being signed are randomized, and in particular, chosen in a manner independent of the λ random bits in the public key.

As a result, we may view the message as having been chosen first, and the λ random bits as having been chosen subsequently and uniformly at random and independently. As a result, as above, the probability that the function is admissible with parameter $\sqrt{s}$ using $\mathsf{PRG}(\mathbf{b})$ as our source of randomness is at most negligibly smaller than the probability that the function is admissible with parameter $\sqrt{s}$ using truly random bits as our source of randomness. To reiterate, this is because a noticeable difference in the probabilities would mean that we have a statistical test that is entirely independent of the seed $\mathbf{b}$, and would thus break the assumed pseudorandomness of PRG.

A simple union bound over the $Q = \mathsf{poly}(\lambda)$ message queries means that the scheme remains correct except with negligible probability. □

Chapter 5

5Gen: A Framework for Prototyping Applications Using Multilinear Maps and Matrix Branching Programs

5.1 Introduction

A multilinear map (mmap) [BS02] is an extremely powerful tool for constructing advanced cryptographic systems including program obfuscation [GGH+13b], n-party non-interactive key exchange [BS02], multi-input functional encryption [GGG+14, BLR+15], optimal broadcast encryption [BWZ14a], witness encryption [GGSW13], and many others. The recent emergence of candidate mmaps [GGH13a, CLT13, CLT15, GGH15] bring these proposals closer to reality, although several of the current candidates have been shown to be too weak for some of these applications, as discussed in Section 5.4.

Despite the remarkable power of mmaps, few published works study the efficiency of the resulting applications, primarily due to the rapid pace of development in the field and the high resource requirements needed for carrying out experiments. In this chapter we develop a

generic framework called `5Gen`[1] (available at `https://github.com/5GenCrypto`) that lets us experiment with powerful applications of current and future mmaps. We focus on two applications in particular: multi-input functional encryption (MIFE) and program obfuscation, both of which can be instantiated with some of the existing mmap candidates (see Section 5.4). Our framework is built as a multi-layer software stack where different layers can be implemented with any of the current candidates or replaced altogether as new constructions emerge.

The top layer of our framework is a system to compile a high-level program written in the Cryptol language [Cry] into a matrix branching program (MBP), as needed for the most efficient MIFE and obfuscation constructions. We introduce several novel optimizations for obtaining efficient MBPs and show that our optimizations reduce both the dimension and the total number of matrices needed.

The next layer implements several variants of MIFE and obfuscation using a provided MBP. This lets us experiment with several constructions and to compare their performance.

The lowest layer is the multilinear map library, `libmmap`. We demonstrate our framework by experimenting with two leading candidate mmaps: GGHLite [GGH13a, LSS14, ACLL15] and CLT [CLT13, CLT15]. Our experiments show that for the same level of security, the CLT mmap performs considerably better than GGHLite in all the applications we tried, as explained in Section 5.7. (Although the GGH15 multilinear map [GGH15] was not included in our implementation or experiments, we hope that future work might integrate this multilinear map into our framework.)

Our framework makes it possible to quickly plug in new mmaps as new proposals emerge, and easily measure their performance in applications like MIFE and obfuscation.

MIFE experiments. Recall that functional encryption [BSW11] is an encryption scheme where the decryption key sk_f is associated with a function f. If c is the encryption of

[1] The name `5Gen` comes from the fact that multilinear maps can be considered the "fifth generation" of cryptography, where the prior four are: symmetric key, public key, bilinear maps, and fully homomorphic encryption.

message m then decrypting c with the key sk_f gives the decryptor the value $f(m)$ and nothing else. An n-input MIFE scheme is the same, except that the function f now takes n inputs. Given independently created ciphertexts $c_1, \ldots, c_n$, with each c_i an encryption of a message m_i and associated with "slot" i, decrypting these ciphertexts using sk_f reveals $f(m_1, \ldots, m_n)$ and nothing else.

One important application of 2-input MIFE is *order-revealing encryption* (ORE) [GGG+14, BLR+15]. Here the function $f(x, y)$ outputs 1 if $x < y$ and 0 otherwise. Thus, the key sk_f applied to ciphertexts c_1 and c_2 reveals the relative order of the corresponding plaintexts. ORE is useful for responding to range queries on an encrypted database. For large domains, the only known constructions for secure ORE are based on mmaps. We conduct experiments on ORE using real-world security parameters where mmaps give the best known secure construction.

We also experiment with 3-input MIFE. Here, we choose a DNF formula f that operates on triples of inputs, which is useful in the context of privacy-preserving fraud detection where a partially trusted gateway needs to flag suspicious transactions without learning anything else about the transactions (see Section 5.5.3). Again, the best known construction for such a scheme uses mmaps.

We use our framework to evaluate the implementation of these schemes using existing mmaps for which they are currently believed to be secure. Clearly these systems are too inefficient to be used in practice. Nevertheless, our experiments provide a data point for the current cost of using them. Moreover, our framework makes it possible to easily plug in better or more secure mmaps as they become available.

Obfuscation experiments. Roughly speaking, an obfuscator takes as input a program and outputs a functionally equivalent program such that the only way to learn information about the program is to run it. We experiment with several obfuscators built on the obfuscator described by Barak et al. [BGK+14], including those inspired by Sahai and Zhandry [SZ14] and Ananth et al. [AGIS14]. These improvements allow for obfuscation of

a point function with increased security at less than half the total obfuscation size reported by Apon *et al.* [AHKM14]. We also implemented the Zimmerman [Zim15] obfuscator, but we ultimately found that it was too inefficient for the settings that we consider in our experiments.

5.1.1 Our Contributions

Summarizing, we make the following contributions:

- An optimizing compiler from programs written in the Cryptol language to MBPs, which are used in many mmap applications including MIFE and obfuscation. Our compiler uses optimizations such as dimension reduction, matrix pre-multiplication, and condensing the input representation, and solves a constraint-satisfaction problem needed to obtain the most efficient MBP. See Section 5.3 for details.

- A library providing a clean API to various underlying mmap implementations. This allows researchers to experiment with different mmaps, as well as to easily plug future mmaps into our framework. See Section 5.4 for more details.

- A general MIFE construction based on the scheme of Boneh et al. [BLR+15] using real-world security parameters. We contribute optimized implementations of two-input MIFE (in particular, order-revealing encryption) and three-input MIFE (in particular, a functionality needed for privacy-preserving fraud detection), as well as performance results that characterize our constructions. See Section 5.5 for details and Section 5.7 for evaluation results.

- Obfuscation constructions [BGK+14, SZ14, AGIS14, Zim15] using real-world security parameters. We experiment with obfuscating point functions and evaluate their performance. See Section 5.6 for details and Section 5.7 for evaluation results.

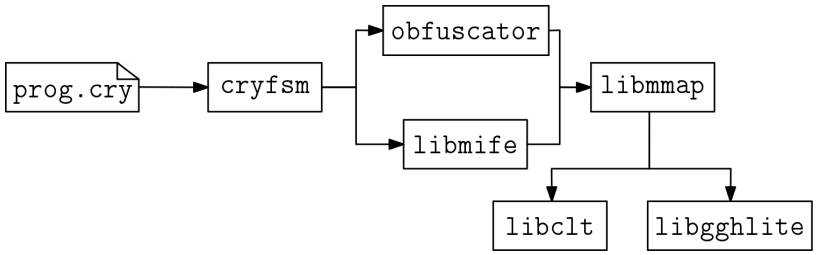

Figure 5.2.1: Framework architecture. We use `cryfsm` to compile a Cryptol program (here denoted by `prog.cry`) to an MBP, which can either be used as input into our MIFE implementation or our obfuscation implementation. Both these implementations use `libmmap` as a building block, which supports both the CLT (`libclt`) and GGHLite (`libgghlite`) mmaps.

5.1.2 Related Work

Several groups have previously implemented mmaps [ACLL15, CLT13, CLT15] to experiment with their performance. However, they did not go so far as experimenting with cryptographic applications of mmaps beyond direct applications such as multi-party non-interactive key exchange. The goal of our work is to explore the performance of more advanced applications such as MIFE and obfuscation. An earlier work implementing obfuscation [AHKM14] experimented with obfuscating point functions, and was only able to successfully obfuscate a 14-bit point function.

Our work builds on the vast amount of previous work showing applications of mmaps—most notably, MIFE [GGG+14, BLR+15] and obfuscation [BR14, BGK+14, PST13, AGIS14, Zim15, AB15, GLSW15, BMSZ15, Lin16, LPST16, GMS16, MSZ16b, DGG+16].

5.2 Framework Architecture

Our framework incorporates several software components that together enable the construction of applications using mmaps and MBPs. In particular, we use our framework to develop implementations of MIFE and program obfuscation. See Figure 5.2.1 for the framework architecture.

The top layer of our framework, `cryfsm`, takes as input a program written in Cryptol [Cry], a high-level language designed to express manipulations over bitstreams in a

concise syntax, and compiles the program into an MBP. This process, and the various optimizations we introduce, are described in more detail in Section 5.3.

The bottom layer of our framework, `libmmap`, provides an API for using various mmaps, which in our case includes the CLT (through the `libclt` library) and GGHLite (through the `libgghlite` library) mmaps. The `libmmap` library, which we describe in Section 5.4, is also designed to allow for a straightforward integration of future mmap implementations.

We combine the above components to realize various applications of mmaps and MBPs: in particular, MIFE and program obfuscation. We demonstrate the applicability of our MIFE implementation (cf. Section 5.5) through two examples: order-revealing encryption (ORE) and three-input DNF (3DNF) encryption. We implement program obfuscation based on two main approaches: the techniques described by Sahai and Zhandry [SZ14], and also the scheme by Zimmerman [Zim15], which operates over arithmetic circuits, but only applies to the CLT mmap.

5.3 From Programs to MBPs

One of our key contributions in this work is a compiler, `cryfsm`, that takes as input a program written in Cryptol [Cry], a domain-specific language for specifying algorithms over generic streams of bits, and produces an MBP for the given input program. `cryfsm` does this by translating a Cryptol specification into a *layered state machine*, which can then be transformed into an optimized corresponding MBP.

Our toolchain proceeds as follows. The user writes a Cryptol function of type `[n] -> Bit` for some `n` (that is, the function takes `n` input bits and produces one output bit). This function is interpreted as deciding membership in a language. The toolchain symbolically evaluates this function to produce a new version of the function suitable for input to an SMT solver, as explained in detail below. Queries to the SMT solver take the form of deciding the prefix equivalence relation between two initial bitstrings, which is sufficient to build the

minimal layered state machine, which we then convert to an MBP.

Our solver-based approach results in a substantial dimension reduction of the corresponding output MBPs that we tested. In contrast, the traditional approach would be to heuristically optimize the state machine design in an attempt to achieve a best-effort optimization. The dimension reduction we achieve recovers the most efficient known MBPs for several previously studied bit-string functions, including MBPs for point functions that are smaller than the MBPs constructed from boolean formulas using existing techniques (e.g., [SZ14]). In the remainder of this section, we describe the key steps in this toolchain, along with several optimizations to the MBPs that we use throughout the remainder of this work.

Specifying functions in Cryptol. Cryptol is an existing language widely used in the intelligence community for describing cryptographic algorithms. A well-formed Cryptol program looks like an algorithm specification, and is executable. The Cryptol tool suite supports such execution, along with capabilities to state, verify, and formally prove properties of Cryptol specifications, and capabilities to both prove equivalence of implementation in other languages to Cryptol specifications and automatically generate such implementations. In our work, a user specifies an MBP in Cryptol, and then we use `cryfsm` to transform the high-level specification into a minimal layered state machine, and further transform it into an efficient MBP.

Minimal layered state machines. There is a standard translation from traditional finite state machines to MBPs: create a sequence of matrix pairs (or matrix triples for three-symbol alphabets, etc.) that describe the adjacency relation between states. If state i transitions to state j on input symbol number b, then the bth MBP matrix will have a 1 in the ith row and jth column and 0 elsewhere. For many languages of interest, this is inefficient: for an automaton with $|S|$ states, each matrix must be of size $|S|^2$, even though many states may be unreachable.

In the applications of mmaps that we study in this work, we consider functions on inputs of a fixed length. Hence, for a positive integer n, we can take advantage of this property by

restricting ourselves to *layered state machines* of depth n, which are simply (deterministic) finite state machines that only accept length-n inputs. Here, the ith "layer" of transitions in the machine is only used when reading the ith digit of the input. As a result, layered state machines are acyclic.

To generate minimal layered state machines, our compiler must introduce machinery to track which states are reachable at each layer, which allows us to reduce the overall MBP matrix dimensions. To do this, `cryfsm` computes the quotient automaton of the layered state machine using an SMT solver to decide the state equivalence relation. The quotient automaton is then used as the new minimal layered state machine for the specified function. Then, from a layered state machine of depth n, we construct the corresponding MBP on base-d inputs of length n in a manner essentially equivalent to the techniques of Ananth *et al.* [AGIS14] for constructing layered branching programs. Intuitively, for each $i \in [n]$ and $j \in [d]$, the ith matrix associated with the jth digit is simply the adjacency matrix corresponding to the transitions belonging to the ith layer of the machine, associated with reading the digit j. Then, the "final matrix" (that defines the output of the MBP being 1) is simply the adjacency matrix linking the initial state to the final state of the layered state machine.

Optimizations for MBP creation. Boneh et al. [BLR+15] describe a simple five-state finite state machine appropriate for ORE applications, and describe the translation to MBPs that produces 5×5 matrices at each depth. The MBP we build and use for our ORE application differs from this one via three transformations that can be generalized to other programs: change of base, matrix premultiplication, and dimension reduction. Of these, matrix premultiplication and dimension reduction are a direct consequence of the technique used by `cryfsm` for constructing MBPs and therefore automatically apply to all programs, whereas choosing an input base remains a manual process because it must be guided by outside knowledge about the performance characteristics of the mmap used to encode the MBPs. While the change of base and matrix pre-multiplication optimizations are described

by Boneh *et al.*, we introduce dimension reduction as a new optimization that is useful for ORE yet generalizable to other applications.

For each optimization, we use the integer d to represent the "input base", the integer n to represent the length (number of digits) of each input, the integer N to represent the input domain size (so, we have that $d^n \geq N$), the integer m to represent the length of the MBP, and the integer M to represent the total number of elements across all the matrices of the MBP.

At a high level, the optimizations are as follows.

- **Condensing the input representation** corresponds to processing multiple bits of the input, by increasing d, to reduce the length of the MBP, at the expense of increasing the number M of total elements.
- **Matrix premultiplication** also aims to reduce the parameter m, but without increasing the parameter M.
- **Dimension reduction** aims to directly reduce the number M of total elements, but may not be fully compatible with matrix premultiplication, depending on the function.

To help with the understanding of the intuition behind these optimizations, we use the simple comparison state machine as a running example—however, we stress that these optimizations are in no way specific to the comparison function, and can be applied more generally to any function expressed as a layered state machine.

Condensing the input representation. The most immediate optimization that we apply is to condense the representation of inputs fed to our state machines. MBPs are traditionally defined as operating on bitstrings, so it is natural to begin with state machines that use bits as their alphabet, but using larger alphabets can cut down on the number of state transitions needed (at the potential cost of increasing the state space).

As an example, for evaluating the comparison state machine, this optimization translates to representing the input strings in a larger base $d > 2$, and to adjust the comparison state

machine to evaluate using base-d representations. The resulting state machine consists of $d + 3$ total states.

A naive representation of an input domain of size N with a state machine that processes the inputs bit-by-bit (in other words, $d = 2$) would induce an MBP length of $m = 2 \cdot \lceil \log_2(N) \rceil$ and $M = 50 \cdot m$ total elements (in two 5×5 matrices). However, by using the corresponding comparison state machine that recognizes the language when the inputs are in base-d, we can then set $m = 2 \cdot \lceil \log_d(N) / \log_2(d) \rceil$ and $M = 2 \cdot (d + 3)^2 \cdot m$.

Concretely, setting $N = 10^{12}$, without condensing the input representation, we require $m = 80$ and $M = 2000$ for the resulting MBP. However, if we represent the input in base-4, we can then obtain $m = 20$ and $M = 1960$, a strict improvement in parameters.

Matrix premultiplication. Boneh *et al.* [BLR+15] informally describe a simple optimization to the comparison state machine, which we explain in more detail here. The natural state machine for evaluating the comparison function on two n-bit inputs x and y reads the bits of x and y in the order $x_1 y_1 x_2 y_2 \cdots x_n y_n$.

However, Boneh *et al.* show that a slight reordering of the processing of these input bits can result in reduced MBP length without compromising in correctness. When the inputs are instead read in the following order:

$$x_1 y_1 y_2 x_2 x_3 y_3 \cdots y_n x_n, \tag{5.1}$$

then, rather than producing one matrix for each input bit position during encryption, the two matrices corresponding to y_1 and y_2 can be pre-multiplied, and the result is a *single* matrix representing two digit positions. Naturally, this premultiplication can be performed for each pair of adjacent bit positions belonging to the same input string (such as for $x_2 x_3$, $y_3 y_4$, and so on), and hence the number of matrices produced is slightly over half of the number of matrices in the naive ordering of input bits.

As a result, for evaluating the comparison state machine, where n is the length of

the base-d representation of an input, applying this optimization implies $m = n + 1$, a reduction from the naive input ordering, which would result in $m = 2n$, and a reduction from $M = 2 \cdot (d+3)^2 \cdot m$ to $M = (d+3)^2 \cdot m$. When applying this optimization in conjunction with representing the input in base $d = 4$, for example, setting $N = 10^{12}$ only requires $m = 21$ and $M = 1029$, a huge reduction in cost that was emphasized by Boneh *et al.*, and another strict improvement in parameters.

A new optimization: dimension reduction. We now describe a more sophisticated optimization that can be applied to general MBPs which also results in a reduced ciphertext size. As an example, we describe this optimization, called dimension reduction, as it applies to the comparison function state machine (without applying the reordering of input bits from matrix premultiplication), but we emphasize that the technique does not inherently use the structure of this state machine in any crucial way, and can naturally be extended to general MBPs.

Our new optimization stems from the observation that, for each bit position in the automaton evaluation, the transitions in the automaton do not involve all of the states in the automaton. This is the same observation that motivates the use of layered state machines over finite state machines.

In particular, for the even-numbered bit positions, the transitions map from a set of d states to a set of only 3 states. Similarly, for the odd-numbered bit positions, the transitions map from a set of (at most) 3 states to a set of d states. As a result, the corresponding matrices for each bit position need only be of dimension $d \times 3$ or $3 \times d$ (depending on the parity), as opposed to the naive interpretation of the Boneh *et al.* construction which requires matrices of dimension $(d+3) \times (d+3)$.

Note, however, that the dimension reduction optimization is not fully compatible with matrix premultiplication, since the effectiveness of dimension reduction can degrade if matrix premultiplication is also applied. In particular, when applying matrix premultiplication to the comparison state machine, we notice that there is less room for improvements with

dimension reduction, as the transitions for the position $y_1 y_2$ correlate from a domain of d states to a range of also d states.

In Section 5.5.1, we concretely show how to apply a mixture of these optimizations to the comparison automaton, and then use these optimizations to obtain asymptotically shorter ciphertexts for order-revealing encryption.

5.4 A Library for Multilinear Maps

In this section we describe our library, `libmmap`, which provides an API for interacting with different mmap backends. In this work we implement GGHLite (`libgghlite`) and CLT (`libclt`) backends[2], although we believe that it should be relatively straightforward to support future mmap implementations.

The `libmmap` library exports as its main interface a virtual method table `mmap_vtable`, which in turn contains virtual method tables for the public parameters (`mmap_pp_vtable`), the secret key (`mmap_sk_vtable`), and the encoded values (`mmap_enc_vtable`). Table 5.4.1 lists the available functions within each table. Each underlying mmap library must export functions matching these function interfaces and write a wrapper within `libmmap` to match the virtual method table interface. A user of `libmmap` then defines a pointer `const mmap_vtable *` which points to the virtual method table corresponding to the mmap of the user's choice (in our case, either `clt_vtable` or `gghlite_vtable`). In the following, we describe the two mmap schemes we support within `libmmap`: `libgghlite` (Section 5.4.1) and `libclt` (Section 5.4.2).

Figure 5.4.1 presents estimates for the size of an encoding using GGHLite and CLT for security parameters $\lambda = 80$ and $\lambda = 40$. We describe our parameter choices for arriving at these estimates in Section 5.9. As we can see, the CLT mmap produces smaller encodings than GGHLite as we vary both λ and κ. This appears to be due to the growth of the lattice

[2]We also have a "dummy" mmap implementation for testing purposes.

vtable	function	comments
`mmap_pp_vtable`	`fread/fwrite`	read/write public parameters
	`clear`	clear public params
`mmap_sk_vtable`	`init/clear`	initialize/clear secret key
	`fread/fwrite`	read/write secret key
`mmap_enc_vtable`	`init/clear`	initialize/clear encoding
	`fread/fwrite`	read/write encoding
	`set`	copy encoding
	`add`	implements Add
	`mul`	implements Mult
	`is_zero`	implements ZeroTest
	`encode`	implements Encode

Table 5.4.1: Interfaces exported by the `libmmap` library.

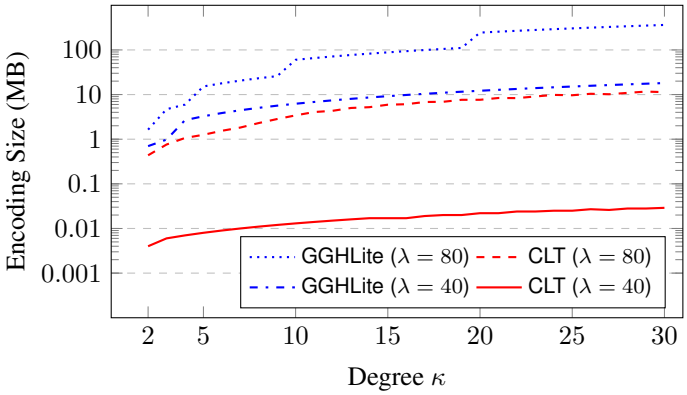

Figure 5.4.1: Estimates for the size of a single encoding in megabytes (MB) produced for security parameters $\lambda = 80$ and $\lambda = 40$ and varying the multilinearity degree $\kappa \in [2, 30]$ for the GGHLite and CLT mmaps.

dimension in GGHLite compared to the number of secret primes required by the CLT scheme, among other factors.

5.4.1 The GGHLite Multilinear Map

Building off of the original mmap candidate construction of Garg *et al.* (GGH) [GGH13a], Langlois *et al.* [LSS14] proposed a modification called GGHLite, along with parameter and performance estimates for the resulting encodings of the scheme. More recently, Albrecht *et al.* [ACLL15] proposed further modifications and optimizations on top of GGHLite, along

with an implementation of their scheme under an open-source license. In this work, we refer to GGHLite as the construction from the work of Albrecht *et al.*, as opposed to the original work of Langlois et al.

Our GGHLite implementation. We use as our starting point the implementation of GGH-Lite[3] released by Albrecht et al. [ACLL15]. We modified this implementation to add functionality for handling the reading and writing of encodings, secret parameters, and public parameters to disk. We also extended the implementation to handle more expressive index sets, which are used in MIFE and obfuscation, as follows.

Typically, multilinear maps only support "levels", where each encoding is created with respect to an integer $i \in [\kappa]$ (for an mmap of degree κ). The GGHLite implementation supports more advanced labelings of encodings, by allowing for a universe U of κ indices to be defined, and each encoding can be created with respect to a singleton subset (containing only one element) of this universe U. Multiplication of two encodings with respect to sets of indices S_1 and S_2 produces an encoding with respect to the multiset union of S_1 and S_2. The zero-testing parameter is then created to test for encodings which are labeled with respect to U. However, this functionality is still not sufficiently expressive to match the needs of our implementation and our definition of mmaps.

Consequently, we upgraded the handling of these encodings to support labelings of an encoding with respect to *any subset* S of indices of the universe U. Then, when two encodings labeled with two different subsets are multiplied, the resulting encoding is labeled with respect to their multi-set union. Finally, as before, the zero-testing parameter allows to check for encodings of 0 labeled at U, only.

Finally, we isolated and rewrote the randomness generation procedures used by GGHLite, since the original implementation relied on the randomness obtained from the GMP library, which is not generated securely. We split this into a separate library, `libaesrand`, which uses AES-NI for efficient randomness generation, and which may be useful in other contexts.

[3] https://bitbucket.com/malb/gghlite-flint

Attacks on GGHLite. Recently, Hu and Jia [HJ16] showed how to perform "zeroizing" attacks on GGHLite, to recover the secret parameters given certain public encodings of 0. However, since neither MIFE nor obfuscation publish any encodings of 0, these applications seem to be unaffected by the zeroizing attacks. More recently, Albrecht, Bai, and Ducas [ABD16] gave a quantum break for GGHLite *without* using any encodings of 0 or the public zero-testing parameter. Subsequently, Cheon, Jeong, and Lee [CJL16] showed how to give a (classical) polynomial-time attack on GGHLite, again without using any encodings of 0. However, their attack requires exponential time if the parameters of GGHLite are sufficiently increased (by a polynomial amount).

In concurrent work, Miles, Sahai, and Zhandry [MSZ16a] gave a completely different form of attack, known as an "annihilation" attack, on *applications* of GGHLite, specifically, MIFE and program obfuscation. They show that provably secure instantiations of these primitives from mmaps are in fact insecure when the mmap is instantiated with GGHLite. Despite the annihilation attacks, our implementations of these primitives from GGHLite still serve as a useful benchmark for the efficiency of GGHLite and for the efficiency of future GGH-like schemes resistant to annihilation attacks, which will inevitably arise from improvements to the GGH framework.

5.4.2 The CLT Multilinear Map

Coron, Lepoint, and Tibouchi [CLT13] proposed a candidate multilinear map over the integers, which works over a composite modulus that is assumed to be hard to factor.

Our CLT implementation. Our implementation started with the implementation[4] of CLT in C++ by Coron et al. [CLT13]. We rewrote it in C and added functionality to save and restore encodings and the public parameters. As in the GGHLite case, we also modified its basic functionality to support indices instead of levels.

Furthermore, in our extension of CLT, we improve the efficiency of the encoding process

[4] https://github.com/tlepoint/multimap

which allows for us to apply the CLT multilinear map to the large parameter settings that we consider in the remainder of this work. The original CLT implementation applies the Chinese Remainder Theorem in the procedure that produces encodings of plaintext elements. Our implementation employs a certain trade-off that allows for the application of the Chinese Remainder Theorem in a recursive manner, resulting in more multiplications to compute the encoding, but with the efficiency gain that the elements being multiplied are much smaller. Experimentally, this yields a large speedup in the encoding time, more noticeably with larger parameters. In particular, for $\lambda = 80$ and $\kappa = 19$, without this optimization, it takes 134 seconds to produce a CLT encoding, whereas with our optimization, this time drops to 33 seconds.

Attacks on CLT. Similarly to other candidate constructions for multilinear maps, the CLT construction was not based on an existing hardness assumption but rather introduced a new assumption. Subsequently Cheon et al. [CHL+15] demonstrated a zeroizing attack against the construction of CLT, which succeeds in recovering the secret parameters of the scheme. This attack was further extended in the work of Coron et al. [CGH+15a], which demonstrated how it can be generalized and applied against some proposed countermeasures [BWZ14b, GGHZ14] to the attack by Cheon et al. [CHL+15]. But again, as with the zeroizing attacks on GGHLite, these results do *not* apply directly to the constructions we consider in this work.

5.5 Multi-Input Functional Encryption

The notion of multi-input functional encryption (MIFE), introduced by Goldwasser et al. [GGG+14], extends the concept of functional encryption [BSW11] so that a decryption key is associated with a *multi-input* function which is evaluated over multiple ciphertexts. More formally, a secret-key, fixed-key MIFE scheme for a function f, on m inputs and with output in a range $\mathcal{R}$, is a tuple of algorithms $(\mathsf{Keygen}, \mathsf{Encrypt}, \mathsf{Eval})$ such that:

- $\mathsf{Keygen}(1^\lambda) \to (pp, sk)$. The algorithm takes as input the security parameter and generates the public parameters pp and a secret key sk.
- $\mathsf{Encrypt}(sk, i, x) \to \mathsf{ct}$. The algorithm takes as input the secret key sk, an input position index i, and an input x, and outputs a ciphertext ct.
- $\mathsf{Eval}(pp, \mathsf{ct}_1, \ldots, \mathsf{ct}_m) \to z$. The algorithm takes as input a secret key sk and m ciphertexts $\mathsf{ct}_1, \ldots, \mathsf{ct}_m$, and produces an output $z \in \mathcal{R}$.

Correctness requires that for any inputs $x_1, \ldots, x_m$, for $(pp, sk) \leftarrow \mathsf{Keygen}(1^\lambda)$, letting $\mathsf{ct}_i = \mathsf{Encrypt}(sk, i, x_i)$ for each $i \in [m]$, we have that

$$\mathsf{Eval}(pp, \mathsf{ct}_1, \ldots, \mathsf{ct}_m) = f(x_1, \ldots, x_m).$$

Informally, an MIFE scheme is *secure* if the information revealed by a collection of ciphertexts is exactly the information that can be obtained by running Eval, and no more. We omit formal security definitions since we do not directly rely on them in this work.

Goldwasser et al. [GGG+14] gave a general MIFE construction that uses indistinguishability obfuscation in a black-box manner. Boneh et al. [BLR+15] proposed a secret-key MIFE construction that is based directly on mmaps (instead of obfuscation) in order to obtain better efficiency. A particular instantiation of this MIFE construction, where the function used in the decryption key is the comparison function, results in a construction for *order revealing encryption* (ORE), which allows comparisons over ciphertexts while hiding all other information about the encrypted messages. More specifically, an ORE scheme is a MIFE scheme with the function $f(x, y)$ that outputs the ordering between x and y. This ORE scheme achieves the optimal security definition for a scheme that allows the comparison functionality over encrypted data, improving over the security level provided by order-*preserving* encryption schemes.

MIFE implementation. We implemented the Boneh et al. [BLR+15] MIFE construction on top of the `libmmap` library, and provide interfaces for Keygen, Encrypt, and Eval, which

perform the respective operations supported by MIFE. We parallelize the computation performed during Keygen, but for Encrypt, we choose to sequentially construct the encodings belonging to the ciphertext, and instead defer the parallelism to the underlying mmap implementation for producing encodings, in the interest of reducing memory usage at the cost of potentially increased running times. We note that, since CLT enjoys much more parallelism than GGHLite when constructing encodings, this optimization causes the Encrypt time for GGHLite to be less efficient. Finally, for Eval, we multiply encodings in parallel for CLT, since the multiplication of CLT encodings natively does not support parallelism. However, for GGHLite, we choose to multiply encodings sequentially, and instead rely on the parallelism afforded by GGHLite encoding multiplication.

The ciphertexts produced by a call to Encrypt on an ℓ-length input are split into ℓ components (one for each input slot), which can be easily separated and combined with different components from other ciphertexts in a later call to Eval. Hence, with a collection of full ciphertexts, an evaluator can specify which components from each ciphertext should be passed as input to Eval, in order to evaluate the function on components originating from different sources.

5.5.1 Optimizing Comparisons

In this section, we describe a case study of applying the optimizations detailed in Section 5.3 to the comparison function. We establish two distinct "variants" of the comparison function which result in shorter ciphertext sizes. Both variants are built from a combination of condensing the input representation into a larger base $d > 2$, followed by dimension reduction, and optionally applying matrix pre-multiplication.

- DC-variant**.** The *degree-compressed* optimization is to first apply matrix pre-multiplication to re-order the reading of the input bits as in Equation (5.1). Then, the dimensions of the resulting matrices from the layered state machine are slightly reduced.

- MC-variant**.** The *matrix-compressed* optimization is to directly apply dimension reduction in the normal interleaved ordering of the bits (as $x_1y_1x_2y_2\cdots x_ny_n$). Here, the dimensions of the matrices can be reduced to depend only linearly in the base representation d, as opposed to quadratically.

We now discuss each optimization in more detail.

The degree-compressed variant (DC-variant**).** By optimizing the (layered) comparison state machine, we obtain that not all matrices need to be of dimension $(d+3)\times(d+3)$. For example, the first matrix need only be of dimension $1\times d$, and the second matrix need only be of dimension $d\times(d+2)$. Also, the last matrix can be of dimension $(d+2)\times 3$. And finally, each of the remaining intermediate matrices need only be of dimension $(d+2)\times(d+2)$. Putting these observations together, the total number of encodings in the ciphertext is

$$\begin{aligned} M &= d + d(d+2) + 3(d+2) + (\kappa-3)(d+2)^2 \\ &= d^2(\kappa-2) + (d+1)(4\kappa-6). \end{aligned} \tag{5.2}$$

The matrix-compressed variant (MC-variant**).** Note, however, that if we do not apply the matrix pre-multiplication optimization, but instead apply dimension reduction directly to the comparison state machine associated with the normal (not interleaved) ordering of the input digits, then the first matrix is of dimension $1\times d$, the second matrix is of dimension $d\times 3$, and all other $\kappa-2$ matrices are of dimension either $3\times(d+2)$ or $(d+2)\times 3$. Putting this together, we have $\kappa = 2n$ and

$$\begin{aligned} M &= d + 3d + 3(\kappa-2)(d+2) \\ &= 3(\kappa-2)(d+2) + 4d. \end{aligned} \tag{5.3}$$

Concretely, for a domain of size $N = 10^{12}$, if we choose to represent the inputs in base $d = 5$, then this implies $n = 18$ (since $12 \le \log_{10}(5^{18}) < 13$), and hence with the matrix

pre-multiplication optimization along with the Cryptol optimization for dimension reduction, we have $\kappa = 19$ and $M = 845$. Without applying matrix pre-multiplication and only using dimension reduction with base $d = 10$, we have $n = 12$, $\kappa = 24$, and $M = 832$.

Experimentally for the mmaps we tested, we found that the matrix pre-multiplication optimization produces shorter ciphertexts than applying dimension reduction without matrix pre-multiplication. However, we only tested this for an input domain of $N = 10^{12}$ and security parameter $\lambda = 80$. As N grows larger, depending on the asymptotic behavior of encoding sizes as κ increases and λ varies, future implementations of the comparison state machine may find that one can produce shorter ciphertexts when applying dimension reduction without matrix pre-multiplication.

5.5.2 Order-Revealing Encryption

To implement order-revealing encryption, we set our plaintext domain to the numbers in the range $[N]$. By taking $N = 10^{12}$, we found that selecting the base representation $d = 5$ and applying the matrix pre-multiplication optimization resulted in using only $\kappa = 19$ levels of the underlying mmap, which achieved the shortest ciphertexts for this domain. In fact, this construction yields the shortest known ciphertexts for ORE on a domain of size 10^{12}, as explained below.

An alternative (basic) construction. The closest competitor to our ORE construction in terms of ciphertext size and overall efficiency is a construction due to Lewi and Wu [LW16], which we refer to as the "basic" ORE scheme, described below.

Let $[N]$ be the message space. Let $F : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}^\lambda$ be a secure pseudorandom function (PRF) and $H : \{0,1\}^\lambda \times \{0,1\}^\lambda \to \{0,1\}$ be a hash function (modeled as a random oracle). Let CMP be the comparison function, defined as $\text{CMP}(x, y) = 1$ if $x < y$ and $\text{CMP}(x, y) = 0$ if $x > y$. The basic ORE scheme Π_{ore} is defined as follows.

- $\mathsf{Keygen}(1^\lambda) \to (pp, \mathsf{sk})$. The algorithm samples a PRF key $k \xleftarrow{R} \{0,1\}^\lambda$ for F, and a random permutation $\pi : [N] \to [N]$. The secret key sk is the pair (k, π), and there are

no public parameters.

- $\mathsf{Encrypt}(\mathsf{sk}, i, x) \to \mathsf{ct}$. Write sk as (k, π). If $i = 1$, the ciphertext output is simply $\mathsf{ct} = (F(k, \pi(x)), \pi(x))$. If $i = 2$, then the encryption algorithm samples a nonce $r \xleftarrow{R} \{0,1\}^\lambda$, and for $j \in [N]$, it computes $v_j = \text{CMP}(\pi^{-1}(j), y) \oplus H(F(k, j), r)$. Finally, it outputs $\mathsf{ct} = (r, v_1, v_2 \ldots, v_N)$.

- $\mathsf{Eval}(pp, \mathsf{ct}_1, \mathsf{ct}_2) \to \{0,1\}$. The compare algorithm first parses $\mathsf{ct}_1 = (k', h)$ and $\mathsf{ct}_2 = (r, v_1, v_2, \ldots, v_n)$, then outputs $v_h \oplus H(k', r)$.

Note that a single ciphertext from this scheme is precisely $N + 2\lambda + \lceil \log_2(N) \rceil$ bits long. For $N = 10^{12}$ and $\lambda = 80$, this amounts to ciphertexts of length 116.42 GB.[5]

Choosing the best optimizations. Our goal is to construct an ORE scheme which achieves shorter ciphertexts than the above construction, without compromising security. To do this, we use our MIFE implementation for the comparison function, and we apply our optimizations to make the ciphertext as short as possible.

We compare the ciphertext sizes for four different ORE constructions, obtained from either using the GGHLite or CLT mmap, and by applying either the DC-variant or MC-variant optimizations. For each of these options, we fix the input domain size $N = 10^{12}$ and vary the input base representation $d \in [2, 25]$. Using Equations (5.2) and (5.3), we can compute the estimated ciphertext size as a function of d (since κ is determined by the choice of d and N). See Figure 5.5.1 for the results. We find that, for $N = 10^{12}$, the shortest ciphertexts for ORE from GGHLite are obtained when $d = 5$ using the DC-variant optimization, and the shortest ciphertexts for ORE from CLT are obtained when $d = 4$ using the DC-variant optimization as well.

Under these settings, the DC-variant optimization for GGHLite reads the inputs in base 5, requiring $\kappa = 19$, to produce a total of 845 encodings per ciphertext, for a total size

[5] Clearly, increasing λ has a relatively unnoticeable effect on the overall ciphertext size for the settings of N we consider.

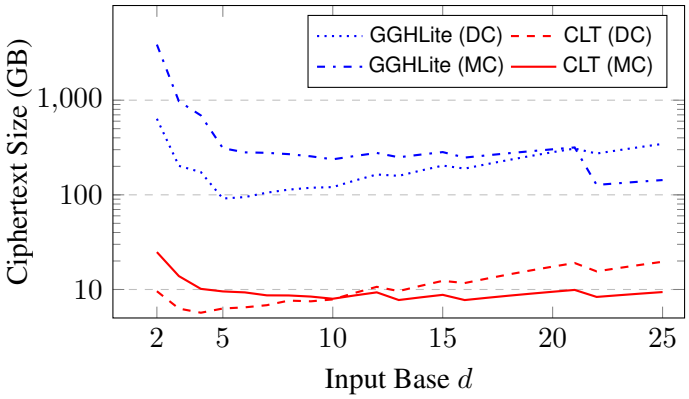

Figure 5.5.1: Estimates of the ciphertext size (in *GB*) for ORE with best-possible semantic security at $\lambda = 80$, for domain size $N = 10^{12}$ and for bases $d \in [2, 25]$. We compare GGHLite and CLT, with the DC-variant and MC-variant optimizations.

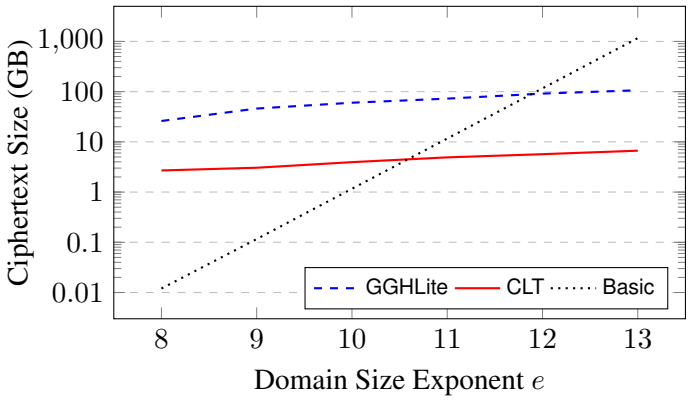

Figure 5.5.2: Estimates of the ciphertext size (in *GB*) for ORE with best-possible semantic security at $\lambda = 80$, for varying domain sizes. The exponent e on the x-axis denotes support for plaintexts in the range from 1 to $N = 10^e$. We compare GGHLite map (DC-variant), the CLT map, and the basic construction Π_{ore} (described in Section 5.5.2).

of 91.4 GB. For CLT, the DC-variant optimization reads in the inputs in base 4, requiring $\kappa = 21$, to produce a total of 694 encodings, for a total size of 5.68 GB.

We also measure the ciphertext size as we vary the domain size; see Figure 5.5.2. We measure the estimated ciphertext size for various domain sizes when using GGHLite, CLT, and the Π_{ore} construction described above. The results for GGHLite and CLT are using the optimal bases as detailed in Figure 5.5.1. We find that for $N = 10^{11}$ and $N = 10^{12}$, ORE using the CLT mmap and GGHLite mmap, respectively, produces a smaller ciphertext than Π_{ore}. This demonstrates that for certain domain sizes, our ORE construction produces the smallest known ciphertexts (versus ORE schemes that do not require mmaps).

5.5.3 Three-Input DNF Encryption

We now explore the applications of MIFE to a function on three inputs, which we call the 3DNF function. For n-bit inputs $x = x_1 \cdots x_n$, $y = y_1 \cdots y_n$, and $z = z_1 \cdots z_n$, the 3DNF function is defined as

$$\mathsf{3DNF}(x, y, z) = (x_1 \wedge y_1 \wedge z_1) \vee \cdots \vee (x_n \wedge y_n \wedge z_n) \in \{0, 1\}.$$

This function bears resemblance to the "tribes" function studied by Gentry *et al.* [GLW14], who also use mmaps to construct tribe instances. We refer to a MIFE scheme for the 3DNF function as a *3DNF encryption scheme*, and we refer to each ciphertext as consisting of three components, one for each input slot: the left encryption, middle encryption, and right encryption. To the best of our knowledge, 3DNF encryption schemes do not appear to follow directly from simpler cryptographic assumptions.

Application to fraud detection. An immediate application of 3DNF encryption is in the fraud detection of encrypted transactions. Consider the scenario where a (stateless) user makes payments through transactions that are audited by a payment authority. In this setting, each transaction is associated with a string of n flags, represented as bits pertaining to a set of n properties of the transaction. A payment authority, in the interest of detecting fraud, restricts the user to make at most (say) $\ell = 3$ transactions per hour, and wants to raise an alarm if a common flag is set in all ℓ of the transactions made in the past hour (if less than ℓ transactions were made in the past hour, then the authority does not need to perform a check).

To protect the privacy of the user, the length-n flag string associated with each transaction can be sent to the payment authority as encrypted under a 3DNF encryption scheme, where the stateless user holds the decryption key. Here, the user would send a left encryption for the first transaction, a middle encryption for the second, and a right encryption for the third. Then, since the payment authority cannot decrypt any of the flag strings for the transactions,

the privacy of the user's transactions is protected. However, the payment authority can still perform the fraud detection check by evaluating a set of ℓ transactions to determine if they satisfy the 3DNF function. Since we require that the user is stateless between transactions, this application fits the model for a 3DNF encryption scheme, and does not seem to directly follow from simpler primitives.

Optimizing 3DNF encryption. Similar to the case of the comparison function, we can apply the branching program optimizations to the 3DNF function as well, in order to reduce the overall efficiency of the resulting 3DNF encryption scheme. We constructed a 3DNF encryption scheme using our MIFE implementation, for $n = 16$ bit inputs at security parameter $\lambda = 80$. We optimized the 3DNF encryption scheme by condensing the input representation into base $d = 4$. Additionally, we applied the matrix pre-multiplication optimization, which meant that our input bits were read in the order $x_1 y_1 z_1 z_2 y_2 x_2 x_3 y_3 \cdots$ (the natural generalization of the interleaving of Equation (5.1) to three inputs). This resulted in a setting of degree $\kappa = 17$ for the underlying mmap. Finally, we used `cryfsm` to generate the corresponding MBP, which automatically applied the appropriate dimension reduction optimizations. Under the CLT mmap, a left encryption is 637 MB, a middle encryption is 1.4 GB, and a right encryption is 680 MB.

5.6 Program Obfuscation

A *program obfuscator* [BGI+01, GGH+13b] is a compiler that aims to make a program "unintelligible" while preserving its functionality. Formally, an obfuscator $\mathcal{O}$ is a tuple of algorithms written as $\mathcal{O} = (\mathsf{obf}, \mathsf{eval})$, where the obfuscation algorithm obf takes as input a program prog (e.g., expressed as a Boolean circuit), and outputs an obfuscated program $\mathsf{obf}(\mathsf{prog})$, and the evaluation algorithm takes an obfuscated program $\mathsf{obf}(\mathsf{prog})$ and produces an output. An obfuscator is *correct* for a program prog if, for all valid inputs x accepted by prog, we have that $\mathsf{eval}(\mathsf{obf}(\mathsf{prog}))(x) = \mathsf{prog}(x)$.

VBB and pseudo-VBB security. An obfuscator $\mathcal{O}$ is *virtual black-box (VBB) secure*[6] for a program prog if for any efficient adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$, given only oracle access to $\mathsf{prog}(\cdot)$, for which the quantity

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathsf{obf}(\mathsf{prog})) = 1\right] - \Pr\left[\mathcal{S}^{\mathsf{prog}(\cdot)}(1^\lambda) = 1\right]\right|$$

is negligible. We say that an obfuscator $\mathcal{O}$ is *pseudo-VBB secure* (pVBB) for program prog and obfuscator $\mathcal{O}' = (\mathsf{obf}', \mathsf{eval}')$ if $\mathcal{O}'$ is both VBB secure and for every efficient adversary $\mathcal{A}$, there exists an efficient adversary $\mathcal{B}$ for which the quantity

$$\left|\Pr\left[\mathcal{A}(1^\lambda, \mathsf{obf}(\mathsf{prog})) = 1\right] - \Pr\left[\mathcal{B}(1^\lambda, \mathsf{obf}'(\mathsf{prog})) = 1\right]\right|$$

is negligible. In other words, if an obfuscator $\mathcal{O}$ is pVBB secure for a program prog and obfuscator $\mathcal{O}'$, then any efficient attack on the security of $\mathcal{O}$ translates directly to an efficient attack on the VBB security of $\mathcal{O}'$ for the program prog.

In our work, we construct a point function obfuscator that is pVBB secure for the Sahai-Zhandry obfuscator [SZ14]. Our obfuscator operates identically to the Sahai-Zhandry obfuscator, which is VBB secure, except that we discard half of the ciphertext that corresponds to the second input in the "dual-input" branching programs that obfuscator uses. Effectively, our obfuscator can be seen as operating on "single-input" branching programs, which do not obtain VBB security, but do obtain pseudo-VBB security. We emphasize that this distinction in security is purely definitional from an attacker's point of view, as any attack on our obfuscator immediately results in an attack on the Sahai-Zhandry obfuscator.

In this section we show how we use `cryfsm` and `libmmap` to build such a program obfuscator. Apon et al. [AHKM14] gave the first implementation of program obfuscation, using the CLT mmap [CLT13] and a program compiler based on the approaches of Barak et

[6]The reason we consider VBB versus *indistinguishability* obfuscation is that we consider point functions, for which VBB obfuscators are believed to exist.

al. [BGK+14] and Ananth et al. [AGIS14]. We extend this codebase in the following ways:

- **Multilinear maps.** We integrate in `libmmap` to support both the CLT and GGHLite mmaps.
- **Program compilers.** We support MBPs output by `cryfsm`, using the Sahai-Zhandry obfuscator [SZ14].

Point function obfuscation. We evaluated our implementation by obfuscating point functions, namely, functions that output 0 on a single (secret) input, and 1 otherwise. Previous work [AHKM14] also evaluated obfuscation for point functions, but was only able to successfully obfuscate 14-bit point functions with an mmap security parameter of $\lambda = 60$. As noted by Bernstein *et al.* [BHLN15], the secret input of an n-bit point function can be recovered by simply enumerating over all 2^n possible inputs. In our experiments, we set $n = \lambda$, and consider point function obfuscation for 40-bit and 80-bit inputs.

The MBP for a λ-bit point function is of length λ and consists of a total of 2λ matrices, each of dimension 2×2. As a small optimization, we can apply dimension reduction to obtain a branching program where the first pair of matrices need only be of dimension 1×2. The more significant optimization comes by condensing the input representation through increasing the input base d.

The total number of encodings that we must publish in the obfuscation of a λ-bit point function can be computed as $M = 2 + 4 \cdot d \cdot \ell$, where ℓ is the length of the MBP. We estimate the ciphertext size for various choices of bases in Figure 5.6.1, which incorporates our estimations for the size of a single encoding in GGHLite and CLT for $\lambda = 40$ and $\lambda = 80$.

- For $\lambda = 40$, we find that the minimal ciphertext size for domain size $N = 2^{40}$ is produced using MBPs under base 9 and length 13 for GGHLite, and base 6 and length 16 for CLT.

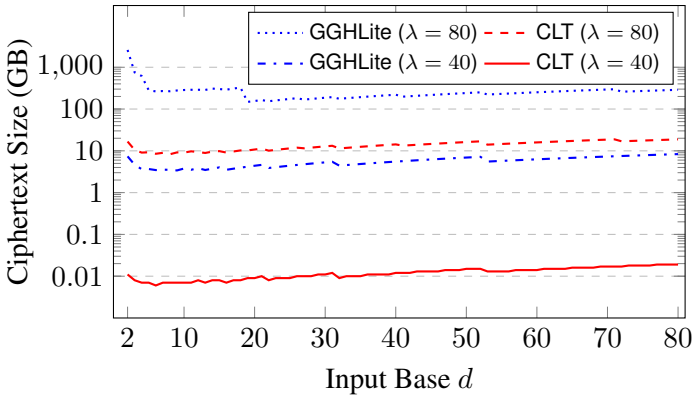

Figure 5.6.1: Estimates for the ciphertext size (in GB) for point function obfuscation, for domain sizes $N = 2^{80} = 2^{\lambda}$ and $N = 2^{40} = 2^{\lambda}$. In the case of $\lambda = 80$, the minimums are achieved at $d = 19$ for GGHLite and $d = 8$ for CLT. In the case of $\lambda = 40$, the minimums are achieved at $d = 9$ for GGHLite and $d = 6$ for CLT.

- For $\lambda = 80$, we find that the minimal ciphertext size for domain size $N = 2^{80}$ is produced using MBPs under base 19 and length 19 for GGHLite, and base 8 and length 27 for CLT.

Obfuscator implementation. Our implementation is in a mix of Python and C, with Python handling the frontend and with C handling all the computationally expensive portions, and provides interfaces to both obfuscate (obf) and evaluate (eval) an MBP. We parallelize the encoding of the elements in the MBP by using a threadpool and delegating each encoding operation to a separate thread. Once all the threads for a given matrix in the MBP complete, we then write the (encoded) matrix to disk. Thus, the threadpool approach has a higher RAM usage (due to keeping multiple encodings in memory as we parallelize) than encoding one element at a time and letting the underlying mmap library handle the parallelization, but is more efficient.

Other obfuscators. Our obfuscator is built upon improvements inspired by the Sahai-Zhandry obfuscator, which is built on the general obfuscator described by Barak et al. [BGK+14] and Ananth et al. [AGIS14]. In addition to these obfuscators, we also implemented the Zimmerman [Zim15] obfuscator. However, because the Zimmerman obfuscator induces a seemingly unavoidable lower bound on the degree of multilinearity for the inputs we

consider, we found that the Zimmerman obfuscator was not competitive with the obfuscator we implemented. More specifically, the Zimmerman obfuscator requires that the degree of multilinearity for the obfuscation of *any* program be at least twice the number of inputs that the circuit accepts—a cost that may be insignificant when obfuscating other programs, but was too high for point functions (even when we tried to increase the input base representation to minimize this cost), and hence unsuitable for our purposes.

5.7 Experimental Analysis

All of our experiments were performed using the Google Compute Engine servers with a 32-core Intel Haswell CPU at 2.5 GHz, 208 GB RAM, and 100 GB disk storage.

5.7.1 MIFE Experiments

We evaluated our multi-input functional encryption constructions with two applications: order-revealing encryption (ORE) (cf. Section 5.5.2) and three-input DNF (3DNF) encryption (cf. Section 5.5.3). In Section 5.5, we showed how we can accurately estimate the ciphertext size from parameters derived from the input size and the security parameter λ, and our experiments confirmed that these parameter estimates are reasonably accurate (all within 1–2% of our reported values).

Additionally, we assessed the performance of the MIFE interface algorithms Keygen, Encrypt, and eval, along with memory utilization during the Encrypt computation, which was by far the most costly step. We note that, since the files that we are working with are so large, a non-trivial amount of time was spent in the reading and writing of these files to disk, and so an exact reproduction of our numbers may also need to mimic the disk storage specification we use.

As another sidenote, we reiterate that our primary interest in selecting the parameters for our applications is to create the most compact ciphertexts possible. As a result, some

λ	mmap	N	d	ℓ	Encrypt	eval	$\lvert\mathsf{ct}\rvert$	RAM
40	CLT	10^{10}	4	19	1 s	0.3 s	13 MB	17 MB
		10^{12}	4	22	3 s	1.6 s	18 MB	18 MB
	GGH	10^{10}	4	19	38 m	47 s	7.1 GB	23 GB
		10^{12}	4	22	52 m	68 s	9.6 GB	25 GB
80	CLT	10^{10}	4	19	28 m	4 m	4.7 GB	5 GB
		10^{12}	4	22	37 m	6 m	6.0 GB	6 GB

Table 5.7.1: ORE experiments. "λ" denotes the security parameter of the underlying multilinear map; "mmap" denotes the multilinear map; "N" denotes the domain size; "d" denotes the MBP base; "ℓ" denotes the MBP length; "Encrypt" denotes the running time of encryption; "eval" denotes the running time of evaluation, "$\lvert\mathsf{ct}\rvert$" denotes the size of the ciphertext; and "RAM" denotes the RAM required to encrypt. We use "h" for hours, "m" for minutes, and "s" for seconds.

of our optimizations come with a cost of increased evaluation time, and hence, we believe that it is possible to reduce our evaluation time (potentially at the expense of having larger ciphertexts).

Experimental results. We summarize our MIFE experiments in Tables 5.7.1 and 5.7.2. We evaluated the MIFE constructions for ORE with input domain sizes $N = 10^{10}$ and $N = 10^{12}$, and for 3DNF encryption on 8-bit inputs, testing both GGHLite and CLT as the underlying mmap. For each experiment, we report the computation wall time for Encrypt and eval, the overall ciphertext size $\lvert\mathsf{ct}\rvert$, along with the memory usage during the Encrypt computation. The Keygenoperation varied from several seconds (for CLT with $\lambda = 40$) to 145 minutes (for GGHLite with $\lambda = 80$). The encryption statistics measured were for generating a complete ciphertext, containing all components, as opposed to containing only the left or right (or middle) components.

Since the CLT mmap produces shorter encodings, the encryption and evaluation time for the experiments using CLT were much faster than the corresponding experiments for GGHLite. This is also partly due to the fact that CLT enjoys much more parallelism than GGHLite. We also only present timings for CLT with $\lambda = 80$ because we ran out of RAM during the encryption procedure when using GGHLite.

λ	mmap	N	d	ℓ	Encrypt	eval	$\lvert\mathsf{ct}\rvert$	RAM
40	CLT	16-bit	4	17	0.6 s	0.2 s	7.4 MB	18 MB
	GGH	16-bit	4	17	20 m	28 s	3.9 GB	22 GB
80	CLT	16-bit	4	17	12 m	3 m	2.5 GB	4 GB

Table 5.7.2: 3DNF experiments. See Table 5.7.1 for the column details.

5.7.2 Program Obfuscation Experiments

To evaluate our program obfuscation implementation, we chose a random secret 40-bit and a random secret 80-bit point, and used `cryfsm` to create the corresponding MBPs for the point functions associated with these points. We selected the input base representation for these programs with the goal of minimizing the total obfuscation size for each obfuscated point function (see Section 5.6 for our calculations). Like with MIFE, optimizing for obfuscation or evaluation time could lead to different optimal input base representations.

Experimental results. We tested three settings for point function obfuscation: 40-bit inputs with $\lambda = 40$, 80-bit inputs with $\lambda = 40$, and finally, 80-bit inputs with $\lambda = 80$. We also obfuscated using both CLT and GGHLite for $\lambda = 40$, but only used CLT for $\lambda = 80$, as the GGHLite experiment was too resource-intensive. Our results are summarized in Table 5.7.3. As we observed in the MIFE experiments, we note that GGHLite performs significantly worse when used in obfuscation compared to CLT. We also note that while obfuscation takes a huge amount of time and resources, evaluation is much less resource-intensive, for both GGHLite and CLT—a consequence of the fact that eval only requires multiplying (encoded) matrices, which is highly parallelizable and also much less costly than the encoding operation itself.

These results, while evidently impractical, are a huge improvement over prior work [AHKM14], which took 7 hours to obfuscate a 14-bit point function with $\lambda = 60$, resulting in an obfuscation of 31 GB. This improvement mainly come from (1) using a much tighter matrix branching program representation of the program, and (2) operating over different sized bases.

λ	mmap	N	d	ℓ	obf	eval	|obf|	RAM
40	CLT	40-bit	6	16	1.7 s	0.1 s	6.3 MB	1.7 GB
		80-bit	7	29	6.6 s	0.3 s	21.7 MB	1.7 GB
	GGH	40-bit	9	13	28 m	5.9 s	3.5 GB	38 GB
		80-bit	6	31	56 m	39 s	13.7 GB	37 GB
80	CLT	80-bit	8	27	3.3 h	180 s	8.3 GB	11 GB

Table 5.7.3: Program obfuscation experiments. "λ" denotes the security parameter of the underlying multilinear map; "mmap" denotes the multilinear map; "N" denotes the domain size; "d" denotes the MBP base; "ℓ" denotes the MBP length; "obf" denotes the obfuscation time; "eval" denotes the evaluation time; "|obf|" denotes the obfuscation size; and "RAM" denotes the RAM required to obfuscate (evaluation RAM usage never exceeded 1 GB). We use "h" for hours, "m" for minutes, and "s" for seconds.

5.8 Conclusions

In this work, we presented `5Gen`, a framework for the prototyping and evaluation of applications that use multilinear maps (mmaps) and matrix branching programs. `5Gen` is built as a multi-layer software stack which offers modularity and easy integration of new constructions for each component type. Our framework offers an optimized compiler that converts programs written in the Cryptol language into matrix branching programs, a representation widely used in mmap-based constructions. `5Gen` includes a library of mmaps available through a common API; we currently support the GGHLite and CLT mmaps, but our library can be easily extended with new candidates. Leveraging the capabilities of our compiler and mmap libraries, we implemented applications from two computing paradigms based on mmaps: multi-input functional encryption (MIFE) and obfuscation.

We measured the efficiency of our MIFE and obfuscation applications with various parameter settings using both the GGHLite and CLT mmaps. While the results show efficiency that is clearly not usable in practice, they provide a useful benchmark for the current efficiency of these techniques.

Constructing multilinear maps is an active and rapidly-evolving area of research. Our `5Gen` framework provides an easy-to-use testbed to evaluate new mmap candidates for various applications and is open-source and freely available at `https://github.com/`

5GenCrypto.

5.9 Parameter Selection

In this section, we discuss the parameter selection for both the GGHLite and CLT mmaps that we consider in this work. Throughout, we use λ as the security parameter and κ as the multilinearity parameter.

5.9.1 GGHLite

We now discuss the parameter selection used for the GGHLite mmap discussed in Section 5.4.1. We discuss the various parameters, and where applicable, the values we set them to and why. As we build off of the code from Albrecht et al. [ACLL15], many of these parameter settings come directly from that work; we simply document them explicitly here.

- n: The dimension of the lattice. We set n by iteratively producing the below parameters and then checking whether they satisfy the security parameter according to [APS15], increasing n by a power of two each iteration (cf. [ACLL15, Section 4.4]).
- σ: The Gaussian parameter. We set $\sigma = 4\pi n\sqrt{e\ln(8n)/\pi}$ according to [LSS14, Eq. (4)].
- ℓ_g^{-1}: The upper bound on the size of $\|g^{-1}\|$. We set $\ell_g^{-1} = 4\sqrt{e\pi n}/\sigma$ according to [LSS14, Eq. (4)].
- ℓ: The number of bits to extract from the level-κ encoding. We set $\ell = \lceil \log_2(8\sigma n)\rceil$ according to the discussion in [ACLL15, Section 4.2].
- σ': The Gaussian parameter for encoding values. We set $\sigma' = \max\left\{\begin{array}{c} 2n^{1.5}\sigma\sqrt{e\log(8n)/\pi} \\ 7n^{2.5}\ln^{1.5}(n)\sigma \end{array}\right\}$ according to the discussion in [LSS14, Section 6].
- q: The modulus. Selected according to [ACLL15, Section 4.2, pg. 10].

5.9.2 CLT

We now discuss the parameter selection used for the CLT mmap discussed in Section 5.4.2. As above, we discuss the various parameters, and where applicable, the values we set them to and why.

- ρ: The bitlength of the randomness used for encodings. We set $\rho = \lambda$ to avoid the attack of Lee and Seo [LS14].
- α: The bitlength of the message slots g_i. We set $\alpha = \lambda$ as suggested by [CLT13].
- β: The bitlength of the random h_i values. We set $\beta = \lambda$ to avoid a GCD attack similar to [CLT13, Section 5.2].
- ρ_f: The maximum bitlength of the randomness in a level-κ encoding. For the specific usecase of obfuscation we can set $\rho_f = \kappa(\rho + \alpha)$.
- n: The number of secret primes p_i. In Section 5.10 we show how to adapt the lattice attack on encodings by Coron et al. [CLT13, Section 5.1] to the "general" setting where no 0-level encodings of zero are available, and thus we need to set $n = \omega(\eta \log \lambda)$. However, rather than setting n to match some asymptotic, such as $\eta\lambda$, we consider the concrete costs of the various attacks to give an accurate estimate of n. We use the approach detailed by Tancrède Lepoint [Lep14, Section 7.2] using the more conservative estimate for the running time of the LLL algorithm [Lep14, Section 7.2.5].
- η: The bitlength of the secret primes p_i. We set $\eta = \rho_f + \alpha + \beta + \log_2(n) + 9$ according to [CLT13, Lemma 8].
- ν: The bitlength of the image of the mmap. We set $\nu = \eta - \beta - \rho_f - \lambda - 3$ according to [CLT13, Lemma 8].

Note that n, η, and ν depend on each other. Thus, to compute these values we simply loop until we reach a fixed point, using the requirement that $\beta + \alpha + \rho_f + \log_2(n) \leq \eta - 9$ and $\nu \geq \alpha + 6$, as detailed in [CLT13, Lemma 8].

5.10 Lattice Attack on Encodings

In this section we describe how to adapt the lattice attack on encodings by Coron et al. [CLT13, Section 5.1] to our particular use of the CLT mmap.[7] In particular, we consider the case where an attacker has access to $t > n$ level-1 encodings. Without loss of generality, we assume a single z.

Let $x_0 = \prod_{i=1}^{n} p_i$. For $j \in [t]$ consider the level-1 encoding $x'_j = x_j/z \bmod x_0$ of secret message $\boldsymbol{m}_j = (m_{ij})_i$, where $x_j \in \mathbb{Z}_{x_0}$ is such that $x_j \bmod p_i = r_{ij}g_i + m_{ij}$. Note that $x_j \bmod p_i$ is of size $\rho + \alpha$ bits and can be considered as a level-0 encoding of $\boldsymbol{m}_j$. Let $\boldsymbol{x'} = (x'_j)_j$ and let $\boldsymbol{x} = (x_j)_j$.

As detailed by Coron et al., we want to use $\boldsymbol{x'}$ to relate the lattice of vectors $\boldsymbol{u}$ orthogonal to $\boldsymbol{x'} \bmod x_0$ to the lattice of vectors orthogonal to each "plaintext" vector $\boldsymbol{s}_i = (s_{ij})_j = (r_{ij}g_i + m_{ij})_j$. We do so as follows.

By construction we have that

$$\begin{aligned} \boldsymbol{u} \cdot \boldsymbol{x'} &\equiv 0 \pmod{x_0} \\ \iff z(\boldsymbol{u} \cdot \boldsymbol{x'}) &\equiv 0 \pmod{x_0} \\ \iff \boldsymbol{u} \cdot \boldsymbol{x} &\equiv 0 \pmod{x_0}, \end{aligned}$$

where the last equivalence comes from the fact that z and x_0 are coprime with high probability. We can thus apply the attack detailed by Coron et al. [CLT13, Section 5.1] on vector $\boldsymbol{x'}$ to directly recover the vectors $\boldsymbol{s}_i$.

Now, given these vectors, we can recover p_i for $i \in [n]$ with high probability as

[7] We thank Tancrède Lepoint for detailing this adaptation.

follows [CHL+15]. Note that

$$\begin{aligned} \frac{x'_1}{s_{i1}} &\equiv \frac{x'_2}{s_{i2}} \pmod{p_i} \\ \iff x'_1 s_{i2} - x'_2 s_{i1} &\equiv 0 \pmod{p_i}. \end{aligned}$$

Thus, we can compute $\gcd(x'_1 s_{i2} - x'_2 s_{i1}, x_0)$ to learn p_i.

Chapter 6

Cryptanalysis of Indistinguishability Obfuscations of Circuits over GGH13

6.1 Introduction

An obfuscator is a program compiler which hides all partial implementation details of a program, intuitively. This is formalized via the notion of indistinguishability obfuscation [BGI+01]: we say an obfuscator obf is an indistinguishability obfuscator if it holds for every pair C_0, C_1 of functionally equivalent circuits (i.e. computing the same function) that $\mathsf{obf}(C_0)$ and $\mathsf{obf}(C_1)$ are indistinguishable. A recent surge of results has highlighted the importance of this notion: virtually "any cryptographic task" can be achieved assuming indistinguishability obfuscation and one-way functions [SW14].

All known candidate constructions of indistinguishability obfuscation, e.g. [GGH+13b, BGK+14, AB15], are based on multilinear-maps [GGH13a, CLT13, GGH15][1], which have been the subjects of various attacks [CHL+15, CGH+15b, CFL+16, HJ16, CLLT16a]. Among them, the attacks (e.g. [GGH13a, HJ16]) on GGH13 [GGH13a] multilinear maps required explicit access to "low-level" encodings of zero, or differently represented low-level

[1]The work of [AJN+16] might be seen as an exception to this: Assuming the (non-explicit) existence of indistinguishability obfuscation, they provide an explicit construction of an indistinguishability obfuscator.

encodings of zero, in the form of an encoded matrix with a zero eigenvalue [CGH+15a]; such low-level zero-encodings do not appear naturally in obfuscation constructions(except for a few specially designed programs [CGH+15b]). Recently Miles, Sahai, and Zhandry [MSZ16a] introduced a new class of polynomial-time[2] attacks without requiring low-level zeros against several obfuscation constructions [BR14, BGK+14, AGIS14, MSW14, PST14, BMSZ16], when instantiated with the GGH13 multilinear maps.

More specifically, Miles et al. [MSZ16a] exhibit two simple *branching programs* (and also programs padded with those) that are functionally equivalent, yet their BGKPS-obfuscations (put forward by Barak et al. in [BGK+14]) and similar constructions [BR14, AGIS14, MSW14, PST14, BMSZ16] are efficiently distinguishable.[3] Additionally they show that their attacks extend to any two branching programs with those two simple programs (respectively) padded into them. However, the branching programs considered there, in particular the *all-identity* branching program, do not appear "in the wild". More specifically, obfuscation constructions for circuits first convert an NC^1 circuit into a branching program (e.g. via Barrington's transformation) that possibly results in programs with complex structures, even if one starts with simple circuits. This brings us to the following open question:

Is it possible to attack obfuscations of complex branching programs generated from NC^1*?*

6.1.1 Our Contributions

In this work, we are able to answer the above question affirmatively. In particular, our main contributions are:

- We first define a *general* and efficiently-testable property of two single-input[4] branch-

[2] Several subexponential-time or quantum-polynomial-time [CDPR16, ABD16, CJL16] attacks on GGH13 multilinear maps also have been considered. We do not consider these in this work.

[3] To avoid repetitions, from now on we will refer to the obfuscation constructions of [BGK+14, BR14, AGIS14, MSW14, PST14] by BGKPS-like constructions.

[4] The branching programs, where any pair of matrices in the sequence depends on a single input location, are called single-input branching programs. Such branching programs naturally evolve from Barrington's transformation on circuits.

ing programs called *partial inequivalence* (discussed below) and demonstrate an annihilation attack against BGKPS-like obfuscations of any two (large enough) branching programs that satisfy this property.

- Next, using implementation in Sage [S+16] (see Section 6.8 for details on the implementation) we give explicit examples of pairs of (functionally equivalent) natural NC^1 circuits, which when processed via Barrington's Theorem yield pairs of branching programs that are partially inequivalent – and thus, attackable.

- As a consequence of the above result, we are also able to show that the "bootstrapping circuit(s)" technique used to boost $i\mathcal{O}$ for NC^1 to $i\mathcal{O}$ for P/poly, for a certain choice of the universal circuit, yield partially inequivalent branching programs in a similar manner – and are, thus, also attackable.

Our general partial inequivalence condition is broad and seems to capture a wide range of natural single-input branching programs. However, we require the program to be large enough.[5] Additionally, we need the program to output 0 on a large number of its inputs.

Finally, our new annihilation attacks are essentially based on linear system solvers and thus quite *systematic*. This is in contrast with the attacks of Miles et al. [MSZ16a] which required an exhaustive search operation rendering it hard to extend their analysis for branching programs with natural structural complexity. Therefore, at a conceptual level, our work enhances the understanding of the powers and the (potential) limits of annihilation attacks.

One limitation of our technique is that they do *not* extend to so-called dual-input branching programs. We leave it as an interesting open question.

A Concurrent and Independent work. Concurrent and independent to our work,[6] Chen et

[5]Note that, for our implementation we consider circuits that are quite small, only depth 3, and the resulting Barrington programs are of length 64. However, using the implementation we then "boost" the attack to a much larger NC^1 circuits that suffice for the real-world attack (discussed in Section 6.6)to go through.

[6]The first draft of this work appeared online concurrently as their first draft [CGH16a]. At the same time another independent work [CLLT16b] appeared that provided attacks against several CLT13 based obfuscators for a broader class of programs.

al. [CGH16a] provides a polynomial time attack against the GGHRSW construction [GGH+13b] based on GGH13 (and also GGH15 [GGH15]) maps that works for so-called "input-partitioning" branching programs. Nonetheless, their attacks are not known to extend [CGH16b] for complex branching programs evolved from NC^1 circuits (e.g. via Barrington's Transformation). Hence, our work stands as the *only* work that breaks obfuscations of NC^1 circuits based on GGH13 till date.

Change in Obfuscation Landscape. Given our work and the work of Chen et al. [CGH16a] the new *attack landscape against GGH13-based obfuscators* is depicted in Figure 6.1.1. We refer the reader to [AJN+16, Figure 13] for the state of the art on obfuscation constructions based on CLT13 and GGH15 multilinear maps.

	Branching Programs	NC^1 Circuits (Barrington's)	NC^1-to-P/poly [GGH+13b, App14] [BGL+15, GIS+10]
GGHRSW [GGH+13b]	⊗	○	○
BGKPS-like constructions [BR14, BGK+14, AGIS14] [PST14, MSW14, BMSZ16]	×	⊗	⊗
Obfuscations from weak mmaps [GMM+16, DGG+16]	○	○	○

Figure 6.1.1: The Attack Landscape for GGH13-based Obfuscators. In all cases, the multilinear map is [GGH13a]. ○ means no attack is known. × means a prior attack is known, and we present more general attacks for this setting. ⊗ means we give the first known attack in this setting and ⊗ means a new attack is discovered concurrently to ours (namely [CGH16a]).

6.1.2 Technical Overview

Below, after providing some additional backgrounds on multilinear maps and known attacks, we provide a overview of our annihilation attacks.

Multilinear Maps: Abstractly. As a first approximation, one can say that a cryptographic multilinear map system encodes a value $a \in \mathbb{Z}_p$ (where p is a large prime) by using a

homomorphic encryption scheme equipped with some additional structure. In other words, given encodings of a and b, one can perform homomorphic computations by computing encodings of $a + b$ and $a \cdot b$. Additionally, each multilinear map encoding is associated with some *level* described by a value $i \in \{1 \dots \kappa\}$ for a fixed universe parameter κ. Encodings can be added only if they are at the same level: $\mathsf{Enc}_i(a) \oplus \mathsf{Enc}_i(b) \rightarrow \mathsf{Enc}_i(a + b)$. Encodings can be multiplied: $\mathsf{Enc}_i(a) \odot \mathsf{Enc}_j(b) \rightarrow \mathsf{Enc}_{i+j}(a \cdot b)$ if $i + j \leq \kappa$ but is meaningless otherwise. We naturally extend the encoding procedure and the homomorphic operations to encode and to compute on matrices, respectively, by encoding each term of the matrix separately. Finally, the multilinear map system comes equipped with a *zero test*: an efficient procedure for testing whether the input is an encoding of 0 at level-κ. However, such zero-test procedure is not perfect as desired when instantiated with concrete candidate multilinear maps. In particular we are interested in the imperfection in GGH13 map.

An Imperfection of the GGH13 Multilinear Maps. Expanding a little on the abstraction above, a fresh multilinear map encoding of a value $a \in \mathbb{Z}_p$ at level i is obtained by first sampling a random value μ from $\mathbb{Z}_p$ and then encoding $\mathsf{Enc}_i(a + \mu \cdot p)$. Homomorphic operations can be performed just as before, except that the randomnesses from different encodings also get computed on. Specifically, $\mathsf{Enc}_i(a + \mu \cdot p) \oplus \mathsf{Enc}_i(b + \nu \cdot p)$ yields $\mathsf{Enc}_i(a + b + (\mu + \nu) \cdot p)$ and multiplication $\mathsf{Enc}_i(a + \mu \cdot p) \odot \mathsf{Enc}_j(b + \nu \cdot p)$ yields $\mathsf{Enc}_{i+j}(a \cdot b + (b \cdot \mu + a \cdot \nu + \mu \cdot \nu \cdot p) \cdot p)$ if $i + j \leq \kappa$ but is meaningless otherwise. An imperfection of the zero-test procedure is a feature characterized by two phenomena:

1. On input $\mathsf{Enc}_\kappa(0 + r \cdot p)$ the zero-test procedure additionally reveals r in a somewhat "scrambled" form.

2. For certain efficiently computable polynomials f and a collection of scrambled values $\{r_i\}$ it is efficient to check if $f(\{r_i\}) = 0 \mod p$ or not for any choice of r_i's.[7]

This imperfection has been exploited to perform attacks in prior works, such as the one by

[7] One can alternatively consider the scrambled values as polynomials over $\{r_i\}$ and then check if $f(\{r_i\})$ is identically zero in $\mathbb{Z}_p$.

Miles et al. [MSZ16a].[8]

Matrix Branching Programs. A matrix branching program of length ℓ for n-bit inputs is a sequence $BP = \left\{A_0, \left\{A_{i,0}, A_{i,1}\right\}_{i=1}^{\ell}, A_{\ell+1}\right\}$, where $A_0 \in \{0,1\}^{1\times 5}$, $A_{i,b}$'s for $i \in [\ell]$ are in $\{0,1\}^{5\times 5}$ and $A_{\ell+1} \in \{0,1\}^{5\times 1}$. Without providing details, we note that the choice of 5×5 matrices comes from Barrington's Theorem [Bar89]. We use the notation $[n]$ to describe the set $\{1, \ldots, n\}$. Let inp be a fixed function such that $\mathsf{inp}(i) \in [n]$ is the input bit position examined in the i^{th} step of the branching program. The function computed by this matrix branching program is

$$f_{BP}(x) = \begin{cases} 0 & \text{if } A_0 \cdot \prod_{i=1}^{\ell} A_{i,x[\mathsf{inp}(i)]} \cdot A_{\ell+1} = 0 \\ 1 & \text{if } A_0 \cdot \prod_{i=1}^{\ell} A_{i,x[\mathsf{inp}(i)]} \cdot A_{\ell+1} \neq 0 \end{cases},$$

where $x[\mathsf{inp}(i)] \in \{0,1\}$ denotes the $\mathsf{inp}(i)^{th}$ bit of x.

The branching program described above inspects one bit of the input in each step. More generally, multi-arity branching programs inspect multiple bits in each step. For example, dual-input programs inspect two bits during each step. Our strategy only works against single-input branching programs, hence we restrict ourselves to that setting.

Exploiting the Imperfection/Weakness. At a high level, obfuscation of a branching program $BP = \{A_0, \{A_{i,0}, A_{i,1}\}_{i=1}^{\ell}, A_{\ell+1}\}$ yields a collection of encodings $\{M_0, \{M_{i,0}, M_{i,1}\}_{i=1}^{\ell}, M_{\ell+1}\}$, say all of which are obtained at level-1.[9] We let $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$ denote the randomnesses used in the generation of these encodings, where each Z corresponds to a matrix of random values (analogous to r above) in $\mathbb{Z}_p$. For every input x such that $BP(x) = 0$, we have that $M_0 \odot \bigodot_{i=1}^{\ell} M_{i,x[\mathsf{inp}(i)]} \odot M_{\ell+1}$ is an encoding of 0, say of the form $\mathsf{Enc}(0 + r_x \cdot p)$ from which r_x can be learned in a scrambled form. The crucial

[8] Recent works such as [GMM+16, DGG+16], have attempted to realize obfuscation schemes secure against such imperfection and are provably secure against our attacks. We refer to them as obfuscations from weak multilinear maps (see Figure 6.1.1).

[9] Many obfuscation constructions use more sophisticate leveling structure, typically referred to as so-called "straddling sets". However we emphasize that, this structure does not affect our attacks. Therefore we will just ignore this in our setting.

observations of Miles et al. [MSZ16a] are: (1) for every known obfuscation construction, r_x is a *program dependent* function of $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$, and (2) for a large enough $m \in \mathbb{Z}$ the values $\{r_{x_k}\}_{k=1}^{m}$ must be correlated, which in turn implies that there exists a (program-dependent) efficiently computable function f^{BP} and input choices $\{x_k^{BP}\}_{k=1}^{m}$ such that for all $k \in [m]$, $BP(x_k^{BP}) = 0$ and $f^{BP}(\{r_{x_k^{BP}}\}_{k=1}^{m}) = 0 \mod p$.[10] Further, just like Miles et al. we are interested in constructing an attacker for the *indistinguishability* notion of obfuscation. In this case, given two arbitrarily distinct programs BP and BP' (such that $\forall x, BP(x) = BP'(x)$) an attacker needs to distinguish between the obfuscations of BP and BP'. Therefore, to complete the attack, it suffices to argue that for the sequence of $\{r'_{x_k^{BP'}}\}$ values obtained from execution of BP' it holds that, $f^{BP}(\{r'_{x_k^{BP'}}\}_{k=1}^{m}) \neq 0 \mod p$. Hence, the task of attacking any obfuscation scheme reduces to the task of finding such distinguishing function f^{BP}.

Miles et al. [MSZ16a] accomplishes that by presenting specific examples of branching programs, both of which implement the constant zero function, and a corresponding distinguishing function. They then extend the attack to other related branching programs that are padded with those constant-zero programs. The details of their attack [MSZ16a] is quite involved, hence we jump directly to the intuition behind our envisioned more general attacks.

Partial Inequivalence of Branching Programs and Our Attacks. We start with the following observation. For BGKPS-like-obfuscations for any branching program $BP = \{A_0, \{A_{i,0}, A_{i,1}\}_{i=1}^{\ell}, A_{\ell+1}\}$ the value $s_x = r_x \mod p$ looks something like: [11]

$$s_x \simeq \prod_{i=1}^{\ell} \alpha_{i,x[\mathsf{inp}(i)]} \underbrace{\left[\sum_{i=0}^{\ell+1} \left(\prod_{j=0}^{i-1} A_{j,x_{\mathsf{inp}(j)}} \cdot Z_{i,x[\mathsf{inp}(i)]} \cdot \prod_{j=i+1}^{\ell+1} A_{j,x_{\mathsf{inp}(j)}}\right)\right]}_{t_x},$$

[10] This follows from the existence of an annihilating polynomial for any over-determined non-linear systems of equations. We refer to [Kay09] for more details.

[11] Obtaining this expression requires careful analysis that is deferred to the main body of the chapter. Also, by abuse of notation let $A_{0,x_{\mathsf{inp}(0)}} = A_0$, $A_{\ell+1,x_{\mathsf{inp}(\ell+1)}} = A_{\ell+1}$, $Z_{0,x_{\mathsf{inp}(0)}} = Z_0$ and $Z_{\ell+1,x_{\mathsf{inp}(\ell+1)}} = Z_{\ell+1}$.

where $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$ are the randomnesses contributed by the corresponding encodings. Let $\overline{x}$ denote the value obtained by flipping every bit of x (a.k.a. the bitwise complement). Now observe that the product value $\Lambda = \prod_{i=1}^{\ell} \alpha_{i,x[\mathsf{inp}(i)]} \cdot \alpha_{i,\overline{x}[\mathsf{inp}(i)]}$ is independent of x. Therefore, $u_x = s_x \cdot s_{\overline{x}} = \Lambda \cdot t_x \cdot t_{\overline{x}}$. Absorbing Λ in the $\{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}$, we have that u_x is quadratic in the randomness values $\{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$, or linear in the random terms ZZ' obtained by multiplying every choice of $Z, Z' \in \{Z_0, \{Z_{i,0}, Z_{i,1}\}_{i=1}^{\ell}, Z_{\ell+1}\}$. In other words if BP evaluates to 0 both on inputs x and $\overline{x}$, the values revealed by two zero-test operations give one linear equation where the coefficients of the linear equations are program dependent. Now, if BP implements a "sufficiently non-evasive" circuit,(e.g. a PRF) such that there exist sufficiently many such inputs $x, \overline{x}$ for which $BP(x) = BP(\overline{x}) = 0$, then collecting sufficiently many values $\{x_k^{BP}, u_{x_k^{BP}}\}_{k=1}^{m}$, we get a dependent system of linear relations. Namely, there exist $\{\nu_k^{BP}\}_{k=1}^{m}$ such that $\sum_{k=1}^{m} \nu_k^{BP} \cdot u_{x_k^{BP}} = 0$. In other words, $\sum_{k=1}^{m} \nu_k^{BP} \cdot r_{x_k^{BP}} \cdot r_{\overline{x}_k^{BP}} = 0 \mod p$, where $\{\nu_k^{BP}\}_{k=1}^{m}$ depends only on the description of the branching program BP.

We remark that, in the process of linearization above we increased (by a quadratic factor) the number of random terms in the system. However, this can be always compensated by using more equations, because the number of random terms is $O(poly(n))$ (n is the input length) whereas the number of choices of input x is $2^{O(n)}$ which implies that there are exponentially many r_x available.

Note that for any branching program BP' that is "different enough" from BP, we could expect that $\sum_{k=1}^{m} \nu_k^{BP} \cdot r'_{x_k^{BP}} \cdot r'_{\overline{x}_k^{BP}} \neq 0 \mod p$ where $r'_{x_k^{BP}}$ are values revealed in executions of an obfuscation of BP'. This is because the values $\{\nu_k^{BP}\}_{k=1}^{m}$ depend on the specific implementation of BP through terms of the form $\prod_{j=0}^{i-1} A_{j,x[\mathsf{inp}(i)]}$ and $\prod_{j=i+1}^{\ell+1} A_{j,x[\mathsf{inp}(i)]}$ in s_x above. Two branching programs that differ from each other in this sense are referred to as *partially inequivalent*.[12]

What Programs are Partially Inequivalent? Attack on NC^1 circuits. The condition we put forth seems to be fairly generic and intuitively should work for large class of programs.

[12]Note that the only other constraint we need is that both BP and BP' evaluates to 0 for sufficiently many inputs, which we include in the definition (c.f. Def. 6.4.2) of partial inequivalence.

In particular, we are interested in the programs generated from NC^1 circuits. However, due to complex structures of such programs the analysis becomes quite non-trivial.[13] Nonetheless, we manage to show via implementation in Sage [S+16] (c.f. Section 6.8) that the attack indeed works on a pair of branching programs obtained from a pair of simple NC^1 circuits, (say C_0, C_1) (see Sec. 6.7 for the circuit descriptions) by applying Barrington's Theorem. The circuits take 4 bits of inputs and on any input they evaluate to 0. In our attack we use all possible 16 inputs. Furthermore, we can escalate the attack to any pair of NC^1 circuits (E_0, E_1) where $E_b = \neg C_b \wedge D_b$ ($b \in \{0, 1\}$) for practically any two NC^1 circuits D_0, D_1 (we need only one input x for which $D(x) = D(\overline{x}) = 0$). We now take again a sequence of 16-inputs such that we vary the parts of all the inputs going into C_b and keep the part of inputs read by D_b fixed to x. Intuitively, since the input to D_b is always the same, each evaluation chooses the exactly same randomnesses (that is Z_i's) always. Hence in the resulting system all the random variables can be replaced by a single random variable and hence $\neg C_b \wedge D_b$ can be effectively "collapsed" to a much smaller circuit $\neg C_b \wedge 0$ (0 refers to the smallest trivial circuit consisting of only identities). Finally, again via our Sage-implementation we show that for circuits $\neg C_0 \wedge 0$ and $\neg C_1 \wedge 0$ the corresponding branching programs are partially inequivelent.

As a consequence of the above we are also able to show examples of universal circuits U_b for which the same attack works. Since the circuit D can be almost any arbitrary NC^1 circuit, we can, in particular use any universal circuit U' and carefully combine that with C to obtain our attackable universal circuit U that results in partially inequivalent Barrigton programs when compiled with any two arbitrary NC^1 circuits. The details are provided in Section 6.7.3.

[13] Note that, the analysis of Miles et al. uses 2×2 matrices in addition to using simple branching programs. These simplifications allow them to base their analysis on many facts related to the structures of these programs. Our aim here is to see if the attack works for programs obtained from NC^1 circuits, in particular via Barrington's Theorem. So, unfortunately it is not clear if their approach can be applicable here as the structure of the programs yielded via Barrington's Theorem become much complex structurally (and also much larger in size) to analyze.

6.1.3 Roadmap

The rest of this chapter is organized as follows. We provide basic definitions in Sec. 6.2. In Sec. 6.3 we formalize our abstract-attack model that is mostly similar to the attack model considered by Miles et al. [MSZ16a]. In Sec. 6.4 we formalize partial inequivalence of two branching programs. In Sec. 6.5 we describe our annihilation attack in the abstract model for two partially inequivalent branching programs. In Sec. 6.6 we then extend the abstract attack to real-world attack in GGH13 setting. Finally in Sec. 6.7 we provide details on our example NC^1 circuits for which the corresponding branching programs generated via Barrington's Theorem are partially inequivalent.

Additionally, in Section 6.8 we provide some details on our implementations in Sage.

6.2 Additional Notations and Preliminaries

6.2.1 Notations

We denote the set of natural numbers $\{1, 2, \ldots\}$ by $\mathbb{N}$, the set of all integers $\{\ldots, -1, 0, 1 \ldots\}$ by $\mathbb{Z}$ and the set of real numbers by $\mathbb{R}$. We use the notation $[n]$ to denote the set of first n natural numbers, namely $[n] \stackrel{\text{def}}{=} \{1, \ldots, n\}$.

For any bit-string $x \in \{0,1\}^n$ we let $x[i]$ denotes the i-th bit. For a matrix A we denote its i-th row by $A[i, \star]$, its j-th column by $A[\star, j]$ and the element in the i-th row and j-th column by $A[i, j]$. The i-th element of a vector $\boldsymbol{v}$ is denoted by $\boldsymbol{v}[i]$.

Bit-Strings. The compliment of $x \in \{0,1\}^n$ is denoted by $\overline{x}$ and defined as: $\overline{x} \stackrel{\text{def}}{=} 1^n \oplus x$, where $\oplus$ denotes the bitwise XOR operation. The hamming weight of $x \in \{0,1\}^n$ denoted by $\mathsf{Ham}(x)$ is equal to $\sum_i x[i]$.

Matrices. The transpose of A is denoted by A^T. We denote matrix multiplications between two matrices A and B by $A \cdot B$ whereas scalar multiplications between one scalar a with a matrix (or scalar) A by aA. A boolean matrix is a matrix for which each of its entries

is from $\{0, 1\}$. A permutation matrix is a boolean matrix such that each of its rows and columns has exactly one 1. Concatenation of two matrices A, B of dimensions $d_1 \times d_2$ and $d_1 \times d_2'$ is a $d_1 \times (d_2 + d_2')$ matrix denoted by $[A \mid B]$. For multiple matrices $A_1, A_2, \ldots, A_m$ the concatenation is denoted as $[\mathbf{\Pi}_{i \in [n]} A_i]$.

Vectors. Matrices of dimension $1 \times d$ and $d \times 1$ are referred to as row-vectors and column-vectors, respectively. Unless otherwise mentioned, by default we assume that a vector is a row-vector. Any matrix operation is also applicable for vectors. For example, the inner product $\boldsymbol{a} \cdot \boldsymbol{b}$ is a scalar defined as $\boldsymbol{a} \cdot \boldsymbol{b} \stackrel{\text{def}}{=} \sum_{i=1}^{d} \boldsymbol{a}[i]\boldsymbol{b}[i]$, where $\boldsymbol{a}$ and $\boldsymbol{b}$ are row and column vectors of dimension d respectively. We define the *vectorization* of any matrix M of dimension $d_1 \times d_2$ to be a column vector of dimension $d_1 d_2 \times 1$ that is obtained by concatenating the rows of the matrix M and then taking the transpose. We denote:

$$\mathbf{vec}\,(M) = \big[M[1, \star] \mid M[2, \star] \mid \cdots \mid M[d_1, \star]\big]^T.$$

Note that if M is a column-vector then $\mathbf{vec}\,(M) = M$ and if M is a row-vector then $\mathbf{vec}\,(M) = M^T$.

6.2.2 Matrix Products

Below, we provide additional notation and background on matrix products that will be needed in our technical sections.

Definition 6.2.1 (Matrix Tensor Product (Kronecker Product))**.** *The Tensor Product of a $d_1 \times d_2$ matrix A and a $d_1' \times d_2'$ matrix B is a $d_1 d_1' \times d_2 d_2'$ matrix defined as:*

$$A \otimes B = \begin{bmatrix} A[1,1]B & \cdots & A[1,d_2]B \\ \vdots & \ddots & \vdots \\ A[d_1,1]B & \cdots & A[d_1,d_2]B \end{bmatrix}$$

where $A[i,j]B$ is a matrix of dimension $d_1' \times d_2'$ that is a scalar product of the scalar $A[i,j]$

and matrix B.

Property 6.2.2 (Rule of Mixed Product)**.** *Let A, B, C and D be matrices for which the matrix multiplications $A \cdot B$ and $C \cdot D$ is defined. Then we have:*

$$(A \cdot B) \otimes (C \cdot D) = (A \otimes C) \cdot (B \otimes D).$$

Property 6.2.3 (Matrix Equation via Tensor Product)**.** *Let A, X and B be matrices such that the multiplication $A \cdot X \cdot B$ is defined, then we have that:*

$$\mathbf{vec}\,(A \cdot X \cdot B) = (A \otimes B^T) \cdot \mathbf{vec}\,(X)$$

We define a new matrix product.

Definition 6.2.4 (Row-wise Tensor Product of Matrices)**.** *Let A and B be two matrices of dimensions $d_1 \times d_2$ and $d_1 \times d_2'$ respectively. Then the row-wise tensor product of A and B is a matrix C of dimension $d_1 \times d_2 d_2'$ such that each row of C is a tensor product of rows of A and B. Formally,*

$$C = A \boxtimes B \text{ where } C[i, \star] \stackrel{\text{def}}{=} A[i, \star] \otimes B[i, \star].$$

The following fact is straightforward to see.

Fact 6.2.5 (Concatenation of Row-wise Tensors)**.** *Let $A \stackrel{\text{def}}{=} [A_1 \mid A_2 \mid \cdots \mid A_m]$ and $B \stackrel{\text{def}}{=} [B_1 \mid B_2 \mid \cdots \mid B_n]$ be two matrices, then we have:*

$$A \boxtimes B = [\mathbf{\Pi}_{i \in [m], j \in [n]} A_i \boxtimes B_j].$$

Definition 6.2.6 (Permutation Equivalence)**.** *Let A, B be matrices with dimensions $d_1 \times d_2$, then A and B are called permutation equivalent if there exists a permutation matrix P such that $A = B \cdot P$. We denote by $A \stackrel{\text{per}}{=} B$*

Property 6.2.7. *For any two matrices A and B of dimensions $d_1 \times d_2$ and $d_1 \times d_2'$ respectively we have that:*

$$A \boxtimes B \overset{\text{per}}{=} B \boxtimes A$$

Proof. Let $C \overset{\text{def}}{=} A \boxtimes B$ then for any $k \in [d_2 d_2']$ the k-th column of C can be written as:

$$C[\star, k] = \begin{bmatrix} A[1,j]B[1,i] \\ \vdots \\ A[d_1,j]B[d_1,i], \end{bmatrix}$$

where $i = k \mod d_2'$ and $j = \frac{k-i}{d_2'} + 1$. For $\ell \in [d_2']$, define the matrix

$$D_\ell = [C[\star, \ell] \mid C[\star, \ell + d_2'] \mid \ldots \mid C[\star, \ell + d_2'(d_2 - 1)]].$$

Observe that we can express $B \boxtimes A$ as follows:

$$B \boxtimes A = [D_1 \mid \ldots \mid D_{d_2'}] = (A \boxtimes B) \cdot P$$

where P is a permutation matrix that maps the k-th column of $A \boxtimes B$ to the $d_2(i-1) + j$-th column where $i = k \mod d_2'$ and $j = \frac{k-i}{d_2'} + 1$. □

6.2.3 Column Space of a Matrix

Our attacks will require certain properties on the column space of certain matrices which we elaborate on below.

Definition 6.2.8 (Column Space of a matrix)**.** *Let A be a matrix of dimension $d_1 \times d_2$. Then the column space of A is defined as the vector space generated by linear combinations of its columns, formally the column space contains all vectors generated as $\sum_{i=1}^{d_2} c_i A[\star, i]$ for all choices of $c_i \in \mathbb{R}$. We denote the column-space of A by $\mathsf{colsp}(A)$.*

Definition 6.2.9 (Null-space of a matrix)**.** [14] *Let A be a matrix of dimension $d_1 \times d_2$. Then the null-space of A consists of all vectors $\boldsymbol{v}$ of dimension $1 \times d_1$ for which $\boldsymbol{v} \cdot A = 0$. We denote the null-space of A by* $\mathsf{nullsp}(A)$.

We state some basic property of the above vector spaces.

Property 6.2.10 ([Ogu16])**.** *Let A and B be two matrices of dimensions $d_1 \times d_2$. Then the following statements are equivalent:*

- $\mathsf{colsp}(A) = \mathsf{colsp}(B)$.
- $\mathsf{nullsp}(A) = \mathsf{nullsp}(B)$.
- *There exists an invertible square matrix C such that $A \cdot C = B$.*

Corollary 6.2.11. *Since $A \overset{\text{per}}{=} B$ is a special case of item-3 in the above property, we have that $A \overset{\text{per}}{=} B \implies \mathsf{colsp}(A) = \mathsf{colsp}(B)$.*

Combining above corollary along with Property 6.2.7 we can get the following corollary.

Corollary 6.2.12. *For any two matrices A and B having equal number of rows we have that*

$$\mathsf{colsp}(A \boxtimes B) = \mathsf{colsp}(B \boxtimes A)$$

Next we prove the following lemma that will be useful later in Sec. 6.7.

Lemma 6.2.13. *Let A and B be two boolean matrices of dimensions $d_1 \times d_2$ and $d_1 \times d_2'$ such that both A and B have equal number of 1's in each of its rows. Then we have:*

$$\mathsf{colsp}(A) \subseteq \mathsf{colsp}(A \boxtimes B) \quad \textit{and} \quad \mathsf{colsp}(B) \subseteq \mathsf{colsp}(A \boxtimes B)$$

[14] Traditionally such space is called left-null space or co-kernel.

Proof. For each column $A[\star, j]$ of A, we define the matrix $W_j \in \{0,1\}^{d_1 \times d_2'}$ as a row-wise tensor product between $A[\star, j]$ and B:

$$W_j = A[\star, j] \boxtimes B.$$

Summing up the columns of W_j we get:

$$\sum_{j'} W_j[\star, j'] = \begin{bmatrix} A[1,j] \sum_{j'} B[1,j'] \\ \vdots \\ A[d_1,j] \sum_{j'} B[d_1,j'] \end{bmatrix} = c(A[\star, j]).$$

for some integer c. Moreover we can write $A \boxtimes B$ as:

$$A \boxtimes B = [W_1 \mid W_2 \mid \ldots \mid W_{d_2}].$$

Hence there is a linear combination of columns of $A \boxtimes B$ that generates the j-th column of A for any $j \in [d_2]$. This allows us to conclude that $\mathsf{colsp}(A) \subseteq \mathsf{colsp}(A \boxtimes B)$. Now similar to the proof of the statement $\mathsf{colsp}(A) \subseteq \mathsf{colsp}(A \boxtimes B)$ we can prove that:

$$\mathsf{colsp}(B) \subseteq \mathsf{colsp}(B \boxtimes A).$$

From Corollary 6.2.12 we get that $\mathsf{colsp}(A \boxtimes B) = \mathsf{colsp}(B \boxtimes A)$. This allows us to conclude that $\mathsf{colsp}(B) \subseteq \mathsf{colsp}(A \boxtimes B)$. □

6.2.4 Branching Programs

In this subsection, we recall definitions of branching programs.

Definition 6.2.14 (w-ary Input-Oblivious Matrix Branching Program [BGK+14])**.** *A w-ary input oblivious matrix branching program of dimension d, length ℓ over n-bit inputs is given*

by a tuple,

$$\mathbf{A} = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}^w}, A_{\ell+1})$$

where $\mathsf{inp}(\cdot) : [\ell] \to [n]^w$ *is a function such that* $\mathsf{inp}(i)$ *is the set of* w *bit locations of the input examined in step* i*;* $A_{i,b}$ *are permutation matrices over* $\{0,1\}^{d\times d}$ *and* $A_0 \in \{0,1\}^{1\times d} \setminus (0^d)^T$*,* $A_{\ell+1} \in \{0,1\}^{d\times 1} \setminus 0^d$ *are fixed bookend vectors such that:*

$$A_0 \cdot A \cdot A_{\ell+1} = \begin{cases} 0 & \text{if and only if } A = I_{d\times d} \\ 1 & \text{otherwise.} \end{cases} \tag{6.1}$$

The output of the matrix branching program on an input $x \in \{0,1\}^n$ *is given by:*

$$\mathbf{A}(x) = \begin{cases} 1 & \text{if } A_0 \left(\prod_{i\in[\ell]} A_{i,x[\mathsf{inp}(i)]}\right) A_{\ell+1} = 1 \\ 0 & \text{if } A_0 \left(\prod_{i\in[\ell]} A_{i,x[\mathsf{inp}(i)]}\right) A_{\ell+1} = 0 \end{cases},$$

where $\mathsf{inp}(i)$ *denotes the set of locations that are inspected at step* i *of* $\mathbf{A}$ *and* $x[\mathsf{inp}(i)]$ *denotes the bits of* x *at locations* $\mathsf{inp}(i)$*. A* w*-ary branching program is said to be* input-oblivious *if the function* inp *is fixed and independent of the program* $\mathbf{A}$*.*

Remark 6.2.15. *A* 1*-ary branching program is also called a* single-input *branching program. Unless otherwise stated we will always assume that any branching program is single-input and input-oblivious.*

Barrington [Bar89] showed that all circuits in NC^1 can be equivalently represented by a branching program of polynomial length. We provide the theorem statement below adapted to our definition of branching programs.

Theorem 6.2.16 (Barrington's Theorem [Bar89])**.** *For any depth-*D*, fan-in-2 boolean circuit* C*, there exists an input oblivious branching program of matrix-dimension 5 and length at most* 4^D *that computes the same function as the circuit* C*.*

Given a circuit C of depth D, Barrington's Theorem provides yield a single-input

branching program of matrix dimension 5 implementing circuit C. We stress that the specific implementation obtained by use of Barrigton's depends on the specific choices made in its implementation and therefore the obtained implementation is *not* unique. Sometimes the branching program obtained via applying Barrington's Theorem to a circuit is called the Barrington-implementation of that circuit. We choose a specific one for our Sage-implementation. The details are provided in Section 6.8.

6.3 Attack Model for Investigating Annihilation Attacks

Miles, Sahai, and Zhandry [MSZ16a] describe an abstract obfuscation scheme, designed to encompass the main ideas of BGKPS-like-obfuscations [BGK+14, BR14, AGIS14, PST14, MSW14, BMSZ16] for the purposes of investigating annihilation attacks. We use the same abstract attack model as the starting point for our new attacks. Below, we first describe the model, obfuscation in this model and what violating indistinguishability obfuscation security means.

6.3.1 Annihilation Attack Model

We describe the abstract annihilation attack model. An abstract model is parameterized with n *arbitrary secret* variables $X_1, ..., X_n$, m *random secret* variables $Z_1, ..., Z_m$, a *special secret* variable g. Then the *public* variables $Y_1, \ldots, Y_m$ are such that $Y_i := q_i(\{X_j\}_{j\in[n]}) + gZ_i$ for some polynomials q_i. The polynomials are defined over a field F.[15] An abstract model attacker $\mathcal{A}$ may adaptively make two types of queries:

- **Type 1: Pre-Zeroizing Computation.** In a **Type 1** query, the adversary $\mathcal{A}$ submits a "valid" polynomial p_k on the public Y_i. Here, valid polynomials are those polynomials

[15] Looking ahead, the arbitrary variables represent the plain-texts (the branching program or circuit to be obfuscated) of encoding, the random variables represent the randomness of encodings generated by the obfuscator, the variable g represents the "short" generator $\boldsymbol{g}$ of the ideal lattice and the public variables represent the encodings available to the attacker.

as enforced by the graded encodings.[16]

Then, we expand the representation of the (public) polynomial on Y_i in order to express p_k as a polynomial of the (private) formal variables X_j, Z_i, g stratified in the powers of g as follows:

$$p_k = p_k^{(0)} + g \cdot p_k^{(1)} + g^2 \cdot p_k^{(2)} + \dots .$$

If p_k is identically 0 or if $p_k^{(0)}$ is not identically 0, then the adversary $\mathcal{A}$ receives $\perp$ in return. Otherwise, the adversary $\mathcal{A}$ receives a new handle to a new variable W_k, which is set to be

$$W_k := p_k/g = p_k^{(1)} + g \cdot p_k^{(2)} + g^2 \cdot p_k^{(3)} + \dots .$$

- **Type 2: Post-Zeroizing Computation.** In a **Type 2** query, the adversary $\mathcal{A}$ is allowed to submit *arbitrary* polynomials r of polynomial degree, on the W_k that it has seen so far. We again view $r(\{W_k\})$ as a polynomial of the (secret) formal variables X_j, Z_i, g, and write it as:

  $$r = r^{(0)} + g \cdot r^{(1)} + g^2 \cdot r^{(2)} + \dots .$$

  If $r^{(0)}$ is identically 0, then the adversary $\mathcal{A}$ receives 0 in return. Otherwise, the adversary $\mathcal{A}$ receives 1 in return.

Comparing the Abstract Model to other Idealized Models. We briefly compare the Abstract Model described above to the ideal graded encoding model that has traditionally been used to argue about obfuscation security in prior works, e.g. as in the [BR14, BGK+14]. All adversarial behavior allowed within the Ideal Graded Encoding model is captured by **Type 1** queries in the Abstract Model and the **Type 2** queries are not considered. The works

[16] For example, for a branching program obfuscation it must be a correct (and complete) evaluation of a branching program on some specific input as directed by the inp function of the program.

of [GMM+16, DGG+16] argue security in this new model also referred to as the *Weak Multilinear Map Model*.

6.3.2 Obfuscation in the Annihilation Attack Model

The abstract obfuscator $\mathcal{O}$ takes as input a single-input branching program A of length ℓ, input length n. We describe our obfuscation using notation slightly different from Miles et al. [MSZ16a] as it suits our setting better and is closer to notation of branching programs (Def. 6.2.4). The branching program has an associated input-indexing function $\mathsf{inp} : [\ell] \to [n]$. The branching program has $2\ell + 2$ matrices $A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1}$. In most generality, in order to evaluate a branching program on input x, we compute the matrix product

$$A(T) = A_0 \cdot \prod_{i=1}^{\ell} A_{i,x[\mathsf{inp}(i)]} \cdot A_{\ell+1},$$

where $x[\mathsf{inp}(i)]$ denotes the bit of x at locations described by the set $\mathsf{inp}(i)$. Finally the program outputs 0 if and only if $A(T) = 0$.

The abstract obfuscator randomizes its input branching program by sampling random matrices $\{R_i\}_{i\in[\ell+1]}$ (Killian randomizers) and random scalars $\{\alpha_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$, then setting

$$\widetilde{A_0} := A_0 \cdot R_1^{adj}, \qquad \widetilde{A_{i,b}} := \alpha_{i,b}(R_i \cdot A_{i,b} \cdot R_{i+1}^{adj}), \qquad \widetilde{A_{\ell+1}} := R_{\ell+1} \cdot A_{\ell+1}.$$

that are the abstract model's arbitrary secret variables. Here R^{adj} denotes the adjugate matrix of R that satisfies $R^{adj} \cdot R = det(R) \cdot I$. Then the obfuscator defines the public variables to be

$$Y_0 := \widetilde{A_0} + gZ_0; \;\; Y_{i,b} := \widetilde{A_{i,b}} + gZ_{i,b}; \;\; Y_{\ell+1} := \widetilde{A_{\ell+1}} + gZ_{\ell+1},$$

where g is the special secret variable and Z_is are the random variables. This defines the abstract obfuscated program $\mathcal{O}(A) = \{Y_i\}_i$. The set of valid **Type 1** polynomials consists

of all the honest evaluations of the branching program. This is, the allowed polynomials are

$$p_x = Y_0 \cdot \prod_{i=1}^{\ell} Y_{i,x[\mathsf{inp}(i)]} \cdot Y_{\ell+1},$$

for all $x \in \{0,1\}^n$.[17]

6.3.3 Abstract Indistinguishability Obfuscation Security

We define security of $i\mathcal{O}$ in the abstract model. Formally consider the following indistinguishability game consisting of three phases.

Set Up. The adversary $\mathcal{A}$ comes up with a pair of matrix branching programs $(\mathbf{A}_0, \mathbf{A}_1)$ that are (i) functionally equivalent, (ii) of same length and (iii) input oblivious and some auxiliary information aux. $\mathcal{A}$ outputs the pair $(\mathbf{A}_0, \mathbf{A}_1)$ to the challenger.

Challenge. The challenger applies the abstract obfuscator $\mathcal{O}$ to a branching program, uniformly chosen as $\mathbf{A}_b \leftarrow \{\mathbf{A}_0, \mathbf{A}_1\}$ and returns the public variables $\{Y_0, \{Y_{i,b}\}, Y_{\ell+1}\}$, generated by applying $\mathcal{O}$ to $\mathbf{A}_b$, to the adversary.

Pre-zeroing (Type-1) Queries. In this phase the adversary makes several type-1 valid queries p_k and gets back handles $\{W_1, W_2, \ldots\}$.

Post-zeroing (Type-2) Query. In this phase the adversary makes one type-2 query r with some degree $poly(\lambda)$ polynomial Q over the formal variables corresponding to handles $\{W_1, W_2, \ldots\}$ and receives a bit as a response from the challenger. Finally $\mathcal{A}$ outputs its guess $b' \in \{0,1\}$.

Definition 6.3.1 (Abstract $i\mathcal{O}$ Security)**.** *An abstract obfuscation candidate $\mathcal{O}$ is called an indistinguishability obfuscator if for any probabilistic polynomial time adversary $\mathcal{A}$ the*

[17] Looking ahead, the Z_is are random noise component sampled in the encoding procedure of GGH13 maps and g is a "short" generator of the ideal lattice. The abstract model is agnostic to the exact choice of those variables, but only depends on the structure of the variables.

probability that $\mathcal{A}$ guesses the choice of $\mathbf{A}_b$ correctly is negligibly close to $1/2$. Formally, in the above game

$$|Pr[b = b'] - 1/2| \leq \mathsf{negl}(\lambda)$$

for any security parameter $\lambda \in \mathbb{N}$, where the probability is over the randomness of $\mathcal{A}$ and the challenger.

6.4 Partially Inequivalent Branching Programs

In this section, we provide a formal condition on two branching programs, namely partial inequivalence, that is sufficient for launching a distinguishing attack in the abstract model. In Section 6.5 we prove that this condition is sufficient for the attack.[18]

All the below definitions consider single-input branching programs, but they naturally extends to multi-input setting.

Definition 6.4.1 (Partial Products)**.** *Let $\mathbf{A} = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1})$ be a single-input branching program of matrix-dimension d and length ℓ over n-bit input.*

1. *For any input $x \in \{0,1\}^n$ and any index $i \in [\ell+1] \cup \{0\}$ we define the vectors $\phi^{(i)}_{\mathbf{A},x}$ as follows:*

$$\phi^{(i)}_{\mathbf{A},x} \stackrel{\text{def}}{=} \begin{cases} \left(A_0 \cdot \prod_{j=1}^{i-1} A_{j,x[\mathsf{inp}(j)]}\right) \otimes \left(\prod_{j=i+1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right)^T \in \{0,1\}^{1\times d^2} & \text{if } i \in [\ell] \\ \left(\prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right)^T \in \{0,1\}^{1\times d} & \text{if } i = 0 \\ A_0 \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \in \{0,1\}^{1\times d} & \text{if } i = \ell+1 \end{cases},$$

[18] We note that this condition is not necessary. Looking ahead, we only consider first order partially inequivalent programs in in this work and remark that higher order partially inequivalent programs could also be distinguished using our techniques.

Additionally, define $\widetilde{\phi}^{(i)}_{\mathbf{A},x}$ *for any such branching program as:*

$$\widetilde{\phi}^{(i)}_{\mathbf{A},x} \stackrel{\text{def}}{=} \begin{cases} [\phi^{(i)}_{\mathbf{A},x} \mid 0^{d^2}] & \text{if } i \in [\ell] \text{ and } x[\mathsf{inp}(i)] = 0 \\ [0^{d^2} \mid \phi^{(i)}_{\mathbf{A},x}] & \text{if } i \in [\ell] \text{ and } x[\mathsf{inp}(i)] = 1 \\ \phi^{(i)}_{\mathbf{A},x} & \text{if } i = 0 \text{ or } \ell+1 \end{cases},$$

where inp *is a function from* $[\ell] \to [n]$ *and that* $x[\mathsf{inp}(i)]$ *denotes the bit of* x *corresponding to location described by* $\mathsf{inp}(x)$.

2. *Then the* ***linear partial product vector*** $\boldsymbol{\phi}_{\mathbf{A},x}$ *and the* ***quadratic partial product vector*** $\boldsymbol{\psi}_{\mathbf{A},x}$ *of* $\mathbf{A}$ *with respect to* x *are defined as:*

$$\boldsymbol{\phi}_{\mathbf{A},x} \stackrel{\text{def}}{=} [\widetilde{\phi}^{(0)}_{\mathbf{A},x} \mid \cdots \mid \widetilde{\phi}^{(\ell+1)}_{\mathbf{A},x}] \in \{0,1\}^{1\times(2d+2\ell d^2)}.$$

$$\boldsymbol{\psi}_{\mathbf{A},x} \stackrel{\text{def}}{=} \boldsymbol{\phi}_{\mathbf{A},x} \otimes \boldsymbol{\phi}_{\mathbf{A},\overline{x}} \in \{0,1\}^{1\times(2d+2\ell d^2)^2},$$

where $\overline{x} = x \oplus 1^n$ *is the compliment of* x.

3. *For a set of inputs* $X = \{x_1, x_2, \ldots, x_m\}$ *the the* ***linear partial product matrix*** $\Phi_{\mathbf{A},X}$ *and the* ***quadratic partial product matrix*** $\Psi_{\mathbf{A},X}$ *of* $\mathbf{A}$ *with respect to* X *are defined as:*

$$\Phi_{\mathbf{A},X} \stackrel{\text{def}}{=} \begin{bmatrix} \boldsymbol{\phi}_{\mathbf{A},x_1} \\ \boldsymbol{\phi}_{\mathbf{A},x_2} \\ \vdots \\ \boldsymbol{\phi}_{\mathbf{A},x_m} \end{bmatrix} \in \{0,1\}^{m\times(2d+2\ell d^2)}$$

$$\Psi_{\mathbf{A},X} \stackrel{\text{def}}{=} \Phi_{\mathbf{A},X} \boxtimes \Phi_{\mathbf{A},\overline{X}} + \Phi_{\mathbf{A},\overline{X}} \boxtimes \Phi_{\mathbf{A},X} = \begin{bmatrix} \boldsymbol{\psi}_{\mathbf{A},x_1} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}_1} \\ \boldsymbol{\psi}_{\mathbf{A},x_2} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}_2} \\ \vdots \\ \boldsymbol{\psi}_{\mathbf{A},x_m} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}_m} \end{bmatrix} \in \{0,1\}^{m\times(2d+2\ell d^2)^2},$$

where $\overline{X} \stackrel{\text{def}}{=} \{\overline{x}_1, \overline{x}_2, \ldots\}$.

[19]

Definition 6.4.2 (Partial Inequivalence)**.** *Let* $\mathbf{A}_0$ *and* $\mathbf{A}_1$ *be two single-input matrix branching programs of matrix-dimension* d *and length* ℓ *over* n*-bit input. Then they are called* ***partially inequivalent*** *if there exists a polynomial in security parameter sized set* X *of inputs such that:*

- *For every* $x \in X$*, we have that* $\mathbf{A}_0(x) = \mathbf{A}_1(x) = 0$ *and* $\mathbf{A}_0(\overline{x}) = \mathbf{A}_1(\overline{x}) = 0$.
- $\mathsf{colsp}\left(\Psi_{\mathbf{A}_0,X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_1,X}\right)$.

Lemma 6.4.3. *For any matrix branching program* $\mathbf{A}$ *we have that for any two inputs* x, x' *the linear partial product vectors* $\boldsymbol{\phi}_{\mathbf{A},x}$ *and* $\boldsymbol{\phi}_{\mathbf{A},x'}$ *contain the same number of* 1*'s.*

Proof. Note that for any input x and index i, via definition of $\phi^{(i)}_{\mathbf{A},x}$, we have:

$$\phi^{(i)}_{\mathbf{A},x} = \left(A_0 \cdot P_{x,1} \otimes A^T_{\ell+1} \cdot P_{x,2}\right)$$

for some x dependent permutations $P_{x,1}$ and $P_{x,2}$. Note that A_0 is a row vector and therefore $A_0 \cdot P_{x,1}$ is also a row vector. Since $P_{x,1}$ is a permutation, we conclude that $\mathsf{Ham}(A_0 \cdot P_{x,1}) = \mathsf{Ham}(A_0)$ where $\mathsf{Ham}(A_0)$ is the hamming weight of the vector A_0 (specifically, the number of locations at which it is 1). Similarly, $\mathsf{Ham}(P_{x,2} \cdot A_{\ell+1}) = \mathsf{Ham}(A_{\ell+1})$. Hence, the $\mathsf{Ham}(\boldsymbol{\phi}_{\mathbf{A},x}) = \mathsf{Ham}(A_0)\mathsf{Ham}(A_{\ell+1})$ which is independent of x. Consequently, $\mathsf{Ham}(\boldsymbol{\phi}_{\mathbf{A},x}) = (\ell+2)\mathsf{Ham}(A_0)\mathsf{Ham}(A_{\ell+1})$ which is also independent of x. This concludes the proof. □

[19] Note that in the above definition we add the row-wise tensors. Looking ahead, this is done to capture the commutativity in the polynomial multiplications. Namely since for any two ring elements z_1, z_2 we have $z_1 z_2 = z_2 z_1$, their coefficients add up. Also note that the sum in the above expression equivalently double the coefficients of the quadratic terms z_1^2, z_2^2. But, due to our choices of inputs $x, \overline{x}$ we would only have such terms for the bookends which are nonetheless always stays the same (in fact they are independent of the actual program) and does not affect the column-space.

6.5 Annihilation Attack for Partially Inequivalent Programs

In this section, we describe an abstract annihilation attack against any two branching programs that are partially inequivalent. In this section, we show an attack only in the abstract model and provide details on how it can be extended to the real GGH13 setting in Section 6.6 . Formally we prove the following theorem.

Theorem 6.5.1. *Let $\mathcal{O}$ be the generic obfuscator described in Sec. 6.3.2. Then for any two functionally equivalent same length single-input branching programs $\mathbf{A}_0, \mathbf{A}_1$ that are partially inequivalent there exists a probabilistic polynomial time attacker that distinguishes between between $\mathcal{O}(\mathbf{A}_0)$ and $\mathcal{O}(\mathbf{A}_1)$ with noticeable probability in the abstract attack model (violating Definition 6.3.1).*

Proof of Theorem 6.5.1. Below we provide the proof.

Setup for the attack. The given branching programs $\mathbf{A}_0$ and $\mathbf{A}_1$ are provided to be functionally equivalent and partially inequivalent. Therefore there exists a set X such that: (1) for all $x \in X, \mathbf{A}_0(x) = \mathbf{A}_0(\overline{x}) = \mathbf{A}_1(x) = \mathbf{A}_1(\overline{x}) = 0$, and (2) $\mathsf{colsp}\left(\Psi_{\mathbf{A}_0,X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_1,X}\right)$. We will assume that the adversary has access to X as auxiliary information.

Challenge. $\mathcal{A}$ receives as a challenge the obfuscation of the branching program: $\mathbf{A} \in \{\mathbf{A}_0, \mathbf{A}_1\}$ by the challenger. Recall from the description of the abstract obfuscator that, the obfuscation of program $\mathbf{A} = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1})$, denoted by $\mathsf{obf}(\mathbf{A})$ consists of the following public variables:

$$Y_0 := A_0 \cdot R_1^{adj} + gZ_0, \;\; Y_{i,b} := \alpha_{i,b} R_i \cdot A_{i,b} \cdot R_{i+1}^{adj} + gZ_{i,b}, \;\; Y_0 := R_{\ell+1} \cdot A_{\ell+1} + gZ_0,$$

where the arbitrary secret variables are:

$$\widetilde{A}_0 \stackrel{\text{def}}{=} A_0 \cdot R_1^{adj}, \;\; \widetilde{A}_{i,b} \stackrel{\text{def}}{=} \alpha_{i,b}(R_{i,b} \cdot A_{i,b} \cdot R_{i,b}^{adj}), \;\; \widetilde{A}_{\ell+1} \stackrel{\text{def}}{=} R_{\ell+1} \cdot A_{\ell+1};$$

for random variables (i.e. Killian randomizers) $R_1, \{R_i\}, R_{\ell+1}$ and the random secret

variables are denoted by $Z_0, \{Z_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, Z_{\ell+1}$ and the special secret variable is g. Via change of variables we can equivalently write:

$$Y_0 := (A_0+gZ_0)\cdot R_1^{adj};\ Y_{i,b} := \alpha_{i,b}R_i\cdot(A_{i,b}+gZ_{i,b})\cdot R_{i+1}^{adj};\ Y_{\ell+1} := R_{\ell+1}\cdot(A_{\ell+1}+gZ_{\ell+1}).$$

Pre-Zeroizing Computation (Type-1 queries). On receiving the obfuscation of $\mathbf{A} \in \{\mathbf{A}_0, \mathbf{A}_1\}$, $\mathsf{obf}(\mathbf{A}) = \{Y_0, \{Y_{i,b}\}, Y_{\ell+1}\}$ the attacker, in the pre-zeroizing step, performs a "valid" Type-1 queries on all the inputs $X, \overline{X}$ where $X = \{x_1, \ldots, x_m\}, \overline{X} = \{\overline{x}_1, \ldots, \overline{x}_m\}$. That is, for any $x \in \{0,1\}^n$, and the abstract obfuscation $\mathsf{obf}(\mathbf{A})$, the attacker queries the polynomial:

$$P_{\mathbf{A},x} = Y_0 \cdot \prod_{i=1}^{\ell} Y_{i,x[\mathsf{inp}(i)]} \cdot Y_{\ell+1}.$$

Then, expressing $P_{\mathbf{A},x}$ stratified as powers of g we obtain:

$$P_{\mathbf{A},x} = P^{(0)}_{\mathbf{A},x}(\{Y_i\}_i) + g \cdot P^{(1)}_{\mathbf{A},x}(\{Y_i\}_i) + ... + g^{\ell+2} \cdot P^{(\ell+2)}_{\mathbf{A},x}(\{Y_i\}_i)$$

for some polynomials $P^{(j)}_{\mathbf{A},x}(\{Y_i\}_i)$ ($j \in \{0, ..., \ell+1\}$). However, by Lemma 6.5.2 we have that:

$$P^{(0)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x\mathbf{A}(x)$$

for $\rho \stackrel{\text{def}}{=} \prod_i \det(R_i)$ (or $\rho I = \prod_i R_i^{adj} R_i$) and $\widehat{\alpha}_x \stackrel{\text{def}}{=} \prod_{i=1}^{\ell} \alpha_{i,x_{\mathsf{inp}(i)}}$. Since for $x \in X$ we have that $\mathbf{A}(x) = 0$, the polynomial $P^{(0)}_{\mathbf{A},x}$ is identically 0. Consequently, for each such Type 1 query the attacker receives a new handle to a variable $W_{\mathbf{A},x}$ that can be expressed as follows:

$$W_{\mathbf{A},x} = P_{\mathbf{A},x}/g = P^{(1)}_{\mathbf{A},x} + g \cdot P^{(2)}_{\mathbf{A},x} + ... + g^{\ell+1} \cdot P^{(\ell+2)}_{\mathbf{A},x}.$$

Analogously, the attacker obtains handles $W_{\mathbf{A},\overline{x}}$. After obtaining handles

$$\{(W_{\mathbf{A},x_1}, W_{\mathbf{A},\overline{x}_1}), ...(W_{\mathbf{A},x_m}, W_{\mathbf{A},\overline{x}_m})\}$$

the attacker starts the post-zeroizing phase.

Post-Zeroizing Computation. The goal of post-zeroizing computation is to *find* a polynomial Q^{ann} of degree $poly(\lambda)$ such that following holds for some $b \in \{0,1\}$:

(i) $Q^{\mathsf{ann}}(P^{(1)}_{\mathbf{A}_b,x_1}, P^{(1)}_{\mathbf{A}_b,\overline{x}_1}..., P^{(1)}_{\mathbf{A}_b,x_m}, P^{(1)}_{\mathbf{A}_b,\overline{x}_m}) \equiv 0.$

(ii) $Q^{\mathsf{ann}}(P^{(1)}_{\mathbf{A}_{1-b},x_1}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_1}..., P^{(1)}_{\mathbf{A}_{1-b},x_m}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_m}) \not\equiv 0.$

Clearly, this leads to an attack on the obfuscation security(c.f. Definition 6.3.1) as $\mathcal{A}$ would receive 0 from the challenger if and only if $Q^{\mathsf{ann}}(P^{(1)}_{\mathbf{A},x_1}, P^{(1)}_{\mathbf{A},\overline{x}_1}..., P^{(1)}_{\mathbf{A},x_m}, P^{(1)}_{\mathbf{A},\overline{x}_m})$ is identically zero, hence it would receive 0 if and only if $\mathbf{A}_b$ is chosen by the challenger in the challenge phase. To find such Q^{ann} the attacker continues as follows. Observe that by Lemma 6.5.2, for every $x \in X$ we have that:

$$P^{(1)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x(\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T) \tag{6.2}$$

$$P^{(1)}_{\mathbf{A},\overline{x}} = \rho\widehat{\alpha}_{\overline{x}}(\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}^T) \tag{6.3}$$

Next, multiplying the polynomials $P^{(1)}_{\mathbf{A},x}$ and $P^{(1)}_{\mathbf{A},\overline{x}}$ (Eq. 6.2 and Eq. 6.3) we get:

$$\widetilde{P}^{(1)}_{\mathbf{A},x} \stackrel{\text{def}}{=} P^{(1)}_{\mathbf{A},x}P^{(1)}_{\mathbf{A},\overline{x}} = \rho^2\widehat{\alpha}\left((\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T) \otimes (\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}^T)\right) \tag{6.4}$$

$$= \rho^2\widehat{\alpha}\left((\boldsymbol{\phi}_{\mathbf{A},x} \otimes \boldsymbol{\phi}_{\mathbf{A},\overline{x}}) \cdot (\boldsymbol{z}^T \otimes \boldsymbol{z}^T)\right) \tag{6.5}$$

$$= \rho^2\widehat{\alpha}(\boldsymbol{\psi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T)$$

where $\widehat{\alpha} \stackrel{\text{def}}{=} \widehat{\alpha}_x \widehat{\alpha}_{\overline{x}}$ is now independent of input x.[20] Similarly we can also have:

$$\begin{aligned}\widetilde{P}^{(1)}_{\mathbf{A},\overline{x}} \stackrel{\text{def}}{=} P^{(1)}_{\mathbf{A},\overline{x}} P^{(1)}_{\mathbf{A},x} &= \rho^2 \widehat{\alpha} \left((\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}^T) \otimes (\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T) \right) \\ &= \rho^2 \widehat{\alpha} \left((\boldsymbol{\phi}_{\mathbf{A},\overline{x}} \otimes \boldsymbol{\phi}_{\mathbf{A},x}) \cdot (\boldsymbol{z}^T \otimes \boldsymbol{z}^T) \right) \\ &= \rho^2 \widehat{\alpha} (\boldsymbol{\psi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T)\end{aligned}$$

However, since field multiplication is commutative, adding we get:

$$\begin{aligned}\widetilde{P}^{(1)}_{\mathbf{A},x} + \widetilde{P}^{(1)}_{\mathbf{A},\overline{x}} = 2P^{(1)}_{\mathbf{A},x} P^{(1)}_{\mathbf{A},\overline{x}} &= \rho^2 \widehat{\alpha} (\boldsymbol{\psi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T) + \rho^2 \widehat{\alpha} (\boldsymbol{\psi}_{\mathbf{A},\overline{x}} \cdot \boldsymbol{z}^T \otimes \boldsymbol{z}^T) \\ &= \rho^2 \widehat{\alpha} (\boldsymbol{\psi}_{\mathbf{A},x} + \boldsymbol{\psi}_{\mathbf{A},\overline{x}}) \cdot (\boldsymbol{z}^T \otimes \boldsymbol{z}^T)\end{aligned}$$

Using the given conditions that $\Psi_{\mathbf{A}_0,X}$ and $\Psi_{\mathbf{A}_1,X}$ have distinct column spaces (and hence distinct left-kernel) the attacker can efficiently compute (e.g. via Gaussian Elimination) a vector $\boldsymbol{v}_{\mathsf{ann}} \in \{0,1\}^{1 \times m}$ that belongs to it left-kernel, call it the *annihilating vector*, such that for some $b \in \{0,1\}$ we have:

$$\boldsymbol{v}_{\mathsf{ann}} \cdot \Psi_{\mathbf{A}_b,X} = 0 \qquad \text{but} \qquad \boldsymbol{v}_{\mathsf{ann}} \cdot \Psi_{\mathbf{A}_{1-b},X} \neq 0.$$

The corresponding annihilation polynomial Q^{ann} can be written as:

$$Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(W_{\mathbf{A},x_1}, W_{\mathbf{A},\overline{x}_1}, \ldots, W_{\mathbf{A},x_m}, W_{\mathbf{A},\overline{x}_m}) = \boldsymbol{v}_{\mathsf{ann}} \cdot \begin{bmatrix} W_{\mathbf{A},x_1} W_{\mathbf{A},\overline{x}_1} \\ \vdots \\ W_{\mathbf{A},x_m} W_{\mathbf{A},\overline{x}_m} \end{bmatrix}$$

Observe that the coefficient of g^0 in the expression $Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(W_{\mathbf{A},x_1}, W_{\mathbf{A},\overline{x}_1}, \ldots, W_{\mathbf{A},x_m}, W_{\mathbf{A},\overline{x}_m})$ from above is equal to $Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(P^{(1)}_{\mathbf{A}_b,x_1}, P^{(1)}_{\mathbf{A}_b,\overline{x}_1} ..., P^{(1)}_{\mathbf{A}_b,x_m}, P^{(1)}_{\mathbf{A}_b,\overline{x}_m})$. Moreover this value for

[20] Here, we use the fact that the branching programs are single-input. For multi-input programs we do *not* know how to make $\widehat{\alpha}$ independent of x. The rest of the analysis does not require the programs to be single-input.

$\mathbf{A} = \mathbf{A}_b$ is:

$$Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(P^{(1)}_{\mathbf{A}_b,x_1}, P^{(1)}_{\mathbf{A}_b,\overline{x}_1}..., P^{(1)}_{\mathbf{A}_b,x_m}, P^{(1)}_{\mathbf{A}_b,\overline{x}_m}) = \boldsymbol{v}_{\mathsf{ann}} \cdot \frac{\Psi_{\mathbf{A}_b,X}}{2} \cdot (\boldsymbol{z} \otimes \boldsymbol{z})^T \equiv 0$$

but for $\mathbf{A}_{1-b}$:

$$Q^{\mathsf{ann}}_{\boldsymbol{v}_{\mathsf{ann}}}(P^{(1)}_{\mathbf{A}_{1-b},x_1}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_1}..., P^{(1)}_{\mathbf{A}_{1-b},x_m}, P^{(1)}_{\mathbf{A}_{1-b},\overline{x}_m}) = \boldsymbol{v}_{\mathsf{ann}} \cdot \frac{\Psi_{\mathbf{A}_{1-b},X}}{2} \cdot (\boldsymbol{z} \otimes \boldsymbol{z})^T \not\equiv 0.$$

Hence, the response to Type 2 query is sufficient to distinguish between obfuscation of $\mathbf{A}_b$ and $\mathbf{A}_{1-b}$ in the abstract model. This concludes the proof.

Evaluations of $P^{(0)}_{\mathbf{A},x}$ and $P^{(1)}_{\mathbf{A},x}$. Below we provide a lemma that described what the terms $P^{(0)}_{\mathbf{A},x}$ and $P^{(1)}_{\mathbf{A},x}$ look like.

Lemma 6.5.2. *For every $x \in \{0,1\}^n$, we have that:*

$$P^{(0)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x\mathbf{A}(x)$$
$$P^{(1)}_{\mathbf{A},x} = \rho\widehat{\alpha}_x(\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T),$$

where $\rho \stackrel{\text{def}}{=} \prod_i \det(R_i)$ and $\widehat{\alpha}_x \stackrel{\text{def}}{=} \prod_{i=1}^{\ell} \alpha_{i,x_{\mathsf{inp}(i)}}$ and $\boldsymbol{z}$ is a vector consisting of the random terms $Z_0, Z_{i,b}$, and $Z_{\ell+1}$ used to generate the obfuscation terms $Y_0, Y_{i,b}$, and $Y_{\ell+1}$ in an appropriate sequence.

Proof of Lemma 6.5.2. For each $x \in \{0,1\}^n$ note that:

$$\begin{aligned}
P^{(0)}_{\mathbf{A},x} &= \widetilde{A}_0 \cdot \prod_{i=1}^{\ell} \widetilde{A}_{i,x[\mathsf{inp}(i)]} \cdot \widetilde{A}_{\ell+1} \\
&= A_0 \cdot R_1^{adj} \times \prod_{i=1}^{\ell} \left(\alpha_{i,x_{\mathsf{inp}(i)}} R_i \cdot A_{i,x[\mathsf{inp}(i)]} \cdot R_{i+1}^{adj}\right) \times R_{\ell+1} \cdot A_{\ell+1} \\
&= \rho\widehat{\alpha}_x\mathbf{A}(x)
\end{aligned}$$

for $\rho \stackrel{\text{def}}{=} \prod_i \det(R_i)$ (or $\rho I = \prod_i R_i^{adj} R_i$) and $\widehat{\alpha}_x \stackrel{\text{def}}{=} \prod_{i=1}^{\ell} \alpha_{i,x_{\mathsf{inp}(i)}}$.

Also, note that for any $x \in \{0,1\}^n$ we can express $P^{(1)}_{\mathbf{A},x}$ as:

$$
\begin{aligned}
P^{(1)}_{\mathbf{A},x} &= Z_0 \cdot R_1^{adj} \cdot \prod_{j=1}^{\ell} \widetilde{A}_{j,x[\mathsf{inp}(j)]} \cdot \widetilde{A}_{\ell+1} \\
&\quad + \sum_{i=1}^{\ell} \left(\widetilde{A}_0 \cdot \prod_{j=1}^{i-1} \widetilde{A}_{j,x[\mathsf{inp}(j)]} \cdot \left(\alpha_{i,x[\mathsf{inp}(i)]} R_i \cdot Z_{i,x[\mathsf{inp}(i)]} \cdot R_{i+1}^{adj} \right) \cdot \prod_{j=i+1}^{\ell} \widetilde{A}_{j,x[\mathsf{inp}(j)]} \cdot \widetilde{A}_{\ell+1} \right) \\
&\quad + \widetilde{A}_0 \cdot \prod_{j=1}^{\ell} \widetilde{A}_{j,x[\mathsf{inp}(j)]} \cdot R_{\ell+1} \cdot Z_{\ell+1} \\
&= \rho\widehat{\alpha}_x \left(Z_0 \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1} \right) \\
&\quad + \rho\widehat{\alpha}_x \sum_{i=1}^{\ell} \left(A_0 \cdot \prod_{j=1}^{i-1} A_{j,x[\mathsf{inp}(j)]} \cdot Z_{i,x[\mathsf{inp}(i)]} \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1} \right) \\
&\quad + \rho\widehat{\alpha}_x \left(A_0 \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot Z_{\ell+1} \right) \qquad (6.6)
\end{aligned}
$$

Now, define:

$$\mathbf{z}_0 \stackrel{\text{def}}{=} \mathbf{vec}\left(Z_0\right) \in \{0,1\}^{d \times 1} \ , \ \mathbf{z}_{\ell+1} \stackrel{\text{def}}{=} \mathbf{vec}\left(Z_{\ell+1}\right) \in \{0,1\}^{d \times 1},$$

and

$$\mathbf{z}_{i,b} \stackrel{\text{def}}{=} \mathbf{vec}\left(Z_{i,b}\right) \in \{0,1\}^{d^2 \times 1}$$

Now, we set

$$\boldsymbol{z}_i = \left[\mathbf{z}_{i,0}^T \mid \mathbf{z}_{i,1}^T\right].$$

And finally set, as

$$\boldsymbol{z} \stackrel{\text{def}}{=} [\boldsymbol{z}_0 \mid \boldsymbol{z}_1 | \ldots | \boldsymbol{z}_\ell \mid \boldsymbol{z}_{\ell+1}] \in \{0,1\}^{1 \times (2\ell+2)d^2}$$

where $\boldsymbol{z}$ consists of all random secret variables involved in $\mathcal{O}(\mathbf{A})$. Next using the property

of tensor products (Property 6.2.3) we can rewrite Eq. 6.6 as:

$$
\begin{aligned}
P^{(1)}_{\mathbf{A},x} = \mathbf{vec}\left(P^{(1)}_{\mathbf{A},x}\right) &= \rho\widehat{\alpha}_x \mathbf{vec}\left(Z_0 \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right) \\
&\quad + \rho\widehat{\alpha}_x \sum_{i=1}^{\ell} \mathbf{vec}\left(A_0 \cdot \prod_{j=1}^{i-1} A_{j,x[\mathsf{inp}(j)]} \cdot Z_{i,x[\mathsf{inp}(i)]} \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right) \\
&\quad + \rho\widehat{\alpha}_x \mathbf{vec}\left(A_0 \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot Z_{\ell+1}\right) \\
&= \rho\widehat{\alpha}_x \left(\prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right)^T \cdot \mathbf{z}_0 \\
&\quad + \rho\widehat{\alpha}_x \sum_{i=1}^{\ell} \left(A_0 \cdot \prod_{j=1}^{i-1} A_{j,x[\mathsf{inp}(j)]}\right) \otimes \left(\prod_{j=i+1}^{\ell} A_{j,x[\mathsf{inp}(j)]} \cdot A_{\ell+1}\right)^T \cdot \mathbf{z}_{i,x[\mathsf{inp}(i)]} \\
&\quad + \rho\widehat{\alpha}_x \left(A_0 \cdot \prod_{j=1}^{\ell} A_{j,x[\mathsf{inp}(j)]}\right) \cdot \mathbf{z}_{\ell+1} \qquad (6.7) \\
&= \rho\widehat{\alpha}_x \left(\phi^{(0)}_{\mathbf{A},x} \cdot \mathbf{z}_0 + \sum_{i=1}^{\ell} \phi^{(i)}_{\mathbf{A},x} \cdot \mathbf{z}_{i,x[\mathsf{inp}(i)]} + \phi^{(\ell+1)}_{\mathbf{A},x} \cdot \mathbf{z}_{\ell+1}\right) \\
&= \rho\widehat{\alpha}_x \left(\widetilde{\phi}^{(0)}_{\mathbf{A},x} \cdot \mathbf{z}_0 + \sum_{i=1}^{\ell} \widetilde{\phi}^{(i)}_{\mathbf{A},x} \cdot \boldsymbol{z}_i + \widetilde{\phi}^{(\ell+1)}_{\mathbf{A},x} \cdot \mathbf{z}_{\ell+1}\right) \\
&= \rho\widehat{\alpha}_x (\boldsymbol{\phi}_{\mathbf{A},x} \cdot \boldsymbol{z}^T). \qquad (6.8)
\end{aligned}
$$

6.6 Extending the Abstract Attack to GGH13 Multilinear Maps

In this section, we show that an attack in abstract model described in Section 6.3.1 can be translated to an attack in the GGH13 setting. This part of the attack is heuristic and analogous to some of the previous attacks on GGH13 such as in [GGH13a, MSZ16a, CHL+15].

6.6.1 The GGH13 Scheme: Background

In the GGH13 scheme [GGH13a], the plaintext space is a quotient ring $R/\boldsymbol{g}R$, where R is the ring of integers in a cyclotomic number field and $\boldsymbol{g} \in R$ is a "small prime element." The space of encodings is $R_q = R/qR$ for a large (exponential in the security parameter λ) modulus q. We write $[\cdot]_q$ to denote operations are done in $\mathbb{Z}_q$.

A uniformly random secret $\boldsymbol{z}_1 \ldots \boldsymbol{z}_k \in R_q$ is chosen, and used to encode plaintext values as follows: A plaintext element $\boldsymbol{a} \in R/\boldsymbol{g}R$ is encoded at the level-1 as $\boldsymbol{u} = [\boldsymbol{c}/\boldsymbol{z}]_q$, where the numerator $\boldsymbol{c}$ is a "small" element in the coset of $\boldsymbol{a}$; i.e. $\boldsymbol{c} = \boldsymbol{a} + \boldsymbol{g}\boldsymbol{r}$ for a small random term $r \in R$, chosen from an appropriate distribution. We describe the GGH13 and our attack assuming use of "symmetric" multilinear maps just for simplicity of notation. Note that in our attacks we compute on provided multilinear maps encodings in a prescribed manner. Furthermore, the $\boldsymbol{z}$ always vanish in our attacks. Therefore, the attack immediately generalize to the "asymmetric GGH" setting, with many distinct choices of $\boldsymbol{z}$'s and we continue to use the "symmetric" notation for simplicity.

Addition and subtraction of encodings at the same level is performed by addition in R_q, and outputs an encoding of the sum of the encoded plaintext values at the same level. Multiplication of encodings at levels t_1 and t_2 yields a new level-$t_1 + t_2$ encoding of the product of the corresponding plaintexts.

The level-k encodings of the *zero plaintext,* $0 \in R/\boldsymbol{g}R$, have the form $\boldsymbol{u} = [\boldsymbol{g}\boldsymbol{r}/\boldsymbol{z}^k]_q$. Public parameter of the GGH13 multilinear maps include a public zero-testing parameter $\mathsf{p_{zt}} = [\boldsymbol{h}\boldsymbol{z}^k/\boldsymbol{g}]_q$, for a "somewhat small' element $\boldsymbol{h} \in R$, which is roughly of size $\sqrt{q}$. The zero-test operation involves multiplying $\mathsf{p_{zt}}$ by a level-k encoding $\boldsymbol{u}$, and checking if the result $[\mathsf{p_{zt}} \cdot \boldsymbol{u}]_q$ is much smaller than the modulus q. Note that if $\boldsymbol{u}$ is indeed an encoding of zero then we have that $[\mathsf{p_{zt}} \cdot \boldsymbol{u}]_q = [\boldsymbol{h}\boldsymbol{r}]_q$. If $\boldsymbol{h}, \boldsymbol{r}$, are much smaller than q then we have that this computed value will also be much smaller than q. On the other hand if $\boldsymbol{u} = [\boldsymbol{c}/\boldsymbol{z}^k]_q$ is not an encoding of zero, then we have that $[\mathsf{p_{zt}} \cdot \boldsymbol{u}]_q = [\boldsymbol{c}/\boldsymbol{g}]_q$ will be large.

6.6.2 Translating the Abstract Attack to GGH13

In this section, we assume that we are provided programs $\mathbf{A}_0$ and $\mathbf{A}_1$, set of inputs X and a vector $\boldsymbol{v}_{\mathsf{ann}}$ such that $\boldsymbol{v}_{\mathsf{ann}} \cdot \Psi_{\mathbf{A}_0,X} \cdot (\boldsymbol{z} \otimes \boldsymbol{z})^T \equiv 0$ and $\boldsymbol{v}_{\mathsf{ann}} \cdot \Psi_{\mathbf{A}_1,X} \cdot (\boldsymbol{z} \otimes \boldsymbol{z})^T \not\equiv 0$. Recall that $\boldsymbol{v}_{\mathsf{ann}}$ is sufficient to complete an attack in the abstract model. Given the above we describe an attack strategy of distinguishing between obfuscations of $\mathbf{A}_0$ and $\mathbf{A}_1$ generated using GGH13 multilinear maps. We do this in two steps. In the first step, we will use the abstract attack to compute an element $\boldsymbol{u}$ whose distribution depends on whether $\mathbf{A}_0$ was used or $\mathbf{A}_1$ was used. We explain this step in this subsection below. The second step that involves efficiently testing the distribution from which $\boldsymbol{u}$ is sampled is described in the next subsection.

Our attack is provided an obfuscation of either $\mathbf{A} \in \{\mathbf{A}_0, \mathbf{A}_1\}$ and it proceeds as follows. It mimics the abstract attack for the pre-zeroing computation queries by computing the values using the provided encodings. Since only "valid" queries were made in the abstract model, therefore the corresponding computation can be done locally. Specifically, for each $x \in X$, we obtain

$$\left[\frac{P_{\mathbf{A},x}}{\boldsymbol{z}^k}\right]_q = \left[\frac{P^{(0)}_{\mathbf{A},x} + \boldsymbol{g}P^{(1)}_{\mathbf{A},x} + \boldsymbol{g}^2(\ldots)}{\boldsymbol{z}^k}\right].$$

Since, $P^{(0)}_{\mathbf{A},x} = 0$, zero-testing on this value yields a value that is unreduced $\mod q$. In particular, zero-test reveals the value:

$$W_{\mathbf{A},x} = \boldsymbol{h}\left(P^{(1)}_{\mathbf{A},x} + \boldsymbol{g}(\ldots)\right).$$

Using these values, we set $\boldsymbol{u} = \boldsymbol{v}_{\mathsf{ann}} \cdot \widetilde{W}_{\mathbf{A},X}$ where

$$\widetilde{W}_{\mathbf{A},X} = \begin{bmatrix} W_{\mathbf{A},x_1}W_{\mathbf{A},\overline{x}_1} \\ \vdots \\ W_{\mathbf{A},x_m}W_{\mathbf{A},\overline{x}_m} \end{bmatrix}$$

Note that if $\mathbf{A} = \mathbf{A}_0$ then we have that $\boldsymbol{u} \in \langle \rho^2 \hat{\alpha} \boldsymbol{h}^2 \boldsymbol{g} \rangle$, where $\rho \stackrel{\text{def}}{=} \prod_i \det(R_i)$ and $\widehat{\alpha} \stackrel{\text{def}}{=} \widehat{\alpha}_x \widehat{\alpha}_{\overline{x}}$ both of which are fixed terms. On the other hand, if $\mathbf{A} = \mathbf{A}_1$ then we have that $\boldsymbol{u} \notin \langle \rho^2 \hat{\alpha} \boldsymbol{h}^2 \boldsymbol{g} \rangle$ with overwhelming probability by Schwartz-Zippel Lemma.

6.6.3 Completing the Attack for Large Enough Circuits

In order to complete the attack we need to check if $\boldsymbol{u}$ obtained in the previous step is in the ideal $\langle \rho^2 \hat{\alpha} \boldsymbol{h}^2 \boldsymbol{g} \rangle$ or not. Below we describe a method to compute several (heuristically) linearly independent elements is in the ideal $\mathcal{J} = \langle \rho^4 \hat{\alpha}^2 \boldsymbol{h}^4 \boldsymbol{g} \rangle$. Note that if $\boldsymbol{u} \in \langle \rho^2 \hat{\alpha} \boldsymbol{h}^2 \boldsymbol{g} \rangle$ then $\boldsymbol{u}^2 \in \mathcal{J}$ as well. However, since $\boldsymbol{g}$ is prime, if $\boldsymbol{u} \notin \langle \rho^2 \hat{\alpha} \boldsymbol{h}^2 \boldsymbol{g} \rangle$ then $\boldsymbol{u}^2$ will not be in $\mathcal{J}$.

Let $X_1, X_2, \ldots$ be disjoint sets of inputs such that for each i we have that $X_i \cap (X \cup \overline{X}) = \emptyset$, $|X_i| = (2d + 2^\ell d^2)^2$ and that $\forall x \in X_i$ we have that $\mathbf{A}(x) = \mathbf{A}(\overline{x}) = 0$.[21] Since, the number of inputs is 2^n for a large enough circuit we can define any polynomial number of such sets X_i.

Note that for each i, since the number of equations is larger than the number of variables, therefore $\exists \boldsymbol{a}_i, \boldsymbol{b}_i$ such that $\boldsymbol{a}_i \cdot \Psi_{\mathbf{A}_0, X_i} = 0$ and $\boldsymbol{b}_i \cdot \Psi_{\mathbf{A}_1, X_i} = 0$. Therefore, for $\mathbf{A} \in \{\mathbf{A}_0, \mathbf{A}_1\}$ we can conclude that $(\boldsymbol{a}_i \cdot \widetilde{W}_{\mathbf{A}, X_i})(\boldsymbol{b}_i \cdot \widetilde{W}_{\mathbf{A}, X_i}) \in \mathcal{J} = \langle \rho^4 \hat{\alpha}^2 \boldsymbol{h}^4 \boldsymbol{g} \rangle$ where

$$\widetilde{W}_{\mathbf{A}, X_i} = \begin{bmatrix} W_{\mathbf{A}, x_{i,1}} W_{\mathbf{A}, \overline{x}_{i,1}} \\ \vdots \\ W_{\mathbf{A}, x_{i,m}} W_{\mathbf{A}, \overline{x}_{i,m}} \end{bmatrix}$$

and $X_i = \{x_{i,1}, x_{i,2}, \ldots, x_{i,m}\}$. Repeating this process for each choice of i we obtain several elements in $\mathcal{J}$. Note that these values are linearly independent except that some of these values (possibly all of them) might actually be in $\mathcal{J}' = \langle \rho^4 \hat{\alpha}^2 \boldsymbol{h}^4 \boldsymbol{g}^2 \rangle$. However, this doesn't affect our attack because $\boldsymbol{u}^2$ is in $\mathcal{J}'$ as well.

[21] Since the programs are functionally equivalent we have this condition.

6.7 Example of Partially Inequivalent Circuits

In this section, we show examples of pairs of NC^1 circuits such that the corresponding Barrington-implemented[22] branching programs are partially inequivalent and therefore are subject to the abstract annihilation attacks shown in Section 6.5. Note that here we extend the notion of partial inequivalence from branching programs to circuits in a natural way. Unless otherwise mentioned, partial inequivaelnce of circuits specifically imply that the corresponding branching programs generated via applying Barrngton's Theorem are partially inequivalent.

6.7.1 Simple Pairs of Circuits that are Partially Inequivalent

Consider the following pair of circuits (C_0, C_1) each of which implements a boolean function $\{0,1\}^4 \to \{0,1\}$:

$$C_0(x)] \stackrel{\text{def}}{=} (x[1] \wedge 1) \bigwedge (x[2] \wedge 0) \bigwedge (x[3] \wedge 1) \bigwedge (x[4] \wedge 0),$$

$$C_1(x) \stackrel{\text{def}}{=} (x[1] \wedge 0) \bigwedge (x[2] \wedge 0) \bigwedge (x[3] \wedge 0) \bigwedge (x[4] \wedge 0).$$

Define the set $X \stackrel{\text{def}}{=} \{0,1\}^4$. Now, we provide an implementation (see Appendix 6.8 for more details on the implementation) in Sage[S+16] that evaluates the column spaces of matrices produced via applying a Barrington-implementation to the above circuits. The outcome from the implementation led us to conclude the following claim:

Claim 6.7.1. *Let $\mathbf{A}_{C_0}, \mathbf{A}_{C_1}$ be the Barrington-Implementation of the circuits C_0, C_1 respectively, then we have that:* $\mathsf{colsp}\left(\Psi_{\mathbf{A}_{C_0},X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_{C_1},X}\right)$.

Remark 6.7.2. *We emphasize that we use branching programs generated with a particular Barrington-implementation that makes a set of specific choices. We remark that Barrington's*

[22] Recall that by Barrington-implementation of a circuit we mean the single-input branching program produced as a result of Barrington Theorem on the circuit. Also we implicitly assume that the branching programs are input-oblivious.

construction can be implemented in many different ways. However, since in this section we aim to find one concrete example for which the condition of our abstract attack satisfies, we restrict ourselves to this specific program. We refer the reader to Appendix 6.8 for the details of our implementation. Throughout this section we refer to this particular Barrington-implementation.

The circuits presented above are of constant size. Looking ahead, though, they are partially inequivalent and hence (by Theorem 6.5.1) are susceptible to the abstract attack that does not translate to a real-world attack in GGH13 setting immediately. For that we need to consider larger (albeit NC^1) circuits which we construct next based on the above circuits.

6.7.2 Larger Pairs of Circuits that are Partially Inequivalent

We construct pairs of NC^1 circuits (in fact, exponentially many of them) that build on the constant-size circuits described in Sec. 6.7.1.

Consider *any* pair of functionally equivalent NC^1 circuits (D_0, D_1) and an input $x^\star \in \{0,1\}^n$ such that $D_0(x^\star) = D_1(x^\star) = D_0(\overline{x^\star}) = D_1(\overline{x^\star}) = 0$. Now define the circuits E_0, E_1 each of which computes a boolean function $\{0,1\}^{n+4} \to \{0,1\}$ as follows:

$$E_0(y) \stackrel{\text{def}}{=} \neg C_0(x) \wedge D_0(x'),$$

$$E_1(y) \stackrel{\text{def}}{=} \neg C_1(x) \wedge D_1(x')$$

($\neg C$ is the circuit C with output negated) such that for each $y \in \{0,1\}^{n+4}$ we have $y = x \circ x'$ ($\circ$ denotes concatenation) where $x \in \{0,1\}^4$ and $x' \in \{0,1\}^n$. Define the input-sequence $Y \stackrel{\text{def}}{=} \{x \circ x^\star \mid x \in \{0,1\}^4\}$ (consisting of 16 inputs). Then we show the following statement.

Lemma 6.7.3. *Let $\mathbf{A}_{E_0}, \mathbf{A}_{E_1}$ be the Barrington-implementations of E_0, E_1 respectively,*

then we have that: $\mathsf{colsp}\left(\Psi_{\mathbf{A}_{E_0},Y}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_{E_1},Y}\right)$.

Proof. As a first step, similar to Claim 6.7.1 we also verify the following claim via our Sage-implementations (c.f. Appendix 6.8 for more details on the implementation).

Claim 6.7.4. *Let* $\mathbf{A}_{\neg C_0 \wedge 0}, \mathbf{A}_{\neg C_1 \wedge 0}$ *be the Barrington-implementations of the circuits* $\neg C_0 \wedge 0, \neg C_1 \wedge 0$ *respectively, then we have that:* $\mathsf{colsp}\left(\Psi_{\mathbf{A}_{\neg C_0 \wedge 0},X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_{\neg C_1 \wedge 0},X}\right)$.

Now, recall (Def. 6.2.4) that any branching program $\mathbf{A}$ has the following representation:

$$\mathbf{A} = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1}).$$

Let us call the "core" of $\mathbf{A}$ as: $\mathbf{A}' \stackrel{\text{def}}{=} \{A_{1,b}, \ldots, A_{\ell,b}\}_{b\in\{0,1\}}$.[23] For any such $\mathbf{A}'$ we define the inverse as $\mathbf{A}'^{-1} \stackrel{\text{def}}{=} \{A^{-1}_{\ell,b}, A^{-1}_{\ell-1,b}, \ldots, A^{-1}_{1,b}\}_{b\in\{0,1\}}$. Furthermore, for any permutation matrix $\boldsymbol{\rho} \in S_5$ (recall that Barrington-implementation works with matrices in the symmetric group S_5). we define an operation on $\mathbf{A}'$:

$$\boldsymbol{\rho}(\mathbf{A}')\boldsymbol{\rho}^{-1} = \left\{(\boldsymbol{\rho} \cdot A_{1,b}), \{A_{i,b}\}_{i\in[\ell]}, (A_{\ell,b} \cdot \boldsymbol{\rho}^{-1})\right\}_{b\in\{0,1\}}$$

Now recall that using the construction from Barrington's Theorem with the above notations we can write for any choice of $E \in (E_0, E_1)$ (where $E = C \wedge D$)

$$\underbrace{\mathbf{A}'_{\neg C\wedge D}}_{\Phi_{\mathbf{A}_{\neg C\wedge D},Y}} = \underbrace{(\boldsymbol{\rho}(\mathbf{A}'_{\neg C})\boldsymbol{\rho}^{-1})}_{M_0} \circ \underbrace{(\boldsymbol{\varrho}(\mathbf{A}'_{D})\boldsymbol{\varrho}^{-1})}_{M_1} \circ \underbrace{(\boldsymbol{\rho}(\mathbf{A}'_{\neg C})^{-1}\boldsymbol{\rho}^{-1})}_{M_2} \circ \underbrace{(\boldsymbol{\varrho}(\mathbf{A}'_{D})^{-1}\boldsymbol{\varrho}^{-1})}_{M_3} \quad (6.9)$$

where $\boldsymbol{\rho}, \boldsymbol{\varrho} \in S_5$ are specific to the Barrington-implementation (see Appendix 6.8 for the exact values we used) and fixed for a particular implementation. Now we can split the linear partial matrix of $\mathbf{A}_{\neg C\wedge D}$ into four parts:

[23] The order of the matrices are taken into account here and the evaluation of branching program depends on that. So, essentially we abuse notations of sets to denote an ordered tuple here. Unless otherwise mentions we assume that the set $\{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$ is ordered as $\{A_{1,b}, \ldots, A_{\ell,b}\}_{b\in\{0,1\}}$

$$\Phi_{\mathbf{A}_{\neg C\wedge D},Y} \stackrel{\text{def}}{=} [M_0 \mid M_1 \mid M_2 \mid M_3]$$

where each M_i is corresponding to a part of the core $\mathbf{A}'_{C\wedge D}$ as shown in Eq. 6.9. However, since we have $D_{x^\star} = 0$ for all $y = x \circ x^\star$ in Y and for any two inputs $y_1, y_2 \in Y$ the sub-input to the circuit D (corresponding to parts M_1 and M_3) is same (equal to $x^\star$) clearly when i is in the range of M_1 or M_3 we get that:

$$\phi^{(i)}_{\mathbf{A}_{\neg C\wedge D},y_1} = \phi^{(i)}_{\mathbf{A}_{\neg C\wedge D},y_2}.$$

which implies that $M_1, M_3 \in \mathcal{T}$ where $\mathcal{T}$ is a family of all "trivial matrices" with columns which are either all 0 or all 1 as follows:

$$\begin{bmatrix} \cdots & 1 \cdots & 0 & \cdots \\ \cdots & 1 \cdots & 0 & \cdots \\ \vdots & \ddots & \vdots & \vdots \\ \cdots & 1 \cdots & 0 & \cdots \end{bmatrix}$$

Again, using Barrington's Theorem for the circuit $\neg C \wedge 0$ we have that:

$$\underbrace{\mathbf{A}'_{\neg C\wedge 0}}_{\Phi_{\mathbf{A}_{\neg C\wedge 0},X}} = \underbrace{(\boldsymbol{\rho}(\mathbf{A}'_{\neg C})\boldsymbol{\rho}^{-1})}_{N_0} \circ \underbrace{(\boldsymbol{\varrho}(ID)\boldsymbol{\varrho}^{-1})}_{N_1} \circ \underbrace{(\boldsymbol{\rho}(\mathbf{A}'_{\neg C})^{-1}\boldsymbol{\rho}^{-1})}_{N_2} \circ \underbrace{(\boldsymbol{\varrho}(ID)^{-1}\boldsymbol{\varrho}^{-1})}_{N_3} \tag{6.10}$$

for $X = \{0,1\}^4$ where we again have $N_1, N_3 \in \mathcal{T}$ that follows using similar arguments.

Moreover, using again the fact that for any $y = x \circ x^\star$ in Y we have that $D(x^\star) = 0$, the core of D would always evaluates to ID on any choice of $y \in Y$. Hence when i lies in the range of M_0, M_2 the partial vectors $\phi^{(i)}_{\mathbf{A}_{\neg C\wedge D},Y}$ are independent of the part of the program corresponding to the ranges of M_1, M_3. Therefore, we can conclude that the i-th partial vectors corresponding to ranges M_0, M_2 would be equal to the i-th partial vectors

corresponding to ranges N_0, N_2. Hence,

$$M_0 = N_0 \text{ and } M_2 = N_2$$

On the other hand, via exactly the same analysis for the inputs $\overline{Y} = \{\overline{y} \mid \overline{x} \circ \overline{x^\star}\}_{x\in\{0,1\}^4}$ we have that:

$$\Phi_{\mathbf{A}_{\neg C\wedge D},\overline{Y}} = [\overline{M}_0 \mid \overline{M}_1 \mid \overline{M}_2 \mid \overline{M}_3]$$
$$\Phi_{\mathbf{A}_{\neg C\wedge 0},\overline{X}} = [\overline{N}_0 \mid \overline{N}_1 \mid \overline{N}_2 \mid \overline{N}_3]$$

where $\overline{M_1}, \overline{M}_3, \overline{N}_1, \overline{N}_3 \in \mathcal{T}$ and

$$\overline{M}_0 = \overline{N}_0 \text{ and } \overline{M}_2 = \overline{N}_2.$$

Hence we conclude:

$$\mathsf{colsp}\left(\Psi_{\mathbf{A}_{\neg C\wedge D},Y}\right) = \mathsf{colsp}\left(\left[\mathbf{\Pi}_{i,j}[M_i \boxtimes \overline{M}_j]\right] + \left[\mathbf{\Pi}_{i,j}[\overline{M}_i \boxtimes M_j]\right]\right) \tag{6.11}$$
$$= \mathsf{colsp}\left(\left[\tau \mid \mathbf{\Pi}_{i,j\in\{0,2\}}[M_i \boxtimes \overline{M}_j]\right] + \left[\tau' \mid \mathbf{\Pi}_{i,j\in\{0,2\}}[\overline{M}_i \boxtimes M_j]\right]\right) \tag{6.12}$$

for some $\tau, \tau' \in \mathcal{T}$. Note that, in the above equations the first step follows from Fact 6.2.5. In the second step we first observe that for $\tau, \tau' \in \mathcal{T}$ and any matrix $M_i, \overline{M}_i$ we have that $\mathsf{colsp}\left(\tau \boxtimes M_i\right) = \mathsf{colsp}\left(M_i\right)$ (resp. $\mathsf{colsp}\left(\tau \boxtimes \overline{M}_i\right) = \mathsf{colsp}\left(\overline{M}_i\right)$). Also applying Lemma 6.4.3 it is straightforward to verify that each of the matrices $\{M_i, \overline{M}_i\}_{i\in[3]\cup\{0\}}$ has the same number of 1's in each row. Hence, then we use Lemma 6.2.13 to obtain the final expression. Similarly we get:

$$\mathsf{colsp}\left(\Psi_{\mathbf{A}_{\neg C\wedge 0},X}\right) = \mathsf{colsp}\left(\left[\mathbf{\Pi}_{i,j}[N_i \boxtimes \overline{N}_j]\right] + \left[\mathbf{\Pi}_{i,j}[\overline{N}_i \boxtimes N_j]\right]\right) \tag{6.13}$$

$$= \mathsf{colsp}\left(\left[\sigma \mid \mathbf{\Pi}_{i,j\in\{0,2\}}[N_i \boxtimes \overline{N}_j]\right] + \left[\sigma' \mid \mathbf{\Pi}_{i,j\in\{0,2\}}[\overline{N}_i \boxtimes N_j]\right]\right) \tag{6.14}$$

for some $\sigma, \sigma' \in \mathcal{T}$.

Using the facts, $M_k = N_k$ and $\overline{M}_k = \overline{N}_k$ for $k \in \{0, 2\}$ from Eq. 6.12 and Eq. 6.14 we get:

$$\mathsf{colsp}\left(\Psi_{\mathbf{A}_{\neg C\wedge D},Y}\right) = \mathsf{colsp}\left(\Psi_{\mathbf{A}_{\neg C\wedge 0},X}\right).$$

Now combining this equation with Claim 6.7.4 the lemma follows.

□

6.7.3 Universal Circuit Leading to Partially Inequivalent Branching Programs

In this section we present constructions of (NC^1) universal circuits that, when compiled with two arbitrary distinct (NC^1) but functionally equivalent circuits as inputs, then the obfuscations of the Barrington-implementation of the compiled circuits are distinguishable by the abstract attack.

For any circuit C, its description is denoted by a bit-string, abusing notation slightly we use the same symbol C to represent the description of C.

Definition 6.7.5 (Universal Circuits)**.** *An universal circuit U is a boolean circuit that computes a function $\{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}$ which takes two inputs, a λ-bit circuit-description of some boolean circuit $C : \{0,1\}^n \to \{0,1\}$ and a n-bit input x to output $C(x)$. We denote $U(C,x) \stackrel{\text{def}}{=} C(x)$. We also denote the compiled universal circuit with the description of C hard-coded into it by $U[C]$.*

Theorem 6.7.6. *There exists a family of* NC^1 *universal circuits* $\mathcal{U} = \{U_1, U_2, \ldots, U_v\}$ *of size* $v = O(poly(\lambda))$ *such that: given two arbitrary functionally equivalent* NC^1 *circuits* G_0, G_1 *that computes arbitrary boolean function* $\{0,1\}^n \to \{0,1\}$ *satisfying (i)* $|G_0| = |G_1| = v$ *and (ii) there exists an input* $x^\star$ *such that* $G_0(x^\star) = G_1(x^\star) = G_0(\overline{x^\star}) = G_1(\overline{x^\star}) = 0$*; then for at least one* $i \in [v]$ *the Barrington-implementations of the circuits* $U_i[G_0]$ *and* $U_i[G_1]$ *are partially inequivalent.*

Proof. Our construction of the family $\mathcal{U}$ is similar to the construction of circuits E_0, E_1 constructed in Section 6.7.1

Construction of the family $\mathcal{U}$. Given a universal circuit U' we construct a family of NC^1 universal circuits $\mathcal{U} = \{U_1, \ldots, U_v\}$ where each U_i is described as follows for any circuit $G : \{0,1\}^n \to \{0,1\}$ we define $U_i[G]$

$$U_i[G](y, x) = \neg C(y) \wedge U_i'(G, x) \text{ where } C = (y[1] \wedge G[i]) \bigwedge (y[2] \wedge 0) \bigwedge (y[3] \wedge G[i]) \bigwedge (y[4] \wedge 0),$$

as the circuit from $\{0,1\}^{n+4} \to \{0,1\}$. Since the given circuits must have different descriptions, they differ by at least one bit location, say i^{th} location. Clearly, assuming that $G_0[i] = 1$ and $G_1[i] = 0$ the circuit $U_i[G_b]$ is the same as the circuit E_b as described in Sec. 6.7.2. Hence applying Lemma 6.7.3 we conclude that, if $G_0[i] = 1$ and $G_1[i] = 0$ then,

$$\mathsf{colsp}\left(\Psi_{\mathbf{A}_{U_i[G_0]}, X}\right) \neq \mathsf{colsp}\left(\Psi_{\mathbf{A}_{U_i[G_1]}, X}\right),$$

where $X = \{x \circ x^\star \mid x \in \{0,1\}^4\}$.

□

6.8 Some details on our implementation

In this section we provide details on our Barrington-implementation and discuss some optimizations in the Sage-code.

Overview of Barrington's Programs [Bar89]. Barrington's construction works over permutations in the symmetric group S_5. We assume that permutations are represented as matrices for all practical purpose. A Barrington-implementation specifies permutations $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}, \boldsymbol{\rho}, \boldsymbol{\varrho} \in S_5$ such that the following holds:

- $\boldsymbol{\alpha}, \boldsymbol{\beta}$ are 5-cycles.
- $\boldsymbol{\gamma} = \boldsymbol{\alpha}\boldsymbol{\beta}\boldsymbol{\alpha}^{-1}\boldsymbol{\beta}^{-1}$ and one can verify that $\boldsymbol{\gamma}$ is also a cycle.
- $\boldsymbol{\rho} \cdot \boldsymbol{\gamma} \cdot \boldsymbol{\rho}^{-1} = \boldsymbol{\alpha}$.
- $\boldsymbol{\varrho} \cdot \boldsymbol{\gamma} \cdot \boldsymbol{\varrho}^{-1} = \boldsymbol{\beta}$.

We define some syntaxes for branching programs. Some of them are redefinitions from Sec. 6.7.2 (provided in the proof of Lemma 6.7.3 as it uses some details on Barrington-implementations).

Core of a Branching Program. Recall (Def. 6.2.4) that any branching program $\mathbf{A}$ has the following representation:

$$\mathbf{A} = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1}).$$

Let us call the "core" of $\mathbf{A}$ as: $\mathbf{A}' \stackrel{\text{def}}{=} \{A_{1,b}, \ldots, A_{\ell,b}\}_{b\in\{0,1\}}$.[24] For any such $\mathbf{A}'$ we define the inverse as $\mathbf{A}'^{-1} \stackrel{\text{def}}{=} \{A^{-1}_{\ell,b}, A^{-1}_{\ell-1,b}, \ldots, A^{-1}_{1,b}\}_{b\in\{0,1\}}$. Furthermore, for any permutation matrix $\boldsymbol{\rho} \in S_5$ we define an operation on $\mathbf{A}'$:

$$\boldsymbol{\rho}(\mathbf{A}')\boldsymbol{\rho}^{-1} \stackrel{\text{def}}{=} \left\{(\boldsymbol{\rho} \cdot A_{1,b}), \{A_{i,b}\}_{i\in[\ell]}, (A_{\ell,b} \cdot \boldsymbol{\rho}^{-1})\right\}_{b\in\{0,1\}}$$

$\boldsymbol{\gamma}$-computation. Any branching program $\mathbf{A}_C = (\mathsf{inp}, A_0, \{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}, A_{\ell+1})$ is said to

[24] The order of the matrices are taken into account here and the evaluation of branching program depends on that. So, essentially we abuse notations of sets to denote an ordered tuple here. Unless otherwise mentioned we assume that the set $\{A_{i,b}\}_{i\in[\ell],b\in\{0,1\}}$ is ordered as $\{A_{1,b}, \ldots, A_{\ell,b}\}_{b\in\{0,1\}}$

be $\boldsymbol{\gamma}$-computes a boolean circuit C if the following holds:

$$\prod_{i=1}^{\ell} A_{i,x[\mathsf{inp}(i)]} = \begin{cases} \boldsymbol{\gamma} & when\ \ C(x)=1 \\ ID^{5\times 5} & when\ \ C(x)=0 \end{cases}$$

If $\mathbf{A}_{C_0}, \mathbf{A}_{C_1}$ $\boldsymbol{\gamma}$-computes C_0, C_1 then one can construct $\mathbf{A}_{C_0 \wedge C_1}$ that $\boldsymbol{\gamma}$-computes $C = C_0 \wedge C_1$ as follows:

$$\mathbf{A}'_{C_0 \wedge C_1} = (\boldsymbol{\rho}(\mathbf{A}'_{C_0})\boldsymbol{\rho}^{-1}) \circ (\boldsymbol{\varrho}(\mathbf{A}'_{C_1})\boldsymbol{\varrho}^{-1}) \circ (\boldsymbol{\rho}(\mathbf{A}'_{C_0})^{-1}\boldsymbol{\rho}^{-1}) \circ (\boldsymbol{\varrho}(\mathbf{A}'_{C_1})^{-1}\boldsymbol{\varrho}^{-1})$$

and with the same bookends.[25]

Let us also define the operation $(\mathbf{A}') \cdot \boldsymbol{\gamma}$ as $(\mathbf{A}') \cdot \boldsymbol{\gamma} \stackrel{\text{def}}{=} \{A_{1,b}, \ldots, A_{\ell_b} \cdot \boldsymbol{\gamma}\}_{b\in\{0,1\}}$ that has the final pairs right-multiplied with $\boldsymbol{\gamma}$. Then one can construct another branching program $\mathbf{A}_{\neg C}$ that $\boldsymbol{\gamma}$-computes the circuit $\neg C$ as follows:

$$\mathbf{A}'_{\neg C} = (\mathbf{A}'^{-1}_{C}) \cdot \boldsymbol{\gamma}$$

Since any boolean circuit can be converted to a circuit containing only NOT ($\neg$) and AND ($\wedge$) gates Barrington's theorem [Bar89] follows.

Our Barrington-implementation. In our implementations the branching programs are single-input and input-oblivious. We stress that the input-obliviousness comes automatically from our choice of circuits.

We choose the following permutations for our implementation:

[25] Our input function is a fixed one and designed as suggested by Barrington's Theorem. Namely to compute a program of size 4 on 2-bit input, $\boldsymbol{\alpha}\boldsymbol{\beta}\boldsymbol{\alpha}^{-1}\boldsymbol{\beta}^{-1}$ we use input function $\mathsf{inp} = (1\to 1, 1\to 2, 3\to 1, 4\to 2)$, that is the first position of the program reads the first bit, the fourth position the second and so on. Similarly for AND operation the input-functions can be extended with adjusted indexes. For more details we refer to Barrington's result [Bar89].

$$\boldsymbol{\alpha} \stackrel{\text{def}}{=} (1 \to 2 \to 3 \to 4 \to 5) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\boldsymbol{\beta} \stackrel{\text{def}}{=} (1 \to 3 \to 5 \to 4 \to 2) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{\gamma} \stackrel{\text{def}}{=} (1 \to 3 \to 2 \to 5 \to 4) = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{\rho} \stackrel{\text{def}}{=} (\alpha \to \gamma) = (1 \to 1, 2 \to 3, 3 \to 2, 4 \to 5, 5 \to 4) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{\varrho} \stackrel{\text{def}}{=} (\beta \to \gamma) = (1 \to 1, 3 \to 3, 5 \to 2, 4 \to 5, 2 \to 4) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

We fix the bookends to:

$$A_0 \stackrel{\text{def}}{=} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} \text{ and } A_{\ell+1} \stackrel{\text{def}}{=} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

Source Code and Experimental Set-Up. We provide an implementation in Sage [S+16]. The sage-executable file named implementations.sagews and a corresponding pdf file (implementations.pdf) of our source-code can be found at `https://people.eecs.berkeley.edu/~pratyay85/Implementations.zip`. The code can be run on the SageMath cloud server (`https://sagemath.cloud/`). The approximate performance for the 2 circuits on the SageMath cloud are given below:

Circuit	Approx time	Approx memory
C	3100 sec ($\sim$ 55 minutes)	4 GB
$\neg C \wedge 0$	33400 sec ($\sim$ 10 hours)	9 GB

Optimizations. Our source-code is not low-level optimized. However, to run the quadratic attack in practical time we required some algorithmic optimization in order to get the

program terminated in reasonable time. In particular, since the number of columns for the quadratic partial matrix, $\Psi_{\mathbf{A},X}$ becomes squared compared to number of columns in the linear matrices $\Phi_{\mathbf{A},X}, \Phi_{\mathbf{A},\overline{X}}$, even for the case of the simplest circuits ($\mathbf{A}_C$ or $\mathbf{A}_{\neg C}$) the estimated time to compute directly $\Psi_{\mathbf{A},X}$ as $(\Phi_{\mathbf{A},X} \boxtimes \Phi_{\mathbf{A},\overline{X}} + \Phi_{\mathbf{A},\overline{X}} \boxtimes \Phi_{\mathbf{A},X})$ becomes huge. Instead, we first remove the columns that are all-zero in both $\Phi_{\mathbf{A},X}, \Phi_{\mathbf{A},\overline{X}}$ since the corresponding random variables $\mathbf{z}_{i,b}$ appear in neither of the linear partial matrices. Then we observe that, even after performing that removal, there are many columns that are all-zero in exactly one of $\Phi_{\mathbf{A},X}, \Phi_{\mathbf{A},\overline{X}}$. Hence we first collect those that appear in both and then those appear in one of them. Let us call these three parts $M_X, M_{\overline{X}}$ and $M_{X,\overline{X}}$. Then we have:

$$\Phi^{\star}_{\mathbf{A},X} = [M_X \mid M_{X,\overline{X}}] \;\; \Phi^{\star}_{\mathbf{A},\overline{X}} = [M_{\overline{X}} \mid M_{X,\overline{X}}]$$

where $\Phi^{\star}_{\mathbf{A},X}$ (resp. $\Phi^{\star}_{\mathbf{A},\overline{X}}$) is the same as $\Phi_{\mathbf{A},X}$ (resp. $\Phi_{\mathbf{A},\overline{X}}$) but without some all 0 columns (those appear in none).

Then we compute

$$N = M_X \boxtimes \Phi_{\mathbf{A},\overline{X}} + M_{\overline{X}} \boxtimes \Phi_{\mathbf{A},X} + \Phi_{\mathbf{A},X} \boxtimes \Phi_{\mathbf{A},\overline{X}} + \Phi_{\mathbf{A},X} \boxtimes \Phi_{\mathbf{A},\overline{X}} \overset{\text{per}}{=} \Psi_{\mathbf{A},X}$$

by combining Fact 6.2.5 with the above observation. This reduced the number of row-wise tensor product by at least 2 (even after removing the all-zero columns) as we are not computing tensor products from both directions for the matrices containing columns that appear only once.

Bibliography

[AB15] Benny Applebaum and Zvika Brakerski. Obfuscating circuits via composite-order graded encoding. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 528–556. Springer, Heidelberg, March 2015.

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 553–572. Springer, Heidelberg, May 2010.

[ABD16] Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 153–178. Springer, Heidelberg, August 2016.

[ABG+14] Adi Akavia, Andrej Bogdanov, Siyao Guo, Akshay Kamath, and Alon Rosen. Candidate weak pseudorandom functions in ac0 (mod2). In Moni Naor, editor, *Innovations in Theoretical Computer Science, ITCS'14, Princeton, NJ, USA, January 12-14, 2014*, pages 251–260. ACM, 2014.

[ABSV15] Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario

Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

[ACLL15] Martin R. Albrecht, Catalin Cocis, Fabien Laguillaumie, and Adeline Langlois. Implementing candidate graded encoding schemes from ideal lattices. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part II*, volume 9453 of *LNCS*, pages 752–775. Springer, Heidelberg, November / December 2015.

[AFL15] Daniel Apon, Xiong Fan, and Feng-Hao Liu. Bi-deniable inner product encryption from LWE. *IACR Cryptology ePrint Archive*, 2015:993, 2015.

[AGIS14] Prabhanjan Vijendra Ananth, Divya Gupta, Yuval Ishai, and Amit Sahai. Optimizing obfuscation: Avoiding Barrington's theorem. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM CCS 14*, pages 646–658. ACM Press, November 2014.

[AHKM14] Daniel Apon, Yan Huang, Jonathan Katz, and Alex J. Malozemoff. Implementing cryptographic program obfuscation. Cryptology ePrint Archive, Report 2014/779, 2014. http://eprint.iacr.org/2014/779.

[AJN+16] Prabhanjan Ananth, Aayush Jain, Moni Naor, Amit Sahai, and Eylon Yogev. Universal constructions and robust combiners for indistinguishability obfuscation and witness encryption. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 491–520. Springer, Heidelberg, August 2016.

[Ajt99] Miklós Ajtai. Determinism versus non-determinism for linear time RAMs (extended abstract). In *31st ACM STOC*, pages 632–641. ACM Press, May 1999.

[Ajt04] Miklós Ajtai. Generating hard instances of lattice problems. *Quaderni di Matematica*, 13:1–32, 2004. Preliminary version in STOC 1996.

[AKPW13] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited - new reduction, properties a nd applications. In *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 57–74, 2013.

[Alp15] Jacob Alperin-Sheriff. Short signatures with short public keys from homomorphic trapdoor functions. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 236–255, 2015.

[AP10] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. *Theory of Computing Systems*, 48(3):535–553, 2010.

[AP12] Jacob Alperin-Sheriff and Chris Peikert. Circular and KDM security for identity-based encryption. In Marc Fischlin, Johannes Buchmann, and Mark Manulis, editors, *PKC 2012*, volume 7293 of *LNCS*, pages 334–352. Springer, Heidelberg, May 2012.

[AP13] Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasilinear time. In *CRYPTO (1)*, pages 1–20, 2013.

[AP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 297–314. Springer, Heidelberg, August 2014.

[App14] Benny Applebaum. Bootstrapping obfuscators via fast pseudorandom functions. In Palash Sarkar and Tetsu Iwata, editors, *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 162–172. Springer, Heidelberg, December 2014.

[APS15] Martin R. Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. Cryptology ePrint Archive, Report 2015/046, 2015. http://eprint.iacr.org/2015/046.

[ASA16] Jacob Alperin-Sheriff and Daniel Apon. Dimension-preserving reductions from lwe to lwr. Cryptology ePrint Archive, Report 2016/589, 2016. http://eprint.iacr.org/2016/589.

[ASP14] Jacob Alperin-Sheriff and Chris Peikert. Faster bootstrapping with polynomial error. In *CRYPTO (1)*, pages 297–314, 2014.

[Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in nc^1. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.

[BB08] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the SDH assumption in bilinear groups. *J. Cryptology*, 21(2):149–177, 2008.

[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 Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 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 Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

[BGK+14] Boaz Barak, Sanjam Garg, Yael Tauman Kalai, Omer Paneth, and Amit Sahai. Protecting obfuscation against algebraic attacks. In Phong Q. Nguyen and

Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 221–238. Springer, Heidelberg, May 2014.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *47th ACM STOC*, pages 439–448. ACM Press, June 2015.

[BGM+16] Andrej Bogdanov, Siyao Guo, Daniel Masny, Silas Richelson, and Alon Rosen. On the hardness of learning with rounding over small modulus. In *Theory of Cryptography - 13th International Conference, TCC 2 016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part I*, pages 209–224, 2016.

[BHJ+15] Florian Böhl, Dennis Hofheinz, Tibor Jager, Jessica Koch, and Christoph Striecks. Confined guessing: New signatures from standard assumptions. *J. Cryptology*, 28(1):176–208, 2015.

[BHLN15] Daniel J. Bernstein, Andreas Hülsing, Tanja Lange, and Ruben Niederhagen. Bad directions in cryptographic hash functions. In Ernest Foo and Douglas Stebila, editors, *ACISP 15*, volume 9144 of *LNCS*, pages 488–508. Springer, Heidelberg, June / July 2015.

[BHY09] Mihir Bellare, Dennis Hofheinz, and Scott Yilek. Possibility and impossibility results for encryption and commitment secure under selective opening. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 1–35. Springer, Heidelberg, April 2009.

[BKKP15] Olivier Blazy, Saqib A. Kakvi, Eike Kiltz, and Jiaxin Pan. Tightly-secure signatures from chameleon hash functions. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 256–279. Springer, Heidelberg, March / April 2015.

[BL16] Xavier Boyen and Qinyi Li. Towards tightly secure lattice short signature and id-based encryption. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, pages 404–434, 2016.

[BLL+15] Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the rényi divergence rather than the statistical distance. In *Advances in Cryptology - ASIACRYPT 2015 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part I*, pages 3–24, 2015.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 575–584. ACM Press, June 2013.

[BLR+15] Dan Boneh, Kevin Lewi, Mariana Raykova, Amit Sahai, Mark Zhandry, and Joe Zimmerman. Semantically secure order-revealing encryption: Multi-input functional encryption without obfuscation. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 563–594. Springer, Heidelberg, April 2015.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, December 2001.

[BLW15] Dan Boneh, Kevin Lewi, and David J. Wu. Constraining pseudorandom functions privately. *IACR Cryptology ePrint Archive*, 2015:1167, 2015.

[BMSZ15] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: The case of evasive circuits. Cryptology ePrint Archive, Report 2015/167, 2015. http://eprint.iacr.org/2015/167.

[BMSZ16] Saikrishna Badrinarayanan, Eric Miles, Amit Sahai, and Mark Zhandry. Post-zeroizing obfuscation: New mathematical tools, and the case of evasive circuits. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 764–791. Springer, Heidelberg, May 2016.

[BNNO11] Rikke Bendlin, Jesper Buus Nielsen, Peter Sebastian Nordholt, and Claudio Orlandi. Lower and upper bounds for deniable public-key encryption. In Dong Hoon Lee and Xiaoyun Wang, editors, *ASIACRYPT 2011*, volume 7073 of *LNCS*, pages 125–142. Springer, Heidelberg, December 2011.

[Boy10] Xavier Boyen. Lattice mixing and vanishing trapdoors: A framework for fully secure short signatures and more. In *Public Key Cryptography*, pages 499–517, 2010.

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, pages 353–370, 2014.

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In *EUROCRYPT*, pages 719–737, 2012.

[BR14] Zvika Brakerski and Guy N. Rothblum. Virtual black-box obfuscation for all circuits via generic graded encoding. In Yehuda Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 1–25. Springer, Heidelberg, February 2014.

[BS02] Dan Boneh and Alice Silverberg. Applications of multilinear forms to cryptography. Cryptology ePrint Archive, Report 2002/080, 2002. http://eprint.iacr.org/2002/080.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Heidelberg, March 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based FHE as secure as PKE. In Moni Naor, editor, *ITCS 2014*, pages 1–12. ACM, January 2014.

[BWZ14a] Dan Boneh, Brent Waters, and Mark Zhandry. Low overhead broadcast encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/195, 2014. http://eprint.iacr.org/2014/195.

[BWZ14b] Dan Boneh, David J. Wu, and Joe Zimmerman. Immunizing multilinear maps against zeroizing attacks. Cryptology ePrint Archive, Report 2014/930, 2014. http://eprint.iacr.org/2014/930.

[CDNO97] Ran Canetti, Cynthia Dwork, Moni Naor, and Rafail Ostrovsky. Deniable encryption. In Burton S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 90–104. Springer, Heidelberg, August 1997.

[CDPR16] Ronald Cramer, Léo Ducas, Chris Peikert, and Oded Regev. Recovering short generators of principal ideals in cyclotomic rings. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 559–585. Springer, Heidelberg, May 2016.

[CDW16] Ronald Cramer, Léo Ducas, and Benjamin Wesolowski. Short stickelberger class relations and application to ideal-svp. *IACR Cryptology ePrint Archive*, 2016:885, 2016.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

[CFL+16] Jung Hee Cheon, Pierre-Alain Fouque, Changmin Lee, Brice Minaud, and Hansol Ryu. Cryptanalysis of the new CLT multilinear map over the integers. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 509–536. Springer, Heidelberg, May 2016.

[CGGI16] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Cryptology ePrint Archive, Report 2016/870, 2016.

[CGH+15a] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 247–266. Springer, Heidelberg, August 2015.

[CGH+15b] Jean-Sébastien Coron, Craig Gentry, Shai Halevi, Tancrède Lepoint, Hemanta K. Maji, Eric Miles, Mariana Raykova, Amit Sahai, and Mehdi Tibouchi. Zeroizing without low-level zeroes: New MMAP attacks and their limitations. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 247–266, 2015.

[CGH16a] Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. Cryptology ePrint Archive, Report 2016/998, To ap-

pear in EUROCRYPT 2017, 2016. http://eprint.iacr.org/2016/998.

[CGH16b] Yilei Chen, Craig Gentry, and Shai Halevi. Cryptanalyses of candidate branching program obfuscators. Personal Communication, 2016.

[CGP15] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 557–585. Springer, Heidelberg, March 2015.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 523–552. Springer, Heidelberg, May 2010.

[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. Preliminary version in Eurocrypt 2010.

[CHL+15] Jung Hee Cheon, Kyoohyung Han, Changmin Lee, Hansol Ryu, and Damien Stehlé. Cryptanalysis of the multilinear map over the integers. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 3–12. Springer, Heidelberg, April 2015.

[CIO16] Angelo De Caro, Vincenzo Iovino, and Adam O'Neill. Deniable functional encryption. In *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part I*, pages 196–222, 2016.

[CJL16] Jung Hee Cheon, Jinhyuck Jeong, and Changmin Lee. An algorithm for NTRU problems and cryptanalysis of the GGH multilinear map without a low

level encoding of zero. Cryptology ePrint Archive, Report 2016/139, 2016. http://eprint.iacr.org/2016/139.

[CLLT16a] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Cryptanalysis of GGH15 multilinear maps. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 607–628. Springer, Heidelberg, August 2016.

[CLLT16b] Jean-Sébastien Coron, Moon Sung Lee, Tancrède Lepoint, and Mehdi Tibouchi. Zeroizing attacks on indistinguishability obfuscation over clt13. Cryptology ePrint Archive, Report 2016/1011, To appear in PKC 2017, 2016. http://eprint.iacr.org/2016/1011.

[CLT13] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. Practical multilinear maps over the integers. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 476–493. Springer, Heidelberg, August 2013.

[CLT15] Jean-Sébastien Coron, Tancrède Lepoint, and Mehdi Tibouchi. New multilinear maps over the integers. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 267–286. Springer, Heidelberg, August 2015.

[CN11] Yuanmi Chen and Phong Q. Nguyen. BKZ 2.0: Better lattice security estimates. In *ASIACRYPT*, pages 1–20, 2011.

[Cry] Cryptol. http://cryptol.net/. Accessed: 2016-05-02.

[CW13] Jie Chen and Hoeteck Wee. Fully, (almost) tightly secure IBE and dual system groups. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA,*

USA, August 18-22, 2013. Proceedings, Part II, volume 8043 of *Lecture Notes in Computer Science*, pages 435–460. Springer, 2013.

[DG14] Nagarjun C. Dwarakanath and Steven D. Galbraith. Sampling from discrete gaussians for lattice-based cryptography on a constrained device. *Appl. Algebra Eng. Commun. Comput.*, 25(3):159–180, 2014.

[DGG+16] Nico Döttling, Sanjam Garg, Divya Gupta, Peihan Miao, and Pratyay Mukherjee. Obfuscation from low noise multilinear maps. Cryptology ePrint Archive, Report 2016/599, 2016.

[DKR15] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 586–613. Springer, Heidelberg, March 2015.

[DLZ15] Dana Dachman-Soled, Feng-Hao Liu, and Hong-Sheng Zhou. Leakage-resilient circuits revisited - optimal number of computing components without leak-free hardware. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 131–158. Springer, Heidelberg, April 2015.

[DM14] Léo Ducas and Daniele Micciancio. Improved short lattice signatures in the standard model. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 335–352. Springer, 2014.

[DM15] Léo Ducas and Daniele Micciancio. FHEW: bootstrapping homomorphic encryption in less than a second. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications*

of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I, pages 617–640, 2015.

[DN02] Ivan Damgrard and Jesper Buus Nielsen. Expanding pseudorandom functions; or: From known-plaintext security to chosen-plaintext security. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 449–464. Springer, 2002.

[DRS04] Yevgeniy Dodis, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 523–540. Springer, Heidelberg, May 2004.

[DS15] Nico Döttling and Dominique Schröder. Efficient pseudorandom functions via on-the-fly adaptation. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, volume 9215 of *Lecture Notes in Computer Science*, pages 329–350. Springer, 2015.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009. http://crypto.stanford.edu/craig.

[GGG+14] Shafi Goldwasser, S. Dov Gordon, Vipul Goyal, Abhishek Jain, Jonathan Katz, Feng-Hao Liu, Amit Sahai, Elaine Shi, and Hong-Sheng Zhou. Multi-input functional encryption. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 578–602. Springer, Heidelberg, May 2014.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 1–17. Springer, Heidelberg, May 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 *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013.

[GGH15] Craig Gentry, Sergey Gorbunov, and Shai Halevi. Graph-induced multilinear maps from lattices. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 498–527. Springer, Heidelberg, March 2015.

[GGHZ14] Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Fully secure attribute based encryption from multilinear maps. Cryptology ePrint Archive, Report 2014/622, 2014. http://eprint.iacr.org/2014/622.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 467–476. ACM Press, June 2013.

[GHR99] Rosario Gennaro, Shai Halevi, and Tal Rabin. Secure hash-and-sign signatures without the random oracle. In Jacques Stern, editor, *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques, Prague, Czech Republic, May 2-6, 1999, Proceeding*, volume 1592 of *Lecture Notes in Computer Science*, pages 123–139. Springer, 1999.

[GHS12a] Craig Gentry, Shai Halevi, and Nigel P. Smart. Better bootstrapping in fully homomorphic encryption. In *Public Key Cryptography*, pages 1–16, 2012.

[GHS12b] Craig Gentry, Shai Halevi, and Nigel P. Smart. Homomorphic evaluation of the AES circuit. In *CRYPTO*, pages 850–867, 2012.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In Daniele Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 308–326. Springer, Heidelberg, February 2010.

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013.

[GLSW15] Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 151–170. IEEE Computer Society Press, October 2015.

[GLW14] Craig Gentry, Allison B. Lewko, and Brent Waters. Witness encryption from instance independent assumptions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 426–443. Springer, Heidelberg, August 2014.

[GMM+16] Sanjam Garg, Eric Miles, Pratyay Mukherjee, Amit Sahai, Akshayaram Srinivasan, and Mark Zhandry. Secure obfuscation in a weak multilinear map model. In *TCC 2016-B*, 2016.

[GMS16] Sanjam Garg, Pratyay Mukherjee, and Akshayaram Srinivasan. Obfuscation without the vulnerabilities of multilinear maps. Cryptology ePrint Archive, Report 2016/390, 2016.

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 614–637. Springer, Heidelberg, March 2015.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 06*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, *40th ACM STOC*, pages 197–206. ACM Press, May 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part I*, volume 8042 of *LNCS*, pages 75–92. Springer, Heidelberg, August 2013.

[GV15] Sergey Gorbunov and Dhinakaran Vinayagamurthy. Riding on asymmetry: Efficient ABE for branching programs. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 550–574. Springer, Heidelberg, November / December 2015.

[GVW15a] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 503–523. Springer, Heidelberg, August 2015.

[GVW15b] Sergey Gorbunov, Vinod Vaikuntanathan, and Daniel Wichs. Leveled fully homomorphic signatures from standard lattices. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 469–477, 2015.

[HAO15] Ryo Hiromasa, Masayuki Abe, and Tatsuaki Okamoto. Packing messages and optimizing bootstrapping in GSW-FHE. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 699–715, 2015.

[HJ16] Yupu Hu and Huiwen Jia. Cryptanalysis of GGH map. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 537–565. Springer, Heidelberg, May 2016.

[HS15] Shai Halevi and Victor Shoup. Bootstrapping for helib. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 641–670, 2015.

[HW09] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 654–670. Springer, 2009.

[Kay09] N. Kayal. The complexity of the annihilating polynomial. In *Computational Complexity, 2009. CCC '09. 24th Annual IEEE Conference on*, pages 184–193, July 2009.

[KR00] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *Proceedings of the Network and Distributed System Security Symposium, NDSS 2000, San Diego, California, USA*. The Internet Society, 2000.

[KTZ13] Jonathan Katz, Aishwarya Thiruvengadam, and Hong-Sheng Zhou. Feasibility and infeasibility of adaptively secure fully homomorphic encryption. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *PKC 2013*, volume 7778 of *LNCS*, pages 14–31. Springer, Heidelberg, February / March 2013.

[KW03] Jonathan Katz and Nan Wang. Efficiency improvements for signature schemes with tight security reductions. In Sushil Jajodia, Vijayalakshmi Atluri, and Trent Jaeger, editors, *Proceedings of the 10th ACM Conference on Computer and Communications Security, CCS 2003, Washington, DC, USA, October 27-30, 2003*, pages 155–164. ACM, 2003.

[Lep14] Tancrède Lepoint. *Design and Implementation of Lattice-based Cryptography*. PhD thesis, Université du Luxembourg, May 2014.

[Lin16] Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In *EUROCRYPT*, 2016.

[LPST16] Huijia Lin, Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation with non-trivial efficiency. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 447–462. Springer, Heidelberg, March 2016.

[LS14] Hyung Tae Lee and Jae Hong Seo. Security analysis of multilinear maps over the integers. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014,*

Part I, volume 8616 of *LNCS*, pages 224–240. Springer, Heidelberg, August 2014.

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Des. Codes Cryptography*, 75(3):565–599, 2015.

[LSS14] Adeline Langlois, Damien Stehlé, and Ron Steinfeld. GGHLite: More efficient multilinear maps from ideal lattices. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 239–256. Springer, Heidelberg, May 2014.

[LW16] Kevin Lewi and David J. Wu. Order-revealing encryption: New constructions, applications, and lower bounds. In *CCS*, 2016.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 465–484. Springer, Heidelberg, August 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Heidelberg, April 2012.

[MR04] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. In *45th FOCS*, pages 372–381. IEEE Computer Society Press, October 2004.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on Gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007. Preliminary version in FOCS 2004.

[MSW14] Eric Miles, Amit Sahai, and Mor Weiss. Protecting obfuscation against arithmetic attacks. Cryptology ePrint Archive, Report 2014/878, 2014. http://eprint.iacr.org/2014/878.

[MSZ16a] Eric Miles, Amit Sahai, and Mark Zhandry. Annihilation attacks for multilinear maps: Cryptanalysis of indistinguishability obfuscation over GGH13. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 629–658. Springer, Heidelberg, August 2016.

[MSZ16b] Eric Miles, Amit Sahai, and Mark Zhandry. Secure obfuscation in a weak multilinear map model: A simple construction secure against all known attacks. Cryptology ePrint Archive, Report 2016/588, 2016.

[NR99] Moni Naor and Omer Reingold. Synthesizers and their application to the parallel construction of pseudo-random functions. *J. Comput. Syst. Sci.*, 58(2):336–375, 1999.

[NR04] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. *J. ACM*, 51(2):231–262, 2004.

[NRR02] Moni Naor, Omer Reingold, and Alon Rosen. Pseudorandom functions and factoring. *SIAM J. Comput.*, 31(5):1383–1404, 2002.

[Ogu16] Arthur Ogus. Row equivalence of matrices (lecture notes). https://math.berkeley.edu/~ogus/old/Math_110-07/Supplements/week6.pdf, 2016. Online; accessed 30 September 2016.

[OPW11] Adam O'Neill, Chris Peikert, and Brent Waters. Bi-deniable public-key encryption. In Phillip Rogaway, editor, *CRYPTO 2011*, volume 6841 of *LNCS*, pages 525–542. Springer, Heidelberg, August 2011.

[OvdPS15] Emmanuela Orsini, Joop van de Pol, and Nigel P. Smart. Bootstrapping BGV ciphertexts with a wider choice of p and q. In *Public-Key Cryptography - PKC 2015 - 18th IACR International Conference on Practice and Theory in Public-Key Cryptography, Gaithersburg, MD, USA, March 30 - April 1, 2015, Proceedings*, pages 673–698, 2015.

[PST13] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multi-linear encodings. Cryptology ePrint Archive, Report 2013/781, 2013. http://eprint.iacr.org/2013/781.

[PST14] Rafael Pass, Karn Seth, and Sidharth Telang. Indistinguishability obfuscation from semantically-secure multilinear encodings. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 500–517. Springer, Heidelberg, August 2014.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6):1–40, 2009. Preliminary version in STOC 2005.

[RR94] Alexander A. Razborov and Steven Rudich. Natural proofs. In Frank Thomson Leighton and Michael T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 204–213. ACM, 1994.

[RT92] John H. Reif and Stephen R. Tate. On threshold circuits and polynomial computation. *SIAM J. Comput.*, 21(5):896–908, 1992.

[S+16] W. A. Stein et al. *Sage Mathematics Software (Version 7.3)*. The Sage Development Team, 2016. http://www.sagemath.org.

[SE94] Claus-Peter Schnorr and M. Euchner. Lattice basis reduction: Improved practical algorithms and solving subset sum problems. *Mathmatical Programming*, 66:181–199, 1994.

[ST01] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

[SZ14] Amit Sahai and Mark Zhandry. Obfuscating low-rank matrix branching programs. Cryptology ePrint Archive, Report 2014/773, 2014. http://eprint.iacr.org/2014/773.

[Ver12] Roman Vershynin. *Compressed Sensing, Theory and Applications*, chapter 5, pages 210–268. Cambridge University Press, 2012. Available at http://www-personal.umich.edu/~romanv/papers/non-asymptotic-rmt-plain.pdf.

[Zim15] Joe Zimmerman. How to obfuscate programs directly. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 439–467. Springer, Heidelberg, April 2015.