Compact Reusable Garbled Circuits

by

Dhinakaran Vinayagamurthy

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Compact Reusable Garbled Circuits

Dhinakaran Vinayagamurthy
Master of Science
Graduate Department of Computer Science
University of Toronto
2014

Garbled circuits are integral to secure function evaluation. A garbled circuit $\hat{C}$ for a circuit C enables a user to compute $C(x)$ and nothing more about C or x, when given an encoding $\hat{x}$ for the input x. Earlier, garbling schemes produced only single-use garbled circuits which did not offer security when used to evaluate more than one input encoding. Very recently, the first reusable version of garbled circuits was constructed by Goldwasser et al. (STOC 2013), which allowed a garbled circuit to evaluate multiple input encodings. But, all these constructions of garbled circuits, including the single-use ones, incur a *multiplicative* blowup in size i.e the size of the garbled circuit is $|C| \cdot \text{poly}(\lambda)$ for some security parameter λ. Hence, a fundamental question about (reusable) garbled circuits is:

How small can a garbled circuit be?

The main result of this thesis is a garbling scheme which produces (reusable) garbled circuits with just an additive overhead in size i.e with size $|C| + \text{poly}(\lambda, d_{\max})$. Here $d_{\max}$ is the maximum depth of circuits which the garbling scheme can handle. Modulo the additive $\text{poly}(\lambda, d_{\max})$ factor which is independent of the size of the circuit, this is the best that one could hope for.

The main technical ingredient of our work is a "fully" key-homomorphic encryption scheme; an object that we define and construct based on learning with errors (LWE) assumption. A fully key-homomorphic encryption scheme enables one to add/multiply ciphertexts encrypted under different (public) keys such that the output ciphertext is the encryption under the addition/multiplication of the public keys. Starting from this fully key-homomorphic encryption scheme, we also obtain an attribute-based and a (single-key) functional encryption schemes with very short secret keys. In both these schemes, the secret key for a circuit C consists of C itself *plus* $\text{poly}(\lambda, d_{\max})$ additional bits.

Acknowledgements

First of all, I thank my advisor Vinod Vaikuntanathan for all the wonderful discussions we have had in the last two years. I am forever indebted to him for his patience in teaching me the math background and in offering simple, insightful explanations whenever we start discussing a new topic. I also thank him for all his support, including the one during the first five days of July 2013 when we spent innumerable hours to move from identifying a bug in our solution to another problem to solving the problem which forms this thesis. Next, I would like to thank Sergey Gorbunov for all his academic and non-academic advice and for hosting me at his house for two days during my trip to Boston. Though he was away when the crux of this thesis was done, he was almost like a 'deputy advisor' to me during his time at Toronto.

I then thank Charles Rackoff for his crypto course and the discussions during the crypto reading groups. As a reader for my thesis, Charlie's comments helped me to obtain an improved understanding of some of the definitions. Also, I thank Wesley George for the discussions during his time here. I also thank my suite-mate Jaiganesh for all the non-crypto and crypto related conversations and cooking in the last two years.

Although we rarely discussed crypto, my friends from the theory lab: Dai, David, Jimmy, Joel, Lalla, Norman, Mika, Minjeong, Robert, Venkatesh, Yuval helped me in having a great time at Toronto. A big thanks to every one of them.

Finally, I thank my parents and grand parents for always providing me a strong mental support.

Contents

Chapter 1

Introduction

1.1 Garbled circuits

Garbling schemes are immensely powerful tools in modern cryptography. A circuit garbling scheme, first suggested by Yao for Secure Function Evaluation, encodes a circuit C into a garbled circuit $\hat{C}$ and then encodes an input $\mathbf{x}$ into $\hat{\mathbf{x}}$, so that any user given $\hat{C}$ and $\hat{\mathbf{x}}$ can obtain only $C(\mathbf{x})$ and nothing more. That is, the garbled circuits and inputs provide circuit and input privacy: an adversary given $\hat{C}$ and $\hat{\mathbf{x}}$ can learn nothing about the circuit C or the input $\mathbf{x}$, other than the information that can be obtained from $C(\mathbf{x})$. Also, we require the garbling schemes to be efficient in terms of computing the input encoding i.e after the generation of a garbled circuit, an input encoding for that should be generated without using the original circuit. Trivially, a garbling scheme on input C and $\mathbf{x}$ could output $(\hat{C} = \phi, \hat{\mathbf{x}} = C(\mathbf{x}))$. This garbled circuit satisfies the security guarantee, but it does not satisfy the efficiency guarantee. Yao provided the first construction of a garbling scheme which satisfies both.

1.1.1 Yao's garbled circuit

Informally, Yao's garbled circuit works as follows: assigns two independently randomly chosen labels $L_{i,0}, L_{i,1}$, corresponding to the two possible values 0 and 1, to each wire of the circuit $i \in \{1, \ldots, |C|\}$. The input encoding $\hat{\mathbf{x}}$ is the tuple consisting of labels of the input wires corresponding to the bits $x_1, \ldots, x_\ell$ of the input i.e $\{L_{i,x_i}\}_{i \in \{1,\ldots,\ell\}}$. Now, the garbled circuit $\hat{C}$ is a composition of $|C|$ garbled tables and a component τ, where a garbled table is provided for each gate of the circuit. The garbled table for the gate $g(u, v; w)$ enables the following transformation

$$L_{u,x_u}, L_{v,x_v} \longmapsto L_{w,x_w = g(x_u, x_v)}$$

The component $\tau = \{L_{|C|,0}||0, L_{|C|,1}||1\}$ i.e the two output labels together with the corresponding output bits. This way, an evaluator who has $\hat{\mathbf{x}}, \hat{C}$ starts with the input labels (obtained from $\hat{\mathbf{x}}$), uses the garbled tables (obtained from $\hat{C}$) to obtain the labels corresponding to the wires level by level and finally obtains the label corresponding to the output wire of the circuit. Then, using τ, he obtains the output bit $C(\mathbf{x})$.

Applications

Over the years, garbled circuits and its variants have found many applications including the following:

- secure two-party computation [Yao86]: two parties with respective private inputs x, y can compute a public function $f(x, y)$, such that an adversary does not learn anything about x, y other than what could be learned from $f(x, y)$. (Also, each party involved does not learn anything about the other party's input other than what he could be learn from $f(x, y)$.)
- secure multi-party computation [GMW87]: multi-party extension of secure two-party computation
- one-time programs [GKR08]: for any function f, a one-time program can be generated which allows one to compute f on a single input x of his choice.
- key-dependent message security [BHHI10]: security is provided even when encryptions of messages of the form $f(\mathsf{sk})$ are provided.
- verifiable computation [GGP10]: allows a computationally weak client to outsource computation of a function f on various inputs to one or more workers. Each worker returns the result and a proof, such that the work done by the client including the verification of the proof is substantially less than the work involved in computing the function by himself.
- homomorphic computations [GHV10]: enables computing a function f on encryptions of messages $x_1, \ldots, x_\kappa$, such that the resulting ciphertext is the encryption of $f(x_1, \ldots, x_\kappa)$.

Optimisations

There are two main disadvantages with Yao's construction of garbled circuits and all its variants.

- **Not reusable :** All the known variants of garbled circuits (till very recently) offered only one-time security. The security of a garbled circuit is compromised if encodings are given for more than one input for a particular garbled circuit. In particular, for the above construction of garbled circuit, when a user obtains encoding for two inputs, $\mathbf{x}$ and its complement $\bar{\mathbf{x}}$, he essentially obtains both the labels $L_{i,0}, L_{i,1}$ corresponding to each bit i of the input. To see this, if the ith bit of $\mathbf{x}$ $x_i = 1$, then the ith bit of $\bar{\mathbf{x}}$ is 0 and hence the user obtains $L_{i,1}$ from $\hat{\mathbf{x}}$ and $L_{i,0}$ from $\hat{\bar{\mathbf{x}}}$. Thus, with $\hat{\mathbf{x}}$ and $\hat{\bar{\mathbf{x}}}$ the user can obtain the garbled input $\hat{\mathbf{y}}$ for any input vector $\mathbf{y} \in \{0,1\}^\ell$ by himself, and hence can learn the output $C(\mathbf{y})$ (having already obtained the garbled circuit $\hat{C}$ with $\hat{\mathbf{x}}$ and $\hat{\bar{\mathbf{x}}}$). Thus, the single-use nature forces the user to generate a fresh set of garbled circuit and input encoding for each evaluation, even if the circuit to be evaluated remains the same.
- **Not compact :** Almost all the known constructions of garbled circuits proceed by garbling each gate and compose the garbled tables to form the garbled circuit. Hence, these garbled circuits have a *multiplicative* size blowup of $\text{poly}(\lambda)$, for some security parameter λ i.e, the size of the garbled circuit is proportional to the number of gates in the circuit.

Prior to our work, some attempts have been made to overcome these problems individually. For instance, Goldwasser, Kalai, Popa, Vaikuntanathan and Zeldovich [GKP+13b] provided the very first construction of reusable garbled circuits based on the learning with errors (LWE) assumption, but they do not care about the second problem. There are also some results which provide partial solutions to the second problem (and ignoring the first problem). Prominent ones among these are:

- A "free XOR" optimization for one-time garbled circuits introduced by Kolesnikov and Schneider [KS08] and further studied in [CKKZ12] and [App13]. This technique allows the size of the garbled

circuit to depend only on the number of non-XOR gates in the circuit, and there are no garbled tables for the XOR gates.

- Garbling of RAM programs for single use, obtained by Lu and Ostrovsky [LO13]. Here, the size of the garbled programs grows as poly(λ) its running time, for some security parameter λ.
- Garbling (reusably) non-uniform Turing machines, shown by Goldwasser et al. [GKP+13a]. Here, they incur a multiplicative poly(λ) overhead in the size of the non-uniformity of the machine, which they show under a non-standard yet non-falsifiable assumption.

But in spite of all these, we do not yet know of a reusable garbled circuit without a multiplicative blowup in size, or even a single-use garbled circuit without a multiplicative blowup in the *multiplicative* size of the circuit.

1.1.2 Our Results

We provide new techniques which almost solves the second problem and at the same time we could apply the techniques of [GKP+13b] to our scheme to solve the first problem. Our result is a circuit garbling scheme which produces reusable garbled circuits for arbitrary depth-$d_{\max}$ circuits with an additive overhead in size, namely $|\hat{C}| = |C| + \text{poly}(\lambda, d_{\max})$. We base the security of our construction on the sub-exponential hardness of the LWE assumption [Reg09a]. Modulo the dependence on the depth $d_{\max}$ of the circuit and the polynomial dependence on the security parameter λ, our construction gives the smallest possible garbled circuit. For large and shallow circuits, such as those that arise from database lookup, search and some machine learning applications, this gives significant bandwidth savings over the previous methods (even in the single use setting).

Theorem 1.1.1 (Informal version of our main theorem)**.** *Assuming sub-exponential LWE, there is a reusable garbled circuit garbling scheme that garbles a depth-$d_{\max}$ circuit C into a garbled circuit $\hat{C}$ such that $|\hat{C}| = |C| + \text{poly}(\lambda, d_{\max})$, and garbles an input $\mathbf{x}$ into an encoded input $\hat{\mathbf{x}}$ such that $|\hat{\mathbf{x}}| = |\mathbf{x}| \cdot \text{poly}(\lambda, d_{\max})$.*

Our *compact* garbled circuits construction follows along the lines of [GKP+13b]. That is, we first construct an attribute-based encryption scheme, and then we apply the reductions in [GKP+13b] first to get a single-key functional encryption scheme and from that, we get compact garbled circuits. The improvement in our construction comes from our new attribute-based encryption scheme, which produces *short* secret keys. The size of the secret key sk_C for a circuit C is independent of the size of the circuit, and depends only on the depth of the circuit.

1.2 Attribute-based encryption

Public key encryption provides the following two aspects: data integrity and confidentiality. When $\mathsf{Enc}(m)$ is transmitted to a receiver, data integrity ensures that no adversary alters the message m during its transmission without being noticed, and confidentiality ensures that no adversary learns *any* information about m. Consider an organisation XYZ Tech where there is a common channel channel for communication to which each employee of the company has access to. If Alice, an employee, wants to send a message m *securely* to another employee Bob, she can do the following:

1. Look up Bob's public key pk_B.

2. Compute $\mathsf{ct} = \mathsf{Enc}_{\mathsf{pk}_B}(m)$ and sends it to Bob.

On receiving ct, Bob can just decrypt it using his secret key sk_B and get back m. The correctness of m is ensured by data integrity and the privacy is ensured by confidentiality. In the same way, Alice can *securely* send multiple messages to multiple employees, just by sending them in an encrypted form, so that only the intended recipient gets to decrypt the ciphertext. Now consider the scenario where Alice wants to send a message to all the Canadian project managers who work in security related projects. With the traditional public key encryption algorithms, it is all-or-nothing access to data. That is, only the person who possesses the secret key can decrypt a ciphertext and all others learn nothing from that. So, Alice has a cumbersome task of identifying the employees who satisfy all the three constraints (Canadian *and* project manager *and* security), look up their public keys and encrypt the note individually to every one of them . This leads to the following essential question: *Does there exist an encryption scheme which allows one to encrypt a message, such that only those who possess* certain specific set of attributes *can decrypt and learn the message?*

This question was first addressed by Sahai and Waters [SW05]. They put forward the notion of attribute-based encryption (ABE). Their attribute-based encryption scheme labels ciphertexts and secret keys with a set of descriptive attributes and a user decrypts a ciphertext only if there are *enough* attributes in common between his secret key and the ciphertext. This is a good first step, but this ABE scheme had many limitations, which prevented its applicability in larger systems. One main limitation is that, this scheme allowed collusion among users i.e a US *project manager* can collude with a *Canadian* employee and a Mexican employee *working on security related projects* to decrypt a message intended for a Canadian project manager working on security related projects.

Goyal, Pandey, Sahai and Waters [GPSW06] provided a concrete definition for attribute-based encryption and a more expressive attribute-based encryption scheme which enables fine-grained access control of data. Their scheme is a key-policy[1] attribute-based encryption, where each ciphertext $\mathsf{ct}_{\mathbf{x}}$ is labelled with a set of attributes, represented by a vector $\mathbf{x}$ and a secret key sk_C is provided for a circuit C, which defines the affiliations of the user who possesses sk_C. A user who possesses sk_C can decrypt a ciphertext $\mathsf{ct}_{\mathbf{x}}$ if and only if $C(\mathbf{x}) = 1$. Though this generalises [SW05] and handles a richer class of functionalities, the scheme in [GPSW06] works only for a restricted class of circuits. In particular, their scheme could handle only boolean formulas which form the class of circuits with logarithmic depth NC^1. Following [GPSW06], significant progress has been made in attribute-based encryption in increasing the efficiency and the security guarantees and in diversifying the security assumptions [Wat09, LW10, LOS+10, CHKP12, ABB10, OT10, Boy13]. But all these schemes could handle only *log*-depth circuits (NC^1). To be more useful, we would like to have attribute-based encryption schemes which could handle all functionalities computable in polynomial time.

The NC^1 barrier was first overcome by Gorbunov, Vaikuntanathan and Wee [GVW13]. They proposed an attribute-based encryption scheme based on Learning With Errors (LWE) assumption, which could handle circuits of polynomial size and a priori bounded polynomial depth. Another attribute-based

[1] In a key-policy ABE, the policies represented as circuits are associated with secret keys whereas in its dual version, a ciphertext-policy ABE, the policies are associated with ciphertexts

encryption scheme which could handle such circuits, has been proposed by Garg, Gentry, Halevi, Sahai and Waters [GGH+13b], but with the security based on multilinear maps [GGH13a]. But all these constructions have a *large* secret key sk_C with their size proportional to the number of the gates in C.

Our new ABE construction facilitates *short* secret keys with sizes dependent only on the depth of the circuit and independent of the size of the circuit. The security of our construction is based on the sub-exponential hardness of the LWE assumption.

Theorem 1.2.1 (informal)**.** *Assuming sub-exponential LWE, there is a key-policy attribute based encryption scheme for depth-$d_{\max}$ circuits C such that $|\mathsf{sk}_C| = |C| + \mathrm{poly}(\lambda, d_{\max})$.*

We use this ABE scheme to build a single-key secure succinct functional encryption scheme with short secret keys using the techniques of [GKP+13b].

1.3 Functional encryption

Functional encryption (FE) is a primitive which is more powerful than ABE. FE allows a user possessing a secret key sk_C for a circuit C and an encryption of $\mathbf{x}$ to learn only $C(\mathbf{x})$ and nothing more about $\mathbf{x}$. An FE scheme is also required to provide security against collusions. That is, users possessing secret keys $\mathsf{sk}_{C1}, \ldots, \mathsf{sk}_{C_n}$ (for any polynomial n) on colluding should not be able to get any information about $\mathbf{x}$ from an encryption of $\mathbf{x}$ other than $C_1(\mathbf{x}), \ldots, C_n(\mathbf{x})$. The notion of FE was first formalised by Boneh, Sahai and Waters [BSW11] and O'Neill [O'N10]. Some previous works [KSW08, LOS+10, AFV11] showed that a FE scheme for inner-product function $f_{\mathbf{y}}$ (which on input $\mathbf{x}$ outputs 1 iff $\langle \mathbf{x}, \mathbf{y} \rangle = 0$) can be constructed. Sahai and Seyalioglu [SS10] constructed an FE scheme for general functions but security is provided only against a single secret key query. Gorbunov, Vaikuntanathan and Wee [GVW12] constructed an FE scheme for general functions secure against bounded number of secret key queries. In both the above schemes, the size of the ciphertext is not succinct (it is proportional to the size of the universal circuit computing the functions allowed by the scheme[2]).

The first succinct (and single-key secure) FE scheme was constructed by Goldwasser et al. [GKP+13b]. They show that this variant of FE is powerful by providing interesting applications like reusable garbled circuits. They provide a way to obtain a single key FE scheme from three ingredients:

1. an ABE scheme which handles polynomial size circuits
2. a fully-homomorphic encryption scheme
3. a one-time garbled circuit (Eg. Yao's garbled circuit)

In this work, we follow the same procedure using from our ABE scheme to obtain a single-key succinct FE scheme with very short secret keys.

Theorem 1.3.1 (informal)**.** *Assuming sub-exponential LWE, there is a single-key secure succinct functional encryption scheme for depth-$d_{\max}$ circuits C such that $|\mathsf{sk}_C| = |C| + \mathrm{poly}(\lambda, d_{\max})$.*

[2] In [GVW12] the size of the ciphertext is also proportional to the secret-key query bound.

1.4 A new tool: Fully Key-homomorphic encryption

The crux of our results is the first construction of a fully key-homomorphic encryption scheme. An encryption scheme is key-homomorphic if the addition/multiplication of ciphertexts under different public keys produced by this scheme, yield a ciphertext under the addition/multiplication of the public keys. Different variants of key-homomorphic encryption have already been used in a few works in the literature. But all the known variants provide only additive key-homomorphism, where only the addition (and not multiplication) of ciphertexts yield a ciphertext under addition of public keys. This additive key-homomorphism has been shown to have some useful applications:

- randomized encryption scheme secure against related-key attacks [AHI11].
- inner-product predicate functional encryption [AFV11]: an encryption of a message m under a private label $\mathbf{x}$ can be decrypted by only those who have secret key $\mathsf{sk}_{\mathbf{y}}$ corresponding to a label $\mathbf{y}$ such that $\langle \mathbf{x}, \mathbf{y} \rangle = 0$.
- ID based encryption scheme without random oracles, secure against bounded number of collusions [GLW12].
- garbling scheme producing (single-use) garbled circuits with *compact* input encodings [AIKW13].

We formally define this primitive of *fully key-homomorphic encryption* and show how to construct an attribute-based encryption scheme with short secret keys from this primitive. We also provide an instantiation of a fully key-homomorphic encryption scheme based on learning with errors (LWE) assumption.

Theorem 1.4.1 (informal)**.** *Assuming sub-exponential LWE, there is a fully key-homomorphic encryption scheme.*

Theorem 1.4.2 (informal)**.** *Assuming the existence of a fully key-homomorphic encryption scheme, there is a key-policy attribute-based encryption scheme for depth-$d_{\max}$ circuits C such that $|\mathsf{sk}_C| = |C| + \mathrm{poly}(\lambda, d_{\max})$.*

One can clearly see that the theorems 1.4.2 and 1.4.1 lead to Theorem 1.2.1.

Organisation of the thesis

In Chapter 2, we present the lattice preliminaries required for our work and a formal definition of attribute-based encryption. Other preliminaries are provided in the corresponding sections.

In Chapter 3, we formally define our variant of fully key-homomorphic encryption (FKHE) and provide a construction of an attribute-based encryption scheme with short secret keys from a fully key-homomorphic encryption scheme.

In Chapter 4, we instantiate the FKHE scheme with its security reduced to the LWE assumption and analyse the parameters involved in this scheme. We also provide an independent description of our LWE based ABE scheme.

In Chapter 5, we discuss about the optimisations and extensions which are inherently provided by our ABE construction.

In Chapter 6, we provide the construction of compact garbled circuits using our ABE scheme.

In Chapter 7, we present some other important applications of the full key-homomorphism techniques used to build our ABE scheme.

In Chapter 8, we provide an ABE scheme with short secret keys based on the ring-LWE assumption and analyse the parameters of this scheme.

In Chapter 9, we conclude our results and provide interesting open problems in the direction of this work.

Chapter 2

Preliminaries

2.1 Lattice preliminaries

In this section we collect the results from the literature that we will need for our construction and the proof of security.

2.1.1 Notations

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

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

We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$, and we use $\mathrm{negl}(n)$ to denote a negligible function of n. We say $f(n)$ is *polynomial* if it is $O(n^c)$ for some $c > 0$, and we use $\mathrm{poly}(n)$ to denote a polynomial function of n. We say an event occurs with *overwhelming probability* if its probability is $1 - \mathrm{negl}(n)$.

For any vector $\mathbf{v}$, $\|\mathbf{v}\|$ refers to the euclidean norm of the vector.

$$\|\mathbf{v}\| := \sqrt{\sum_i (v_i^2)}$$

The infinite norm for the vector $\mathbf{v}$, $\|\mathbf{v}\|_\infty = \max_i (v_i)$. For any matrix $\mathbf{A} \in \mathbb{Z}^{n \times m}$, we refer $\|\mathbf{A}\|$ to the operator norm of $\mathbf{A}$.

$$\|\mathbf{A}\| := \sup_{\mathbf{x} \in S^{m-1}} \|\mathbf{A} \cdot \mathbf{x}\|$$

where S^m is the m-sphere $\{\mathbf{x} \in \mathbb{R}^{m+1} \ : \ \|\mathbf{x}\| = 1\}$. We will also use the maximum norm $\|\cdot\|_{\max}$ for a

matrix where $\|\mathbf{A}\|_{\max} = \max(a_{ij})$ for all $i \in [n], j \in [m]$.

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

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

2.1.2 Lattices

An m-dimensional lattice Λ is a discrete additive subgroup of $\mathbb{R}^m$. A basis of a lattice is a set of linearly independent vectors whose span is Λ. We usually deal with full-dimensional lattices where the basis set has m linearly independent vectors spanning Λ, with m being the lattice dimension. Integer lattices are those whose points have co-ordinates in $\mathbb{Z}^m$. The q-ary lattices are those defined as follows: for any integer $q \geq 2$ and any matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we define

$$\Lambda_q(\mathbf{A}) := \{\mathbf{r} \in \mathbb{Z}^m : \exists \mathbf{s} \in \mathbb{Z}_q^n \text{ s.t. } \mathbf{A}^t \cdot \mathbf{s} = \mathbf{r} \mod q\}$$

Here the matrix $\mathbf{A}$ is called the "basis" of the lattice. That is, a matrix $\mathbf{A}$ is a basis of a lattice Λ if $\Lambda = \Lambda_q(\mathbf{A})$. There can be multiple bases for the same lattice. It is usually hard to find a basis of 'low' norm given the lattice i.e given some arbitrary basis of the lattice.

Also we define a kernel lattice $\Lambda_q^{\perp}(\mathbf{A})$ for $\mathbf{A}$ as follows:

$$\Lambda_q^{\perp}(\mathbf{A}) := \{\mathbf{r} \in \mathbb{Z}_q^m : \mathbf{A} \cdot \mathbf{r} = \mathbf{0} \mod q\}$$

For any $\mathbf{u}$ admitting integral solution to $\mathbf{A} \cdot \mathbf{t} = \mathbf{u} \mod q$,we define a coset (or shifted) lattice of $\Lambda_q^{\perp}(\mathbf{A})$:

$$\Lambda_q^{\mathbf{u}}(\mathbf{A}) := \{\mathbf{r} \in \mathbb{Z}_q^m : \mathbf{A} \cdot \mathbf{r} = \mathbf{u} \mod q\} = \Lambda_q^{\perp}(\mathbf{A}) + \mathbf{t}$$

Gram-Schmidt orthogonalisation. The Gram-Schmidt orthogonalisation of an ordered set of vectors $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\} \in \mathbb{R}^n$ is $\widetilde{\mathbf{V}} = \{\widetilde{\mathbf{v}}_1, \ldots, \widetilde{\mathbf{v}}_k\}$ where $\widetilde{\mathbf{v}}_i$ is the component of $\mathbf{v}_i$ that is orthogonal to the span of $(\mathbf{v}_1, \ldots, \mathbf{v}_{i-1})$, for all $i \in [k]$. That is,

$$\widetilde{\mathbf{v}}_i = \mathbf{v}_i - \sum_{j=1}^{i-1} \frac{\langle \mathbf{v}_i, \widetilde{\mathbf{v}}_j \rangle}{\langle \widetilde{\mathbf{v}}_j, \widetilde{\mathbf{v}}_j \rangle} \cdot \widetilde{\mathbf{v}}_j$$

We refer to $\|\widetilde{\mathbf{V}}\|$ as the Gram-Schmidt norm of a matrix $\mathbf{V}$.

Note: We perform all the arithmetic operations $\mod q$. But, we make exemptions during the calculation of norms and Gram-Schmidt orthogonalisation, for which it is more natural to work in the reals.

Gaussian distributions. Let $D_{\mathbb{Z}^m,s,\mathbf{c}}$ be the truncated discrete Gaussian distribution over $\mathbb{Z}^m$ with standard deviation s and center $\mathbf{c}$, that is, we replace the output by $\mathbf{0}$ whenever the $||\cdot||_\infty$ norm exceeds $\sqrt{m} \cdot s$. For notational convenience, we would assume $\mathbf{c} = \mathbf{0}$ in most cases and hence abbreviate $D_{\mathbb{Z}^m,s,\mathbf{0}}$

to $D_{\mathbb{Z}^m,s}$.

The following lemma gives a bound on the length of vectors sampled from a discrete Gaussian. The result follows from [MR07, Lemma 4.4], using [GPV08, Lemma 5.3] to bound the smoothing parameter.

Lemma 2.1.1. *Let Λ be an n-dimensional lattice, let $\mathbf{T}$ be a basis for Λ, and suppose $s \geq \|\widetilde{\mathbf{T}}\| \cdot \omega(\sqrt{\log n})$. Then for any $\mathbf{c} \in \mathbb{R}^n$ we have*

$$\Pr\left[\|\mathbf{x} - \mathbf{c}\| > s\sqrt{n} : \mathbf{x} \xleftarrow{\mathrm{R}} D_{\Lambda,\sigma,\mathbf{c}}\right] \leq \mathrm{negl}(n)$$

Thus, the probability for a random sample from $D_{\mathbb{Z}^m,s}$ to be $\mathbf{0}$ is negligible and hence $D_{\mathbb{Z}^m,s}$ is $\sqrt{m} \cdot s$-bounded.

2.1.3 Learning With Errors (LWE) Assumption

The LWE problem was introduced by Regev [Reg09b] as a generalisation of the well-studied learning parity with noise (LPN) problem. Informally, the LWE problem states that, for any $q \geq 2$, a vector $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and an error value $\mathbf{e}$ sampled from a probability distribution χ on $\mathbb{Z}_q$, when an adversary is given the tuple $(\mathbf{a}, \langle \mathbf{a}, \mathbf{s} \rangle + e \pmod q)$ for any $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^n$, it is not possible for him to output $\mathbf{s}$ with noticeable probability. Regev showed that solving it *on the average* is as hard as (quantumly) solving several standard lattice problems *in the worst case*. We will formally define the decisional version of this problem which we will use in the thesis.

Definition 2.1.1 (dLWE)**.** *For an integer $q = q(n) \geq 2$ and an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the decisional version of the learning with errors problem $\mathsf{dLWE}_{n,m,q,\chi}$ is to distinguish between the following pairs of distributions:*

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

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

Throughout this paper, the parameter $m = \mathrm{poly}(n)$, in which case we will shorten the notation slightly to $\mathsf{dLWE}_{n,q,\chi}$.

Connection to lattices.

There are known quantum [Reg09b] and classical [Pei09] reductions between $\mathsf{dLWE}_{n,q,\chi}$ and approximating gapSVP problems in lattices in the worst case, where χ is a B-bounded (truncated) discretized Gaussian for some appropriate B. For a lattice dimension parameter n and a number d, gapSVP_γ is the promise problem of distinguishing whether a n-dimensional lattice has a vector shorter than d or no vector shorter than $\gamma(n) \cdot d$. We now state a corollary of the reductions of [Reg09b, Pei09] from gapSVP to LWE, in terms of the bound B on the distribution χ after applying the search to decision reduction for LWE [MP12].

Theorem 2.1.2 ([Reg09b, Pei09])**.** *Let $q = q(n) \in \mathbb{N}$ be product of powers of primes (with each distinct prime of size $\mathrm{poly}(n)$), and let $B \geq \omega(\log n) \cdot \sqrt{n}$. Then there exists an efficient sampleable distribution χ such that if there is an efficient algorithm that solves the average case $\mathsf{dLWE}_{n,q,\chi}$ problem, then:*

- *There is a quantum algorithm that solves* $\mathsf{gapSVP}_{\tilde{O}(nq/B)}$ *on any* n*-dimensional lattice.*
- *If* $q \geq \tilde{O}(2^{n/2})$*, there is an efficient classical algorithm for* $\mathsf{gapSVP}_{\tilde{O}(nq/B)}$ *for any* n*-dimensional lattice.*

When the error distribution is discrete Gaussian with parameter α, the above reductions in Theorem 2.1.2 can be interpreted with $\alpha q \geq 2B$, hence with $\mathsf{gapSVP}_{\tilde{O}(n/\alpha)}$. Note that the restriction on q to be a product of powers bounded primes is due to the reduction from dLWE to *lwe*. The reductions from LWE to the lattice problems do not require this property of q.

Recently, Brakerski et al. [BLP+13] provided an improved version of [Pei09]. They have obtained a classical reduction from LWE to gapSVP which works for polynomial modulus.

Theorem 2.1.3 (Informal theorem of [BLP+13])**.** *Solving* n*-dimensional* LWE *with* poly(n) *modulus implies an equally efficient solution to a worst-case lattice problem in dimension* $\sqrt{n}$*.*

Their main result is Lemma 2.1.4, which gives a reduction from the LWE problem with exponential modulus to the LWE problem with binary secrets and polynomial modulus. It is clear that this lemma when combined with the classical reduction in Theorem 2.1.2, leads to Theorem 2.1.3. And this new reduction works when the error distribution χ is a discrete Gaussian distribution with some parameter α.

Lemma 2.1.4 ([BLP+13])**.** *Let* $k, q \geq 1$ *and* $n \geq 1$ *be integers, and let* $\epsilon \in (0, 1/2), \alpha, \delta > 0$*, be such that* $n \geq (k+1)\log_2 q + 2\log_2(1/\delta), \alpha \geq \sqrt{\ln(2n(1+1/\epsilon))/\pi}/q$*. There exists a transformation from* $\mathsf{LWE}_{k,q,\alpha}$ *to* $\mathsf{LWE}_{n,q',\alpha'}$*, the latter with binary secrets, for any* $q' \geq 1$ *and* $\xi > 0$ *where*

$$\alpha' := \left(10n\alpha^2 + \frac{4n}{\pi q'^2}\ln(2n(1+1/\epsilon))\right)^{1/2}$$

Since the LWE problem with dimension $k = \sqrt{n}$ and modulus $q = 2^{k/2}$ is classically as hard as $\sqrt{n}$-dimensional gapSVP problem, Theorem 2.1.3 follows (for some q' as small as $\sqrt{n}$). The state-of-the-art algorithms for these lattice problems (Eg. gapSVP) run in time nearly exponential in the dimension n [AKS01, MV10]; more generally, we can get a 2^k-approximation in time $2^{\tilde{O}(n/k)}$.

Thus the $\mathsf{dLWE}_{n,q,\chi}$ assumption states that any adversary for the LWE problem with a discrete Gaussain distribution $\chi = D_{\mathbb{Z}^m,s}$ for any $s = \text{poly}(n)$ and q as large as 2^{n^ϵ} (for any constant $0 < \epsilon < 1$) can succeed with only a negl(n) advantage.

2.1.4 Trapdoors for Lattices

Trapdoor of a lattice is usually a "short" basis of its kernel lattice. For the lattice $\Lambda_q(\mathbf{A})$ defined by some matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, its trapdoor is a "short" basis $\mathbf{T_A} \in \mathbb{Z}_q^{m\times m}$ of its kernel lattice $\Lambda_q^{\perp}(\mathbf{A})$ i.e $\mathbf{A}\cdot\mathbf{T_A} = \mathbf{0}$. The most improved lattice trapdoor generation algorithm (which outputs a $(\mathbf{A}, \mathbf{T}_A)$ pair for some $\mathbf{A}$ which is statistically or computationally indistinguishable from random) that is available now, has been constructed by Micciancio and Peikert [MP12].

Lemma 2.1.5 (Lattice Trapdoors [Ajt99, MP12]). *There is an efficient randomized algorithm* $\mathsf{TrapSamp}(1^n, 1^m, q)$ *that, given any integers* $n \geq 1$, $q \geq 2$, *and sufficiently large* $m = \Omega(n \log q)$, *outputs a parity check matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ *and a trapdoor matrix* $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ *such that the distribution of* $\mathbf{A}$ *is* $\mathrm{negl}(n)$*-close to uniform and* $\|\widetilde{\mathbf{T_A}}\| \leq O(n \log q)$.

Here are some titbits about trapdoors. From Lemma 2.1.5, we can see that one can obtain a "random" matrix along with its trapdoor by running the $\mathsf{TrapSamp}$ algorithm. But when given a matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, it is hard to obtain its trapdoor in polynomial time. Note that one can easily obtain some basis of the kernel lattice since the equation $\mathbf{A} \cdot \mathbf{T}' = \mathbf{0}$ has multiple solutions for some $\mathbf{T}' \in \mathbb{Z}_q^{m \times m}$. But it is hard to get a "short" solution. In fact getting even a single column $\mathbf{t}$ of low norm such that $\mathbf{A} \cdot \mathbf{t} = \mathbf{0}$ is equivalent to the SIS (Short Integer Solution) problem, which itself is assumed to be hard.

But when a trapdoor is provided along with the matrix, a lot of operations which are originally hard can be performed easily. For instance, we will show (informally) how to solve the search version of the LWE problem given the trapdoor in polynomial time. In the following sections we will provide other applications of trapdoors which we will use in this work. Now lets look at the LWE problem. We are given a tuple $(\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e} \mod q)$ for some matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, secret vector $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$ and error vector $\mathbf{e} \xleftarrow{\$} \chi^m$ for some error distribution χ. If we are also provided with a trapdoor $\mathbf{T_A} \in \mathbb{Z}_q^{m \times m}$ for the matrix $\mathbf{A}$, we can calculate $\mathbf{T_A}^T \cdot (\mathbf{A}^T\mathbf{s} + \mathbf{e}) = \mathbf{T_A}^T\mathbf{e} \mod q$, since $\mathbf{T_A}^T\mathbf{A}^T\mathbf{s} = (\mathbf{AT_A})^T\mathbf{s} = \mathbf{0}$. If the parameters are chosen appropriately, since by our assumption both $\mathbf{T_A}$ and $\mathbf{e}$ have low norm, all the values of $\mathbf{T_A}^T\mathbf{e}$ are small compared to q even without the $\pmod q$ operation i.e the values of $\mathbf{T_A}^T\mathbf{e}$ do not wrap around q. Now getting $\mathbf{e}$ from this value is equivalent to solving m equations in m variables over integers. This set of equations have a unique solution since the columns of $\mathbf{T_A}$ are linearly independent. After getting $\mathbf{e}$ we can subtract this from the LWE instance to get $\mathbf{A}^T\mathbf{s}$ from which $\mathbf{s}$ can be obtained easily. Thus we can solve the LWE problem when the trapdoor is given. Note that if we try to perform these steps with some basis $\mathbf{T}'$ (not necessarily one with low norm) of the kernel lattice, we would not be able to find the correct error vector $\mathbf{e}$ from $\mathbf{T}'\mathbf{e} \mod q$.

Sampling algorithms

Lattice trapdoors can also be used to obtain "short" pre-images. Given a matrix $\mathbf{A}$ and a vector $\mathbf{u}$, the pre-image of $\mathbf{A}$ is a vector $\mathbf{r}$ of low norm such that $\mathbf{A} \cdot \mathbf{r} = \mathbf{u} \mod q$. There are two versions of pre-image sampling algorithms:

1. Deterministic algorithm, where the output is of low norm.

   Lemma 2.1.6 (Deterministic pre-image sampling [Bab86]). *There is an efficient deterministic algorithm* Inv *that on input a matrix* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, *its trapdoor* $\mathbf{T_A} \in \mathbb{Z}_q^{m \times m}$, *a vector* $\mathbf{u} \in \mathbb{Z}_q^n$, *outputs a vector* $\mathbf{r} \in \mathbb{Z}_q^m$ *such that* $\mathbf{A} \cdot \mathbf{r} = \mathbf{u} \mod q$ *and* $\|\mathbf{r}\| \leq \frac{1}{2}\sqrt{m} \cdot \|\widetilde{\mathbf{T_A}}\|$.

   We prominently use the algorithm $\mathsf{InvB} : \mathbb{Z}_q^{n \times m} \to \mathbb{Z}_q^{m \times m}$. InvB is a deterministic algorithm having a matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$ and its trapdoor $\mathbf{T_B}$ hardcoded in it. InvB takes as input a matrix $\mathbf{A}$ and outputs a low norm matrix $\mathbf{D}$ such that $\mathbf{B} \cdot \mathbf{D} = \mathbf{A}$. This algorithm essentially runs Babai's algorithm Inv (Lemma 2.1.6) m times, for every column of $\mathbf{A}$. Here, $\|\mathbf{D}\| \leq \frac{1}{2}\sqrt{m} \cdot \|\widetilde{\mathbf{T_B}}\|$.

2. Randomised algorithm, where the output is a random sample from a discrete Gaussian distribution.

Lemma 2.1.7 (Randomised pre-image sampling [GPV08, MP12]). *There is an efficient algorithm* $\mathsf{SampleD}(\mathbf{A}, \mathbf{T_A}, \mathbf{u}, s)$ *that with overwhelming probability over all random choices, does the following: For any* $\mathbf{u} \in \mathbb{Z}_q^n$*, and large enough* $s = \|\widetilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log m})$*, the randomized algorithm* SampleD *outputs a vector* $\mathbf{r} \in \mathbb{Z}^m$ *with norm* $||\mathbf{r}||_\infty \leq ||\mathbf{r}||_2 \leq s\sqrt{n}$ *(with probability* 1*). Furthermore, the following distributions of the tuple* $(\mathbf{A}, \mathbf{T_A}, \mathbf{U}, \mathbf{R})$ *are within* $\mathrm{negl}(n)$ *statistical distance of each other for any polynomial* $k \in \mathbb{N}$*:*

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

Other Sampling algorithms

We will use the following algorithms to sample short vectors from specific lattices. Looking ahead the algorithm SampleLeft [ABB10, CHKP12] will be used to sample keys in the real system, while the algorithm SampleRight [ABB10] will be used to sample keys in the simulation.

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

Inputs: a full rank matrix $\mathbf{A}$ in $\mathbb{Z}_q^{n \times m}$, a "short" basis $\mathbf{T_A}$ of $\Lambda_q^\perp(\mathbf{A})$, a matrix $\mathbf{B}$ in $\mathbb{Z}_q^{n \times m_1}$, a vector $\mathbf{u} \in \mathbb{Z}_q^n$, and a Gaussian parameter α. (2.1)

Output: Let $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$. The algorithm outputs a vector $\mathbf{r} \in \mathbb{Z}^{m+m_1}$ in the coset $\Lambda_q^{\mathbf{u}}(\mathbf{F})$.

Theorem 2.1.8 ([ABB10, Theorem 17], [CHKP12, Lemma 3.2]). *Let* $q > 2$*,* $m > n$ *and* $\alpha > \|\widetilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m + m_1)})$*. Then* $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{B}, \mathbf{T_A}, \mathbf{u}, \alpha)$ *taking inputs as in (2.1) outputs a vector* $\mathbf{r} \in \mathbb{Z}^{m+m_1}$ *distributed statistically close to* $D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}), \alpha}$*, where* $\mathbf{F} := (\mathbf{A} \parallel \mathbf{B})$.

Proof. We would like to output a vector $\mathbf{r} \xleftarrow{\$} D_{\mathbb{Z}^{m+m_1}, \alpha}$ such that $\mathbf{F} \cdot \mathbf{r} = \mathbf{u}$. We do this as follows:

- Sample a random vector $\mathbf{r}_2 \xleftarrow{\$} D_{\mathbb{Z}^{m_1}, \alpha}$.
- Compute the vector $\mathbf{y} = \mathbf{u} - \mathbf{B} \cdot \mathbf{r}_2$.
- Run $\mathsf{SampleD}(\mathbf{A}, \mathbf{T_A}, \mathbf{y}, \alpha)$ and obtain a random vector $\mathbf{r}_1$. This is a random sample from $D_{\mathbb{Z}^m, \alpha}$ by the correctness of the SampleD algorithm, since $\alpha > \|\widetilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log(m + m_1)})$.
- Output $\mathbf{r} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix}$.

Thus, $\mathbf{r} \in \mathbb{Z}^{m+m_1}$ is a distributed statistically close to $D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}), \alpha}$. □

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

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

Often the matrix $\mathbf{R}$ given to the algorithm as input will be a random matrix in $\{1,-1\}^{m\times m}$.

Theorem 2.1.9 ([ABB10, Theorem 19]). *Let $q > 2, m > n$ and $\alpha > \|\widetilde{\mathbf{T_B}}\| \cdot \|\mathbf{R}\| \cdot \omega(\sqrt{\log m})$. Then $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{T_B}, \mathbf{u}, \alpha)$ taking inputs as in (2.2) outputs a vector $\mathbf{r} \in \mathbb{Z}^{k+m}$ distributed statistically close to $D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}),\alpha}$, where $\mathbf{F} := (\mathbf{A} \parallel \mathbf{AR} + \mathbf{B})$.*

Proof. We would like to output a vector $\mathbf{r} \xleftarrow{\$} D_{\mathbb{Z}^{k+m},\alpha}$ such that $\mathbf{F}\cdot\mathbf{r} = \mathbf{u}$.

- First, we find $k+m$ linearly independent vectors $\mathbf{t}_i \in \Lambda_q^{\perp}(\mathbf{F})$, when given a trapdoor $\mathbf{T_B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \in \mathbb{Z}^{m\times m}$ for $\mathbf{B}$, such that $\|\widetilde{\mathbf{T_F}}\| \le \|\widetilde{\mathbf{T_B}}\| \cdot (\|\mathbf{R}\| + 1)$.
- Then we use Lemma 7.1 of [MG02] to convert this set $\mathbf{T_F}$ into a basis $\mathbf{T'_F}$ for $\Lambda_q^{\perp}(\mathbf{F})$ with the same Gram-Schmidt norm.
- Now, we can run $\mathsf{SampleD}(\mathbf{F}, \mathbf{T'_F}, \mathbf{u}, \alpha)$ to obtain $\mathbf{r} \in D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}),\alpha}$, because $\alpha > \|\widetilde{\mathbf{T'_F}}\| \cdot \omega(\log m)$.

Now we explain the first step of obtaining $k+m$ linearly independent vectors $\mathbf{t}_i \in \Lambda_q^{\perp}(\mathbf{F})$, with $\|\widetilde{\mathbf{T_F}}\| \le \|\widetilde{\mathbf{T_B}}\| \cdot (\|\mathbf{R}\| + 1)$ in detail. We are provided with a trapdoor $\mathbf{T_B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \in \mathbb{Z}^{m\times m}$ for $\mathbf{B}$.

- For $i = \{1, \ldots, m\}$, set $\mathbf{t}_i = \begin{bmatrix} -\mathbf{R}\mathbf{b}_i \\ \mathbf{b}_i \end{bmatrix} \in \mathbb{Z}^{k+m}$. Clearly, $\mathbf{F}\cdot\mathbf{t}_i = -\mathbf{ARb}_i + \mathbf{ARb}_i + \mathbf{Bb}_i = \mathbf{0} \mod q$.
- For $i = \{1, \ldots, k\}$, consider $\mathbf{w}_i$ to be the ith column of the identity matrix $\mathbf{I}_k$ and $\mathbf{u}_i$ to be an arbitrary vector in $\mathbb{Z}^m$ satisfying $\mathbf{Aw}_i + \mathbf{Bu}_i = 0 \mod q$. This $\mathbf{u}_i$ exists since rank of $\mathbf{B}$ is n. Now, set
$$\mathbf{t}_{i+m} = \begin{bmatrix} \mathbf{w}_i - \mathbf{R}\mathbf{u}_i \\ \mathbf{u}_i \end{bmatrix}$$
Here, $\mathbf{F}\cdot\mathbf{t}_{i+m} = \mathbf{Aw}_i - \mathbf{ARu}_i + \mathbf{ARu}_i + \mathbf{Bu}_i = \mathbf{Aw}_i + \mathbf{Bu}_i = 0 \mod q$. Hence, $\mathbf{t}_{i+m} \in \Lambda_q^{\perp}(\mathbf{F})$.

We still have to show that all these $k+m$ vectors are linearly independent and that $\|\widetilde{\mathbf{T_F}}\| \le \|\widetilde{\mathbf{T_B}}\| \cdot (\|\mathbf{R}\| + 1)$. The linear independence of the vectors follows from the following discussion:

- The first m vectors span the linear space $\mathbf{V}$ of vectors of the form $\begin{bmatrix} -\mathbf{R}\mathbf{x}_i \\ \mathbf{b}_i \end{bmatrix}$, where the vectors $\mathbf{b}_i$ are linearly independent. Hence, $\mathbf{t}_1, \ldots, \mathbf{t}_m$ are linearly independent.
- For $i \in \{m, \ldots, m+k\}$, each vector $\mathbf{t}_i$ is the sum of a unit vector $\begin{bmatrix} \mathbf{w}_i \\ 0^m \end{bmatrix}$ plus some vector in $\mathbf{V}$. These vectors are clearly linearly independent (within themselves and with the first m vectors), since $\mathbf{w}_i$ are the rows of an identity matrix.

The bound in the Gram-Schmidt norm of $\mathbf{r}$ can be shown as follows:

- For $i > m$, since the vectors are of the form $\begin{bmatrix} \mathbf{w}_i \\ 0^m \end{bmatrix}$ plus a vector in $\mathbf{V}$, the corresponding Gram-Schmidt vectors will be $\begin{bmatrix} \mathbf{w}_i \\ 0^m \end{bmatrix}$. Hence, the norm of these vectors are bounded by 1.

- We apply Lemma 6 in [ABB10] to bound the Gram-Schmidt norm of the vectors for $i \in \{1, \ldots, m\}$ and hence for $\|\widetilde{\mathbf{T_F}}\|$. According to that lemma, for any $\mathbf{R} \in \mathbb{R}^{k \times m}$, for any linearly independent set of vectors $\mathbf{T_B} = \{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \in \mathbb{R}^{m \times m'}$,

$$\|\widetilde{\mathbf{RT_B}}\| \leq \max_{1 \leq i \leq m'} \|\mathbf{R}\widetilde{\mathbf{b}}_i\|$$

Since the trapdoor $\mathbf{T_B}$ is a basis for $\Lambda_q^{\perp}(\mathbf{B})$, we can apply this lemma to get an upper bound for $\|\widetilde{\mathbf{T_F}}\|$. Let $\mathbf{R}' = \begin{bmatrix} -\mathbf{R} \\ \mathbf{I}_m \end{bmatrix} \in \mathbb{Z}^{(k+m) \times m}$. Then,

$$\begin{aligned} \|\widetilde{\mathbf{T_F}}\| &\leq \max_{1 \leq i \leq m} \|\mathbf{R}'\widetilde{\mathbf{b}}_i\| \leq \max_{1 \leq i \leq m} \|\mathbf{R}\widetilde{\mathbf{b}}_i\| + \|\widetilde{\mathbf{b}}_i\| \leq \max_{1 \leq i \leq m} \|\mathbf{R}\|\|\widetilde{\mathbf{b}}_i\| + \|\widetilde{\mathbf{b}}_i\| \\ &\leq \max_{1 \leq i \leq m} \|\widetilde{\mathbf{b}}_i\| \cdot (\|\mathbf{R}\| + 1) \leq \|\widetilde{\mathbf{T_B}}\| \cdot (\|\mathbf{R}\| + 1) \end{aligned}$$

Thus, $\mathbf{T_F}$ is a set of $k + m$ linearly independent vectors in $\Lambda_q^{\perp}(\mathbf{F})$ with its Gram-Schmidt norm upper bounded by $\|\widetilde{\mathbf{T_B}}\| \cdot (\|\mathbf{R}\| + 1)$. We can now perform the next two steps to obtain a random vector $\mathbf{r} \in D_{\Lambda_q^{\mathbf{u}}(\mathbf{F}),\alpha}$.

□

Primitive matrices

A matrix is termed as a primitive matrix, if its columns generate all of $\mathbb{Z}_q^n$. As shown in [MP12], there exists primitive matrices which define lattices with excellent properties. One such use is that, one can easily find its trapdoor, which is a short basis for its kernel matrix.

Here, for instance, consider the following matrix,

$$\mathbf{G} = \mathsf{Powersof2}(\mathbf{I}_n) = \begin{bmatrix} \mathbf{g} & & & \\ & \mathbf{g} & & \\ & & \ddots & \\ & & & \mathbf{g} \end{bmatrix} \in \mathbb{Z}_q^{n \times ny}$$

where $\mathbf{g} = \mathsf{Powersof2}(1) = \begin{bmatrix} 1 & 2 & \ldots & 2^{y-1} \end{bmatrix} \in \mathbb{Z}_q^{1 \times y}$. This matrix $\mathbf{G}$ satisfies the properties of a *primitive* matrix. Its trapdoor is

$$\mathbf{T_G} = \begin{bmatrix} \mathbf{T}_t & & & \\ & \mathbf{T}_t & & \\ & & \ddots & \\ & & & \mathbf{T}_t \end{bmatrix} \in \mathbb{Z}_q^{ny \times ny}$$

where each $\mathbf{T}_t = \begin{bmatrix} 2 & & & & q_0 \\ -1 & 2 & & & q_1 \\ & -1 & & & q_2 \\ & & \ddots & & \vdots \\ & & & 2 & q_{y-2} \\ & & & -1 & q_{y-1} \end{bmatrix} \in \mathbb{Z}_q^{y \times y}$.

We summarise the properties that we require of the primitive matrices in the following theorem.

Theorem 2.1.10 ([MP12], Theorem 4.1)**.** *For any integers $q \geq 2, n \geq 1, y = \lfloor \log q \rfloor + 1$ and $m = ny$, there is a primitive matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ such that the lattice $\Lambda_q^{\perp}(\mathbf{G})$ has a known basis $\mathbf{T_G} \in \mathbb{Z}^{m \times m}$ with $\|\widetilde{\mathbf{T_G}}\| \leq \sqrt{5}$ and $\|\mathbf{T_G}\| \leq \max\{\sqrt{5}, \sqrt{y}\}$. If $q = 2^y$, we have $\|\mathbf{T_G}\| = \sqrt{5}$ and $\widetilde{\mathbf{T_G}} = 2\mathbf{I}_m$ and hence $\|\widetilde{\mathbf{T_G}}\| = 2$.*

Also, for $m > ny$, we can append columns of zeros (0^n) to obtain primitive matrices $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$. Note that, the Gram-Schmidt norms of the trapdoors of these matrices are still upper bounded by $\sqrt{5}$ i.e $\|\widetilde{\mathbf{T_B}}\| \leq \sqrt{5}$.

2.2 Attribute-Based Encryption

We define attribute-based encryption, following [GPSW06]. An attribute-based encryption scheme $\mathcal{ABE}$ for a class of boolean predicate circuits $\mathcal{C}$[1] consists of four polynomial time algorithms $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Keygen}, \mathsf{Dec})$. For some security parameter λ, for some input length $\ell = \text{poly}(\lambda)$ for circuits in $\mathcal{C}$, for some bounded depth $d_{\max}$ for circuits in $\mathcal{C}$, the ABE algorithms are defined as follows:

$\mathsf{Setup}(1^\lambda, 1^{d_{\max}}, 1^\ell) \to (\mathsf{pp}, \mathsf{mpk}, \mathsf{msk})$: The probabilistic setup algorithm gets as input $\lambda, d_{\max}, \ell$ and outputs the public parameters pp, the master public key mpk, and the master key msk. pp is implicitly given to all the other algorithms of the scheme.

$\mathsf{Enc}(\mathsf{mpk}, x, \mu) \to \mathsf{ct}_x$: The probabilistic encryption algorithm gets as input mpk, an attribute vector $x \in \{0,1\}^\ell$ and a message $\mu \in \{0,1\}$. It outputs a ciphertext $\mathsf{ct}_\mathbf{x}$.

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

$\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_\mathbf{x}) \to \mu$: The deterministic decryption algorithm gets as input $\mathsf{ct}_\mathbf{x}$ and sk_C, and outputs either $\perp$ or a message $\mu \in \{0,1\}$.

Definition 2.2.1 (Correctness)**.** *We require that for all λ, for all circuits $C \in \mathcal{C}$ of bounded depth $d_{\max}$ and input length ℓ, for all $\mathbf{x} \in \{0,1\}^\ell$ and for all $\mu \in \{0,1\}$, if $C(\mathbf{x}) = 1$ we have $\Pr[\mathsf{ct}_\mathbf{x} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu); \mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_\mathbf{x}) = \mu)] = 1$ where the probability is taken over $(\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{d_{\max}} 1^\ell)$ and the coins of all the algorithms in the expression above.*

[1] Predicate circuits are those with single bit outputs.

2.2.1 Security of Attribute-based Encryption

Informally, the security of attribute-based encryption ensures the following. When an adversary is provided with a ciphertext $\mathsf{ct}_{\mathbf{x}}$, corresponding to a vector $\mathbf{x}$, it cannot learn anything about the message μ encrypted in $\mathsf{ct}_{\mathbf{x}}$, unless it has the secret key sk_C for a circuit C such that $C(\mathbf{x}) = 1$. We provide an indistinguishability based definition which ensures that the adversary cannot distinguish between encryptions of μ_0 and μ_1, without having an appropriate secret key.

Definition 2.2.2 (Fully Secure ABE)**.** *For a stateful adversary $\mathcal{A}$, we define the advantage function $\mathsf{Adv}^{\mathrm{ABE}}_{\mathcal{A}}(\lambda)$ to be*

$$\Pr\left[b = b' : \begin{array}{l}(d_{\max}, \ell) \leftarrow \mathcal{A}(\lambda);\\ (\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{d_{\max}}, 1^\ell);\\ \mathbf{x} \leftarrow \mathcal{A}^{\mathsf{Keygen}(\mathsf{msk},\cdot)}(\mathsf{mpk});\\ b \xleftarrow{\$} \{0,1\};\\ \mathsf{ct}_{\mathbf{x}} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, b);\\ b' \leftarrow \mathcal{A}^{\mathsf{Keygen}(\mathsf{msk},\cdot)}(\mathsf{ct}_{\mathbf{x}})\end{array}\right] - \frac{1}{2}$$

with the restriction that all the secret key queries that $\mathcal{A}$ makes to $\mathsf{Keygen}(\mathsf{msk}, \cdot)$ for circuits C are of depth bounded by $d_{\max}$, input length ℓ and satisfy $C(\mathbf{x}) = 0$ (that is, sk_C does not decrypt $\mathsf{ct}_{\mathbf{x}}$). We call an attribute-based encryption scheme fully secure *if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathrm{ABE}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.*

We will use a weaker version of security called "selective" security. Here, the adversary should provide the challenge attribute vector $\mathbf{x}^*$ at the beginning of the experiment. The Setup algorithm produces the keys based on the challenge attribute vector $\mathbf{x}^*$.

Definition 2.2.3 (Selectively Secure ABE)**.** *For a stateful adversary $\mathcal{A}$, we define the advantage function $\mathsf{Adv}^{\mathrm{ABE}}_{\mathcal{A}}(\lambda)$ to be*

$$\Pr\left[b = b' : \begin{array}{l}(\mathbf{x}^*, d_{\max}, \ell) \leftarrow \mathcal{A}(1^\lambda);\\ (\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{d_{\max}}, 1^\ell, \mathbf{x}^*);\\ \mathcal{A}^{\mathsf{Keygen}(\mathsf{msk},\cdot)}(\mathsf{mpk});\\ b \xleftarrow{\$} \{0,1\};\\ \mathsf{ct}_{\mathbf{x}^*} \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}^*, b);\\ b' \leftarrow \mathcal{A}^{\mathsf{Keygen}(\mathsf{msk},\cdot)}(\mathsf{ct}_{\mathbf{x}^*})\end{array}\right] - \frac{1}{2}$$

with the same restriction that all the secret key queries that $\mathcal{A}$ makes to $\mathsf{Keygen}(\mathsf{msk}, \cdot)$ for circuits C are of depth bounded by $d_{\max}$, input length ℓ and satisfy $C(\mathbf{x}^*) = 0$ (that is, sk_C does not decrypt $\mathsf{ct}_{\mathbf{x}^*}$). Here, $\mathbf{x}^* \in \{0,1\}^\ell$. An attribute-based encryption scheme is selectively secure *if for all PPT adversaries $\mathcal{A}$, the advantage $\mathsf{Adv}^{\mathrm{ABE}}_{\mathcal{A}}(\lambda)$ is a negligible function in λ.*

Note: Our results can be easily extended to a more general message space $\{0,1\}^{\mathrm{poly}(\lambda)}$.

Any selectively secure ABE can be made adaptively secure by a standard complexity leveraging argument as in [BB11] who give one such transformation for ID-based encryption schemes.

Theorem 2.2.1. *Any selectively secure attribute-based encryption scheme with advantage ϵ is also adaptively secure with advantage $2^\ell\epsilon$, where ℓ is the length of the attribute vector*

Proof. This proof closely follows the proof of [BB11]. First, assume the existence of an adversary $\mathcal{A}$ which has an advantage of atleast $2^\ell\epsilon$ in the full security game of an ABE scheme. We will construct an adversary $\mathcal{B}$ using $\mathcal{A}$ for the selective security game of the same ABE scheme which has an advantage of atleast ϵ. $\mathcal{B}$ acts as the challenger to $\mathcal{A}$ in the adaptive security game.

- $\mathcal{B}$ chooses an attribute vector $\mathbf{x}^* \xleftarrow{\$} \{0,1\}^\ell$ and gives it to its challenger as its challenge attribute vector. $\mathcal{B}$ then receives public parameters of the scheme from its challenger. Note that the distribution of the public parameters is independent of the challenge attribute vector $\mathbf{x}^*$. $\mathcal{B}$ forwards these public parameters to $\mathcal{A}$.

- During the training phase I, whenever $\mathcal{A}$ queries the Keygen oracle, $\mathcal{B}$ answers them by querying its Keygen oracle except when $\mathcal{A}$ queries for circuits C such that $C(\mathbf{x}^*) = 1$. When $\mathcal{A}$ queries for such circuits, $\mathcal{B}$ aborts the game, and outputs b' at random.

- During the challenge phase, $\mathcal{A}$ outputs a challenge attribute vector $\mathbf{x}^*_A$ and a pair of messages μ_0, μ_1. If $\mathbf{x}^*_A \neq \mathbf{x}^*$, then $\mathcal{B}$ aborts again outputting b' at random.

- If $\mathbf{x}^*_A = \mathbf{x}^*$, $\mathcal{B}$ gives the messages μ_0, μ_1 to its challenger and proceeds to the training phase II. $\mathcal{B}$ answers $\mathcal{A}$'s Keygen queries as before by forwarding the queries to its challenger. Note that there will be no abort during this stage since all queries of $\mathcal{A}$ will be such that $C(\mathbf{x}^*_A) = C(\mathbf{x}^*) = 0$.

- Finally, $\mathcal{A}$ outputs its guess $b'_A \in \{0,1\}$. $\mathcal{B}$ outputs the same bit as its guess b'.

We now analyse the advantage of $\mathcal{B}$ in the above selective security game i.e $\mathcal{B}$'s chances of outputting the correct bit $b' = b$. Let $q \leq q_A < 2^\ell$ be the number of distinct Keygen queries that $\mathcal{A}$ makes during the training phase I. Let $X_1 \subseteq \{0,1\}^\ell$ be the set of $\mathbf{x} \in \{0,1\}^\ell$ such that $C_1(\mathbf{x}) = 0$ for $\mathcal{A}$'s first query C_1. Thus the probability for not aborting after the first query is $\frac{\alpha}{2^\ell}$, where $\alpha_1 = |X_1|$. Now, for any $i \in [2, q]$, let $X_i \subseteq X_{i-1}$ be the set of $\mathbf{x} \in \{0,1\}^\ell$ for which $C_j(\mathbf{x}) = 0$ for all $j \in [1, i]$. If S_i is the event that there is no abort after i queries, then the probability $\Pr[S_i|S_{i-1}] = \frac{\alpha_i}{\alpha_{i-1}}$, where $\alpha_i = |X_i|$. Thus, the probability that there is no abort in the training phase I is $\Pr[S_q]$ where

$$\begin{aligned}
\Pr[S_q] &= \Pr[S_q|S_{q-1}] \cdot \Pr[S_{q-1}] \\
&= \Pr[S_q|S_{q-1}] \cdot \Pr[S_{q-1}|S_{q-2}] \cdots \Pr[S_2|S_1] \cdot \Pr[S_1] \\
&= \frac{\alpha_q}{\alpha_{q-1}} \cdot \frac{\alpha_{q-1}}{\alpha_{q-2}} \cdots \frac{\alpha_2}{\alpha_1} \cdot \frac{\alpha_1}{2^\ell} \\
&= \frac{\alpha_q}{2^\ell}
\end{aligned}$$

Note that $\alpha_q \geq 1$ because the challenge attribute vector $\mathbf{x}^*_A$ given by $\mathcal{A}$ should be such that $C_i(\mathbf{x}^*_A) = 0$ for all $i \in [1, q]$. Therefore, the probability for no abort during the above game is

$$\Pr[NA] = \Pr[S_q \cap (\mathbf{x}^*_A = \mathbf{x}^*)] = \Pr[S_q] \cdot \Pr[\mathbf{x}^*_A = \mathbf{x}^*|S_q] = \frac{\alpha_q}{2^\ell} \cdot \frac{1}{\alpha_q} = \frac{1}{2^\ell}$$

When no abort happens during the above game, the view of $\mathcal{A}$ is identical to the view in the real security game. And, by our assumption $\Pr[b' = b|NA] - \frac{1}{2} \geq 2^\ell\epsilon$. When abort happens during the above game,

$\mathcal{B}$ always chooses b' at random. Hence, $\Pr[b' = b|\overline{NA}] = \frac{1}{2}$. Thus, the advantage of $\mathcal{B}$ in the selective security game is given by

$$\begin{aligned}
\big|\Pr[b' = b] - \frac{1}{2}\big| &= \big|\Pr[b' = b|NA] \cdot \Pr[NA]\big| + \Pr[b' = b|\overline{NA}] \cdot \Pr[\overline{NA}] - \frac{1}{2}\big| \\
&= \big|\Pr[b' = b|NA] \cdot \frac{1}{2^\ell} + \frac{1}{2} \cdot (1 - \frac{1}{2^\ell}) - \frac{1}{2}\big| \\
&= \big|\Pr[b' = b|NA] - \frac{1}{2}\big| \cdot \frac{1}{2^\ell} \\
&\geq \epsilon
\end{aligned}$$

□

Chapter 3

Fully Key homomorphic encryption

In this chapter, we present a formal definition of our variant of fully key homomorphic encryption (FKHE) scheme. We then use it to build an ABE scheme with short secret keys. We define our variant in such a way that it would be easy to build the ABE scheme using the FKHE algorithms as black boxes.

A naive, informal definition of key homomorphism would be: anyone can transform an encryption under a vector of ℓ public keys $\mathbf{x} \in \mathcal{X}^\ell$ into an encryption under the public key $C(\mathbf{x})$ for some arbitrary circuit C. In other words, if $\mathbf{c_x}$ is an encryption of message μ under a public key vector $\mathbf{x} \in \mathcal{X}^\ell$, with key-homomorphism we would expect to have a public evaluation algorithm $\mathsf{Eval}_{\mathrm{ct}}(C, \mathbf{x}, \mathbf{c_x}) \to \mathbf{c}_C$ such that the output (evaluated) ciphertext $\mathbf{c}_C$ is an encryption of μ under the public key $C(\mathbf{x}) \in \mathcal{Y}$[1]. But when one tries to build an ABE system from this key homomorphic encryption scheme as such, it becomes insecure. To see this, consider the scenario where a user, with a policy represented by circuit C, has been given a secret key corresponding to the public key $y \in \mathcal{Y}$. When given a ciphertext $\mathbf{c_x}$ under the public key vector $\mathbf{x}$, he can run the $\mathsf{Eval}_{\mathrm{ct}}$ algorithm and then use the secret key to decrypt the evaluated ciphertext $\mathbf{c}_C$ if $C(\mathbf{x}) = y$. Hence, correctness holds. But, even when $C(\mathbf{x}) \neq y$, he can pick any other circuit C' such that $C'(\mathbf{x}) = y$ and run $\mathsf{Eval}_{\mathrm{ct}}(C', \mathbf{x}, \mathbf{c_x})$ to get $\mathbf{c}_{C'}$ and use the secret key to decrypt $\mathbf{c}_{C'}$ to get the message. He can do this because $\mathbf{c}_{C'}$ is also a valid encryption under the public key y. The problem here is that the secret key is bound only to the evaluated pubic key $C(\mathbf{x}) = y$ and not to the circuit C. Hence, to construct a secure ABE we slightly extend the basic key homomorphism idea to include C in the (evaluated) public key. Thus, a 'base' encryption public key is a vector $\mathbf{x} \in \mathcal{X}^\ell$ as before, however $\mathsf{Eval}_{\mathrm{ct}}$ produces ciphertexts encrypted under the (evaluated) public key $(C(\mathbf{x}), C)$ where $C(\mathbf{x}) \in \mathcal{Y}$. To be a bit more precise we define our FKHE variant such that the vector $\mathbf{x}$ is not the actual public key vector. Instead we use each x_i to choose a public key pk_{i,x_i} from the master public key $\mathsf{mpk}_{\mathsf{FKHE}}$. And the key-homomorphism property ensures the following: for any two wires $i, j \in [1, |C|]$ having the public keys pk_{i,x_i} and pk_{j,x_j} allows one to compute pk_{k,x_k} without requiring any secret components. Here $x_k = x_i \odot x_j$ for some boolean gate $\odot$ with input wires i, j and output wire k. Thus there is a deterministic way to publicly determine the (evaluated) public key corresponding to $(C(\mathbf{x}), C)$ from the public keys determined by $\mathbf{x}$.

[1] It is convenient to group the ciphertexts under a public key vector together, because to achieve key homomorphism and maintain semantic security simultaneously, the individual ciphertexts have to be encrypted using "correlated" randomness. Also, it is more appropriate to denote the evaluated ciphertext as $\mathbf{c}_{\mathbf{x},C}$ or $\mathbf{c}_{y,C}$. But we will simplify the notation and use $\mathbf{c}_C$.

Our variant of key homomorphism can now described as follows. Suppose $\mathbf{c_x}$ is an encryption of μ under the vector of public keys determined by $\mathbf{x} \in \mathcal{X}^\ell$, anyone can transform $\mathbf{c_x}$ to an encryption of μ under the public-key determined by $(C(\mathbf{x}), C)$. Transforming the ciphertext $\mathbf{c_x}$ from the public keys determined by $\mathbf{x}$ to that determined by $(C(\mathbf{x}), C)$ is done using algorithm $\mathsf{Eval}_{\mathrm{ct}}(C, \mathbf{x}, \mathbf{c_x}) \to \mathbf{c}_C$. Now, in order to decrypt, we give the user the secret key for the public key corresponding to (y, C), ensuring that the user with $\mathsf{sk}_{y,C}$ can decrypt $\mathbf{c}_C$ only when $C(\mathbf{x}) = y$.

Now, to build an ABE from this key homomorphic encryption scheme, we simply publish the parameters of the key homomorphic system. We can encrypt a message μ under the attribute vector $\mathbf{x} = (x_1, \ldots, x_\ell) \in \mathcal{X}^\ell$ that determines the 'base' public keys and obtain the ciphertext $\mathbf{c_x}$. Now a user with his policy represented by a circuit C can transform $\mathbf{c_x}$ into an encryption $\mathbf{c}_C$ of μ under the public key determined by $(C(\mathbf{x}), C)$ using the $\mathsf{Eval}_{\mathrm{ct}}$ algorithm. The point is that now the secret key for the circuit C can simply be the decryption key for the public key determined by $(1, C)$. This key enables the decryption of $\mathbf{c}_C$ when $C(\mathbf{x}) = 1$.

In our construction we deal with the class $\mathcal{C}$ of predicate boolean circuits with ℓ input wires. Hence we set $\mathcal{X} = \{0, 1\}$ and $\mathcal{Y} = \{0, 1\}$. This can be trivially extended to a construction with $\mathcal{X} = \mathcal{Y} = \mathbb{Z}_p$ for any bounded p and hence $\mathcal{C}$ would be the set of arithmetic circuits with ℓ input wires.

Remark: We have defined this notion of FKHE to abstract the techniques that we used for building ABE with short keys so that more applications could be found for our techniques or in general, full key-homomorphism. We leave it to future work to study the notion of full key-homomorphism and its applications in more detail.

3.1 Definition

Now, we give a formal definition of our fully key homomorphic encryption (FKHE) system.

Let $\mathcal{C}$ be a class of boolean predicate circuits, namely $\mathcal{C} = \{C : \{0,1\}^\ell \to \{0,1\}\}$ for some $\ell > 0$. Public keys in an FKHE scheme are pairs $(y, C) \in \{0, 1\} \times \mathcal{C}$. We call y the "value" and C the associated circuit. All such pairs are valid public keys. We also allow tuples $\mathbf{x} \in \{0,1\}^\ell$ to function as public keys. Now, an FKHE scheme for $\mathcal{C}$ consists of five PPT algorithms:

$\mathsf{Setup}_{\mathsf{FKHE}}(1^\lambda, 1^\ell) \to (\mathsf{mpk}_{\mathsf{FKHE}}, \mathsf{msk}_{\mathsf{FKHE}})$: The probabilistic Setup algorithm on input the security parameter λ and input length of circuits ℓ, outputs a master secret key $\mathsf{msk}_{\mathsf{FKHE}}$ and public parameters $\mathsf{mpk}_{\mathsf{FKHE}}$.

$\mathsf{KeyGen}_{\mathsf{FKHE}}\big(\mathsf{msk}_{\mathsf{FKHE}},\ (y, C)\big) \to \mathsf{sk}_{y,C}$: The probabilistic key generation algorithm on input a value, circuit pair $(y, C) \in \{0, 1\} \times \mathcal{C}$ outputs a decryption key $\mathsf{sk}_{y,C}$ for the public key determined by (y, C). Note that $\mathsf{sk}_{y,C}$ contains (y, C).

$\mathsf{E}_{\mathsf{FKHE}}\big(\mathsf{mpk}_{\mathsf{FKHE}},\ \mathbf{x},\ \mu\big) \longrightarrow \mathbf{c_x}$: The randomised encryption algorithm on input a vector $\mathbf{x} \in \{0,1\}^\ell$ and a message $\mu \in \{0, 1\}$, creates a ciphertext $\mathbf{c_x}$ which is an encryption of μ under the public keys determined by $\mathbf{x}$. Note that $\mathbf{c_x}$ contains $\mathbf{x}$.

$\mathsf{Eval}_{\mathbf{ct}}\big(C,\ \mathbf{x},\ \mathbf{c_x}\big) \longrightarrow \mathbf{c}_C$ [2] : A deterministic evaluation algorithm that implements key-homomorphism. Let $\mathbf{c_x}$ be an encryption of message μ under the public keys determined by $\mathbf{x} \in \{0,1\}^\ell$. For a circuit C the algorithm outputs the evaluated ciphertext $\mathbf{c}_C$ where if $y = C(x_1, \ldots, x_\ell)$ then $\mathbf{c}_C$ is an encryption of message μ under the (evaluated) public-key determined by (y, C).

$\mathsf{D}_{\mathsf{FKHE}}(\mathsf{sk}_{y,C},\ \mathbf{c}_{C'})$: The decryption algorithm on input a ciphertext $\mathbf{c}_{C'}$ with a secret key $\mathsf{sk}_{y,C}$ and outputs a message $\mu \in \{0,1\}$ or $\perp$.

Algorithm $\mathsf{Eval}_{\mathrm{ct}}$ captures the key-homomorphic property of the system: ciphertext $\mathbf{c_x}$ encrypted with the public keys determined by $\mathbf{x} = (x_1, \ldots, x_\ell)$ is transformed to a ciphertext $\mathbf{c}_C$ encrypted under the public key determined by $\big(C(x_1, \ldots, x_\ell),\ C\big)$.

Correctness. The key-homomorphic property is stated formally in the following requirement: For all $(\mathsf{mpk}_{\mathsf{FKHE}}, \mathsf{msk}_{\mathsf{FKHE}})$ output by $\mathsf{Setup}_{\mathsf{FKHE}}(1^\lambda, 1^\ell)$, all messages $\mu \in \{0,1\}$, all $C \in \mathcal{C}$, and $\mathbf{x} = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ where ℓ is the input length of C:

If $\quad \mathbf{c_x} \leftarrow \mathsf{E}_{\mathsf{FKHE}}\big(\mathsf{mpk}_{\mathsf{FKHE}},\ \mathbf{x},\ \mu\big), \quad y = C(x_1, \ldots, x_\ell),$

$\mathbf{c}_C = \mathsf{Eval}_{\mathrm{ct}}\big(C,\ \mathbf{x},\ \mathbf{c_x}\big), \quad \mathsf{sk}_{y,C} \leftarrow \mathsf{KeyGen}_{\mathsf{FKHE}}(\mathsf{msk}_{\mathsf{FKHE}}, (y, C))$

Then $\mathsf{D}_{\mathsf{FKHE}}(\mathsf{sk}_{y,C}, \mathbf{c}_C) = \mu$.

Another version of FKHE definition

The above definition is more general for an FKHE scheme, but we can obtain a construction which works only for a variant of the above version. In particular, similar to our definition of ABE, we introduce a depth bound. The $\mathsf{Setup}_{\mathsf{FKHE}}$ algortihm can be modified as follows to accommodate this. We will use this version for our construction.

$\mathsf{Setup}_{\mathsf{FKHE}}(1^\lambda, 1^{d_{\max}}, 1^\ell) \rightarrow (\mathsf{mpk}_{\mathsf{FKHE}}, \mathsf{msk}_{\mathsf{FKHE}})$: The probabilistic Setup algorithm on input the security parameter λ, the input length ℓ and the depth bound $d_{\max}$ of the circuits in $\mathcal{C}$, outputs a master secret key $\mathsf{msk}_{\mathsf{FKHE}}$ and public parameters $\mathsf{mpk}_{\mathsf{FKHE}}$.

All the other algorithms are the same as in the above definition.

3.2 ABE from FKHE

A FKHE for a family of predicate boolean circuits $\mathcal{C}$ immediately gives a key-policy ABE scheme for the same family of circuits. Attribute vectors for the ABE are ℓ-tuples over $\{0,1\}$ and the supported key-policies are circuits in $\mathcal{C}$. Note that there is no apriori depth bound $d_{\max}$ unless the FKHE scheme has similar constraints on the circuits that it can support. The ABE system works as follows:

- $\mathsf{Setup}(1^\lambda, 1^{d_{\max}}, 1^\ell)$:
  1. Run $\mathsf{Setup}_{\mathsf{FKHE}}(1^\lambda, 1^{d_{\max}}, 1^\ell)$ to get public parameters $\mathsf{mpk}_{\mathsf{FKHE}}$ and master secret $\mathsf{msk}_{\mathsf{FKHE}}$.
  2. These respectively function as the ABE master public key mpk and the master secret key msk.

[2] We use a subscript of ct with $\mathsf{Eval}_{\mathrm{ct}}$ because later we are going to use two more evaluation algorithms as a part of our FKHE construction, to improve the presentation of the construction.

- $\mathsf{Enc}(\mathsf{mpk},\ \mathbf{x},\ \mu)$:

  1. Run the $\mathsf{E}_{\mathsf{FKHE}}(\mathsf{mpk}_{\mathsf{FKHE}},\ \mathbf{x},\ \mu)$ algorithm to get the ciphertext $\mathbf{c}_{\mathbf{x}}$ under the attribute vector $\mathbf{x}$ (which is used to determine the public keys for the FKHE encryption).
  2. Output $\mathsf{ct}_{\mathbf{x}} = \mathbf{c}_{\mathbf{x}}$.

- $\mathsf{Keygen}(\mathsf{msk}, C)$:

  1. Run the $\mathsf{KeyGen}_{\mathsf{FKHE}}\big(\mathsf{msk}_{\mathsf{FKHE}},\ (1, C)\big)$ algorithm to get the secret key $\mathsf{sk}_{1,C}$. This is the secret key sk_C of the ABE scheme for the circuit C since by our definition of ABE we allow decryption only when $C(\mathbf{x}) = 1$. Output sk_C.
  2. Jumping ahead, we remark that in our FKHE instantiation, the size of sk_C does not depend on the size of C. (Actually our FKHE implementation will also need an apriori depth bound $d_{\max}$, and hence the size of secret key will depend only on the (max) depth but not on the size of the circuit.)

- $\mathsf{Dec}\big(\mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_C\big)$:

  1. If $C(\mathbf{x}) = 0$, output $\perp$.
  2. If $C(\mathbf{x}) = 1$, proceed as follows: run the $\mathsf{Eval}_{\mathrm{ct}}\big(C,\ \mathbf{x},\ \mathsf{ct}_{\mathbf{x}}\big)$ to get the (evaluated) ciphertext $\mathbf{c}_C$.
  3. Now run the decryption algorithm $\mathsf{D}_{\mathsf{FKHE}}(\mathsf{sk}_C, \mathbf{c}_C)$ and output the decrypted answer.
  4. Note that $\mathbf{c}_C$ is the encryption of the message under the public key determined by $(C(\mathbf{x}),\ C)$. Since sk_C is the decryption key for the public key determined by $(1, C)$, decryption will succeed whenever $C(\mathbf{x}) = 1$ as required.

Lemma 3.2.1. *The above ABE scheme is correct, given that the FKHE scheme is correct.*

Proof. The public parameters and the master secret key of the ABE system are those of the underlying FKHE system. Now we need to show that for all ciphertexts $\mathsf{ct}_{\mathbf{x}}$ encrypted under an attribute vector $\mathbf{x}$, any secret key sk_C such that $C(\mathbf{x}) = 1$ should be able to decrypt $\mathsf{ct}_{\mathbf{x}}$. This follows from the correctness of the FKHE scheme since the secret key sk_C is the secret key $\mathsf{sk}_{1,C}$ corresponding to the evaluated public key determined by $(1, C)$ and this decrypts any ciphertext under the set of public keys determined by $\mathbf{x}$ such that $C(\mathbf{x}) = 1$. □

3.3 Security of FKHE systems

Our security definition of FKHE is as follows:

Definition 3.3.1 (Selectively-secure FKHE)**.** *For a stateful adversary $\mathcal{A}$ we define the advantage function for a fully key homomorphic encryption scheme $\mathcal{FKHE} = (\mathsf{Setup}_{\mathsf{FKHE}}, \mathsf{KeyGen}_{\mathsf{FKHE}}, \mathsf{E}_{\mathsf{FKHE}}, \mathsf{Eval}_{ct})$*

for a class of circuits $\mathcal{C} = \{C : \{0,1\}^\ell \to \{0,1\}\}$ *we define the advantage function* $\mathrm{Adv}^{FKHE}_{\mathcal{A}}(\lambda)$ *as follows:*

$$\Pr\left[b = b' : \begin{array}{l} \mathbf{x}^* \leftarrow \mathcal{A}(\lambda, 1^{d_{\max}}, 1^\ell) \\ (\mathsf{mpk}_{\mathsf{FKHE}}, \mathsf{msk}_{\mathsf{FKHE}}) \leftarrow \mathsf{Setup}_{\mathsf{FKHE}}(\lambda) \\ \mathcal{A}^{\mathsf{KG}_{\mathsf{KH}}(\mathsf{msk}_{\mathsf{FKHE}}, x^*, \cdot, \cdot)}(\mathsf{mpk}_{\mathsf{FKHE}}) \\ b \xleftarrow{\$} \{0,1\}; \\ \mathbf{c}_{\mathbf{x}^*} \leftarrow \mathsf{E}_{\mathsf{FKHE}}(\mathsf{mpk}_{\mathsf{FKHE}},\ \mathbf{x}^*,\ b) \\ b' \leftarrow \mathcal{A}^{\mathsf{KG}_{\mathsf{KH}}(\mathsf{msk}_{\mathsf{FKHE}}, x^*, \cdot, \cdot)}(\mathbf{c}_{\mathbf{x}^*}) \end{array}\right] - \frac{1}{2}$$

where $\mathbf{x}^* \in \{0,1\}^\ell$ *and* $\mathsf{KG}_{\mathsf{KH}}(\mathsf{msk}_{\mathsf{FKHE}}, x^*, y, C)$ *is an oracle that on input* $C \in \mathcal{C}$ *and* $y \in \{0,1\}$*, returns* $\perp$ *whenever* $C(\mathbf{x}^*) = y$*, and otherwise returns* $\mathsf{KeyGen}_{\mathsf{FKHE}}\big(\mathsf{msk}_{\mathsf{FKHE}}, (y, C)\big)$*. The FKHE scheme* $\mathcal{FKHE}$ *is* selectively secure *if for all PPT adversaries* $\mathcal{A}$*,* $\mathsf{Adv}^{FKHE}_{\mathcal{A}}(\lambda)$ *is a negligible function in* λ*.*

With Definition 3.3.1 the following theorem is now immediate.

Theorem 3.3.1. *The ABE system above is selectively secure provided the underlying FKHE is selectively secure.*

Proof. Assume that we have an adversary $\mathcal{A}$ which breaks the selectively security of the ABE system with advantage ϵ. We will then build an adversary $\mathcal{B}$ who would have the same advantage during the selective security game for the FKHE scheme.

- $\mathcal{A}$ gives its challenge attribute vector $\mathbf{x}^*$ to $\mathcal{B}$ and $\mathcal{B}$ forwards this to its challenger.
- $\mathcal{B}$ gets the public parameters of the FKHE scheme from its challenger. $\mathcal{B}$ forwards these as the public parameters of the ABE scheme to $\mathcal{A}$.
- Whenever $\mathcal{A}$ asks for secret key queries for any circuit C, $\mathcal{B}$ answers by querying the $\mathsf{KG}_{\mathsf{KH}}$ with the evaluated public key $(1, C)$. $\mathcal{B}$ could always do this because it is allowed to discard the queries of $\mathcal{A}$ whenever $C(\mathbf{x}^*) = 1$.
- $\mathcal{B}$ continues answering $\mathcal{A}$'s Keygen queries in the same way as before.
- When $\mathcal{B}$ gets its challenge ciphertext, it forwards it to $\mathcal{A}$.
- Finally, when $\mathcal{A}$ outputs its guess b', $\mathcal{B}$ outputs the same bit b' as its guess.

From the game, it is obvious that if $\mathcal{A}$ breaks the selective security of ABE with advantage ϵ, $\mathcal{B}$ has the same advantage ϵ in breaking the selective security of the FKHE scheme. □

Chapter 4

An FKHE scheme under LWE assumption

We now instantiate our FKHE system using the LWE assumption. Recall that the (decisional) LWE assumption states that the vector $\mathbf{As} + \mathbf{e} \pmod q$ for a matrix $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, a secret $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, an error $\mathbf{e} \xleftarrow{\$} \chi^m$ is indistinguishable from a randomly chosen vector in $\mathbb{Z}_q^m$. We will first give an overview of our techniques before giving a complete description of the FKHE algorithms.

Fix some $n \in \mathbb{Z}^+$, prime q, and $m = \Theta(n \log q)$. Let $\mathbf{A}, \mathbf{A}_1, \ldots, \mathbf{A}_\ell$ be matrices sampled from $\mathbb{Z}_q^{n \times m}$ and let $\mathbf{B}$ be a matrix again chosen from $\mathbb{Z}_q^{n \times m}$ but with an additional property that we also need to know its trapdoor. Recall that when we choose $\mathbf{B}$ to be a primitive matrix as defined in Theorem 2.1.10 (and appropriately append columns of 0^n), we can trivially get its trapdoor[1]. These $\ell + 2$ matrices will be part of the system parameters. To encrypt a message μ under the public key $\mathbf{x} = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ we use a variant of dual Regev encryption ($\mathsf{MAT}^T\mathbf{s} + \mathbf{e}$, for some secret $\mathbf{s}$, error $\mathbf{e}$) using the following matrix as the public key MAT for the dual Regev system:

$$\left(\mathbf{A} \mid \mathbf{A}_1 + x_1 \cdot \mathbf{B} \mid \cdots \mid \mathbf{A}_\ell + x_\ell \cdot \mathbf{B}\right) \in \mathbb{Z}_q^{n \times (\ell+1)m} \tag{4.1}$$

A similar variant was used by Agrawal, Boneh and Boyen [ABB10] to get hierarchical IBE and by Agrawal, Freeman and Vaikuntanathan [AFV11] to get inner-product predicate functional encryption. Both of them choose $\mathbf{B}$ randomly from $\mathbb{Z}_q^{n \times m}$. [AFV11] use the additive key homomorphism property of this to construct their inner-product predicate encryption scheme. One of the main contributions of our work is a novel method of multiplying the dual Regev public key matrices. We achieve this by picking $\mathbf{B}$ so that we know its trapdoor (and not from a uniform distribution in $\mathbb{Z}_q^{n \times m}$). We also observe that the above variant of dual Regev encryption is secure as long as $\mathbf{B}$ is a matrix with rank n.

First, we obtain a ciphertext $\mathbf{c_x}$ with the above matrix as the public key of Regev's encryption. We show that, this system is key homomorphic: given a circuit C, anyone can transform the ciphertext $\mathbf{c_x}$

[1] We can also run $\mathsf{TrapSamp}$ algorithm to get $\mathbf{B}$ along with its trapdoor and then publish both of them in the master public key. But choosing $\mathbf{B}$ to be the primitive matrix yields more efficient parameters for our scheme.

into a dual Regev encryption for the (evaluated) public-key matrix

$$\left(\mathbf{A} \mid \mathbf{A}_C + C(\mathbf{x}) \cdot \mathbf{B}\right) \in \mathbb{Z}_q^{n \times 2m}$$

where the matrix $\mathbf{A}_C \in \mathbb{Z}_q^{n \times m}$ serves as the encoding of the circuit C. This way, we are also able to achieve the crucial properties (required for ABE) that:

- $\mathbf{A}_C$ is uniquely determined by C and $\mathbf{A}_1, \ldots, \mathbf{A}_\ell$.
- $\mathbf{A}_C$ can be constructed without the knowledge of $\mathbf{x}$.

We illustrate the key homomorphism in the above dual Regev variant as follows. Assume that we have the ciphertext under the public key $\mathbf{x} = (x_1 x_2)$: $\mathbf{c}_\mathbf{x} = (\mathbf{c}_0 \mid \mathbf{c}_{x_1} \mid \mathbf{c}_{x_2})$ where $\mathbf{c}_0 = \mathbf{A}^T\mathbf{s} + \mathbf{e}$, $\mathbf{c}_{x_1} = (\mathbf{A}_1 + x_1 \cdot \mathbf{B})^T\mathbf{s} + \mathbf{e}_1$ and $\mathbf{c}_{x_2} = (\mathbf{A}_2 + x_2 \cdot \mathbf{B})^T\mathbf{s} + \mathbf{e}_2$. We present here a way to perform the NAND operation (which can also be seen as a circuit with a single NAND gate). To compute the ciphertext under the public key $(1 - x_1 x_2,\ \mathsf{NAND})$, we first compute a low norm matrix $\mathbf{D}_1 \in \mathbb{Z}_q^{m \times m}$, by running $\mathsf{InvB}(\mathbf{A}_1)$. With this in mind we compute

$$\mathbf{D}_1^T \mathbf{c}_{x_2} = \mathbf{D}_1^T \cdot \left[(\mathbf{A}_2 + x_2 \cdot \mathbf{B})^T\mathbf{s} + \mathbf{e}_2\right] \approx (x_2 \cdot \mathbf{A}_1 + \mathbf{A}_2\mathbf{D}_1)^T\mathbf{s}$$
$$x_2 \cdot \mathbf{c}_{x_1} = x_2 \left[(\mathbf{A}_1 + x_1 \cdot \mathbf{B})^T\mathbf{s} + \mathbf{e}_1\right] \approx (x_2 \cdot \mathbf{A}_1 + x_1 x_2 \cdot \mathbf{B})^T\mathbf{s}$$

Subtracting the second expression above from the first gives us

$$\left((\mathbf{A}_2\mathbf{D}_1 - \mathbf{B}) + (1 - x_1 x_2) \cdot \mathbf{B}\right)^T\mathbf{s} + \mathbf{noise}$$

$$= \left(\mathbf{A}_{\mathsf{NAND}} + (x_1 \mathsf{NAND} x_2) \cdot \mathbf{B}\right)^T\mathbf{s} + \mathbf{noise}$$

which is a ciphertext under the public key $(x_1 x_2,\ \mathbf{A}_{\mathsf{NAND}})$ where $\mathbf{A}_{\mathsf{NAND}} = \mathbf{A}_2\mathbf{D}_1 - \mathbf{B}$. Note that performing this operation requires that we know x_2. This is reason why this method gives just an ABE and not (private index) predicate encryption.

This key-homomorphism gives us an ABE for circuits since any boolean circuit can be converted into a circuit with NAND gates alone. A decryption key sk_C for a circuit C is a decryption key for the dual Regev ciphertext with public key matrix $(\mathbf{A}||\mathbf{A}_C + 1 \cdot \mathbf{B}) = (\mathbf{A}||\mathbf{A}_C + \mathbf{B})$. This key enables decryption whenever $C(\mathbf{x}) = 1$. The key sk_C can be easily generated using the trapdoor $\mathbf{T}_\mathbf{A}$ for the matrix $\mathbf{A}$ which is a short basis for the lattice $\Lambda_q^\perp(\mathbf{A})$. $\mathbf{T}_\mathbf{A}$ serves as the master secret key. We prove selective security from the learning with errors problem (LWE) by using another homomorphic property of the system implemented in an algorithm called $\mathsf{Eval}_{\mathrm{SIM}}$. Using $\mathsf{Eval}_{\mathrm{SIM}}$ the simulator responds to the adversary's private key queries and then solves the given LWE challenge.

4.1 Construction

We now present the instantiation of FKHE algorithms in Section 3.1 under the LWE assumption building on the ideas presented above. We first assume the existence of the following three deterministic evaluation algorithms[2] and instantiate them after describing the other algorithms.

[2]In addition to $\mathsf{Eval}_{\mathrm{ct}}$, we define two new algorithms $\mathsf{Eval}_{\mathrm{pk}}$ and $\mathsf{Eval}_{\mathrm{SIM}}$ which help to improve the presentation of the LWE instantiation.

- $\mathsf{Eval}_{\mathrm{pk}}(C, \{\mathbf{A}_i\}_{i\in[\ell]}) \to \mathbf{A}_C$: The evaluation algorithm for the public keys of dual Regev encryption. Inputs are ℓ matrices $\mathbf{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ along with a circuit $C \in \mathcal{C}$. Output is the evaluated matrix $\mathbf{A}_C \in \mathbb{Z}_q^{n\times m}$.

- $\mathsf{Eval}_{\mathrm{ct}}(C, \{x_i, \mathbf{A}_i, \mathbf{c}_i\}_{i\in[\ell]}) \to \mathbf{c}_C$: The ciphertext evaluation algorithm as defined in the FKHE definition with a circuit $C \in \mathcal{C}$, a vector $\mathbf{x} \in \{0,1\}^\ell$ and a set of ciphertexts $\mathbf{c}_\mathbf{x} = (\mathbf{c}_i)_{i\in[\ell]}$. The third component, a set of matrices $(\mathbf{A}_i)_{i\in[\ell]}$ comes from the master public key mpk. These matrices form the public keys for encryption The output is the evaluated ciphertext $\mathbf{c}_C$ under $(C(\mathbf{x}), C)$ which is used to determine the evaluated public key. $\mathbf{c}_C$ can thus be viewed as a dual Regev ciphertext under the public key matrix $\mathbf{A}_C + C(\mathbf{x})\cdot\mathbf{B}$.

- $\mathsf{Eval}_{\mathrm{SIM}}(C, \{x_i^*, \mathbf{R}_i\}_{i\in[\ell]}, \mathbf{A}) \to \mathbf{R}_C$: This algorithm is used during simulation. On input a circuit $C \in \mathcal{C}$, matrices $\mathbf{R}_i \in \{-1,1\}^{m\times m}$ and a challenge public key vector $\mathbf{x}^* = (x_1^*, \ldots, x_\ell^*)$, this algorithm outputs a low-norm matrix $\mathbf{R}_C$ such that

$$\mathbf{A}\mathbf{R}_C - C(\mathbf{x}^*)\cdot\mathbf{B} = \mathbf{A}_C, \text{ where } \mathbf{A}_C = \mathsf{Eval}_{\mathrm{pk}}(C, \{\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}\}_{i\in[\ell]})$$

Now, assuming the existence of the evaluation algorithms $\mathsf{Eval}_{\mathrm{pk}}, \mathsf{Eval}_{\mathrm{ct}}, \mathsf{Eval}_{\mathrm{SIM}}$ for a family $\mathcal{C}$ of boolean predicate circuits with depth atmost $d_{\max}$, we first present the algorithms $\mathsf{Params}, \mathsf{Setup}_{\mathsf{FKHE}}, \mathsf{KeyGen}_{\mathsf{FKHE}}, \mathsf{E}_{\mathsf{FKHE}}, \mathsf{D}_{\mathsf{FKHE}}$ of our FKHE scheme $\mathcal{FKHE}$ for the same family of circuits.

- $\mathsf{Params}(1^\lambda, d_{\max})$: Let $n = n(\lambda, d_{\max})$, $q = q(n, d_{\max})$ and $m = m(n, d_{\max})$. Set the error distribution $\chi = \chi(n)$ and the error bound $B = B(n)$. We also additionally choose two Gaussian parameters: a "small" Gaussian parameter $s = s(n)$ (which the reader should think of as polynomially bounded) and a "large" Gaussian parameter $\alpha = \alpha(n, d_{\max})$ (which the reader should think of as growing exponentially in $d_{\max}$). Output the global public parameters $\mathsf{pp} = (n, \chi, B, q, m, s, \alpha)$. This is implicitly given to all of the algorithms defined below[3].

- $\mathsf{Setup}_{\mathsf{FKHE}}(1^\ell)$:

  1. Run an instance of the trapdoor generation algorithm $\mathsf{TrapSamp}(1^n, 1^m, q)$ to obtain $(\mathbf{A}, \mathbf{T}_\mathbf{A})$.
  2. Let the matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$ be obtained by appending columns of zeros (0^n) to the primitive matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times n(\lfloor \log q\rfloor+1)}$ defined in Theorem 2.1.10. Let its trapdoor be $\mathbf{T}_\mathbf{B}$.
  3. Choose ℓ matrices $\{\mathbf{A}_i\}_{i\in[\ell]}$ at random from $\mathbb{Z}_q^{n\times m}$.
  4. Choose a vector $\mathbf{u}$ at random from $\mathbb{Z}_q^n$.
  5. Run the InvB algorithm ℓ times for each $\mathbf{A}_i$ to obtain $\mathbf{D}_i$ such that $\mathbf{B}\cdot\mathbf{D}_i = \mathbf{A}_i$.
  6. Define and output the master public key as

$$\mathsf{mpk} := \left(\mathbf{A}, \{\mathbf{A}_i, \mathbf{D}_i\}_{i\in[\ell]}, \mathbf{B}, \mathbf{u}\right)$$

  and the master secret key as $\mathsf{msk} := (\mathbf{T}_\mathbf{A})$.

- $\mathsf{KeyGen}_{\mathsf{FKHE}}(\mathsf{msk}_{\mathsf{FKHE}}, (y, C))$:

  1. Let $\mathbf{A}_C = \mathsf{Eval}_{\mathrm{pk}}(C, (\mathbf{A}_1, \ldots, \mathbf{A}_\ell))$.

[3] See Sections 4.2 and 4.4 for concrete instantiation of the parameters.

2. Define $\mathbf{F} = [\mathbf{A}||(\mathbf{A}_C + y \cdot \mathbf{B})] \in \mathbb{Z}_q^{n \times 2m}$ where $y \in \{0,1\}$. Compute $\mathbf{r}_{\text{out}} \in \mathbb{Z}_q^{2m}$ by running $\mathsf{SampleLeft}(\mathbf{A}, (\mathbf{A}_C + y \cdot \mathbf{B}), \mathbf{T}_{\mathbf{A}}, \mathbf{u}, \alpha)$, satisfying $\mathbf{F} \cdot \mathbf{r}_{\text{out}} = \mathbf{u}$.
3. Output the secret key for the public key (y, C), $\mathsf{sk}_{y,C} := (C, \mathbf{r}_{\text{out}})$.

- $\mathsf{E}_{\mathsf{FKHE}}(\mathsf{mpk}_{\mathsf{FKHE}}, \mathbf{x} = (x_1, \ldots, x_\ell), \mu)$: For encrypting a message $\mu \in \{0,1\}$, do the following:

1. Choose a vector $\mathbf{s} \in \mathbb{Z}_q^n$ at random.
2. Choose a noise term $\mathbf{e}_0 \leftarrow \chi^m$ and compute $\mathbf{c}_0 = \mathbf{A}^T\mathbf{s} + \mathbf{e}_0$.
3. Choose ℓ random matrices $\mathbf{R}_i \leftarrow \{-1,1\}^{m \times m}$ for all $i \in [\ell]$.
4. Set $\mathbf{H} \in \mathbb{Z}_q^{n \times (\ell+1)}$ and $\mathbf{e} \in \mathbb{Z}_q^{(\ell+1)m}$ as

$$\mathbf{H} := [\mathbf{A}||(\mathbf{A}_1 + x_1 \cdot \mathbf{B})|| \cdots ||(\mathbf{A}_\ell + x_\ell \cdot \mathbf{B})]$$
$$\mathbf{e} := [\mathbf{I}_m||\mathbf{R}_1|| \cdots ||\mathbf{R}_\ell]$$

5. Compute $\mathbf{c} = (\mathbf{H}^T\mathbf{s} + \mathbf{e}, \tau = \mathbf{u}\mathbf{s} + e + \lfloor q/2 \rfloor \mu)$, where $e \xleftarrow{\$} \chi$.
6. Output the ciphertext as $\mathbf{c}_{\mathbf{x}}$.

- $\mathsf{D}_{\mathsf{FKHE}}(\mathsf{sk}_{y,C}, \mathbf{c})$:

1. Let $\mathbf{c}$ be an (evaluated) encryption of μ under the (evaluated) public key (y', C'). If $y' \neq y$, or if C and C' are not identical circuits, output $\perp$.
2. Otherwise, let $\mathbf{c} = (\mathbf{c}_0||\mathbf{c}_1|| \cdots ||\mathbf{c}_\ell, \tau)$. Calculate $\mathbf{c}_C = \mathsf{Eval}_{\text{ct}}(C, \{(x_i, \mathbf{A}_i, \mathbf{c}_i)\}_{i \in [\ell]})$.
3. Compute $\beta = \mathbf{r}_{\text{out}}^T \cdot [\mathbf{c}_0||\mathbf{c}_C]$, where $\beta \approx \mathbf{u}^T\mathbf{s} \pmod q$ (by Lemma 4.2.1).
4. Output $\mu = 0$ if $|\tau - \beta| < q/4$ and $\mu = 1$ otherwise.

To complete the description of the scheme, we now instantiate the three evaluation algorithms.

- $\mathsf{Eval}_{\text{pk}}(C, \{\mathbf{A}_i\}_{i \in [\ell]}) \rightarrow \mathbf{A}_C$: The matrices $\mathbf{A}_i$ for each input wire $i \in [\ell]$ are given as input and we need to calculate the (evaluated) matrix $\mathbf{A}_C$ for the output wire for the given circuit C.

1. Inductively, from input to output, consider a gate $g = (u, v; w)$. Without loss of generality assume g is a binary NAND gate, since any boolean operation can be computed by a set of NAND gates.
2. The matrices for the input wires (u, v) are fixed as $\mathbf{A}_u, \mathbf{A}_v$ by induction and the matrix for the output wire w is assigned the value

$$\mathbf{A}_w = \mathbf{A}_v \cdot \mathbf{D}_u - \mathbf{B}$$

3. Finally, let $\mathbf{A}_C$ be the matrix obtained at the output wire by this process. Output this matrix $\mathbf{A}_C$.

- $\mathsf{Eval}_{\text{ct}}(C, \{x_i, \mathbf{A}_i, \mathbf{c}_i\}_{i \in [\ell]}) \rightarrow \mathbf{c}_C$: The components $\mathbf{c}_i, \forall i \in [\ell]$ of the ciphertext $\mathbf{c}$ under the public key vector $\mathbf{x}$ are given as input along with $\mathbf{x}$ and the matrices $\mathbf{A}_i, \forall i \in [\ell]$ (where $\mathbf{A}_i$ acts as the public key when $\mathbf{c}_i$ is viewed as a dual Regev ciphertext). We need to calculate the (evaluated) ciphertext $\mathbf{c}_C$ for the given circuit C.

1. We again proceed inductively from the input level to the output level of C. Consider a NAND gate $g = (u, v; w)$ carrying input values x_u, x_v and hence output value $x_w = x_u \text{ NAND } x_v = 1 - x_u x_v$.
2. Assume by induction, the user holds $\mathbf{c}_u = (\mathbf{A}_u + x_u \cdot \mathbf{B})^T \mathbf{s} + \mathbf{e}_u$ and $\mathbf{c}_v = (\mathbf{A}_v + x_v \cdot \mathbf{B})^T \mathbf{s} + \mathbf{e}_v$ (for some error vectors $\mathbf{e}_u$ and $\mathbf{e}_v$).
3. Compute $\mathbf{c}_w$ as follows:
$$\mathbf{c}_w = \mathbf{D}_u^T \mathbf{c}_v - x_v \mathbf{c}_u$$
As we show below (in Claim 4.2.1), this new ciphertext has the form $\left(\mathbf{A}_w + x_w \mathbf{B}\right)^T \mathbf{s} + \mathbf{e}_w$ (for a small enough noise term $\mathbf{e}_w$).
4. Proceeding this way, at the output gate we obtain $\mathbf{c}_C = (\mathbf{A}_C + C(\mathbf{x}) \cdot \mathbf{B})^T \mathbf{s} + \mathbf{e}_C$, where $\mathbf{A}_C \leftarrow \mathsf{Eval}_{\text{pk}}(C, \{\mathbf{A}_i\}_{i \in [\ell]})$ and $\mathbf{e}_C$ is some bounded error (as proved in Claim 4.2.1). Output $\mathbf{c}_C$.

- $\mathsf{Eval}_{\text{SIM}}(C, \{x_i^*, \mathbf{R}_i\}_{i \in [\ell]}, \mathbf{A}) \to \mathbf{R}_C$: On input a set of matrices $\mathbf{R}_i, \forall i \in [\ell]$ and $\mathbf{A}$ such that $\mathbf{A}_i := \mathbf{A}\mathbf{R}_i - x_i^* \cdot \mathbf{B}$, we need to calculate a low norm matrix $\mathbf{R}_C$ such that $\mathbf{A}_C = \mathbf{A}\mathbf{R}_C - C(\mathbf{x}^*) \cdot \mathbf{B}$.
  1. This algorithm proceeds very similar to $\mathsf{Eval}_{\text{ct}}$. First consider a NAND gate $g = (u, v; w)$ with input wires u, v carrying values $C(\mathbf{x}^*)_u, C(\mathbf{x}^*)_v$ respectively. Here, $C(\mathbf{x}^*)_u$ is the value which passes through the wire u of C when C is evaluated with $\mathbf{x}^*$ as input.
  2. Let the '$\mathbf{R}$' matrices corresponding to the incoming wires (u, v) be $\mathbf{R}_u, \mathbf{R}_v$ by induction. The matrix $\mathbf{R}_w$ for the output wire w is assigned the value
$$\mathbf{R}_w = \mathbf{R}_v \cdot \mathbf{D}_u - C(\mathbf{x}^*)_v \mathbf{R}_u$$
  3. Finally, let $\mathbf{R}_C$ be the matrix obtained at the outgoing wire by this process. Output this matrix $\mathbf{R}_C$ which has low norm as proved in Claim 4.4.1.

This completes the full description of our FKHE scheme based on the LWE assumption.

Note: This scheme can be easily generalised to work for any message space $\mathcal{M} = \{0,1\}^v$ for a bounded message length $v = \text{poly}(\lambda)$ by working with a matrix $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n \times v}$ instead of the vector $\mathbf{u} \in \mathbb{Z}_q^n$. Here the size of the secret keys will increase multiplicatively with v.

4.2 Correctness and Compactness

Lemma 4.2.1 (Correctness)**.** *Let $\mathcal{C}$ be a family of boolean predicate circuits with their depth bounded by $d_{\max}$ and let $\mathcal{FKHE} = (\mathsf{Params}, \mathsf{Setup}_{\mathsf{FKHE}}, \mathsf{KeyGen}_{\mathsf{FKHE}}, \mathsf{E}_{\mathsf{FKHE}}, \mathsf{Eval}_{ct}, \mathsf{D}_{\mathsf{FKHE}})$ be our fully key-homomorphic encryption scheme. Assuming that, for a LWE dimension $n = n(\lambda, d_{\max})$, the parameters for $\mathcal{FKHE}$ are instantiated as follows:*

$$\chi = D_{\mathbb{Z}, \sqrt{n}} \qquad B = O(n)$$
$$q = \tilde{O}(n d_{\max})^{O(d_{\max})} n \qquad s = O(\sqrt{n \log q})$$
$$m = O(n \log q) \qquad \alpha = O(n \log q)^{O(d_{\max})}$$

then the scheme $\mathcal{FKHE}$ is correct, according to the definition in Section 3.1.

Proof. Let us consider a circuit $C \in \mathcal{C}$ of depth atmost $d_{\max}$, such that $C(\mathbf{x}) = 1$. First we show the correctness of the $\mathsf{Eval}_{\mathrm{ct}}$ algorithm.

Claim 4.2.1. *On input $\big(C, \{x_i, \mathbf{A}_i, \mathbf{c}_i\}_{i\in[\ell]}\big)$, Eval_{ct} outputs $\mathbf{c}_C$ which is a ciphertext of the form $(\mathbf{A}_C + C(\mathbf{x})\cdot\mathbf{B})^T\mathbf{s} + \mathbf{e}_C$, where $\mathbf{A}_C \leftarrow \mathsf{Eval}_{pk}(C, \{\mathbf{A}_i\}_{i\in[\ell]})$ and $\mathbf{e}_C \in D_{Z,\alpha}$.*

Proof. First, we show that for each ciphertext $\mathbf{c}_u$ for wire u at level j, the user holds $(\mathbf{A}_u + x_u\mathbf{B})^T\mathbf{s} + \mathbf{e}_u$, where $||\mathbf{e}_u||_\infty \le B\cdot(\frac{\sqrt{5}}{2})^j(m^{1.5j+1})$ and $\mathbf{A}_u$ is the output of $\mathsf{Eval}_{\mathrm{pk}}$ for the part of C which has u as its output wire. Note that when $\mathbf{e}_0 \leftarrow \chi^m$, $||\mathbf{e}_0||_\infty \le B$ by the definition of χ and B. Hence, for the noise term $\mathbf{e}_i = \mathbf{R}_i^T\mathbf{e}_0$, $||\mathbf{e}_i||_\infty \le mB$ since $\mathbf{R}_i \in \{-1,1\}^{m\times m}$. Thus, the base case for the input ciphertext components holds.

- Now, consider a NAND gate $g = (u, v, w)$ and any two ciphertext components $\mathbf{c}_u = (\mathbf{A}_u + x_u\mathbf{B})^T\mathbf{s} + \mathbf{e}_u, \mathbf{c}_v = (\mathbf{A}_v + x_v\mathbf{B})^T\mathbf{s} + \mathbf{e}_v$ at depths j_0, j_1 respectively, where $||\mathbf{e}_u||_\infty \le B\cdot(\frac{\sqrt{5}}{2})^j(m^{1.5j_0+1})$ and $||\mathbf{e}_1||_\infty \le B\cdot(\frac{\sqrt{5}}{2})^j(m^{1.5j_1+1})$. Then, the evaluated ciphertext $\mathbf{c}_w$ for the gate g is computed as follows:

$$\begin{aligned}
\mathbf{c}_w &= \mathbf{D}_u^T\mathbf{c}_v - x_v\mathbf{c}_u \\
&= \big(\mathbf{A}_v\cdot\mathbf{D}_u + x_v\cdot\mathbf{B}\cdot\mathbf{D}_u\big)^T\mathbf{s} + \mathbf{D}_u^T\mathbf{e}_v - x_v\Big(\big(\mathbf{A}_u + x_u\mathbf{B}\big)^T\mathbf{s} + \mathbf{e}_u\Big) \\
&= \big(\mathbf{A}_v\cdot\mathbf{D}_u + x_v\cdot\mathbf{A}_u\big)^T\mathbf{s} + \mathbf{D}_u^T\mathbf{e}_v - x_v\Big(\big(\mathbf{A}_u + x_u\mathbf{B}\big)^T\mathbf{s} + \mathbf{e}_u\Big) \\
&= \big(\mathbf{A}_w + (1 - x_ux_v)\cdot\mathbf{B}\big)^T\mathbf{s} + \mathbf{e}_w
\end{aligned}$$

  where the last equation is because $\mathbf{A}_w = \mathbf{A}_v\cdot\mathbf{D}_u - \mathbf{B}$. Also, we define $\mathbf{e}_w := \mathbf{D}_u^T\mathbf{e}_v - x_v\mathbf{e}_u$.

- Thus,

$$\begin{aligned}
||\mathbf{e}_w||_\infty &\le m\cdot\|\mathbf{D}_u\|_{\max}\cdot||\mathbf{e}_u||_\infty + ||\mathbf{e}_v||_\infty \\
&\le m\cdot\big((\frac{1}{2}\sqrt{5m})\cdot(B\cdot O(m^{1.5j_0+1})) + B\cdot O(m^{1.5j_1+1})\big) \\
&\le B\cdot((\frac{\sqrt{5}}{2})(m^{1.5})\cdot(\frac{\sqrt{5}}{2})^j(m^{1.5j_0+1}) + m\cdot(\frac{\sqrt{5}}{2})^j(m^{1.5j_1+1})) \\
&\le B\cdot(\frac{\sqrt{5}}{2})^{\max(j_0,j_1)+1}(m^{1.5\left(\max(j_0,j_1)+1\right)+1})
\end{aligned}$$

  as claimed (the second inequality is because $\|\mathbf{D}_u\|_{\max} \le \frac{1}{2}\sqrt{m}\cdot\|\widetilde{\mathbf{T}_\mathbf{B}}\| \le \frac{1}{2}\sqrt{5m}$).

Thus, if $C(\mathbf{x}) = 1$, the user obtains $\mathbf{c}_C = (\mathbf{A}_C + C(\mathbf{x})\cdot\mathbf{B})^T\mathbf{s} + \mathbf{e}_C = (\mathbf{A}_C + \mathbf{B})^T\mathbf{s} + \mathbf{e}_C$, where $\|\mathbf{e}_C\|_\infty \le B\cdot(\frac{\sqrt{5}}{2})^{d_{\max}}(m^{1.5d_{\max}+1})$. □

Now to complete the proof of this lemma, we need to show that the decryption proceeds correctly. That is, we need to show that the LWE error obtained during the calculation of β is bound by $q/4$. The user obtains β as $\beta := \mathbf{r}_{\mathrm{out}}^T[\mathbf{c}_0||\mathbf{c}_C] = \mathbf{u}^T\mathbf{s} + e_f$, where $e_f = [\mathbf{e}_0^T||\mathbf{e}_C^T]\mathbf{r}_{\mathrm{out}}$. Here, $\mathbf{r}_{\mathrm{out}}$ is the output of $\mathsf{SampleLeft}$ algorithm (Algorithm 2.1. Hence, it has an infinite norm $\|\mathbf{r}_{\mathrm{out}}\|_\infty \le \alpha\sqrt{m} \le O(n\log q)^{O(d_{\max})}$. Thus,

$$\|e_f\|_\infty \le m\cdot(\|\mathbf{e}_0\|_\infty + \|\mathbf{e}_C\|_\infty)\cdot\|\mathbf{r}_C\|_\infty \le O(B\cdot O(n\log q)^{O(d_{\max})})$$

Also, ciphertext component τ is computed using noise term $||e||_\infty \leq B$. Hence,

$$||e_f - e||_\infty \leq O(B \cdot O(n \log q)^{O(d_{\max})}) < q/4,$$

which holds given the above setting of the parameters.Thus, when $\mathbf{c}$ and $\mathbf{c}'$ are "close", message $\mu \in \{0,1\}$ is decoded correctly as required. □

Corollary 4.2.2. *For any depth $d_{\max}$ family of circuits $\mathcal{C}$ and $y \in \{0,1\}$, the secret key size for an evaluated public key (y, C) in our FKHE is $O(n \log q) = \text{poly}(d_{\max}, \lambda)$.*

4.3 Security Proof

Theorem 4.3.1 (Selective security)**.** *For all ℓ and polynomial $d_{\max} = d_{\max}(\ell)$, there exists a selectively-secure fully key-homomorphic encryption for any family of polynomial-size circuits with ℓ inputs and depth at most $d_{\max}$ with constant secret key size (independent on the number of gates in the circuits), assuming hardness of $\mathsf{dLWE}_{n,q,\chi}$ for sufficiently large $n = \text{poly}(\lambda, d_{\max}), q = n^{O(d_{\max})}$ and $\text{poly}(n)$ bounded error distribution χ.*

Proof. We follow the security strategy of [ABB10, AFV11] to prove the security of our construction. In particular, we define a series of hybrid games, where the first and last games correspond to the real experiments encrypting messages μ_0, μ_1, respectively. We show that these games are indistinguishable. Recall that in the selective security game, the challenge $\mathbf{x}^*$ is declared before the $\mathsf{Setup}_{\mathsf{FKHE}}$ algorithm and all secret key queries (y, C) must be such that $C(\mathbf{x}^*) \neq y$ for some $y \in \{0,1\}$. Now, consider the following simulated ABE algorithms:

- $\mathsf{Setup}^*_{\mathsf{FKHE}}\big(1^\lambda, \mathbf{x}^* = (x_1^*, \ldots, x_\ell^*)\big)$: Takes as input the challenge $\mathbf{x}^*$ and does the following:
  1. Choose a random matrix $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ and a random vector $\mathbf{u} \in \mathbb{Z}_p^n$.
  2. Let the matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$ be obtained by appending columns of zeros (0^n) to the primitive matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times n(\lfloor \log q \rfloor + 1)}$ as defined in 2.1.10. Let its trapdoor be $\mathbf{T_B}$.
  3. For all $i \in [\ell]$, sample a short matrix $\mathbf{R}_i \in \{-1, 1\}^{m \times m}$ at random and let $\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i^* \mathbf{B}$.
  4. Run the InvB algorithm ℓ times for each $\mathbf{A}_i$ to obtain $\mathbf{D}_i$ such that $\mathbf{B} \cdot \mathbf{D}_i = \mathbf{A}_i$.
  5. Define and output the master public key as

$$\mathsf{mpk} := \big(\mathbf{A}, \{\mathbf{A}_i, \mathbf{D}_i)\}_{i \in [\ell]}, \mathbf{B}, \mathbf{u}\big)$$

- $\mathsf{KeyGen}^*_{\mathsf{FKHE}}((1, C), \{\mathbf{R}_i\}_{i \in [\ell]})$: If $C(\mathbf{x}^*) = y$, return $\perp$. Else, proceed as follows:
  1. The challenger runs $\mathsf{Eval}_{\text{SIM}}(C, \{x_i^*, \mathbf{R}_i\}_{i \in [\ell]}, \mathbf{A})$ and obtains $\mathbf{R}_C$. By the correctness of $\mathsf{Eval}_{\text{SIM}}$, we have $\mathbf{A}\mathbf{R}_C - C(x^*) \cdot \mathbf{B} = \mathbf{A}_C$.
  2. Let $\mathbf{F} = [\mathbf{A}||(\mathbf{A}_C + C(\mathbf{x}^*) \cdot \mathbf{B})] = [\mathbf{A}||\mathbf{A}\mathbf{R}_C]$. Compute $\mathbf{r}_{\text{out}} \in \mathbb{Z}^{2m}$ by running $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}_C, \mathbf{T_B}, \mathbf{u}, \alpha)$, satisfying $\mathbf{F} \cdot \mathbf{r}_{\text{out}} = \mathbf{u}$.
  3. The secret key for the evaluated public key (y, C) is $\mathsf{sk}_{y,C} := (C, \mathbf{r}_{\text{out}})$.

- $\mathsf{E}^*_{\mathsf{FKHE}}\big(\mathsf{mpk}, x^* = (x_1^*, \ldots, x_\ell^*), \mu\big)$: For encrypting a message $\mu \in \{0,1\}$, do the following:

1. Choose a vector $\mathbf{s} \in \mathbb{Z}_q^n$ at random.
2. Choose a noise term $\mathbf{e}_0 \leftarrow \chi^m$ and compute $\mathbf{c}^* = (\mathbf{A})^T\mathbf{s} + \mathbf{e}_0$.
3. Choose ℓ random matrices $\mathbf{R}_i \leftarrow \{-1, 1\}^{m \times m}$ for all $i \in [\ell]$.
4. Set $\mathbf{H} \in \mathbb{Z}_q^{n \times (\ell+1)}$ and $\mathbf{e} \in \mathbb{Z}_q^{(\ell+1)m}$ where

$$\mathbf{H} := [\mathbf{A}||(\mathbf{A}_1 + x_1^* \cdot \mathbf{B})|| \cdots ||(\mathbf{A}_\ell + x_\ell^* \cdot \mathbf{B})]$$
$$\mathbf{e} := [\mathbf{I}_m||\mathbf{R}_1|| \cdots ||\mathbf{R}_\ell]$$

   Note that $\mathbf{H} = [\mathbf{A}||\mathbf{A}\mathbf{R}_1|| \cdots ||\mathbf{A}\mathbf{R}_\ell]$.
5. Compute $\mathbf{c} = (\mathbf{H}^T\mathbf{s} + \mathbf{e}, \tau = \mathbf{u}\mathbf{s} + e + \lfloor q/2 \rfloor \mu)$, where $e \xleftarrow{\$} \chi$.
6. Output the ciphertext as $\mathbf{c}_{\mathbf{x}^*}$.

Game Sequence We now define a series of games and then prove that all games **Game i** and **Game i+1** are either statistically or computationally indistinguishable.

- **Game 0**: The challenger runs the real FKHE algorithms and encrypts message μ_0 for the challenge index $\mathbf{x}^*$.
- **Game 1**: The challenger runs the simulated FKHE algorithms $\mathsf{Setup}^*_{\mathsf{FKHE}}, \mathsf{KeyGen}^*_{\mathsf{FKHE}}, \mathsf{E}^*_{\mathsf{FKHE}}$ and encrypts message μ_0 for the challenge index $\mathbf{x}^*$.
- **Game 2**: The challenger runs the simulated FKHE algorithms $\mathsf{Setup}^*_{\mathsf{FKHE}}, \mathsf{KeyGen}^*_{\mathsf{FKHE}}$, but chooses a uniformly random element of the ciphertext space for challenge index $\mathbf{x}^*$.
- **Game 3**: The challenger runs the simulated FKHE algorithms $\mathsf{Setup}^*_{\mathsf{FKHE}}, \mathsf{KeyGen}^*_{\mathsf{FKHE}}, \mathsf{E}^*_{\mathsf{FKHE}}$ and encrypts message μ_1 for the challenge index $\mathbf{x}^*$.
- **Game 4**: The challenger runs the real FKHE algorithms and encrypts message μ_1 for the challenge index $\mathbf{x}^*$.

Lemma 4.3.2. *The view of an adversary in* **Game 0** *is statistically indistinguishable from* **Game 1**. *Similarly, the view of an adversary in* **Game 4** *is statistically indistinguishable from* **Game 3**.

Proof. We prove for the case of **Game 0** and **Game 1**, as the other case is identical. First, note the differences between the games:

- In **Game 0**, matrix $\mathbf{A}$ is sampled using $\mathsf{TrapSamp}$ algorithm and matrices $\mathbf{A}_i \in \mathbb{Z}_p^{m \times m}$ are randomly chosen. In **Game 1**, matrix $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$ is chosen uniformly at random and matrices $\mathbf{A}_i = \mathbf{A}\mathbf{R}_i - x_i^*\mathbf{B}$ for uniformly random $\mathbf{R}_i \in \{-1, 1\}^{m \times m}$.
- In **Game 0**, each ciphertext component is computed as:

$$\mathbf{c}_i = (\mathbf{A}_i + x_i^*\mathbf{B})^T\mathbf{s} + \mathbf{e}_i = (\mathbf{A}_i + x_i^*\mathbf{B})^T\mathbf{s} + \mathbf{R}_i^T\mathbf{e}_0$$

  On the other hand, in **Game 1** each ciphertext component is computed as:

$$\mathbf{c}_i = (\mathbf{A}_i + x_i^*\mathbf{B})^T\mathbf{s} + \mathbf{R}_i^T\mathbf{e}_0 = (\mathbf{A}\mathbf{R}_i)^T\mathbf{s} + \mathbf{R}_i^T\mathbf{e}_0 = \mathbf{R}_i^T\big((\mathbf{A})^T\mathbf{s} + \mathbf{e}_0\big)$$

- Finally, in **Game 0** the vector $\mathbf{r}_{\text{out}}$ is sampled using SampleLeft, whereas in **Game 1** it is sampled using SampleRight algorithm.

For sufficiently large α (See Section-4.4), the distributions produced in two games are statistically indistinguishable. This follows readily from [AFV11, Lemma 4.3], Theorem-2.1.8 and Theorem-2.1.9. We will provide the proof here for completeness.

We would like to prove that the tuple $\mathbf{A}, \{\mathbf{A}_i, \mathbf{c}_i\}_{i\in[\ell]}$ in **Game 0** is statistically indistinguishable from the set from **Game 1**. The generalisation of left-over hash lemma [DORS08, ABB10] states that, for two matrices $\mathbf{R}_i \xleftarrow{\$} \{-1,1\}^{m\times m}, \mathbf{A}, \mathbf{A}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and any vector $\mathbf{e}_0 \in \mathbb{Z}_q^m$, the following is true, when q is square-free (q does not have a square of a prime number as its factor).

$$(\mathbf{A}, \mathbf{A}\mathbf{R}_i, \mathbf{R}_i^T\mathbf{e}_0) \approx_s (\mathbf{A}, \mathbf{A}_i, \mathbf{R}_i^T\mathbf{e}_0)$$

Hence, for every fixed matrix $\mathbf{B} \in \mathbb{Z}_q^{n\times m}$ and every bit $x_i^* \in \{0,1\}$,

$$(\mathbf{A}, \mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}, \mathbf{R}_i^T\mathbf{e}_0) \approx_s (\mathbf{A}, \mathbf{A}_i, \mathbf{R}_i^T\mathbf{e}_0)$$

Now, we can extend this statistical indistinguishability to the joint distribution of these tuples for all $i \in [\ell]$, since the matrices $\mathbf{R}_i$ are independently chosen from $\{-1,1\}^{m\times m}, \forall i \in [\ell]$. Thus,

$$\left(\mathbf{A}, \{\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}, \mathbf{R}_i^T\mathbf{e}_0\}_{i\in[\ell]}\right) \approx_s \left(\mathbf{A}, \{\mathbf{A}_i, \mathbf{R}_i^T\mathbf{e}_0\}_{i\in[\ell]}\right)$$

Also, due to the fact that applying any function to two statistically indistinguishable entities results in entities which are atleast statistically indistinguishable as the original entities, we can perform the following steps:

$$\left(\mathbf{A}, \left\{\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}, (\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B})^T\mathbf{s}, \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right) \approx_s \left(\mathbf{A}, \left\{\mathbf{A}_i, (\mathbf{A}_i)^T\mathbf{s}, \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right)$$

$$\left(\mathbf{A}, \left\{\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}, (\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B} + x_i^*\cdot\mathbf{B})^T\mathbf{s}, \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right) \approx_s \left(\mathbf{A}, \left\{\mathbf{A}_i, (\mathbf{A}_i + x_i^*\cdot\mathbf{B})^T\mathbf{s}, \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right)$$

$$\left(\mathbf{A}, \left\{\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}, (\mathbf{A}\mathbf{R}_i)^T\mathbf{s} + \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right) \approx_s \left(\mathbf{A}, \left\{\mathbf{A}_i, (\mathbf{A}_i + x_i^*\cdot\mathbf{B})^T\mathbf{s} + \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right)$$

Finally,

$$\left(\mathbf{A}, \mathbf{A}\mathbf{s}+\mathbf{e}_0, \left\{\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}, \mathsf{InvB}(\mathbf{A}\mathbf{R}_i - x_i^*\cdot\mathbf{B}), (\mathbf{A}\mathbf{R}_i)^T\mathbf{s} + \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right)$$
$$\approx_s \left(\mathbf{A}, \mathbf{A}\mathbf{s}+\mathbf{e}_0, \left\{\mathbf{A}_i, \mathsf{InvB}(\mathbf{A}_i), (\mathbf{A}_i + x_i^*\cdot\mathbf{B})^T\mathbf{s} + \mathbf{R}_i^T\mathbf{e}_0\right\}_{i\in[\ell]}\right)$$

Thus, we can conclude that the public parameters in **Game 0** are statistically indistinguishable from those in **Game 1**, and that the output of $\mathsf{E}_{\mathsf{FKHE}}$ is statistically indistinguishable from that of $\mathsf{E}^*_{\mathsf{FKHE}}$. When the "large" Gaussian parameter α is chosen appropriately (as discussed in 4.4), the output of the $\mathsf{KeyGen}_{\mathsf{FKHE}}$ and $\mathsf{KeyGen}^*_{\mathsf{FKHE}}$ algorithms are also statistically indistinguishable. Thus, the view of an adversary in **Game 0** is statistically indistinguishable from the view in **Game 1**. □

Lemma 4.3.3. *If the decisional* LWE *assumption holds, then the view of an adversary in* **Game 1** *is computationally indistinguishable from* **Game 2**. *Similarly, if the decisional* LWE *assumption holds, then the view of an adversary in* **Game 3** *is computationally indistinguishable from* **Game 2**.

Proof. Assume there exist an adversary Adv that distinguishes between **Game 1** and **Game 2**. We show how to break LWE problem given a challenge $\{(\mathbf{a}_i, y_i)\}_{i\in[m+1]}$ where each y_i is either a random sample in $\mathbb{Z}_q$ or $\mathbf{a}_i^T \cdot \mathbf{s} + e_i$ (for a fixed, random $\mathbf{s} \in \mathbb{Z}_q^n$ and a noise term sampled from the error distribution $e_i \leftarrow \chi$). Let $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_m] \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{u} = \mathbf{a}_{m+1}$. Let $\mathbf{c}^* = [y_1, y_2, \ldots, y_m]$ and $\tau = y_{m+1} + \mu \lfloor q/2 \rfloor$.

Now, run the simulated $\mathsf{Setup}^*_{\mathsf{FKHE}}$ algorithm where $\mathbf{A}, \mathbf{u}$ are as defined above. Run the simulated $\mathsf{KeyGen}^*_{\mathsf{FKHE}}$ algorithm. Finally, to simulate the challenge ciphertext set $\mathbf{c}^*, \tau$ as defined above and compute

$$\mathbf{c}_i = \mathbf{R}_i^T \mathbf{c}^* = \mathbf{R}_i^T \left((\mathbf{A})^T \mathbf{s} + \mathbf{e}_0\right)$$

Note that if y_i's are LWE samples, then this corresponds exactly to the **Game 1**. Otherwise, the ciphertext corresponds to an independent random sample as in **Game 2** by the left-over hash lemma. Thus, an adversary which distinguishes between **Game 1** and **Game 2** can also be used to break the decisional LWE assumption with almost the same advantage.

The computational indistinguishability of **Game 3** and **Game 2** follows from the same argument.

□

To conclude, note that **Game 0** always corresponds to an encryption of the challenge message μ_0 in the real experiment and **Game 4** corresponds to an encryption of the challenge message μ_1 (also in the real experiment). Hence, by the standard hybrid argument, no adversary can distinguish between encryptions of μ_0 and μ_1 with non-negligible advantage establishing the selective security of our FKHE scheme.

□

4.4 Parameter Selection

This section provides a detailed description on the selection of parameters for our scheme, so that both correctness (see Section 4.2) and security (see Section 4.3) of our scheme are satisfied.

For a family of circuits $\mathcal{C}$ of bounded depth $d_{\max}$, with the LWE dimension n, the parameters can be chosen as follows:

- The error distribution χ is typically a truncated discrete Gaussian distribution $D_{\mathbb{Z},\sqrt{n}}$ with parameter $\sigma = \sqrt{n}$. And, the error bound $B = O(\sigma\sqrt{n}) = O(n)$. For this B, the probability for a random sample from $D_{\mathbb{Z},\sqrt{n}}$ to be $\mathbf{0}$ would be $\mathrm{negl}(n)$, according to Lemma 2.1.1.

From now, we will consider the LWE modulus parameter $q = q(n, d_{\max})$, without instantiating it, to calculate the other parameters m, s, α. Later, we will instantiate q with a value which would make m, s, α satisfy the correctness and security properties.

- The parameter $m = O(n \log q)$.
- The "small" Gaussian parameter s is chosen to be $O(\sqrt{n \log q})$.
- Now, let us calculate the value of the "large" Gaussian parameter $\alpha = \alpha(n, d_{\max})$. We should choose α such that the output of the SampleLeft and the SampleRight algorithms are statistically indistinguishable from each other, when provided with the same set of inputs $\mathbf{F}$ and $\mathbf{u}$.

  The SampleRight algorithm (Algorithm 2.2) requires

$$\alpha > \|\widetilde{\mathbf{T_B}}\| \cdot \|\mathbf{R}_C\| \cdot \omega(\sqrt{\log m}) \tag{4.2}$$

  Hence, we proceed as follows:

  1. First, we calculate the value of $\|\mathbf{R}_C\|$, where $\mathbf{R}_C$ is the matrix obtained corresponding to $\mathbf{A}_C$ as the output of the $\mathsf{Eval}_{\mathrm{SIM}}$ algorithm. In $\mathsf{Eval}_{\mathrm{SIM}}$, at each step of the induction, we get the matrix

$$\mathbf{R}_w = \mathbf{R}_v \cdot \mathbf{D}_u - C(\mathbf{x}^*)_v \mathbf{R}_u$$

     corresponding to the outgoing wire w of the gate $g(u, v; w)$. We prove the following claim which would help us deduce the value of $\|\mathbf{R}_C\|_{\max}$.

     Claim 4.4.1. *Suppose that for a* NAND *gate $g(u, v; w)$, with the incoming wires u, v at depths j_0, j_1 respectively, $\|\mathbf{R}_u\|_{\max} \leq (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{j_0}$ and $\|\mathbf{R}_v\|_{\max} \leq (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{j_1}$. Then, for the outgoing wire w, the maximum norm of the matrix $\mathbf{R}_w$ would be $\|\mathbf{R}_w\|_{\max} \leq (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{\max(j_0, j_1)+1}$*

     Proof. This proof proceeds similar to the calculation of $\|\mathbf{e}_w\|_\infty$ in Claim-4.2.1. In particular,

$$\begin{aligned}
\|\mathbf{R}_w\|_{\max} &\leq m \cdot \|\mathbf{D}_u\|_{\max} \cdot \|\mathbf{R}_u\|_{\max} + \|\mathbf{R}_v\|_{\max} \\
&\leq m\Big(\frac{1}{2}\sqrt{5m} \cdot (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{j_0} + (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{j_1} \\
&\leq \big((\frac{\sqrt{5}}{2} \cdot m^{1.5}) \cdot (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{j_0} + m \cdot (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{j_1} \\
&\leq O((\frac{\sqrt{5}}{2} \cdot m^{1.5})^{\max(j_0, j_1)+1})
\end{aligned}$$

     Thus, the maximum norm of the matrix $\mathbf{R}_w$ would be $\|\mathbf{R}_w\|_{\max} \leq (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{\max(j_0, j_1)+1)}$. □

     Thus, each element of the matrix $\mathbf{R}$ corresponding to $\mathbf{A}_C$, has an absolute value of at most $\|\mathbf{R}\|_{\max} \leq (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{d_{\max}}$.

  2. We then get $\|\mathbf{R}_C\|$ as follows:

$$\|\mathbf{R}_C\| := \sup_{\mathbf{x} \in S^{m-1}} \|\mathbf{R}_C \cdot \mathbf{x}\| \leq m \cdot \|\mathbf{R}_C\|_{\max} \leq m \cdot (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{d_{\max}}$$

  3. By Theorem 2.1.10, $\|\widetilde{\mathbf{T_B}}\| = \sqrt{5}$.

4. Finally, we substitute these values in Equation 4.2 to get the value of α required for the SampleRight algorithm.

$$\alpha \geq \sqrt{5} \cdot (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{d_{\max}} \cdot m \cdot \omega(\sqrt{\log m}) \geq 2 \cdot (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{d_{\max}+1} \cdot \omega(\sqrt{\log m}) \tag{4.3}$$

The value of the parameter α required for the SampleLeft algorithm (Algorithm 2.1) is

$$\alpha \geq \|\widetilde{\mathbf{T}_{\mathbf{A}}}\| \cdot \omega(\sqrt{\log 2m}) \geq O(\sqrt{n \log q}) \cdot \omega(\sqrt{\log 2m}) \tag{4.4}$$

Thus, to satisfy both Equation 4.3 and Equation 4.4, we set the parameter

$$\alpha \geq 2 \cdot (\frac{\sqrt{5}}{2} \cdot m^{1.5})^{d_{\max}+1} \cdot \omega(\sqrt{\log m})$$

Thus, the outputs of the SampleLeft and the SampleRight algorithms will be statistically indistinguishable from each other, when provided with the same set of inputs $\mathbf{F}$ and $\mathbf{u}$. Hiding the constants, we assign $\alpha = O(n \log q)^{O(d_{\max})}$.

When our scheme is instantiated with these parameters, the correctness (see Section 4.2) of the scheme is satisfied when

$$O(B \cdot (n \log q)^{O(d_{\max})}) < q/4$$

Clearly, this condition is satisfied when $q = \tilde{O}(n \log d_{\max})^{O(d_{\max})}$. Also, this value of $q = \text{poly}(n)$, enables both the quantum reduction [Reg09b] and the classical reduction [Pei09] from $\mathsf{dLWE}_{n,q,\chi}$ to approximating lattice problems in the worst case, when n, χ chosen as described above. To conclude this section, for a given max depth $d_{\max}$ and an LWE dimension $n = n(d_{\max})$, we set the parameters for our scheme to satisfy both the correctness and security, as follows:

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

4.5 ABE scheme from LWE

For completeness we will describe our ABE construction based on LWE assumption now. As opposed to [GVW13] (where there are two *randomly chosen* matrices assigned for each wire and the transformation key for each gate consists of 4 "recoding" matrices) in our construction the matrices for each wire are assigned as *functions* of the input matrices fixed at the setup and there are **no** explicit "recoding" matrices. We now define algorithms (Params, Setup, Keygen, Enc, Dec) for a family of circuits $\mathcal{C}$ of bounded depth $d_{\max}$.

- $\mathsf{Params}(1^\lambda, d_{\max})$: Let $n = n(\lambda, d_{\max})$, $q = q(n, d_{\max})$ and $m = m(n, d_{\max})$. Set the error distribution $\chi = \chi(n)$ and the error bound $B = B(n)$. We also additionally choose two Gaussian

parameters: a "small" Gaussian parameter $s = s(n)$ (which the reader should think of as polynomially bounded), and a "large" Gaussian parameter $\alpha = \alpha(n, d_{\max})$ (which the reader should think of as growing exponentially in $d_{\max}$.) Output the global public parameters $\mathsf{pp} = (n, \chi, B, q, m, s, \alpha)$. This is implicitly given to all of the algorithms defined below[4].

- $\mathsf{Setup}(1^\ell)$:
  1. Run an instance of the trapdoor generation algorithm $\mathsf{TrapSamp}(1^n, 1^m, q)$ to obtain $(\mathbf{A}, \mathbf{T_A})$.
  2. Let the matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$ be obtained by appending columns of zeros (0^n) to the primitive matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times n(\lfloor \log q \rfloor + 1)}$ as defined in 2.1.10. Let its trapdoor be $\mathbf{T_B}$.
  3. Choose ℓ matrices $\{\mathbf{A}_i\}_{i \in [\ell]}$ at random from $\mathbb{Z}_q^{n \times m}$.
  4. Choose a vector $\mathbf{u}$ at random from $\mathbb{Z}_q^n$.
  5. Run the InvB algorithm ℓ times for each $\mathbf{A}_i$ to obtain $\mathbf{D}_i$ such that $\mathbf{B} \cdot \mathbf{D}_i = \mathbf{A}_i$.
  6. Define and output the master public key as
  $$\mathsf{mpk} := \left(\mathbf{A}, \{\mathbf{A}_i, \mathbf{D}_i\}_{i \in [\ell]}, \mathbf{B}, \mathbf{u}\right)$$
  and the master secret key as $\mathsf{msk} := (\mathbf{T_A})$.
- $\mathsf{Enc}(\mathsf{mpk}, \mathbf{x} = (x_1, \ldots, x_\ell), \mu)$: For encrypting a message $\mu \in \{0, 1\}$, do the following:
  1. Choose a vector $\mathbf{s} \in \mathbb{Z}_q^n$ at random.
  2. Choose a noise term $\mathbf{e}_0 \leftarrow \chi^m$ and compute $\psi_0 = \mathbf{A}^T\mathbf{s} + \mathbf{e}_0$.
  3. For all input wires $i \in [\ell]$:
     (a) Choose a random matrix $\mathbf{R}_i \leftarrow \{-1, 1\}^{m \times m}$ and let $\mathbf{e}_i = \mathbf{R}_i^T \mathbf{e}_0$.
     (b) Compute $\psi_i = (\mathbf{A}_i + x_i\mathbf{B})^T\mathbf{s} + \mathbf{e}_i$.
  4. Encrypt the message μ as $\tau = \mathbf{u}^T\mathbf{s} + e + \lfloor q/2 \rfloor \mu$, where $e \leftarrow \chi$.
  5. Output the ciphertext as $\mathsf{ct}_\mathbf{x} = \left(\mathbf{x}, \psi_0, \{\psi_i\}_{i \in [\ell]}, \tau\right)$.
- $\mathsf{Keygen}(\mathsf{msk}, C)$:
  1. Inductively, from input to output, consider a gate $g = (u, v; w)$. Without loss of generality assume g is a binary NAND gate. The matrices for the input wires (u, v) are fixed as $\mathbf{A}_u, \mathbf{A}_v$ by induction and the matrix for the output wire w is assigned the value
  $$\mathbf{A}_w = \mathbf{A}_v \cdot \mathbf{D}_u - \mathbf{B}$$
  2. Finally, let $\mathbf{A}_C$ be the matrix defined at the output wire by this process. Define $\mathbf{F} = [\mathbf{A} || (\mathbf{A}_C + \mathbf{B})] \in \mathbb{Z}_q^{n \times 2m}$. Compute $\mathbf{r}_{\text{out}} \in \mathbb{Z}_q^{2m}$ by running $\mathsf{SampleLeft}(\mathbf{A}, (\mathbf{A}_C + \mathbf{B}), \mathbf{T_A}, \mathbf{u}, \alpha)$, satisfying $\mathbf{F} \cdot \mathbf{r}_{\text{out}} = \mathbf{u}$.
  3. Output the secret key for the circuit C, $\mathsf{sk}_C := (C, \mathbf{r}_{\text{out}})$.
- $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_\mathbf{x})$: If $C(\mathbf{x}) = 0$, output $\perp$. Otherwise, proceed the evaluation from input to output as follows:

[4] See Sections 4.2 and 4.4 for concrete instantiation of the parameters.

1. Consider a gate $g = (u, v; w)$ carrying input values x_u, x_v and hence output value $x_w = x_u \ \mathsf{NAND} \ x_v = 1 - x_u x_v$. By induction, the user holds $\psi_u = (\mathbf{A}_u + x_u \mathbf{B})^T \mathbf{s} + \mathbf{e}_u$ and $\psi_v = (\mathbf{A}_v + x_v \mathbf{B})^T \mathbf{s} + \mathbf{e}_v$ (for some error vectors $\mathbf{e}_u$ and $\mathbf{e}_v$).

   Compute ψ_w as follows:

   $$\psi_w = \mathbf{D}_u^T \psi_v - x_v \psi_u$$

   As we show below (in Lemma 4.2.1), this new ciphertext has the form $\left(\mathbf{A}_w + x_w \mathbf{B}\right)^T \mathbf{s} + \mathbf{e}_w$ (for a small enough noise term $\mathbf{e}_w$).

2. Finally, at the output gate the user computes

   $$\psi_C = (\mathbf{A}_C + C(\mathbf{x}) \cdot \mathbf{B})^T \mathbf{s} + \mathbf{e}_C = (\mathbf{A}_C + \mathbf{B})^T \mathbf{s} + \mathbf{e}_C$$

   (since $C(\mathbf{x}) = 1$).

3. Compute $\beta = \mathbf{r}_{\text{out}}^T \cdot [\psi || \psi_C]$. As we show below (in Lemma 4.2.1), $\beta \approx \mathbf{u}^T \mathbf{s} \mod q$.

   Output $\mu = 0$ if $|\tau - \beta| < q/4$ and $\mu = 1$ otherwise.

Note: This scheme can also be easily generalised to work for any message space $\mathcal{M} = \{0,1\}^v$ for a bounded message length $v = \text{poly}(\lambda)$ in the same way as the FKHE scheme i.e by working with a matrix $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n \times v}$ instead of the vector $\mathbf{u} \in \mathbb{Z}_q^n$. And in the same way, the size of the secret keys will increase multiplicatively with v.

Chapter 5

Optimisations and Extensions for Our ABE

5.1 Optimisations

Attribute-based encryption schemes require extensive computational resources (when compared to say, an identity-based encryption scheme), because an ABE scheme is usually instantiated with large parameters to satisfy the security constraints. A few optimisations have been proposed in the literature attempting to alleviate this:

1. Outsourcing computations: Algorithms are designed so that most of the computations are "public" which do not require any secret component (msk for Keygen and sk_C for Dec) and hence can be outsourced without revealing the secret.

2. Offline computations: Most of the computations are performed "offline", when the inputs to the algorithms are not known (and say during when the device is connected to power), so that very few computations have to be performed during the "online" phase of the algorithm, after the inputs are received. For instance, the ABE encryption algorithms can be optimised by performing most of the computations before receiving both the attribute vector and the message, so that very few operations have to be performed after receiving them.

Our scheme naturally supports these optimisations.

5.1.1 Outsourcing computations

Here we discuss about outsourcing of computations in the ABE decryption algorithm [GHW11]. ABE decryption is often computationally expensive and has to be performed by the end user who does not have extensive computational resources in many instances. Here, we expect "bulk" of the computation to be done by some other entity (when given some evaluation key), but still learn nothing about the message encrypted in the ciphertext. Gorbunov et al. [GVW13] showed that decryption in their scheme can be outsourced easily: roughly, the user associated to circuit C gives the server its entire secret key sk_C, except for the secret key material associated to the final gate (used in the final step of decryption). In our scheme, we can do exactly the same thing, but the payoff is greater: since there are no such

recoding keys as a part of our secret keys, they is no need for any component to be *transmitted* to or *stored* by the server; the server only needs the circuit C associated to each user to perform the bulk of the decryption process.

Thus, if a decryptor sends a ciphertext $\mathsf{ct}_{\mathbf{x}}$ (except τ, which is not required) to the server, the server runs almost the entire Dec algorithm i.e all steps except Step 3 of our Dec algorithm (refer Section 4.5) and returns ψ_C. Now the decryptor just calculates $\beta = \mathsf{sk}_C^T \cdot [\psi_0 || \psi_C]$ and subtracts this from τ to get the message μ.

5.1.2 Offline computations

Online/Offline version for an attribute-based encryption scheme was first proposed by Hohenberger and Waters [HW14] based on discrete logarithm-esque assumptions. In those variants, bilinear pairings and exponentiations involve intensive computations followed by multiplication, sampling of random elements and addition. The online phase of their encryption algorithm involves one multiplication per attribute[1].

For the LWE based attribute-based encryption schemes, the most computation intensive operation is sampling from the Gaussian distribution whereas multiplication and addition are much less intensive. Our ABE encryption algorithm allows us perform all samplings and multiplications in the offline phase before the encryptor receives the attribute vector and the message to be encrypted. We are left with only a few additions to perform during the online phase after receiving them. In particular, we can dissect the offline and online phases of our encryption algorithm as follows:

- Offline.Enc(mpk): Repeat the following steps to create tuples of $\left\{\psi_0, \{\psi_i'\}_{i\in[\ell]}, \psi_B, \tau'\right\}$:

  1. Choose a vector $\mathbf{s} \in \mathbb{Z}_q^n$ at random.
  2. Choose a noise term $\mathbf{e} \leftarrow \chi^m$ and compute $\psi_0 = \mathbf{A}^T\mathbf{s} + \mathbf{e}$.
  3. Compute $\psi_B = \mathbf{B}^T\mathbf{s}$.
  4. For all $i \in [\ell]$:
     (a) Choose a random matrix $\mathbf{R}_i \leftarrow \{-1, 1\}^{m\times m}$ and let $\mathbf{e}_i = \mathbf{R}_i^T\mathbf{e}$.
     (b) Compute $\psi_i' = \mathbf{A}_i^T\mathbf{s} + \mathbf{e}_i$.
  5. Compute $\tau' = \mathbf{u}^T\mathbf{s} + e$, where $e \leftarrow \chi$.

  Output $\left\{\psi_0, \{\psi_i'\}_{i\in[\ell]}, \psi_B, \tau'\right\}, \mu' = \lfloor q/2 \rfloor \mu$.

- Online.Enc(mpk, $\mathbf{x} = (x_1, \ldots, x_\ell), \mu, \left\{\psi_0, \{\psi_i'\}_{i\in[\ell]}, \psi_B, \tau'\right\}, \mu'$): For encrypting a message $\mu \in \{0, 1\}$ labelled with an attribute vector $\mathbf{x} \in \{0, 1\}^\ell$, do the following:

  1. Choose one tuple from $\left\{\psi_0, \{\psi_i'\}_{i\in[\ell]}, \psi_B, \tau'\right\}$ and compute $\{\psi_i\}_{i\in[\ell]}$ as follows:

$$\psi_i = \begin{cases} \psi_i' + \psi_B & , \text{ if } x_i = 1 \\ \psi_i' & , \text{ if } x_i = 0 \end{cases}$$

  2. Encrypt the message μ as $\tau = \tau' + \mu'$, if $\mu = 1$ and as $\tau = \tau'$, if $\mu = 0$.

[1] Their representation of attributes $\mathbf{x}$ and access structures C are slightly different from our scheme. But one can consider the number of attributes to be $\log \ell$ where ℓ is the length of the attribute vector $\mathbf{x}$ of our scheme.

3. Output the ciphertext as $\mathsf{ct}_{\mathbf{x}} = \left(\mathbf{x}, \psi_0, \{\psi_i\}_{i\in[\ell]}, \tau\right)$.

Thus without any additional pre-processing, our ABE encryption algorithm inherently supports offline computations for everything, except some a additions, where $a = 1$+hamming weight of $\mathbf{x}$.

One issue with this optimisation in general is that the values calculated during the offline phase should not be leaked.

5.2 Extensions

5.2.1 Ciphertext policy ABE

In this work, we deal with key policy ABE where the secret keys sk_C are generated for policies represented by circuits C and ciphertext $\mathsf{ct}_{\mathbf{x}}$ are generated for attribute vectors $\mathbf{x}$. There is a dual version of this, the ciphertext policy ABE, where ciphertexts ct_C are generated for policies and the secret keys $\mathsf{sk}_{\mathbf{x}}$ are generated for attributes.

Almost all the known efficient constructions of ABE are for key policy ABE. A natural way to transform a key-policy ABE to a ciphertext policy ABE is as follows: first construct a universal circuit U such that $U(C, \mathbf{x}) = C(\mathbf{x})$, then give secret keys corresponding to attribute vectors $\mathbf{x}$ by running the Keygen algorithm of the key policy ABE with $U(., \mathbf{x})$ as input, and give the ciphertexts corresponding to circuits C by running the Enc algorithm of the key policy ABE with the bit string representation of C (in the way it would be input to $U(\cdot, \mathbf{x})$.

But there is an issue with this transformation since the size of the secret key is not independent of the size of the circuit. All the previously known constructions of ABE had this issue, since each of them produced sk_C with sizes proportional to the size (atleast the multiplicative size) of the circuit C. Also there is no other transformation known which transforms a key-policy ABE into a ciphertext policy ABE such that $|\mathsf{sk}_{\mathbf{x}}|$ is independent of $|C|$ and ct_C is independent of $|\mathbf{x}|$. But, since our construction generates secrete keys whose size is independent of the size of the circuit, the above transformation gives a ciphertext policy ABE from our key policy ABE with $\mathsf{ct}_C \propto |C|$ and independent of $|\mathbf{x}|$. Also, $|\mathsf{sk}_{\mathbf{x}}|$ is independent of $|C|$ (but it still depends on the depth of U).

5.2.2 Key delegation

Any user with secret key sk_{C_1} for a circuit C_1 can generate a secret key $\mathsf{sk}_{C_1\wedge C_2}$ for any circuit C_2 such that the delegatee with the delegated secret key $\mathsf{sk}_{C_1\wedge C_2}$ can decrypt a ciphertext under the attribute vector $\mathbf{x}$ can be decrypted by the delegatee only when both $C_1(\mathbf{x}) = 1$ and $C_2(\mathbf{x}) = 1$. Gorbunov et al. [GVW13] provided a restricted form of key delegation where $\mathsf{sk}_{C_1\wedge C_2}$ can be generated only when both sk_{C_1} and sk_{C_2} are known to the delegator. Our ABE scheme provides full key delegation with minor changes.

To obtain key delegation properties, we modify our Keygen algorithm slightly. On input a circuit C_1, the Keygen algorithm provides the trapdoor $\mathbf{T}_{\mathbf{F}_1}$ corresponding to $\mathbf{F}_1 = [\mathbf{A}||\mathbf{A}_{C_1}]$ as the secret key (instead of $\mathbf{r}_{\text{out}1}$) by running $2m$ instances of $\mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}_{C_1} + \mathbf{B}, \mathbf{T}_{\mathbf{A}}, \mathbf{0}, \alpha)$. Note that $\mathbf{r}_{\text{out}1}$ can be obtained

from $\mathbf{T}_{\mathbf{F}_1}$ by running $\mathsf{SampleD}(\mathbf{F}_1, \mathbf{T}_{\mathbf{F}_1}, \mathbf{u}, \alpha)$. Now delegation is done i.e $\mathsf{sk}_{C_1 \wedge C_2}$ is generated by running $3m$ instances of $\mathsf{SampleLeft}(\mathbf{F}_1, \mathbf{A}_{C_2} + \mathbf{B}, \mathbf{T}_{\mathbf{F}_1}, \mathbf{0}, \alpha)$. The delegated secret key is $\mathbf{T}_{\mathbf{F}_{12}} \in \mathbb{Z}_q^{3m \times 3m}$ where $\mathbf{F}_{12} := [\mathbf{F}_1 || (\mathbf{A}_{C_2} + \mathbf{B})] \in \mathbb{Z}_q^{n \times 3m}$. Thus the size of the secret key increases quadratically increases with the number of delegations. Please refer to [BGG+14] for further details.

Chapter 6

Compact Reusable Garbled Circuits and FE with Short Keys

In this chapter, we use our ABE scheme to construct *compact* reusable garbled circuits, and a single-key secure succinct FE scheme (SKFE) with short secret keys. By compact, we mean that the size of our garbled circuit is independent of the size of the circuit, that is being garbled. Starting from our ABE scheme, we essentially use the techniques from the work of Goldwasser et al. [GKP+13b] to first construct the SKFE scheme which produce short secret keys where as in the ABE scheme the size of sk_C is independent of the size of C (and dependent on the depth of C). Recall that their succinctness property assures that the size of the ciphertext is independent of the size of the circuits used to evaluate the attribute vector.

6.1 Definitions

6.1.1 Functional Encryption (FE)

We recall the functional encryption definition from the literature [KSW08, BSW11, GVW12] with some notational changes.

A functional encryption scheme $\mathcal{FE}$ for a class of boolean circuits with an n-bit input $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ is a tuple of four p.p.t. algorithms ($\mathsf{FE.Setup}$, $\mathsf{FE.Keygen}$, $\mathsf{FE.Enc}$, $\mathsf{FE.Dec}$) such that:

$\mathsf{FE.Setup}(1^\lambda) \to (\mathsf{mpk}, \mathsf{msk})$: The setup algorithm takes as input the security parameter 1^λ and outputs a master public key mpk and a master secret key msk.

$\mathsf{FE.Keygen}(\mathsf{msk}, C) \to \mathsf{sk}_C$: The key generation algorithm takes as input the master secret key msk and a circuit $C \in \mathcal{C}$ and outputs a key sk_C.

$\mathsf{FE.Enc}(\mathsf{mpk}, x) \to \mathsf{ct}_x$: The encryption algorithm takes as input the master public key mpk and an input $x \in \{0,1\}^*$ and outputs a ciphertext ct_x.

$\mathsf{FE.Dec}(\mathsf{sk}_C, \mathsf{ct}_x)$: The decryption algorithm takes as input a key sk_C and a ciphertext ct_x and outputs a value y.

Definition 6.1.1 (Correctness)**.** *For any polynomial $n(\cdot)$, for every sufficiently large security parameter λ, for $n = n(\lambda)$, for all $C \in \mathcal{C}_n$, and all $x \in \{0,1\}^n$,*

$$\Pr[(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda); \mathsf{sk}_C \leftarrow \mathsf{FE.Keygen}(\mathsf{msk}, C); \mathsf{ct}_x \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x) : \mathsf{FE.Dec}(\mathsf{sk}_C, \mathsf{ct}_x) = C(x)] = 1 - \mathrm{negl}(\lambda).$$

Security of Single Key Functional Encryption

Intuitively, the security of SKFE requires that an adversary should not learn anything about the input x other than the computation result $C(x)$, for some circuit C for which a key was issued (the adversary can learn the circuit C). Two notions of security have been used in the previous works: full and selective security, with the same meaning as for ABE. We present both definitions because we achieve them with different parameters of the gapSVP assumption. Our definitions are simulation-based: the security definition states that whatever information an adversary is able to learn from the ciphertext and the function keys can be simulated given only the function keys and the output of the function on the inputs.

Definition 6.1.2 (FULL-SIM- FE Security)**.** *Let $\mathcal{FE}$ be a functional encryption scheme for the family of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$. For every p.p.t. adversary $A = (A_1, A_2)$ and p.p.t. simulator S, consider the following two experiments:*

$\exp^{\mathsf{real}}_{\mathcal{FE},A}(1^\lambda)$**:**	$\exp^{\mathsf{ideal}}_{\mathcal{FE},A,S}(1^\lambda)$**:**
	1: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{FE.Setup}(1^\lambda)$
	2: $(C, \mathsf{state}_A) \leftarrow A_1(\mathsf{mpk})$
	3: $\mathsf{sk}_C \leftarrow \mathsf{FE.Keygen}(\mathsf{msk}, C)$
	4: $(x, \mathsf{state}'_A) \leftarrow A_2(\mathsf{state}_A, \mathsf{sk}_C)$
5: $\mathsf{ct}_x \leftarrow \mathsf{FE.Enc}(\mathsf{mpk}, x)$	5: $\tilde{\mathsf{ct}}_x \leftarrow S(\mathsf{mpk}, \mathsf{sk}_C, C, C(x), 1^{\lvert x\rvert})$
6: Output $(\mathsf{state}'_A, \mathsf{ct}_x)$	6: Output $(\mathsf{state}'_A, \tilde{\mathsf{ct}}_x)$

The scheme is said to be (single-key) FULL-SIM*−secure if there exists a p.p.t. simulator S such that for all pairs of p.p.t. adversaries (A_1, A_2), the outcomes of the two experiments are computationally indistinguishable:*

$$\left\{ \exp^{\mathsf{real}}_{\mathcal{FE},A}(1^\lambda) \right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{ \exp^{\mathsf{ideal}}_{\mathcal{FE},A,S}(1^\lambda) \right\}_{\lambda\in\mathbb{N}}$$

We now define selective security, which is a weakening of full security, by requiring the adversary to provide the challenge input x before seeing the public key or any other information besides the security parameter. We simply specify the difference from full security.

Definition 6.1.3 (SEL-SIM-FE Security)**.** *The same as Def. 6.1.2, but modify the game so that the first step consists of A specifying the challenge input x given only the security parameter.*

6.1.2 Reusable Garbled Circuits

We use the term reusable garbled circuit to refer to the most interesting variant of garbled circuits: the ones that can run on an arbitrary number of encoded inputs without compromising the privacy of the

circuit or of the input. We recall the definition of a reusable garbled circuit presented in [GKP+13b].

A reusable garbling scheme for a family of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$ with $\mathcal{C}_n$ being a set of boolean circuits taking n bits as input, is a tuple of p.p.t. algorithms $\mathsf{RGb} = (\mathsf{RGb.Garble}, \mathsf{RGb.Enc}, \mathsf{RGb.Eval})$ such that

$\mathsf{RGb.Garble}(1^\lambda, C) \to (\hat{C}, \mathsf{gsk})$: The garbling algorithm takes as input the security parameter λ and a circuit $C \in \mathcal{C}_n$ for some n, and outputs the garbled circuit $\hat{C}$ and a secret key gsk.

$\mathsf{RGb.Enc}(\mathsf{gsk}, x) \to \hat{x}$: The encoding algorithm takes as input $x \in \{0,1\}^n$ and outputs an encoding $\hat{x}$.

$\mathsf{RGb.Eval}(\hat{C}, \hat{x})$: The evaluation algorithm takes as input a garbled circuit $\hat{C}$, an encoding $\hat{x}$ and outputs a value y which should be $C(x)$.

Definition 6.1.4 (Correctness)**.** *For any polynomial $n(\cdot)$, for all sufficiently large security parameters λ, for $n = n(\lambda)$, for all circuits $C \in \mathcal{C}_n$ and all $x \in \{0,1\}^n$,*

$$\Pr[(\hat{C}, \mathsf{gsk}) \leftarrow \mathsf{RGb.Garble}(1^\lambda, C); \hat{x} \leftarrow \mathsf{RGb.Enc}(\mathsf{gsk}, x); y \leftarrow \mathsf{RGb.Eval}(\hat{C}, \hat{x}) : C(x) = y] = 1 - \mathrm{negl}(\lambda).$$

Definition 6.1.5 (Efficiency)**.** *There exists a universal polynomial $p = p(\lambda, n)$ (p is the same for all classes of circuits $\mathcal{C}$) such that for all input sizes n, security parameters λ, for all boolean circuits C with n bits of input, for all $x \in \{0,1\}^n$,*

$$\Pr[(\hat{C}, \mathsf{gsk}) \leftarrow \mathsf{RGb.Garble}(1^\lambda, C) : |\mathsf{gsk}| \le \mathrm{poly}(\lambda, n) \text{ and } \mathit{runtime}(\mathsf{RGb.Enc}(\mathsf{gsk}, x)) \le \mathrm{poly}(\lambda, n)] = 1.$$

Note that since $\mathsf{RGb.Enc}$ is a p.p.t. algorithm, it suffices to ensure that $|\mathsf{gsk}| \le \mathrm{poly}(\lambda, n)$ and obtain that $\mathsf{RGb.Enc}$'s runtime is also at most a polynomial. We prefer to keep the runtime of $\mathsf{RGb.Enc}$ in the definition as well for clarity.

Security of Reusable Garbled Circuits

Here, we present the security definition of the reusable garbled circuits, as presented in [GKP+13b].

Definition 6.1.6. *(Input and circuit privacy with reusability)*

Let RGb be a garbling scheme for a family of circuits $\mathcal{C} = \{\mathcal{C}_n\}_{n\in\mathbb{N}}$. For a pair of p.p.t. algorithms $A = (A_1, A_2)$ and a p.p.t. simulator $S = (S_1, S_2)$, consider the following two experiments:

$\exp^{\mathsf{real}}_{\mathsf{RGb},A}(1^\lambda)$:	$\exp^{\mathsf{ideal}}_{\mathsf{RGb},A,S}(1^\lambda)$:
1: $(C, \mathsf{state}_A) \leftarrow A_1(1^\lambda)$	1: $(C, \mathsf{state}_A) \leftarrow A_1(1^\lambda)$
2: $(\mathsf{gsk}, \hat{C}) \leftarrow \mathsf{RGb.Garble}(1^\lambda, C)$	2: $(\tilde{C}, \mathsf{state}_S) \leftarrow S_1(1^\lambda, 1^{\lvert C\rvert})$
3: $\alpha \leftarrow A_2^{\mathsf{RGb.Enc}(\mathsf{gsk},\cdot)}(C, \hat{C}, \mathsf{state}_A)$	3: $\alpha \leftarrow A_2^{\mathsf{O}(\cdot,C)[[\mathsf{state}_S]]}(C, \tilde{C}, \mathsf{state}_A)$
4: Output α	4: Output α

In the above, $\mathsf{O}(\cdot, C)[[\mathsf{state}_S]]$ is an oracle that on input x from A_2, runs S_2 with inputs $C(x)$, $1^{|x|}$, and the latest state of S; it returns the output of S_2 (storing the new simulator state for the next invocation).

We say that the garbling scheme RGb is input- and circuit-private with reusability if there exists a p.p.t. simulator S such that for all pairs of p.p.t. adversaries $A = (A_1, A_2)$, the following two distributions

are computationally indistinguishable:

$$\left\{ \exp^{\mathsf{real}}_{\mathsf{RGb},A}(1^\lambda) \right\}_{\lambda\in\mathbb{N}} \overset{c}{\approx} \left\{ \exp^{\mathsf{ideal}}_{\mathsf{RGb},A,S}(1^\lambda) \right\}_{\lambda\in\mathbb{N}}.$$

We can see that this security definition enables reusability of the garbled circuit: A_2 is allowed to make as many queries for input encodings as it wants.

6.2 Constructions

6.2.1 Single-Key Functional Encryption and Reusable Garbled Circuits

In this section we show the gain in efficiency in the size of the secret key for the SKFE scheme that we obtain by using our ABE schemes.

From [GKP+13b], we know how to obtain a Single-Key Functional Encryption from:

1. Attribute-based Encryption
2. Fully-Homomorphic Encryption
3. "one-time" Garbled Circuits

Theorem 6.2.1 ([GKP+13b]). *There is a (fully/selectively secure) single-key functional encryption scheme $\mathcal{FE}$ for any class of circuits $\mathcal{C}$ that take ℓ bits of input and produce a one-bit output, assuming the existence of (1) $\mathcal{C}$-homomorphic encryption scheme, (2) a (fully/selectively) secure ABE scheme for a related class of predicates and (3) Yao's Garbling Scheme, where:*

1. *The size of the secret key is $2 \cdot \alpha \cdot \mathsf{abe.keysize}$, where $\mathsf{abe.keysize}$ is the size of the ABE key for circuit performing homomorphic evaluation of C and outputting a bit of the resulting ciphertext.*
2. *The size of the ciphertext is $2 \cdot \alpha \cdot \mathsf{abe.ctsize}(\ell \cdot \alpha + \gamma) + \mathrm{poly}(\lambda, \alpha, \beta)$*

where (α, β, γ) denote the sizes of the FHE (ciphertext, secret key, public key), respectively. $\mathsf{abe.keysize}, \mathsf{abe.ctsize}(k)$ are the size of ABE secret key, ciphertext on k-bit attribute vector and λ is the security parameter.

We use the same transformation by replacing the ABE scheme of [GVW13] with our ABE scheme with short secret keys. Since FHE (and Yao's Garbled Circuits) can also be instantiated assuming the sub-exponential hardness of LWE ([BV11, BGV12]), we obtain the following corollaries.

Corollary 6.2.2. *Combining our short secret key ABE construction (Theorem-4.3.1 and Section 4.5) and Theorem-6.2.1, we obtain a single-key functional encryption scheme for a circuit class $\mathcal{C}$ with depth at most $d_{\max}$, where the secret key size is some $\mathrm{poly}(d_{\max}, \lambda)$ and λ is the security parameter.*

6.2.2 Compact garbled circuits

From any SKFE scheme, [GKP+13b] show how to construct a compact reusable garble circuits. We apply our results to get the optimal construction of reusable garbled circuits.

Theorem 6.2.3 ([GKP+13b]). *There exists a reusable garbling scheme for any class of circuits $\mathcal{C}$ that take ℓ bits of input, assuming the existence (1) symmetric-encryption algorithm, (2) a single-key functional encryption for $\mathcal{C}$, where:*

1. *The size of the secret key is* $|C| + \mathsf{fe.keysize} + \mathrm{poly}(\lambda)$*, where* $\mathsf{fe.keysize}$ *is the size of the FE key for circuit performing symmetric-key decryption and evaluation of* C*.*
2. *The size of the ciphertext is* $\mathsf{fe.ctsize}(\lambda + \ell)$

where $\mathsf{fe.ctsize}(\lambda + \ell)$ *is the size of FE ciphertext on* $\lambda + \ell$*-bit input.*

Corollary 6.2.4. *From Corollary-6.2.2 and Theorem-6.2.3, we obtain a reusable garbled circuits scheme for any class of polynomial-size circuits with depth at most* $d_{\max}$*, where the secret key size is* $|C| + \mathrm{poly}(d_{\max}, \lambda)$*.*

Construction

We now partially unwind the construction of the garbling scheme in [GKP+13b], which shows the correctness of Theorem 6.2.3 and Corollary 6.2.4. The following construction uses the algorithms of our attribute-based encryption scheme, the fully-homomorphic encryption scheme in [**?**, BGV12] and a semantically secure symmetric-key encryption scheme as black boxes. We now define the algorithms $(\mathsf{RGb.Garble}, \mathsf{RGb.Enc}, \mathsf{RGb.Eval})$ for a family of circuits $\mathcal{C}$ of bounded depth $d_{\max}$.

- We consider the existence of the following three sets of algorithms:
  1. An attribute-based encryption scheme $\mathcal{ABE}(\mathsf{ABE.Setup}, \mathsf{ABE.Keygen}, \mathsf{ABE.Enc}, \mathsf{ABE.Dec})$
  2. A (levelled) fully homomorphic encryption scheme $\mathcal{FHE}(\mathsf{FHE.Keygen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval})$
  3. A semantically secure symmetric-key encryption scheme $\mathcal{E}(\mathsf{E.Enc}, \mathsf{E.Dec})$
- $\mathsf{RGb.Garble}(1^\lambda, C)$
  1. Run 2κ instances of $\mathsf{ABE.Setup}(1^\lambda)$ to obtain $(\mathsf{fmpk}_{i,0}, \mathsf{fmsk}_{i,1})$ and $(\mathsf{fmpk}_{i,0}, \mathsf{fmsk}_{i,1})$, for $i \in [\kappa]$, where κ is the length of FHE ciphertexts.
  2. Generate the secret key for the symmetric-key encryption scheme as $\mathsf{sk} \leftarrow \mathsf{E.Keygen}(1^\lambda)$.
  3. Generate an encrypted form of the circuit C as $C_E := \mathsf{E.Enc}(\mathsf{sk}, C)$.
  4. Define the universal circuit $U_E(\mathsf{sk}, x)$ as the one which performs the computation, for some sk, x:
     (a) Compute $C = \mathsf{E.Dec}(\mathsf{sk}, C_E)$.
     (b) Run C on x.
  5. The (reusable) garbled circuit for C is the collection of ABE secret keys corresponding to the FHE evaluation circuits that evaluate the input ciphertexts to produce the ith bit of the output ciphertext.
     $$\hat{C} := \big((\mathsf{fsk}_{1,0}, \mathsf{fsk}_{1,1}), \ldots, (\mathsf{fsk}_{\kappa,0}, \mathsf{fsk}_{\kappa,1})\big)$$
     where $\mathsf{fsk}_{i,0} \leftarrow \mathsf{ABE.Keygen}(\mathsf{fmsk}_{i,0}, \neg\mathsf{FHE.Eval}_i^{U_E})$, $\mathsf{fsk}_{i,1} \leftarrow \mathsf{ABE.Keygen}(\mathsf{fmsk}_{i,1}, \mathsf{FHE.Eval}_i^{U_E})$ for $i \in [\kappa]$.
  6. The component $\mathsf{gsk} := \big((\mathsf{fmpk}_{1,0}, \mathsf{fmpk}_{1,1}), \ldots, (\mathsf{fmpk}_{\kappa,0}, \mathsf{fmpk}_{\kappa,1}), \mathsf{sk}\big)$ is also output by this algorithm. This is used in the generation of input encodings.[1]

[1] Note that, due to the impossibility result proved in [GKP+13b], input encodings cannot be generated for a reusable garbled circuit without the knowledge of a "secret" component. In our scheme, the knowledge of gsk allows one to generate input encodings to be evaluated by $\hat{C}$.

- $\mathsf{RGb.Enc}(1^\lambda, \mathsf{gsk}, x)$

  1. Run an instance of the key generation algorithm of the fully homomorphic encryption scheme $(\mathsf{hpk}, \mathsf{hsk}) \leftarrow \mathsf{FHE.Keygen}(1^\lambda)$. And generate the encryption of each bit of the input (sk, x) to U_E as follows:

$$\psi := \left(\{\psi_i \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, \mathsf{sk}_i)\}_{i\in[1,\ldots,|\mathsf{sk}|]}, \{\psi_i \leftarrow \mathsf{FHE.Enc}(\mathsf{hpk}, x_i)\}_{i\in[|\mathsf{sk}|+1,\ldots,|\mathsf{sk}|+n]}\right)$$

  2. Run Yao's garbling scheme on the FHE decryption circuit with its secret key hardcoded $(\mathsf{FHE.Dec}(\mathsf{hsk}, \cdot) : \{0,1\}^\kappa \to \{0,1\})$ to produce a (one-time) garbled circuit $\hat{D}$, together with 2κ labels L_i^b, for $b \in \{0,1\}, i \in [\kappa]$.
  3. Now, the input encoding is $\hat{x} := \left(\{c_i\}_{i\in[\kappa]}, \hat{D}\right)$ where

$$c_i = (c_{i,0}, c_{i,1}) := \left(\mathsf{ABE.Enc}(\mathsf{fmpk}_{i,0}, (\mathsf{hpk}, \psi), L_i^0), \mathsf{ABE.Enc}(\mathsf{fmpk}_{i,0}, (\mathsf{hpk}, \psi), L_i^1)\right)$$

- $\mathsf{RGb.Eval}(\hat{C}, \hat{x})$

  1. Obtain the labels $L_i^{d_i}$ corresponding to the bits d_i of the evaluated FHE ciphertext ($d_i = \mathsf{FHE.Eval}_i^{U_E}(\mathsf{hpk}, \psi)$) as follows:
     - First run $\mathsf{ABE.Dec}(\mathsf{fsk}_{i,0}, c_{i,0})$. If the output is not $\perp$, then $d_i = 0$ and hence L_i^0 has been obtained as output.
     - If the output of the previous step is $\perp$, run $\mathsf{ABE.Dec}(\mathsf{fsk}_{i,1}, c_{i,1})$. If this output is not $\perp$, then $d_i = 1$ and hence L_i^1 has been obtained.
     - If both the ABE decryption algorithms output $\perp$, output $\perp$.
  2. Evaluate the garbled circuit $\hat{D}$ of the FHE decryption algorithm with the obtained labels $\{L_i^{d_i}\}_{i\in[\kappa]}$ to obtain $C(x)$.

$$\mathsf{Gb.Eval}(\hat{D}, L_1^{d_i}, \ldots, L_\kappa^{d_\kappa}) = \mathsf{FHE.Dec}(\mathsf{hsk}, d_1 \ldots d_\kappa) = U_E(\mathsf{sk}, x) = C(x)$$

Chapter 7

Other Applications of our ABE

Other than single-key functional encryption and compact garbled circuits, our ABE scheme with short secret keys also results in efficiency improvements in some other interesting primitives like attribute-based fully homomorphic encryption and verifiable computation.

7.1 Attribute-Based Fully Homomorphic Encryption (ABFHE)

An attribute-based fully homomorphic encryption (ABFHE) scheme has all of the functionality of ABE, but also allows messages encrypted under the same attribute vector to be processed homomorphically by anyone (without any public or private keys), such that the final ciphertexts can still be decrypted by any party that was entitled to decrypt the original ciphertexts. Recently, Gentry, Sahai and Waters [GSW13] constructed the first ABFHE scheme, building on the ABE scheme of Gorbunov, Vaikuntanathan and Wee [GVW13].

An ABFHE scheme is defined as follows:

- The four algorithms $\mathsf{Setup}, \mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec}$ defined as in the $\mathcal{ABE}$ scheme.
- $\mathsf{Eval}(\mathsf{mpk}, \mathbf{x}, f, \mathsf{ct}_{\mathbf{x}}^1, \ldots, \mathsf{ct}_{\mathbf{x}}^t) \rightarrow \mathsf{ct}_{\mathbf{x}}^f$: The algorithm, associated to some family function $\mathbb{F}$. For any function $f \in \mathbb{F}$, Eval homomorphically evaluates f on ciphertexts $\left\{\mathsf{ct}_{\mathbf{x}}^i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu_i)\right\}_{i \in [t]}$ generated under the same attribute vector $\mathbf{x} \in \{0,1\}^\ell$.

Correctness

For any $(\mathsf{msk}, \mathsf{mpk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^{d_{\max}})$, $\mathsf{sk}_C \leftarrow \mathsf{Keygen}(\mathsf{msk}, C)$ for any C of depth atmost $d_{\max}$, for any t and ciphertexts $\left\{\mathsf{ct}_{\mathbf{x}}^i \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathbf{x}, \mu_i)\right\}_{i \in [t]}$ generated under the same attribute vector $\mathbf{x} \in \{0,1\}^\ell$ for which $C(\mathbf{x}) = 1$, for any t-ary function $f \in \mathbb{F}$, $\mathsf{ct}_{\mathbf{x}}^f \leftarrow \mathsf{Eval}(\mathsf{mpk}, \mathbf{x}, f, \mathsf{ct}_{\mathbf{x}}^1, \ldots, \mathsf{ct}_{\mathbf{x}}^t)$ is a ciphertext that satisfies $\mathsf{Dec}(\mathsf{sk}_C, \mathsf{ct}_{\mathbf{x}}^f) = f(\mu_1, \ldots, \mu_t)$.

Security

The notions of security for ABFHE are the same as for ABE (refer Section 2.2.1) except that the adversary will have access to the (public) algorithm Eval (in addition to the Enc algorithm), which it could

exploit during the security game. Hence, ABFHE schemes can either be *selectively* secure or be *adaptively* secure, depending on when the adversary provides the challenge attribute vector $\mathbf{x}$.

By applying Gentry et al.'s techniques to our ABE scheme, we obtain an ABFHE scheme with short secret keys, with security based on LWE. Gentry et al. actually provided a general compiler to transform *any* LWE-based ABE scheme with suitable properties into an LWE-based ABFHE scheme, the properties being:

1. **Property 1 (Ciphertext and decryption key vectors)**: Decryption keys $\mathbf{s}_{C,\mathbf{x}}$ and ciphertexts $\mathsf{ct}_{\mathbf{x}}$ are vectors over $\mathbb{Z}_q$. The first coefficient of each $\mathbf{s}_{C,\mathbf{x}}$ is 1.
2. **Property 2 (Small Dot Product)**: If $\mathsf{ct}_{\mathbf{x}}$ encrypts 0, then $\langle \mathsf{ct}_{\mathbf{x}}, \mathbf{s}_{C,\mathbf{x}} \rangle$ is "small".
3. **Property 3 (Security)**: Encryptions of 0 are indistinguishable from uniform vectors over $\mathbb{Z}_q$ (under LWE).

This transformation results in a (levelled) FHE scheme, and hence $\mathbb{F}$ consists of all circuits of depth $d_{\max}{}'$ for some $d_{\max}{}'$. The decryption keys in our LWE-based ABE scheme do not natively have the form needed by the compiler. But, one can re-interpret our decryption as a two-step process in the same way [GSW13] treated the decryption in [GVW13] when it faced the same problem.

The first step involves the "recoding of recodings" process. This is a recursive process used to calculate the recoding key $\mathbf{t}_{C,\mathbf{x}} \in \mathbb{Z}_q^{m+m\ell}$ transforming encodings of the bits of the attribute vector i.e $[\psi_0||\psi_1||\cdots||\psi_\ell] \in \mathbb{Z}_q^{m+m\ell}$ to the encryption of the message $\tau \in \mathbb{Z}_q$, where $\psi_0, \psi_1, \ldots, \psi_\ell, \tau$ form part of some ciphertext $\mathsf{ct}_{\mathbf{x}}$.

This process proceeds (from output to input) as follows:

1. The secret key sk_C for the circuit C, is the "recoding" element $\mathbf{r}_{\text{out}} = [\mathbf{r}_0^T||\mathbf{r}_C^T]^T \in \mathbb{Z}_q^{2m}$ that is used to transform $[\psi_0||\psi_C]$ into τ i.e $[\mathbf{r}_0^T||\mathbf{r}_C^T] \cdot [\psi_0^T||\psi_C^T]^T = \tau$.
2. For any NAND gate $g(u, v; w)$, ψ_w can be calculated from ψ_u, ψ_v as $\psi_w = [-x_v\mathbf{I}_m||\mathbf{D}_u] \cdot [\psi_u^T||\psi_v^T]^T$. Thus, the recoding element for this gate is $[-x_v\mathbf{I}_m||\mathbf{D}_u]$ where $-x_v\mathbf{I}_m$ is the recoding matrix for ψ_u and $\mathbf{D}_u$ is the recoding matrix for ψ_v. We recursively perform this operation from the output level to the input level so tha at the input level we obtain the big recoding matrix $\mathbf{Q} := (\mathbf{Q}_1||\cdots||\mathbf{Q}_\ell) \in \mathbb{Z}_q^{m\times m\ell}$ such that $[\mathbf{Q}_1||\cdots||\mathbf{Q}_\ell] \cdot [\psi_1^T||\cdots||\psi_\ell^T]^T = \psi_C$.
3. Now the final recoding element $\mathbf{t}_{C,\mathbf{x}}$ also performs the final transformation of $[\psi_0||\psi_C]$ to β i.e $\mathbf{t}_{C,\mathbf{x}} := [\mathbf{r}_0^T||\mathbf{r}_{\text{out}}^T \cdot \mathbf{Q}] \in \mathbb{Z}_q^{m+m\ell}$.
4. The secret key for the ABFHE scheme is $\mathbf{s}_{C,\mathbf{x}} = (1, -\mathbf{t}_{C,\mathbf{x}})$. Also the ciphertext $\mathsf{ct}_{\mathbf{x}}$ is viewed as a single long vector $[\tau^T||\psi_0^T||\psi_1^T||\cdots||\psi_\ell^T]^T \in \mathbb{Z}_q^{1+m+m\ell}$.

Thus, rather than decrypting "incrementally" gate-by-gate, we "holistically" view decryption as applying an overall linear transformation (in particular, a dot product) to the ciphertext vector $\mathsf{ct}_{\mathbf{x}}$: we thereby obtain a "sub-key" $\mathbf{s}_{C,\mathbf{x}}$, a vector that depends on sk_C and $\mathbf{x}$, that represents this overall linear transformation from input encodings to the encryption of the message bit.

In the second step of decryption, we simply determine whether the dot product $\langle \mathsf{ct}_{\mathbf{x}}, \mathbf{s}_{C,x} \rangle$ is small or not.

Viewed in this way, our ABE scheme has all of the properties required by the compiler in [GSW13], and hence we obtain ABFHE with short secret keys.

7.2 Verifiable Computation

As mentioned earlier, a verifiable computation protocol allows a computationally weak client to *verifiably* outsource computations to one or more workers. The efficiency requirement of these protocols is that the computational complexity involved in outsourcing the computation and in the verification of the result(s) returned by the worker(s) should be significantly less than the complexity involved in the original computation by the client himself. This problem of (non-interactive) verifiable computation was initially studied in the works of Kilian [Kil92], Micali [Mic94] and Goldwasser, Kalai and Rothblum [GKR08]. Later, after several other results, Parno, Raykova and Vaikuntanathan [PRV12] showed that one could use a key-policy attribute-based encryption scheme to perform verifiable computation with public delegation and public verification. After the pre-processing stage (which is required for almost all the known results on verifiable computation) is performed by some user, the delegation of the computation and the verification of the result returned by the worker can be performed by any user. Without these two properties, these tasks would be restricted to the user who performed the pre-processing stage.

Our ABE scheme results in a verifiable computation protocol for any polynomial size circuits with short evaluation keys, along with the public delegation and public verification properties.

VC protocol from ABE

Intuition: To calculate $C(\mathbf{x})$, the client generates ABE encryptions for $(\mathbf{x}, m_0)$ and $(\mathbf{x}, m_1)$ for some m_0, m_1. The evaluation key is the pair of ABE secret keys for $C, \bar{C}$. Thus worker with the evaluation key can correctly decrypt only the ciphertext corresponding to $(\mathbf{x}, m_0)$ when $\bar{C}(\mathbf{x}) = 1$ i.e $C(\mathbf{x}) = 0$. On the other hand, when $C(\mathbf{x}) = 0$, he can correctly decrypt only the ciphertext corresponding to $(\mathbf{x}, m_1)$ A verifiable computation protocol for a family of circuits $\mathcal{C}$, $\mathcal{VC}(\mathsf{VC.Keygen}, \mathsf{VC.ProbGen}, \mathsf{VC.Compute}, \mathsf{VC.Verify})$ can be obtained from an attribute-based encryption scheme for a family of circuits $\mathcal{C}'$ (which includes $\mathcal{C}$ and the complements of all the circuits in $\mathcal{C}$) $\mathcal{ABE}(\mathsf{ABE.Setup}, \mathsf{ABE.Enc}, \mathsf{ABE.Keygen}, \mathsf{ABE.Dec})$ as follows:

- $\mathsf{VC.Keygen}(C, 1^\lambda)$
  1. Run two instances of $\mathsf{ABE.Setup}(1^\lambda)$ to obtain $(\mathsf{mpk}_0, \mathsf{msk}_0), (\mathsf{mpk}_1, \mathsf{msk}_1)$.
  2. Set $\mathsf{sk}_{\bar{C}} \leftarrow \mathsf{ABE.Keygen}(\mathsf{msk}_0, \bar{C})$ and $\mathsf{sk}_C \leftarrow \mathsf{ABE.Keygen}(\mathsf{msk}_1, C)$, where $\bar{C}$ is the complement of the circuit C.
  3. The public key for the verifiable computation protocol for the circuit C is $\mathsf{pk}_C := (\mathsf{mpk}_0, \mathsf{mpk}_1)$ and the evaluation key is $\mathsf{ek}_C := (\mathsf{sk}_C, \mathsf{sk}_{\bar{C}})$.
- $\mathsf{VC.ProbGen}(\mathsf{pk}_C, \mathbf{x})$
  1. Generate a pair of random messages $m_0, m_1 \xleftarrow{\$} \{0,1\}^\lambda$.

2. Compute ciphertexts $c_0 \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}_0, \mathbf{x}, m_0), c_1 \leftarrow \mathsf{ABE.Enc}(\mathsf{mpk}_1, \mathbf{x}, m_1)$.
3. Output $\big(c_{C,\mathbf{x}} := (c_0, c_1), \mathsf{vk}_{C,\mathbf{x}} := (\mathsf{OWF}(m_0), \mathsf{OWF}(m_1))\big)$.

- $\mathsf{VC.Compute}(\mathsf{ek}_C, c_{C,\mathbf{x}})$
  1. Parse ek_C as $(\mathsf{sk}_C, \mathsf{sk}_{\bar{C}})$ and $c_{C,\mathbf{x}}$ as (c_0, c_1).
  2. Compute $d_0 \leftarrow \mathsf{ABE.Dec}(\mathsf{sk}_{\bar{C}}, c_0)$ and $d_1 \leftarrow \mathsf{ABE.Dec}(\mathsf{sk}_C, c_1)$.
  3. Output $d := (d_0, d_1)$.
- $\mathsf{VC.Verify}(\mathsf{vk}_{C,\mathbf{x}}, d)$
  1. Parse $\mathsf{vk}_{C,\mathbf{x}}$ as $(\mathsf{OWF}(m_0), \mathsf{OWF}(m_1))$ and d as (d_0, d_1).
  2. If $\mathsf{OWF}(d_0) = \mathsf{OWF}(m_0)$, output 0, else if $\mathsf{OWF}(d_1) = \mathsf{OWF}(m_1)$, output 1, else output $\perp$.

Theorem 7.2.1. *There exists a verifiable computation protocol for a family of circuits $\mathcal{C}$ containing polynomial size circuits of depth bounded $d_{\max}$, with the properties of public delegation and public verification. Also, the size of the evaluation keys for any circuit $C \in \mathcal{C}$ is $O(\lambda, d_{\max})$, and hence independent of the size of C.*

Proof. Our attribute-based encryption scheme in Chapter 5 is one for a family of polynomial size circuits $\mathcal{C}$ of depth bounded by $d_{\max}$. This implies the existence of a verifiable computation protocol for the family of circuits $\mathcal{C}$ [PRV12].

From the above construction of VC from ABE, we can see that the size of the evaluation key for a circuit $C \in \mathcal{C}$ is the sum of the size of secret keys produced by the ABE scheme for $C, \bar{C}$. Since, the size of secret keys in our ABE scheme for any circuit $C \in \mathcal{C}$ is $O(\lambda, d_{\max})$, the size of the evaluation key for C in the resulting VC protocol is also $O(\lambda, d_{\max})$. □

Chapter 8

ABE with Short Secret Keys from Ring-LWE

8.1 Ring LWE

The ring-LWE problem, introduced by Lyubashevsky, Peikert and Regev [LPR10], is the ring analogue of the standard learning with errors (LWE) problem [Reg09a]. Informally, in a ring $R = \mathbb{Z}[X]/(f(X))$ for a monic irreducible polynomial $f(X)$ of degree n and for an integer q defining the quotient ring $R_q := R/qR = \mathbb{Z}_q[X]/(f(X))$, the ring-LWE problem is to distinguish the tuples of $(a_i, b = a_i \cdot s + e_i) \in R_q \times R_q$ from uniformly random tuples in the same domain, where $a_i \in R_q$ are uniformly random and independent, $s \xleftarrow{\$} R_q$ is the secret (which stays the same for all the tuples) and $e_i \in R$ are the noise terms which are *short* and independent.

8.2 Preliminaries

Recently, a toolkit for ring-LWE was provided by Lyubashevsky, Peikert and Regev [LPR13], which includes fast algorithms for the important cryptographic operations. Their algorithms work for arbitrary cyclotomic polynomials $\Phi_m(X)$ (for any m). They use canonical embedding for defining the geometric norms (such as norms and inner products). This embedding maps every element of $K = \mathbb{R}[X]/(\Phi_m(X))$ to a set of n complex numbers, where $n = \varphi(m)$ and it ensures that addition and multiplication of two ring elements are component-wise. Also, the work of Micciancio and Peikert [MP12] ensures that all the algorithms, such as $\mathsf{TrapSamp}, \mathsf{SampleD}$ and hence $\mathsf{SampleLeft}, \mathsf{SampleRight}$, that we used in our LWE scheme work have analogues for the ring-LWE version. We refer the ring analogues with a subscript R.

8.2.1 Notations

For some constant $c \in \mathbb{Z}_q$, we will refer to $\vec{c} \in R_q$ the ring element with all of its coordinates to be that constant c. Now, if all coordinates of an element $a \in R_q$ have to be multiplied (or added) with a particular constant c, it can be performed by multiplying (or adding) a with $\vec{c}$. But we will postpone these details to the analysis and use $c \cdot a$ (or $c + a$) in the scheme skipping the arrow above the number.

8.2.2 Additional algorithms

We will also use the following two algorithms.

- $\mathsf{Powersof2}(a)$: The algorithm takes an element $a \in R_q$ as input and outputs $\mathbf{a}' = [2^0 \cdot a, 2^1 \cdot a, \ldots, 2^{y-1} \cdot a] \in R_q^{1\times y}$, where $y = \lfloor \log q \rfloor + 1$. Given a matrix $\mathbf{A} \in R_q^{n\times n}$ as input, $\mathsf{Powersof2}(\mathbf{A})$ returns a matrix in $\mathbb{Z}_q^{n\times ny}$, with $\mathsf{Powersof2}$ algorithm applied to each element of $\mathbf{A}$.
- $\mathsf{BD}(a)$: The algorithm takes in an element $a \in R_q$ and outputs a vector $\mathbf{a}^T = [a_0, \ldots, a_{y-1}]] \in R_q^y$, where a_i is a ring element whose coordinates are the ith bits (when ordered from LSB to MSB) in the binary decomposition of the coordinates of a. Given a matrix $\mathbf{A} \in R_q^{n\times m}$ as input, $\mathsf{BD}(\mathbf{A})$ returns a $ny \times m$ matrix with BD applied to each element of $\mathbf{A}$.

The primitive vector $\mathbf{b}$ is defined as $\mathbf{b} = \mathsf{Powersof2}(1) = (b_1, b_2, \ldots, b_y) \in R_q^y$, where $b_i = 2^i$. We note that for any vector $\mathbf{a} \in R_q^y$: $\mathsf{BD}(\mathbf{a}) \cdot \mathbf{b} = \mathbf{a}$.

8.3 Construction

Note that n,m used in this section are different from those used in the LWE scheme. We now define algorithms $\mathcal{ABE}_R(\mathsf{Params}_R, \mathsf{Setup}_R, \mathsf{Enc}_R, \mathsf{Keygen}_R, \mathsf{Dec}_R)$ for a family of circuits $\mathcal{C}$ of bounded depth $d_{\max}$.

- $\mathsf{Params}_R(1^\lambda, d_{\max})$: Let us consider the use of the ring $R = \mathbb{Z}[X]/(\Phi_m(X))$, where $\Phi_m(X)$ is the mth cyclotomic polynomial of degree $n = \varphi(m)$. Let $R_q = R/qR$ be the ring modulo the ideal generated by an integer q. Here, $m = m(\lambda, d_{\max})$, $q = q(d_{\max})$. Set the error distribution $\chi = \chi(n)$, which is a continuous LWE error distribution over K and the error bound $B = B(n)$. We also additionally choose two Gaussian parameters: a "small" Gaussian parameter $s = s(n)$ (which the reader should think of as polynomially bounded), and a "large" Gaussian parameter $\alpha = \alpha(n, d_{\max})$ (which the reader should think of as growing exponentially in $d_{\max}$.) Output the global public parameters $\mathsf{pp} = (n, \chi, B, q, m, s, \alpha)$. This is implicitly given to all of the algorithms defined below[1].
- $\mathsf{Setup}_R(1^\ell)$:
  1. Run the trapdoor generation algorithm $\mathsf{TrapSamp}_R(1^y, R_q)$ to obtain $(\mathbf{a}, \mathbf{t_a})$, where $y = \lfloor \log q \rfloor + 1$.
  2. Choose ℓ vectors $\{\mathbf{a}_i\}_{i\in[\ell]}$ at random from R_q^y.
  3. Choose u at random from R_q.
  4. Define and output the master public key as
  $$\mathsf{mpk} := \left(\mathbf{a}, \{\mathbf{a}_i\}_{i\in[\ell]}, \mathbf{b}, u\right)$$
  and the master secret key as $\mathsf{msk} := (\mathbf{t_a})$.
- $\mathsf{Enc}_R(\mathsf{mpk}, \mathbf{x} = (x_1, \ldots, x_\ell), \mu)$: For encrypting a message $\mu \in \{0, 1\}$, do the following:

[1] See Sections 8.4 for concrete instantiation of the parameters.

1. Choose a vector $s \in R_q$ at random.
2. Choose a noise term $\mathbf{e} \leftarrow \chi^y$ and compute $\psi_0 = s \cdot \mathbf{a} + \mathbf{e}$.
3. For all input wires $i \in [\ell]$:
   (a) Choose a random matrix $\mathbf{R}_i \leftarrow \{-1, 1\}^{y \times y}$ and let $\mathbf{e}_i = \mathbf{R}_i^T \mathbf{e}$.
   (b) Compute $\psi_i = s \cdot (\mathbf{a}_i + x_i \cdot \mathbf{b}) + \mathbf{e}_i$.
4. Encrypt the message μ as $\tau = s \cdot u + e + \lfloor q/2 \rfloor \mu$, where $e \leftarrow \chi$.
5. Output the ciphertext as $\mathsf{ct}_{\mathbf{x}} = \left(\mathbf{x}, \psi_0, \{\psi_i\}_{i \in [\ell]}, \tau\right)$.

- $\mathsf{Keygen}_R(\mathsf{msk}, C)$:

1. Inductively, from input to output, consider a gate $g = (u, v; w)$. Without loss of generality assume g is a binary NAND gate. The matrices for the input wires (u, v) are fixed as $\mathbf{a}_u, \mathbf{a}_v$ by induction and the matrix for the output wire w is assigned the value

$$\mathbf{a}_w = \mathsf{BD}(\mathbf{a}_u) \cdot \mathbf{a}_v - \mathbf{b}$$

2. Finally, let $\mathbf{a}_{\text{out}}$ be the vector defined at the output wire by this process. Define $\mathbf{f} = [\mathbf{a} || (\mathbf{a}_{\text{out}} + \mathbf{b})] \in R_q^{2y}$. Compute $\mathbf{r}_{\text{out}} \in R_q^{2m}$ by running $\mathsf{SampleLeft}_R(\mathbf{a}, (\mathbf{a}_{\text{out}} + \mathbf{b}), \mathbf{t}_{\mathbf{a}}, u, \alpha)$, satisfying $\mathbf{f} \cdot \mathbf{r}_{\text{out}} = u$.
3. Output the secret key for the circuit C, $\mathsf{sk}_C := (C, \mathbf{r}_{\text{out}})$.

- $\mathsf{Dec}_R(\mathsf{sk}_C, \mathsf{ct}_{\mathbf{x}})$: If $C(\mathbf{x}) = 0$, output $\perp$. Otherwise, proceed the evaluation from input to output as follows:

1. Consider a gate $g = (u, v; w)$ carrying input values x_u, x_v and hence output value $x_w = x_u \ \mathsf{NAND} \ x_v = 1 - x_u x_v$. By induction, the user holds $\psi_u = s \cdot (\mathbf{a}_u + x_u \mathbf{b}) + \mathbf{e}_u$ and $\psi_v = s \cdot (\mathbf{a}_v + x_v \mathbf{b}) + \mathbf{e}_v$ (for some error vectors $\mathbf{e}_u$ and $\mathbf{e}_v$).
   Compute ψ_w as follows:
$$\psi_w = \mathsf{BD}(\mathbf{a}_u) \cdot \psi_v - x_v \psi_u$$
   As we show below (in Lemma 8.4.1), this new ciphertext has the form $s \cdot \left(\mathbf{a}_w + x_w \mathbf{b}\right) + \mathbf{e}_w$ (for a small enough noise term $\mathbf{e}_w$).
2. Finally, at the output gate the user computes
$$\psi_{\text{out}} = s \cdot (\mathbf{a}_{\text{out}} + C(\mathbf{x}) \cdot \mathbf{b}) + \mathbf{e}_{\text{out}} = s \cdot (\mathbf{a}_{\text{out}} + \mathbf{b}) + \mathbf{e}_{\text{out}}$$
   (since $C(\mathbf{x}) = 1$).
3. Compute $\beta = \mathbf{r}_{\text{out}}^T \cdot [\psi || \psi_{\text{out}}]$ As we show below (in Lemma 8.4.1), $\beta \approx s \cdot u \mod qR$.
   Output $\mu = 0$ if $|\tau - \beta| < q/4$ and $\mu = 1$ otherwise.

We prove the correctness of our ring-LWE scheme below, which would help us calculate the parameters for this scheme. The security proof of the ring-LWE scheme is similar to our LWE scheme, hence we skip it here.

8.4 Correctness and Compactness

Lemma 8.4.1 (Correctness)**.** *Let $\mathcal{C}$ be a family of circuits with their depth bounded by $d_{\max}$ and let $\mathcal{ABE}_r = (\mathsf{Params}_R, \mathsf{Setup}_R, \mathsf{Enc}_R, \mathsf{Keygen}_R, \mathsf{Dec}_R)$ be the ring-LWE version of our attribute-based encryption scheme. Assuming that, for a LWE dimension $n = n(\lambda, d_{\max})$, the parameters for $\mathcal{ABE}_R$ are instantiated as follows:*

$$\chi = D_{\mathbb{Z},\sqrt{n}} \qquad B = O(n)$$
$$q = \tilde{O}(nd_{\max})^{O(d_{\max})}n \qquad s = O(\sqrt{n\log q})$$
$$m = O(n\log q) \qquad \alpha = O(n\log q)^{O(d_{\max})}$$

then the scheme $\mathcal{ABE}_R$ is correct, according to Definition 2.2.1.

Proof. Let us consider a circuit $C \in \mathcal{C}$ of depth atmost $d_{\max}$, such that $C(\mathbf{x}) = 1$. Informally, we have to prove that the encoding obtained at the output wire is of the "correct" form and that the noise component e of β is "small" enough.

Claim 8.4.1. *For each encoding ψ_u for wire u at level j, the user holds $[s\cdot(\mathbf{a}_u + \vec{x_u}\mathbf{b}) + \mathbf{e}_u]$, where $||\mathbf{e}_u||_\infty \le B\cdot(2y)^{j+1}$.*

Proof. First, note that when $\mathbf{e} \leftarrow \chi^y$, $||\mathbf{e}||_\infty \le B$ by the definition of χ and B. Hence, for the noise term $\mathbf{e}_i = \mathbf{R}_i^T\mathbf{e}$, $||\mathbf{e}_i||_\infty \le yB$ since $\mathbf{R}_i \in \{\vec{-1}, \vec{1}\}^{y\times y}$. Thus, the base case for the input encodings holds.

- Now, consider a gate $g = (u, v, w)$ and any two input encodings $\psi_u = s\cdot(\mathbf{a}_u + \vec{x_u}\mathbf{b}) + \mathbf{e}_u, \psi_v = s\cdot(\mathbf{a}_v + \vec{x_v}\mathbf{b}) + \mathbf{e}_v$ at depths j_0, j_1 respectively, where $||\mathbf{e}_u||_\infty \le B\cdot(2y)^{j_0+1}$ and $||\mathbf{e}_1||_\infty \le B\cdot(2y)^{j_1+1}$. Then, the recoded encoding ψ_w is computed as follows:

$$\begin{aligned}
\psi_w &= (\mathsf{BD}(\mathbf{a}_u))\,\psi_v - \vec{x_v}\psi_u \\
&= s\cdot\left(\mathsf{BD}(\mathbf{a}_u)\cdot\mathbf{a}_v + \vec{x_v}\cdot\mathsf{BD}(\mathbf{a}_u)\cdot\mathbf{b}\right) + \mathsf{BD}(\mathbf{a}_u)\cdot\mathbf{e}_v - \vec{x_v}\cdot(s\cdot(\mathbf{a}_u + \vec{x_u}\mathbf{b}) + \mathbf{e}_u) \\
&= s\cdot\left(\mathsf{BD}(\mathbf{a}_u)\cdot\mathbf{a}_v + \vec{x_v}\cdot\mathbf{a}_u\right) + \mathsf{BD}(\mathbf{a}_u)\cdot\mathbf{e}_v - \vec{x_v}\cdot(s\cdot(\mathbf{a}_u + \vec{x_u}\mathbf{b}) + \mathbf{e}_u) \\
&= s\cdot\left(\mathbf{a}_w + \overrightarrow{(1 - x_u x_v)}\cdot\mathbf{b}\right) + \mathbf{e}_w
\end{aligned}$$

  where the last equation is because $\mathbf{a}_w = \mathbf{a}_v\cdot\mathsf{BD}(\mathbf{a}_u) - \mathbf{b}$. Also, we define $\mathbf{e}_w := \mathsf{BD}(\mathbf{a}_u)\cdot\mathbf{e}_v - \vec{x_v}\cdot\mathbf{e}_u$.

- Thus,

$$\begin{aligned}
||\mathbf{e}_w||_\infty &\le y\cdot\|\mathsf{BD}(\mathbf{a}_u)\|_{\max}\cdot||\mathbf{e}_u||_\infty + ||\mathbf{e}_v||_\infty \\
&\le y\cdot\left(B\cdot(2y)^{j_0+1}\right) + B\cdot(2y)^{j_1+1} \\
&= B\cdot(y^{j_0+2}2^{j_0+1} + (2y)^{j_1+1}) \\
&\le B\cdot(2y)^{\max(j_0+1, j_1+1)+1}
\end{aligned}$$

  as claimed (the second inequality is because $\|\mathsf{BD}(\mathbf{a}_u)\|_{\max} \le 1$).

□

Hence, by Claim-8.4.1, if $C(\mathbf{x}) = 1$, the user obtains $\psi_{\text{out}} = s\cdot(\mathbf{a}_{\text{out}} + C(\mathbf{x})\cdot\mathbf{b}) + \mathbf{e}_{\text{out}} = s\cdot(\mathbf{a}_{\text{out}} + \mathbf{b}) + \mathbf{e}_{\text{out}}$, where $\|\mathbf{e}_{\text{out}}\|_\infty \le B\cdot(2y)^{d_{\max}+1}$. After final re-encoding, the user obtains $\beta := \mathbf{r}_{\text{out}}\cdot[\psi^*||\psi_{\text{out}}] = s\cdot\mathbf{u} + e_f$,

where $e_f = [\mathbf{e}^T || \mathbf{e}_{\text{out}}^T] \cdot \mathbf{r}_{\text{out}}$. Here, $\mathbf{r}_{\text{out}}$ is the output of SampleLeft algorithm (Algorithm 2.1. Hence, it has an infinite norm $\|\mathbf{r}_{\text{out}}\|_\infty \leq \alpha\sqrt{y} \leq O(n \log q)^{d_{\max}} \cdot \omega(\sqrt{\log y})$.

Thus,

$$\|e_f\|_\infty \leq y \cdot (\|\mathbf{e}\|_\infty + \|\mathbf{e}_{\text{out}}\|_\infty) \cdot \|\mathbf{r}_{\text{out}}\|_\infty \leq y \cdot \big(B + B \cdot (2y)^{d_{\max}+1}\big) \cdot \big(O(y^{d_{\max}}) \cdot \omega(\sqrt{\log y})\big) \leq O(B \cdot (n \log q)^{2d_{\max}+2}) \cdot \omega(\sqrt{\log y})$$

Also, ciphertext component τ is computed using noise term $||e||_\infty \leq B$. Hence,

$$||e_f - e||_\infty \leq O(B \cdot (n \log q)^{2d_{\max}+2}) \cdot \omega(\sqrt{\log y}) < q/4,$$

which holds given the above setting of the parameters.Thus, ψ and ψ' are "close", message $\mu \in \{0, 1\}$ is decoded correctly as required. □

Corollary 8.4.2. *For any depth $d_{\max}$ family of circuits $\mathcal{C}$, the secret key size in our Attribute-Based Encryption is $O(n \log q) = \text{poly}(d_{\max}, \lambda)$.*

8.4.1 Parameter Selection

This section provides a detailed description on the selection of parameters for our scheme, so that both correctness (see Section 8.4) and security of our scheme are satisfied.

For a family of circuits $\mathcal{C}$ of bounded depth $d_{\max}$, with the LWE dimension n, the parameters can be chosen as follows:

- The error distribution χ is typically a truncated discrete Gaussian distribution $D_{\mathbb{Z},\sqrt{n}}$ with parameter $\sqrt{n}$. And, the error bound $B = O(\sqrt{n}\sqrt{n}) = O(n)$. For this B, the probability for a random sample from $D_{\mathbb{Z},\sqrt{n}}$ to be $\mathbf{0}$ would be $\text{negl}(n)$, according to Lemma 2.1.1.

From now, we will consider the LWE modulus parameter $q = q(n, d_{\max})$, without instantiating it, to calculate the other parameters m, s, α. Later, we will instantiate q with a value which would make m, s, α satisfy the correctness and security properties.

- The parameter $y = \lfloor \log q \rfloor + 1$.
- The "small" Gaussian parameter s is chosen to be $O(\sqrt{n \log q})$.
- Now, let us calculate the value of the "large" Gaussian parameter $\alpha = \alpha(n, d_{\max})$. We should choose α such that the output of the SampleLeft and the SampleRight algorithms are statistically indistinguishable from each other, when provided with the same set of inputs $\mathbf{F}$ and $\mathbf{u}$.

  The SampleRight algorithm (Algorithm 2.2) requires

  $$\alpha > \|\widetilde{\mathbf{T}_\mathbf{B}}\| \cdot \|\mathbf{R}\| \cdot \omega(\sqrt{\log y}) \tag{8.1}$$

  Hence, we proceed as follows:

1. First, we calculate the value of $\|\mathbf{R}\|$, where $\mathbf{R}$ is the matrix obtained corresponding to $\mathbf{a}_{\text{out}}$ in the simulated $\mathcal{ABE}_R$ algorithm KeyGen^*_R. In KeyGen^*_R, at each step of the induction, we get the matrix
$$\mathbf{R}_w = \mathbf{R}_v \cdot \mathsf{BD}(\mathbf{a}_u) - C(\mathbf{x}^*)_v \mathbf{R}_u$$
corresponding to the matrix $\mathbf{a}_w$ of the outgoing wire w of the gate $g(u, v; w)$. We prove the following claim which would help us deduce the value of $\|\mathbf{R}\|_{\max}$.

   Claim 8.4.2. *Suppose that for the gate $g(u, v; w)$, with the incoming wires u, v at depths j_0, j_1 respectively, $\|\mathbf{R}_u\|_{\max} \leq (2y)^{j_0}$ and $\|\mathbf{R}_v\|_{\max} \leq (2y)^{q_1}$. Then, for the outgoing wire w, the maximum norm of the matrix $\mathbf{R}_w$ would be $\|\mathbf{R}_w\|_{\max} \leq (2y)^{\max(j_0,j_1)+1}$*

   Proof. This proof proceeds similar to the calculation of $\|\mathbf{e}_w\|_\infty$ in Claim-8.4.1. In particular,
$$\begin{aligned}\|\mathbf{R}_w\|_{\max} &\leq m \cdot \|\mathsf{BD}(\mathbf{a}_u)\|_{\max} \cdot \|\mathbf{R}_u\|_{\max} + \|\mathbf{R}_v\|_{\max} \\ &\leq y\big((2y)^{j_0} + (2y)^{j_1}\big) \\ &= \big(2^{j_0}(y)^{j_0+1} + (2y)^{j_1}\big) \\ &\leq (2y)^{\max(j_0,j_1)+1}\end{aligned}$$
   Thus, the maximum norm of the matrix $\mathbf{R}_w$ corresponding to $\mathbf{a}_w$ would be $\|\mathbf{R}_w\|_{\max} \leq (2y)^{\max(j_0,j_1)+1}$. □

   Thus, each element of the matrix $\mathbf{R}$ corresponding to $\mathbf{a}_{\text{out}}$, has an absolute value of atmost $\|\mathbf{R}\|_{\max} \leq (2y)^{d_{\max}}$.

2. We then get $\|\mathbf{R}\|$ as follows:
$$\|\mathbf{R}\| := \sup_{\mathbf{x} \in S^{m-1}} \|\mathbf{R} \cdot \mathbf{x}\| \leq y \cdot \|\mathbf{R}\|_{\max} \leq (2y)^{d_{\max}}$$

3. For a matrix $\mathbf{T_B}$ corresponding to a primitive vector $\mathbf{b}$ [MP12], $\|\widetilde{\mathbf{T_B}}\| = \sqrt{5}$.

4. Finally, we substitute these values in Equation 8.1 to get the value of α required for the $\mathsf{SampleRight}$ algorithm.
$$\alpha \geq \sqrt{5} \cdot (2y)^{d_{\max}} \cdot \omega(\sqrt{\log y}) \tag{8.2}$$

The value of the parameter α required for the $\mathsf{SampleLeft}$ algorithm (Algorithm 2.1) is
$$\alpha \geq O(\sqrt{n \log q}) \cdot \omega(\sqrt{\log 2y}) \tag{8.3}$$

Thus, to satisfy both Equation 8.2 and Equation 8.3, we set the parameter
$$\alpha \geq (2y)^{d_{\max}} \cdot \omega(\sqrt{\log y})$$

Thus, the outputs of the $\mathsf{SampleLeft}$ and the $\mathsf{SampleRight}$ algorithms will be statistically indistinguishable from each other, when provided with the same set of inputs $\mathbf{f}$ and $\mathbf{u}$. Hiding the constants, we assign $\alpha = O(n \log q)^{d_{\max}} \cdot \omega(\sqrt{\log y})$.

When our scheme is instantiated with these parameters, the correctness (see Section 8.4) of the scheme

is satisfied when

$$O(B \cdot (n \log q)^{2d_{\max}+2}) \cdot \omega(\sqrt{\log y}) < q/4$$

Clearly, this condition is satisfied when $q = \tilde{O}(nd_{\max})^{O(d_{\max})}$. Also, this value of $q = \text{poly}(n)$, enables both the quantum reduction [Reg09b] and the classical reduction [Pei09] from $\mathsf{dLWE}_{n,q,\chi}$ to approximating lattice problems in the worst case, when n, χ chosen as described above. To conclude this section, for a given max depth $d_{\max}$ and using the $m = m(\lambda, d_{\max})$th cyclotomic polynomial $\Phi_m(X)$, with $n = \varphi(m)$, we set the parameters for our scheme to satisfy both the correctness and security, as follows:

$$\begin{aligned}
\chi &= D_{\mathbb{Z},\sqrt{n}} \\
B &= O(n) \\
q &= \tilde{O}(nd_{\max})^{O(d_{\max})} \\
y &= \lfloor \log q \rfloor + 1 \\
s &= O(n \log q) \\
\alpha &= O(n \log q)^{2(d_{\max}+1)} \cdot \omega(\sqrt{\log y})
\end{aligned}$$

Chapter 9

Conclusions and Future Work

We have thus provided new techniques which help us construct reusable garbled circuits whose size is independent of the size of the circuit being garbled. We have formally defined our technique and call it a fully key-homomorphic encryption. Using FKHE, we obtain interesting results along the way. First we construct an ABE scheme with very short secret keys where the size of the secret keys are independent of the size of the circuits, and dependent only on the depth of those circuits. We then obtain a single-key functional encryption with short secret keys and finally the construction of compact (reusable) garbled circuits. Our way of constructing the ABE scheme leads to optimisations in many of ABE's other applications too. In particular, we obtain an attribute-based fully homomorphic encryption with short secret keys, a verifiable computation procedure with short evaluation keys (and with public delegation and public verification properties).

For completeness, we have given a direct construction for reusable garbled circuits starting from the algorithms of ABE, fully homomorphic encryption scheme and one-time garbled circuits and thus partly unwinding the construction in [GKP+13b]. We have also instantiated the ABE algorithms from ring-LWE assumption which generally leads to much better parameters for implementation than the regular LWE version. With the optimisations like outsourcing decryption and offline computations being inherently present, our ABE scheme would be very efficient for practical implementations.

We have obtained all these optimisations starting from the ABE scheme. One interesting future work to use FKHE scheme or our techniques directly to build or optimise other primitives. This is the first time one has defined or used any variant of full key-homomorphism. With additive key-homomorphism leading to interesting cryptographic applications, it would be interesting to find other applications for variants of full key-homomorphism which can be built from standard assumptions.

9.1 Future work

The following are some of the interesting future directions along this line of work that one could tackle.

- Remove the dependence on the depth of the circuit in the secret key sizes.
- Obtain full adaptive security for FKHE/ABE (without any leveraging).

- Identify other applications for our FKHE and for other variants of full key homomorphism.
- Completely unwind and optimise the reusable garbled circuit construction in [GKP+13b] and obtain a compact construction which reduces the security directly to standard hardness assumptions.
- Obtain a variant of full key-homomorphism that optimises one-time garbled circuits with efficient parameters for practical implementation.

Bibliography

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010. Full Version in http://crypto.stanford.edu/~dabo/pubs/papers/latticebb.pdf.

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors. In *ASIACRYPT*, pages 21–40, 2011.

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In *ICS*, pages 45–60, 2011.

[AIKW13] Benny Applebaum, Yuval Ishai, Eyal Kushilevitz, and Brent Waters. Encoding functions with constant online rate or how to compress garbled circuits keys. In *CRYPTO (2)*, pages 166–184, 2013.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, pages 1–9, 1999.

[AKS01] Miklós Ajtai, Ravi Kumar, and D. Sivakumar. A sieve algorithm for the shortest lattice vector problem. In *STOC*, pages 601–610, 2001.

[App13] Benny Applebaum. Garbling xor gates "for free" in the standard model. In *TCC*, pages 162–181, 2013.

[Bab86] László Babai. On lovász' lattice reduction and the nearest lattice point problem. *Combinatorica*, 6(1):1–13, 1986.

[BB11] Dan Boneh and Xavier Boyen. Efficient selective identity-based encryption without random oracles. *J. Cryptology*, 24(4):659–693, 2011.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit abe and compact garbled circuits. In *EUROCRYPT*, 2014.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, pages 309–325, 2012.

[BHHI10] Boaz Barak, Iftach Haitner, Dennis Hofheinz, and Yuval Ishai. Bounded key-dependent message security. In *EUROCRYPT*, pages 423–444, 2010.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, pages 575–584, 2013.

[Boy13] Xavier Boyen. Attribute-based functional encryption on lattices. In *TCC*, pages 122–142, 2013.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, pages 253–273, 2011.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In *FOCS*, pages 97–106, 2011.

[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.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-xor" technique. In *TCC*, pages 39–53, 2012.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[GGH13a] Sanjam Garg, Craig Gentry, and Shai Halevi. Candidate multilinear maps from ideal lattices. In *EUROCRYPT*, pages 1–17, 2013.

[GGH+13b] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO (2)*, pages 479–499, 2013.

[GGP10] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive verifiable computing: Outsourcing computation to untrusted workers. In *CRYPTO*, pages 465–482, 2010.

[GHV10] Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. A simple BGN-type cryptosystem from LWE. In *EUROCRYPT*, pages 506–522, 2010.

[GHW11] Matthew Green, Susan Hohenberger, and Brent Waters. Outsourcing the decryption of abe ciphertexts. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 34–34, Berkeley, CA, USA, 2011. USENIX Association.

[GKP+13a] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. How to run turing machines on encrypted data. In *CRYPTO (2)*, pages 536–553, 2013.

[GKP+13b] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *STOC*, pages 555–564, 2013.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: interactive proofs for muggles. In *STOC*, pages 113–122, 2008.

[GLW12] Shafi Goldwasser, Allison B. Lewko, and David A. Wilson. Bounded-collusion IBE from key homomorphism. In *TCC*, pages 564–581, 2012.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, pages 197–206, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO (1)*, pages 75–92, 2013.

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In *CRYPTO*, pages 162–179, 2012.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.

[HW14] Susan Hohenberger and Brent Waters. Online/offline attribute-based encryption. In *Public Key Cryptography*, 2014.

[Kil92] Joe Kilian. A note on efficient zero-knowledge proofs and arguments (extended abstract). In *STOC*, pages 723–732, 1992.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *ICALP (2)*, pages 486–498, 2008.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, pages 146–162, 2008.

[LO13] Steve Lu and Rafail Ostrovsky. How to garble ram programs. In *EUROCRYPT*, pages 719–734, 2013.

[LOS+10] Allison B. Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In *EUROCRYPT*, pages 1–23, 2010.

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-lwe cryptography. In *EUROCRYPT*, pages 35–54, 2013. Full Version in http://eprint.iacr.org/2013/293.pdf.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC*, pages 455–479, 2010.

[MG02] Daniele Micciancio and Shafi Goldwasser. *Complexity of Lattice Problems: a cryptographic perspective*, volume 671 of *The Kluwer International Series in Engineering and Computer Science*. Kluwer Academic Publishers, Boston, Massachusetts, March 2002.

[Mic94] Silvio Micali. Cs proofs (extended abstracts). In *FOCS*, pages 436–453, 1994.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, pages 700–718, 2012.

[MR07] Daniele Micciancio and Oded Regev. Worst-case to average-case reductions based on gaussian measures. *SIAM J. Comput.*, 37(1):267–302, 2007.

[MV10] Daniele Micciancio and Panagiotis Voulgaris. A deterministic single exponential time algorithm for most lattice problems based on voronoi cell computations. In *STOC*, pages 351–358, 2010.

[O'N10] Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, pages 191–208, 2010.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, pages 333–342, 2009.

[PRV12] Bryan Parno, Mariana Raykova, and Vinod Vaikuntanathan. How to delegate and verify in public: Verifiable computation from attribute-based encryption. In *TCC*, pages 422–439, 2012.

[Reg09a] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[Reg09b] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In *ACM Conference on Computer and Communications Security*, pages 463–472, 2010.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *FOCS*, pages 162–167, 1986.