Thresholdizing Lattice Based Encryption Schemes

by

Andrew Xia

B.S., Massachusetts Institute of Technology (2017)

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

Masters of Engineering in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 2018

Author . . . .
Department of Electrical Engineering and Computer Science
December 31, 2018

Certified by . . . .
Yael Tauman Kalai
Adjunct Professor, Department of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Chairman, Department Committee on Graduate Theses

Thresholdizing Lattice Based Encryption Schemes

by

Andrew Xia

Submitted to the Department of Electrical Engineering and Computer Science
on December 31, 2018, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Computer Science and Engineering

Abstract

In this thesis, we examine a variety of constructions based on secret sharing techniques applied on lattice-based cryptographic primitives constructed from the learning with erros (LWE) assumption. Using secret sharing techniques from [BGG$^+$17], we show how to construct paradigms of threshold multi-key fully homomorphic encryption and predicate encryption. Through multi-key fully homomorphic encryption [MW16] and threshold fully homomorphic encryption, we can construct a low-round multi party computation (MPC) scheme with guaranteed output delivery, assuming honest majority in the semi-honest and malicious settings. Applying the secret sharing scheme on predicate encryption constructions from LWE [GVW15], we can obtain a distributed predicate encryption scheme.

Thesis Supervisor: Yael Tauman Kalai
Title: Adjunct Professor, Department of Electrical Engineering and Computer Science

Acknowledgments

First of all, I would like to thank my advisor Yael Kalai, for all her support throughout the past year during my M.Eng. During our meetings, she always provided a lot of inspiring insight in my research direction, and made sure that my thoughts would be grounded in provable statements. From when I first took her class, 6.857 in Spring 2017, to today, I really appreciate all the support she has given me as I have developed as a researcher.

I would also like to extend special thanks to Adam Sealfon, who has always been available for meeting and discussing ideas. I have been so much happier this year working with you, not only on our discussions in cryptography but also in life in general.

As my time at MIT comes to a close, I'd also like to extend thanks to all the communities I've been a part of. My extracurricular activities during undergraduate, Amphibious Achievement and MIT Technique being the most transformative experiences, and my undergraduate living groups, my fraternity Pi Lambda Phi and East Campus, have shaped much of who I am today. This past year as a Masters student, having the MIT Theory Group as a social and academic community has made my time much more enjoyable. Every experience at this institute has shaped who I am and I am thankful for everyone who I have crossed paths with, who have made an impact on my life.

And finally, I would like to thank my parents, and my sister, for their support and love through the first part of my journey in life.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of cryptography has evolved over time, starting from Diffie-Hellman Key Exchange [DH76] and public key cryptography such as RSA [RSA78]. From the original extension of symmetric key cryptography to public key cryptography, recent development in the field has allowed for further flexibility of encryption protocols.

One such direction is that of *fully homomorphic encryption* (FHE) [GSW13], in which an encryption protocol has the additional property such that computation can be done on the encrypted data. Given the encryption of x and encryption of y, denoted as $Enc(x), Enc(y)$ respectively, it is possible to add the encryptions such that $Dec(Sum(Enc(x), Enc(y))) = x + y$, and similarly compute the product of encryptions such that $Dec(Prod(Enc(x), Enc(y))) = x * y$. This allows for computation of arbitrary circuits on the encrypted space, potentially allowing clients with low memory or compute power to delegate their data to a server, allowing the server to compute on their encrypted data, while having the server learn nothing about the client's data itself.

A recent extension of FHE is *multi-key fully homomorphic encryption* [MW16, LTV13]. This is a distributed FHE scheme where parties are able to encrypt plain text under their own (independently) generated ciphertexts, and yet still be able to jointly compute on the ciphertexts from multiple parties. The decryption protocol in multi-key FHE is a multi-party protocol where all parties' secret keys must be used for correctness and security of the scheme.

Another such direction is that of *function encryption* (FE) [ABB10], which is a more fine-grained approach to the ability do decrypt data. With the encryption of x, a party with the secret key sk_f associated with a function f can only decrypt to reveal $f(x)$ and learn nothing more about x. While it is not known how to construct functional encryption from the learning with errors assumption, we know how to construct a weaker notion of *predicate encryption* (PE), in which a ciphertext of x has a set of attributes (μ) such that nothing is learned about (x, μ) unless if the a corresponding secret key can decrypt the ciphertext.

Recent constructions of FHE and PE can be constructed by lattice-based assumptions, which have the positive feature of believed to be quantum-resistant.

Threshold cryptography [DF89] is the process of constructing a cryptographic protocol in which certain algorithms in the protocol may be completed in a distributed manner, instead of by a single party. We allow the property where we can split a cryptographic secret into N shares, storing each share on a different server, and allowing any subset of t servers to use the secret without ever re-constructing and revealing the secret. An example of threshold cryptography may be for *threshold decryption*, where we distribute the secret key of a public key encryption scheme among N parties. Any subset of t parties may output partial decryptions such that the decrypted plaintext can be reconstructed from the partial decryption.

In this thesis, we develop techniques to construct threshold cryptosystems for existing lattice-based fully homomorphic encryption and predicate encryption schemes.

For the FHE setting, we extend recent work in threshold FHE to the threshold multi-key FHE setting. While existing multi-key FHE schemes require a N-of-N threshold decryption process, through secret sharing the secret keys of the parties, we can achieve a t-of-N threshold decryption protocol. We propose constructions of multi-key FHE based on the Gentry-Sahai-Waters FHE construction [GSW13], and also off of the NTRU FHE construction [LTV13] based off of ring lattices. The tradeoffs of the GSW FHE scheme is such that a common public parameter is required for the setting up public keys, though the decryption process requires a single round. On the other hand, the NTRU FHE construction does not require public parameters to set up the public keys for independent users, but a [two-round] MPC is required for decryption. For more details, see section 3.1.2 and 3.1.4 for the FHE constructions respectively.

For the Predicate Encryption setting, we extend work by [BGG$^+$14, GVW15] to build a threshold predicate encryption scheme. Modifying trapdoor generation algorithms for lattices [MP12, BKP13], we can also construct a threshold key generation algorithm, in which there no longer is a single master authority generating secret keys sk_f for particular functions.

Applications Fully homomorphic encryption and predicate encryption provide a lot of flexibility in creating new protocols with security guarantees.

In one setting, imagine there are n hospitals with a lot of sensitive data. This data cannot be shared with the other hospitals, and researchers who may wish to access the hospitals data are also constrained. Through multi-key threshold fully homomorphic encryption, the each hospital i can encrypt their data under their own public key $Enc(pk_i, x_i)$ and broadcast the encrypted data. In addition, the hospitals can give secret shares of their secret key to the other hospitals (or a set of trusted parties). Now, a researcher can process the *encrypted* data as he wishes, and when he wishes to see the decrypted output of his function, when t of the n hospitals allow the decryption –whether they are online, or accept that the computed function respects privacy –then the researcher can see the output of the function.

In another setting, imagine that there is an investigation on credit card fraud [GVW15]. By encrypting credit card transactions labeled with a set of attributes, including the timestamp, costs, and zipcodes, it would be possible to issue decryption keys where the transactions exceed a certain threshold, and originated from a particular range of zip codes. This in turns protects the privacy of transactions which would not be considered “fradulent."

1.1 Related Work

This work builds off of recent work on threshold FHE by Boneh et al [BGG$^+$17]. Concurrent work on threshold multi-key FHE was done by Badrinarayanan et al [BJMS18]. Multi-key FHE in a sense achieves N-of-N threshold decryption [MW16, LTV13, PS16, BP16], however generalizing the scheme to a t-of-N threshold setting is nontrivial and requires secret sharing techniques from [BGG$^+$17]. Gordon et al also mention the use of a threshold FHE to create a three round multi-party computation scheme with guaranteed output delivery, however without specifying the specific underlying secret sharing scheme used [GLS15].

Bendlin et al [BKP13] also present a threshold Gaussian sampling protocol. However the limitation of this system is that only a bounded number of *online* non-interactive decentralized Gaussian samplings can be done before an *offline* interactive step must be performed. On the line of constructing lattice-based threshold cryptosystems, distributed PRFs [BLMR15] have also been constructed.

1.2 Outline of Thesis

In chapter 2, we detail preliminary concepts such as lattices, gaussian trapdoors, and secret sharing. In chapter 3, we present the definitions for fully homomorphic encryption, threshold FHE, and multi-key FHE. In chapter 4, we present our extensions: constructions of decentralized FHE, threshold multi-key FHE, modifications of the NTRU FHE scheme, and applications to MPC based on threshold multi-key FHE. In chapter 5, we present constructions of predicate encryption and modifications to allow threshold decryption protocols. Finally, in chapter 6, we detail potential future directions of this work.

Chapter 2

Preliminaries

In this chapter, we detail the preliminary concepts for this thesis, which include lattices, Gaussian Trapdoors for lattices, and secret sharing.

We say a function $f(n)$ is *negligible* if it is $O(n^{-c})$ for all $c > 0$ and we use $\text{negl}(n)$ to denote a negligible function of n. We say that an event occurs with *overwhelming probability* if its probability is $1-\text{negl}(n)$. We also define computational indistinguishability below:

Definition 1 (Computational Indistinguishability). *We say that two distributions X, Y over $\{0,1\}^n$ are* computationally indistinguishable *if for every PPT algorithm A, there exists a negligible function $\epsilon(n)$ such that the following holds:*

$$|Pr[A(X_n) = 1] - Pr[A(Y_n) = 1]| \leq \epsilon(n)$$

2.1 Lattices

A *lattice* Λ is a discrete additive subgroup of $\mathbb{R}^m$ for some $m \geq 0$. For this thesis, we consider, full-rank integer lattices, which are additive subgroups of $\mathbb{Z}^m$ with finite index. Recent cryptographic protocols use a particular family of *q-ary* integer lattices, which contain $q\mathbb{Z}^m$ as a sublattice for some integer q, which will be bounded by $poly(n)$. For positive integers n, q, let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ be arbitrary, and define a full-rank m-dimensional q-ary lattice as

$$\Lambda^{\perp}(\mathbf{A}) = \{z \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{0} \mod q\}$$

For any $\mathbf{u} \in \mathbb{Z}_q^n$ with an integral solution $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{A}\mathbf{x} = \mathbf{u} \mod q$, we define the *coset* (or shifted lattice) as

$$\Lambda_u^{\perp}(\mathbf{A}) = \Lambda^{\perp}(\mathbf{A}) + \mathbf{x} = \{z \in \mathbb{Z}^m : \mathbf{A}\mathbf{z} = \mathbf{u} \mod q\}$$

2.1.1 Ideal Lattices

In addition to constructing lattices over $\mathbb{R}^n$, lattices can also be constructed over rings.

A ring R is defined as $\mathbb{Z}[x]/\langle\phi(x)\rangle$ for some degree n polynomial $\phi(x) \in \mathbb{Z}[x]$. For the context of this thesis and for the NTRU encryption scheme, we let $\phi(x) = x^n + 1$ where n is a power of 2. We define $R_q \overset{\text{def}}{=} R/qR$ for some prime integer q. An element in R (or R_q) can be viewed as a degree $(n-1)$ polynomial over $\mathbb{Z}$ (or $\mathbb{Z}_q$). We can represent each element as a vector of its n coefficients.

2.2 Gaussian Trapdoors

This subsection details some trapdoor generators and algorithms for lattices.

Given a lattice $\Lambda \subset \mathbb{R}^m$, let $c \in \mathbb{R}^m$ and let $\Sigma > 0$ be a positive definite matrix. The *discrete Gaussian distribution* $D_{\Lambda+c,\sqrt{\Sigma}}$ is simply the Gaussian distribution $D_{\sqrt{\Sigma}}$ is distributed such that the support is over coset $\Lambda+c$. In other words, for all $x \in \Lambda+c$

$$D_{\Lambda+c,\sqrt{\Sigma}}(x) = \frac{\rho_{\sqrt{\Sigma}}(x)}{\rho_{\sqrt{\Sigma}}(\Lambda+c)} \propto \rho_{\sqrt{\Sigma}}(x)$$

A discrete Gaussian is *spherical* with parameter $s > 0$ if the covariance matrix is $s^2\mathbf{I}$.

The binary decomposition algorithm, denoted $\mathsf{BD}(A) \to R$, or denoted $G^{-1}()$ otherwise, is a deterministic algorithm that takes a matrix $A \in \mathbb{Z}_q^{n\times m}$ and outputs a matrix $R \in \mathbb{Z}_q^{m\times m}$ where R is a binary matrix such that each element $a \in \mathbb{Z}_q$

that belongs to A gets transformed to a column vector $r \in \mathbb{Z}_q^{\lceil \log q \rceil}$. r is defined as $r = [a_0, \dots a_{\lceil \log q \rceil - 1}]^T$ where $\sum_i a_i 2^i = a$.

Lemma 2 ([BGG$^+$14])**.** *Let $n, m, q > 0$ be integers with q prime. Then, there exists PPT algorithms with the properties below:*

- $\mathsf{TrapGen(1^n, 1^m, q)} \to (A, T_A)$. *The trapdoor generation algorithm is a randomized algorithm such that when given $m = \Theta(n \log q)$, outputs a full-rank matrix $A \in \mathbb{Z}_q^{n \times m}$ and basis $T_A \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^{\perp}(A)$ such that A is negl(n)-close to uniform and the Gram-Schmidt orthogonalization of T_A is $O(\sqrt{n \log q})$ with all but negligible probability in n.*

- $\mathsf{ExtendRight(A, T_A, B)} \to T_{(A|B)}$ *[MP12]. The extend right algorithm is a deterministic algorithm where, given full rank matrices A, B and trapdoor T_A, outputs a basis $T_{(A|B)}$ of $\Lambda_q^{\perp}(A|B)$*

- $\mathsf{ExtendLeft(A, G, T_G, S)} \to T_H$ *where $H = (A|G + AS)$ [ABB10]. The extend left algorithm is a deterministic algorithm where, given full-rank matrices $A, G \in \mathbb{Z}_q^{n \times m}$ and a basis T_G for $\Lambda_q^{\perp}(G)$, outputs a basis T_H of $\Lambda_q^{\perp}(H)$ such that $||T_H||_{GS} \leq ||T_G||_{GS} \cdot (1 + ||S||_2)$.*

- *For $m = n\lceil \log q \rceil$, there is a fixed full-rank matrix $G \in \mathbb{Z}_q^{n \times m}$ such that the lattice $\Lambda_q^{\perp}(G)$ has a publicly known basis $T_G \in \mathbb{Z}^{m \times m}$ with $||T_G||_{GS} \leq \sqrt{5}$. The gadget matrix G is defined via a gadget vector $g = (1, 2, 4, \dots 2^{k-1}) \in \mathbb{Z}_q^k$ for $k = \lceil \lg q \rceil$, such that $G = I_n \otimes g^t$. This gives the property that for any matrix $A \in \mathbb{Z}_q^{n \times m}$ we have $G \cdot \mathsf{BD}(A) = A$.*

Discrete Gaussians Regev [Reg05] defined a natural distribution on lattice $\Lambda_q^u(A)$ called a *discrete Gausian* parameterized by a scalar $\sigma > 0$. We let $\mathcal{D}_\sigma(\Lambda_q^u(A))$ to represent this distribution. For a random matrix $A \in \mathbb{Z}_q^{n \times m}$ and $\sigma = \tilde{\Omega}(\sqrt{n})$, a vector x sampled from $\mathcal{D}_\theta(\Lambda_q^u(A))$ has l_2 norm less than $\sigma\sqrt{m}$ with probability at least $1 - \text{negl}(m)$.

With these algorithms in mind, we recall algorithms that allow us to solve $AX = U$. In other words, given $A \in \mathbb{Z}_q^{n\times m}, U \in \mathbb{Z}_q^{n\times k}$ we wish to find a low-norm matrix $X \in \mathbb{Z}^{m\times k}$ such that $AX = U$.

Lemma 3. *Let $A \in \mathbb{Z}_q^{n\times m}$ and $T_A \in \mathbb{Z}^{m\times m}$ be the basis for $\Lambda_q^{\perp}(A)$. Let $U \in \mathbb{Z}_q^{n\times k}$. There are PPT algorithms that output X satisfying $AX = U$ with the properties below:*

- $\mathsf{SampleD}(A, T_A, U, \sigma) \to X$ *SampleD is a randomized algorithm that, when $\sigma = ||T_A|_{GS} \cdot \omega(\sqrt{\log m})$, outputs a random sample X from a distribution that is statistically close to $\mathcal{D}_\theta(\Lambda_q^u(A))$.*

In addition, we also have the following algorithms:

Lemma 4. *We present the* SampleRight *and* SampleLeft *algorithms below [MP12]:*

- $\mathsf{SampleRight}(A, T_A, B, U, \sigma)$*: This is a randomize algorithm that given full rank matrices $A, B \in \mathbb{Z}_q^{n\times m}$, matrix $U \in \mathbb{Z}_q^{n\times m}$, and a basis $T_A \in \mathbb{Z}^{m\times m}$ for $\Lambda_q^{\perp}(A)$ where $\sigma = ||T_A|_{GS} \cdot \omega(\sqrt{\log m})$, outputs a random sample $X \in \mathbb{Z}_q^{2m\times m}$ from a distribution statistically close to $\mathcal{D}_\theta(\Lambda_q^u(A|B))$. More specifically this algorithm is a composition of* $\mathsf{ExtendRight(A, T_A, B)} \to T_{(A|B)}$ *and* $\mathsf{SampleD}((A|B), T_{(A|B)}, U, \sigma) \to X$

- $\mathsf{SampleLeft}(A, T_A, B, u, \alpha)$*: This is a randomized algorithm that given full rank matrices $A \in \mathbb{Z}_q^{n\times m}, B \in \mathbb{Z}_q^{n\times m_1}$, a short basis T_A of $\Lambda_q^{\perp}(A)$, a vector $u \in \mathbb{Z}_q^n$, along with Gaussian parameter α, this algorithm outputs the following. Let $F = (A||B)$, and algorithm outputs a vector $e \in \mathbb{Z}^{m+m_1}$ in coset Λ_{F+u}.*

2.3 Learning with Errors

The LWE assumption was introduced by Regev [Reg05], which shows that solving it *on the average* is as hard as solving several standard lattice problems even *in the worst case.* It is conjectured that the solving the LWE problem is hard even on quantum computers, such that lattice-based cryptographic primitives may offer post-quantum security.

Definition 5 (Learning with Errors [Reg05]). *For an integer $q = q(n) \geq 2$ and an error distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the decision learning with errors problem $\mathsf{dLWE}_{n,m,q,\chi}$ is the assumption that the following pairs of distributions are computationally indistinguishable*

$$(A, As + x) \approx_c (A, u)$$

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

We say that an error distribution χ is *B-bounded* if its support is in $[-B, B]$. The error distribution is a truncated discrete Gaussian distribution.

When working over rings, we introduce the Ring Learning with Errors (RLWE) assumption.

Definition 6 (Ring Learning with Errors [LPR12]). *The* RLWE *assumption states that:*

$$\{(a_i, a_i \cdot s_i + e_i)\} \overset{c}{\approx} \{(a_i, u_i)\}$$

For uniformly random chosen a_i, u_i in R_q and where s_i, e_i are drawn from an error distribution χ.

2.4 Secret Sharing

In this subsection, we provide the basic definitions and notations for secret sharing, similar to [BGG+17].

Definition 7 (Monotone Access Structure). *Let $P = \{P_1, \ldots P_N\}$ be a set of parties. A collection $\mathbb{A} \subseteq \mathcal{P}(P)$ is monotone if for any sets B, C satisfying $B \in \mathbb{A}$ and $B \subseteq C \subseteq P$, then $C \in \mathbb{A}$. A monotone access structure on P is a collection $\mathbb{A} \subseteq \mathcal{P}(P) \setminus \varnothing$. The sets in $\mathbb{A}$ are* valid sets *while the sets not in $\mathbb{A}$ are* invalid sets. *In other words, supersets of a valid set are also valid.*

We now present the definition of secret sharing [Sha79]:

Definition 8 (Secret Sharing Scheme)**.** *Let $P = \{P_1, \dots P_N\}$ be a set of parties. A secret sharing scheme $\mathcal{SS}$ for a secret space $\mathcal{K}$ is a pair of PPT algorithms $\mathcal{SS}$ = ($\mathsf{SS.Share}$, $\mathsf{SS.Combine}$) such that:*

- $\mathsf{SS.Share}$ $(k, \mathbb{A}) \rightarrow (s_1, \dots s_N)$*: On input with a secret $k \in \mathcal{K}$ and a defined access structure $\mathbb{A}$, the sharing algorithm returns shares $(s_1, \dots s_N)$ for each party.*

- $\mathsf{SS.Combine}$ $(S) \rightarrow k$*: On input with a set of shares $S = \{s_i\}_{i \in S}$, if S is a valid set within the access structure then the combining algorithm outputs the secret k.*

Usually, there is a dealer who runs the $\mathsf{SS.Share}$ *algorithm to deal secrets to the N parties, while the N parties jointly run* $\mathsf{SS.Combine}$ *to reconstruct the secret.*

Correctness is defined as follows. For all $S \in \mathbb{A}$ and for all $k \in \mathcal{K}$, given shares $(s_1, \dots s_N) \leftarrow \mathsf{SS.Share}(k, \mathbb{A})$, we see that $\mathsf{SS.Combine}(\{s_i\}_{i \in S}) \rightarrow k$ with probability 1.

Security is defined as follows. For all $S \notin \mathbb{A}$ and for $k_0, k_1 \in \mathcal{K}$, given shares $(s_{1,b}, \dots s_{N,b}) \leftarrow \mathsf{SS.Share}(k_b, \mathbb{A})$, the following distributions are indistinguishable:

$$\{s_{i,0}\}_{i \in S} \approx \{s_{i,1}\}_{i \in S}$$

The indistinguishability definition is either *computational* or *statistical* depending on the setting of the secret sharing scheme.

As an example, we present Shamir's secret sharing scheme [Sha79]:

Definition 9 (Shamir's Secret Sharing)**.** *Let $\mathcal{F}$ be a finite field. Shamir's secret-sharing scheme is a secret sharing scheme with a t-of-n access structure, constructed as follows:*

- $\mathsf{SS.Share}(s)$*: Given a secret s, the sharing algorithm randomly chooses a degree $t-1$ polynomial $p(x)$, where the constant coefficient is s. Party i receives share $p(i)$, with n total shares distributed among the n parties. Note that t or fewer shares look uniformly random.*

- *SS.Combine(S): Given at least t shares, the combining algorithm uses Lagrange interpolation to find a unique degree $t-1$ polynomial $p'(x)$ passing through the t shares. The secret is recovered from $p'(0)$.*

The correctness of Shamir's secret sharing is as follows. Given at least t shares, in which t points uniquely determine a degree $t-1$ polynomial, the secret will be uniquely and correctly reconstructed.

The security of Shamir's secret sharing is as follows. Given at most $t-1$ shares, the distribution of the $t-1$ shares is uniformly random. Because for every secret $s' \neq s$, there exists a polynomial that could pass through s' and the $t-1$ shares, Shamir's secret sharing scheme is information-theoretically secure (as there are no computational assumptions for this scheme).

2.4.1 Linear Secret Sharing

We consider a special class of secret sharing schemes, known as *linear secret sharing schemes*, where the combining algorithm SS.Combine consists of linear operations. A linear secret sharing scheme has been shown to be equivalent to a monotone span program [Bei].

Definition 10 (Linear Secret Sharing Scheme). *Let $P = \{P_1, \dots P_N\}$ be a set of parties, and $\mathbb{A}$ be an access structure. A linear secret sharing scheme consists of a secret space $\mathcal{K} = \mathbb{Z}_p$ for some prime p such that the following properties for SS.Share, SS.Combine are met:*

- *SS.Share: There exists a matrix $M \in \mathbb{Z}_p^{l \times N}$ called the* share matrix, *such that each party P_i is associated with a row(s) $T_i \subseteq [l]$ of the matrix. To create the shares of a secret k, the sharing algorithm first samples random values $r_2, \dots r_n \xleftarrow{R} \mathbb{Z}_p$, and we define a vector $\mathbf{w} = \mathbf{M} \cdot (\mathbf{k}, \mathbf{r_2}, \dots \mathbf{r_n})^{\mathbf{T}}$. The share for P_i consists of the entries $\{w_j\}_{j \in T_i}$*

- *SS.Combine(S): for any valid set $S \in \mathbb{A}$, we have that*

$$(1, 0, \dots 0) \in \mathsf{span}(\{M[j]\}_{j \in \cup_{i \in S} T_i})$$

over $\mathbb{Z}_p$*, where we let* $M[j]$ *denote the* j^{th} *row of* M*. Any valid set of parties* $S \in \mathbb{A}$ *can efficiently find coefficients* c_j *such that*

$$\sum_{j \in \cup_{i \in S} T_i} c_j \cdot M[j] = (1, 0, \dots 0)$$

This in turn allows for the reconstruction of the secret, in that $k = \sum_{j \in \cup_{i \in S} T_i} c_j w_j$*. The coefficients* c_j *are known as the* reconstruction coefficients.

For the purpose of lattice based encryption schemes, due to the noise in the ciphertext as computation is done, we wish to keep our reconstruction coefficents small. To do so, [BGG+17] defined a special class of access structures, known as $\{0, 1\}$-Linear Secret Sharing Schemes, such that the reconstruction coefficients are always binary.

Definition 11 ($\{0, 1\}$-Linear Secret Sharing Schemes)**.** *Let* $P = \{P_1, \dots P_N\}$ *be a set of parties, and* $\mathbb{A}$ *be an access structure. A linear secret sharing scheme consists of a secret space* $\mathcal{K} = \mathbb{Z}_p$ *for some prime* p *such that during* $\mathsf{SS.Combine}$*, the reconstruction coefficients* $c_j \in \{0, 1\}$*.*

A threshold access structure with parameter t is a special type of access structure, such that $S \in \mathbb{A}$ exactly when $|S| \geq t$. It has been shown that the class of all threshold access structures is in $\{0, 1\} - LSSS$ [BGG+17].

We describe a construction that realizes the $\{0, 1\} - LSSS$ in figure 2.4.1.

We can represent every threshold access structure as a special monotone Boolean formula $C : \{0, 1\}^N \to \{0, 1\}$ where all of the input wires of C have fan-out of 1. Given this formula, we can define C as a tree where we associate the root note as the output of the C, and the two children of each node are the inputs of a gate.

We associate the root node with value $m_r = 1$. For every node v, if it is associated with a OR gate, we let the two children v_l, v_r have values $m_{v_l} = m_{v_r} = m_v$. If node v

“Folklore Algorithm"

Input: A special monotone Boolean formula $C : \{0,1\}^N \to \{0,1\}$
Output: A LSSS share Matrix $\mathbf{M}$ for the access structure induced by C.

1. Label root r with length 1 vector $\mathbf{m}_r = (1)$
2. Initialize counter $\mathsf{count} = 1$.
3. For $v = \mathsf{v}(i)$ for $i = 1, \ldots, n$.
   (a) If v is an OR gate, label its children with the same vector as m
   (b) if v is an AND gate with vector $\mathbf{m}_v$ associated with it, then pad $\mathbf{m}$ with 0s at the end to make it length count. Denote the new vector by $\mathbf{m'}$. Let one of its children have value $(\mathbf{m'}, 1)$ and the other vector with children $(0, \ldots, 0, -1)$ with length $\mathsf{count} + 1$. Finally, increase count by 1.
4. Finally, once the entire tree is labeled, the vectors of the leaf nodes are the rows associated with the sharing matrix $\mathbf{M}$. If some of the vectors have different lengths, the shorter vectors are padded with 0s.
5. For a monotone span program where $\mathbf{w} = \mathbf{M} \cdot (\mathbf{k}, \mathbf{r_2}, \ldots, \mathbf{r_n})^{\mathbf{T}}$ where k is the secret and r_i are random values in $\mathbb{Z}_q$, the shares given to party P_i consists of the entries $\{w_j\}_{j \in T_i}$, or the corresponding leafs in the circuit C.

Figure 2-1: A “folklore” linear secret sharing algorithm [BGG+17].

is associated with an AND gate, we define $m_{v_l} = (m_l, 1)$ and $m_{v_r} = (0, 0, \ldots, 0, -1)$. In other words, we define the children to have it's value length increased by 1. Once the entire tree is labeled, we associate the leaf nodes of the tree with values the parties associated with inputs to C, and we append 0s at the end of every node such that every node's value will be the same length.

We can now define a monotone span program $Ar = b$, where each row in matrix A contains the values associated with the leafs in the tree of C, r is a random vector except where the first entry is the secret to be shared, and each entry of b is a share for the party associated with the row in A.

Using this secret sharing scheme, we can guarantee that during the final decryption protocol, the coefficients associated with the partial shares will be binary [BGG+17].

2.4.2 Extensions of {0,1}-LSSS

One thing worth noting is that the folklore secret sharing scheme requires $O(n^{5.2})$ shares for a threshold access structure with n parties [Val84, Gol14]. However, the secret sharing construction uses a formula, which is graphically represented as a tree, instead of a circuit. [VNS+03] shows a computational secret sharing scheme which uses a circuit-based structure, with AND, OR, and FAN-IN gates (of fan-in two). For our folklore algorithm, we keep AND and OR gates as is, and FAN-IN gates can be masked with one-time pads to preserve security. A share size less than that of the formula can be obtained using algorithms from [Weg87], in which the threshold function can be done with $O(n^2)$ input wires with n^2 fanout gates, beating the $O(n^{5.2})$ formula upper bound.

Chapter 3

Fully Homomorphic Encryption

3.1 Fully Homomorphic Encryption

3.1.1 Definitions of FHE

We briefly present the definitions of a public-key fully homomorphic encryption (FHE) scheme in this section.

Definition 12 (Fully Homomorphic Encryption)**.** *A FHE scheme consists of the following PPT algorithms ((FHE.KeyGen, FHE.Enc, FHE.Eval, FHE.Dec)):*

- $\mathsf{FHE.KeyGen}(1^\lambda, 1^d, 1^k) \to (pk, sk)$. *The key generation algorithm is a probabilistic algorithm that takes in the security parameter λ, depth bound d, and message length k and outputs a secret key sk and public key pk.*

- $\mathsf{FHE.Enc}(pk, \mu) \to ct$. *The encryption algorithm is a probabilistic algorithm that takes in input the public key of the FHE scheme, and a message $\mu \in \{0,1\}^k$ and outputs a ciphertext ct.*

- $\mathsf{FHE.Eval}(\mathcal{C}, ct) \to ct'$. *The evaluation algorithm is a deterministic algorithm that takes in as input a boolean circuit $\mathcal{C} : \{0,1\}^k \to \{0,1\}$, a ciphertext ct, and outputs another ciphertext ct' which is the evaluation of the circuit on the ciphertext input.*

- $\mathsf{FHE.Dec}(sk, ct') \to \mu$. *The decryption algorithm is a deterministic algorithm that takes as input the secret key sk and a ciphertext ct' and outputs a bit corresponding to the ciphertext.*

Note that a symmetric-key FHE scheme can be similarly defined if there is no public key and the secret key is used for both encryption and decryption.

A fully homomorphic encryption scheme has the following requirements for correctness, security, and compactness. For correctness, we wish that the decryption of an evaluated ciphertext should output a plaintext that is equal to the evaluation of the plaintexts corresponding to the original ciphertext. For the semantic security, we require the following distributions to be computationally indistinguishable, for any two messages μ_0, μ_1.

$$(pk, \mathsf{Enc}(pk, \mu_0)) \approx_c (pk, \mathsf{Enc}(pk, \mu_0))$$

There is also a notion of *compactness*, in that the size of the ciphertext should not grow with the evaluation circuit depth –otherwise a trivial construction of FHE can be attained. More formally, a ciphertext ct is bounded by a polynomial $p(\dots)$ such that $|ct| \leq p(\lambda, d)$ for depth bound d, but it should be independent of $\mathcal{C}$

Current constructions of FHE are based on lattices [GSW13], specifically the Learning with Errors (LWE) assumption [Reg05]. Fully Homomorphic Encryption was first constructed by Gentry in 2009 [Gen09]. Subsequent second generation FHE schemes were introduced by Brakerski and Vaikuntanathan, where optimizations such as evaluation keys and modulus reduction were introduced [BV11]. The so-called "third-generation" of FHE schemes include that of Gentry, Sahai and Waters [GSW13], which eliminated the need of an evaluation key.

3.1.2 Construction of GSW FHE

Our foundation FHE scheme uses [GSW13]. To recap, a fully homomorphic scheme has the following properties. We first have a SetUp phase such that public parameters are generated. As with all encryption schemes, we include PPT algorithms

KeyGen, $\mathsf{Encrypt}$, $\mathsf{Decrypt}$. In addition, a FHE scheme also has two additional algorithms, Add, Mult which operate on two ciphertexts to attain fully homomorphic evaluation operations.

More formally, the operations are defined below:

- $\mathsf{GSW.SetUp}(1^\lambda, 1^d)$: Output a random matrix $\mathbf{B} \in \mathbb{Z}_q^{n-1\times m}$
- $\mathsf{GSW.KeyGen}$:
  - To generate the secret key, first randomly sample $n-1$ random elements and define $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^{n-1}$. Define secret key $\mathbf{t} = (-s, 1) \in \mathbb{Z}_q^n$.
  - To generate the public key, sample m elements from the error distribution $e \leftarrow \chi^m$. Set $\mathbf{b} := \mathbf{sB} + \mathbf{e} \in \mathbb{Z}_q^m$. We define the public key to be $\mathbf{A} := \begin{bmatrix} B \\ b \end{bmatrix} \in \mathbb{Z}_q^{n\times m}$.
- $\mathsf{GSW.Encrypt}(pk, \mu)$: To encrypt a bit $\mu \in \{= 0, 1\}$, first choose a random binary matrix $R \xleftarrow{\$} \{0, 1\}^{m\times m}$. Output ciphertext $C := AR + \mu G$
- $\mathsf{GSW.Decrypt}(sk, C)$: Given a ciphertext, and secret key $t := sk$, define a vector $\mathbf{w}$ of length n where
$$w = [0, 0, \dots 0, \lceil q/2 \rceil]$$
Next, compute $v = tCG^{-1}(w) = e + \mu(q/2)$ and output $\left|\lfloor \frac{v}{q/2} \rceil\right|$ as the decryption[1].
- $\mathsf{GSW.Eval}$: for the evaluation operation, we define homomorphic addition and multiplication as follows
  - $\mathsf{GSW.Add}(C_1, C_2)$: Output $C^+ = C_1 + C_2 \in \mathbb{Z}_q^{n\times m}$.
  - $\mathsf{GSW.Mult}(C_1, C_2)$: Output $C^* = C_1 G^{-1}(C_2) \in \mathbb{Z}_q^{n\times m}$.

[1]Note that $G^{-1}(w)$ is simply a vector where the last entry is 1 and all other entries are 0

3.1.3 Noise Analysis of GSW

In this subsection, we will see how the ciphertext noise increases with certain functions in the GSW encryption scheme. We analyze the noise to prove the correctness of the GSW scheme. Note that GSW.SetUp, GSW.KeyGen do not contribute to the noise of the ciphertext, while GSW.Encrypt, GSW.Eval contribute to the noise of the ciphertext and GSW.Decrypt will only work correctly if the ciphertext is not too noisy. First, we use the following definition:

Definition 13 (β-noisy ciphertext [MW16])**.** *We define a β-noisy ciphertext C of some message μ such that when the ciphertext is multiplied with secret key t, we see $tC = \mu tG + e$ where $||e||_\infty \leq \beta$.*

Now, we will show that GSW.Encrypt, GSW.Eval, GSW.Decrypt preserve correctness under the consideration of noise.

- GSW.Encrypt: We define a fresh ciphertext to have β_{init} noise. First, note that $tA = e$ where $||e||_\infty \leq B_\chi$. We see that $C = AR + \mu G$ so $tC = \mu tG + e'$ where $e' = eR$. Thus, we see that $||e'||_\infty \leq mB_\chi$. Finally, we can define $\beta_{init} = mB_\chi$.

- GSW.Add: First, let us assume that ciphertext C_1 has β_1 noise and C_2 has β_2 noise. To perform the addition operation, we see that $C^+ = C_1 + C_2$, so the noise of C^+ is $\beta_1 + \beta_2$.

- GSW.Mult: The noise analysis of multiplying ciphertexts is slightly trickier. Once again, let ciphertext C_1 have β_1 noise and C_2 have β_2 noise. The multiplication operation consists of $C^* = C_1 G^{-1} C_2$. Thus, we see that $tC^* = e'' + \mu_1\mu_2 G$, where $e'' = eG^{-1}(C_2) + \mu_1 e_2$. Thus we see that $||e''||_\infty \leq m\beta_1 + \beta_2$ so the resulting ciphertext noise is $(m\beta_1 + \beta_2)$.

- GSW.Decrypt: Assume our ciphertext C (which may be an evaluated ciphertext) has β-noise. In other words, we have $tC = e + \mu tG$ where $||e||_\infty = \beta$. When deecrypting, we see that $v = tCG^{-1}(w^T) = e' + \mu(q/2)$ where $e' = \langle e, G^{-1}(w^T)\rangle$. We know that $||e'|| \leq m\beta$. Thus, we can see that as long as $||e'|| < q/4$, decryption will work properly. All ciphertexts with noise less than β_{max} can be decrypted properly when $\beta_{max} := q/4m$.

Evaluation Depth Note that in order for ciphertexts to have less than β_{max} noise, we must also consider the evaluation depth of the circuit. Let d be the depth of a circuit f, namely we assume that there are d layers of multiplication. Assume the circuit takes in fresh ciphertexts $C_1, C_2, \dots C_n$ each with β_{init} noise. Each multiplication will increase the noise by a multiplicative factor of $(m+1)$. Thus evaluated ciphertext $C = f(C_1, C_2, \dots C_n)$ will have $(m+1)^d \beta_{init}$ noise. As we know $\beta_{init} = mB_\chi$ and we want decryption to work properly, we want $(m+1)^d \beta_{init} < q/(4m)$. In other words, we want $(m+1)^d 4m^2 B_\chi < q$. Given our initial parameter choice, we see that correctness is preserved when $q = B_\chi 2^{\omega(d\lambda \log \lambda)}$.

3.1.4 Construction of NTRU FHE

In this subsection, we present an alternative FHE scheme based on the NTRU encryption scheme of Hofftein et al [HPS98]. This encryption scheme was subsequently modified by [SS11] and [LTV13] to based the encryption scheme on lattice-based assumptions and to modify to become a FHE scheme respectively.

The scheme is defined as follows:

- $\mathsf{Keygen}(1^\lambda)$: Given security parameter, sample polynomials, $f', g \leftarrow \chi$ and set $f \stackrel{\mathsf{def}}{=} 2f' + 1$ so that $f \equiv 1 \mod 2$. If f is not invertible in R_q, then resample f', otherwise let f^{-1} be the inverse of f in R_1. Let the public key be $\mathsf{pk} \stackrel{\mathsf{def}}{=} h = [2gf^{-1}]_q \in R_q$ and the secret key be $\mathsf{sk} \stackrel{\mathsf{def}}{=} f \in R$.

- $\mathsf{Enc}(pk, m)$: The encryption algorithm is a randomized algorithm where we wish to encrypt a bit $, \in \{0,1\}$ with public key h. We sample polynomials $s, e \leftarrow \chi$ from the error distribution and output the ciphertext as

$$c \stackrel{\mathsf{def}}{=} [hs + 2e + m]_q \in R_q$$

- $\mathsf{Dec}(sk, c)$: The decryption algorithm is a deterministic algorithm where given the secret key f, let $\mu = \overset{\mathsf{def}}{=} [fc]_q$ and output $m \overset{\mathsf{def}}{=} \mu \mod 2$.

Correctness of the scheme is seen because as long as there is no reduction modulo q, we see that the decryption algorithm computes

$$[fc]_q \mod 2 = [fhs + 2fe + fm]_q \mod 2 = 2gs + 2fe + fm \mod 2 = m$$

The choice of parameter $\phi(x) = x^n + 1$ ensures that there is no reduction modulo q.

The security of the NTRU encryption scheme presented above is based on the Ring Learning with Errors assumption and also the Decisional Small Polynomial Ratio (DSPR) assumption.

Definition 14 (Decisional Small Polynomial Ratio Assumption)**.** *Let $\phi(x) = \mathbb{Z}[x]$ be a polynomial of degree n, and let $q \in \mathbb{Z}$ be a prime integer, and let χ denote a distribution over the ring $R \overset{\mathsf{def}}{=} \mathbb{Z}[x]/\langle\phi(x)\rangle$. The DSPR assumption says that it is hard to distinguish between the following two distributions:*

- *A polynomial $h \overset{\mathsf{def}}{=} [2gf^{-1}]_q$, where f' and g are sampled from distribution χ and $f = 2f' + 1$ and f^{-1} is the inverse of f in R_q*

- *A polynomial u sampled uniformly random over R_q.*

The security proof of the NTRU scheme uses a hybrid argument as follows:

- **Hybrid 0**: The NTRU scheme with public key h, private key f

- **Hybrid 1**: The NTRU scheme, instead replacing the public key h with a uniformly sampled h. This relies on the DSPR assumption.

- **Hybrid 2**: The NTRU scheme, except changing the ciphertext from $c = [hs + 2e + m]_q$ to $c = [u + m]_q$ where u is uniformly sampled from R_q. This relies on the RLWE assumption.

Lopez-Alt et al noticed that the NTRU encryption scheme can also be converted to a fully homomorphic scheme. This is presented in the multi-key NTRU FHE in section 3.2.4.

3.2 Multi-Key FHE

Multi-key Fully Homomorphic Encryption (MFHE) is a paradigm introduced by Lopez-Alt, Tromer, and Vaikuntanathan [LTV13], in which ciphertexts encrypted under separate keys can still be used in computation in the same function. Assume there are N parties who jointly wish to encrypt their data under their own respective keys, and share the ciphertexts C with a third party. Through a *ciphertext expansion* process, we can derive a new ciphertext $\hat{C}$ encrypted under the set of all N parties public key.

More formally, the Multi-Key FHE scheme is a tuple of algorithms defined as follows:

Definition 15 (Multi-Key FHE). *A multi-key FHE scheme is a tuple of algorithms described as follows:*

- $\mathsf{Setup}(1^\lambda, 1^d) \to \mathsf{params}$: *Setup is a PPT algorithm given security parameter λ, circuit depth d, and outputs the system parameters, where all the other algorithms take in the parameters implicitly.*

- $\mathsf{Keygen}(\mathsf{params}) \to (pk, sk)$: *The key generation algorithm outputs the public and secret keys.*

- $\mathsf{Encrypt}(pk, \mu) \to ct$. *The encryption algorithm is a randomized algorithm that uses a public key to encrypt a message μ, outputting ciphertext ct.*

- $\mathsf{Expand}((pk_1, \ldots, pk_N), i, ct) \to \widehat{ct}$. *The expansion algorithm is a deterministic algorithm that, given a sequence of N public-keys and a fresh ciphertext ct, output an expanded ciphertext $\widehat{ct}$.*

- $\mathsf{Eval}(params, \mathcal{C}, \widehat{ct_1}, \ldots, \widehat{ct_l}) \to \widehat{ct}$. *The evaluation algorithm is a deterministic algorithm and is given a circuit $\mathcal{C}$ along with l expanded ciphertext, and outputs an evaluated ciphertext $\widehat{ct}$.*

- $\mathsf{Decrypt}(params, (sk_1, \ldots, sk_N), \widehat{ct}) \to \mu$. *The decryption algorithm is a deterministic algorithm that is given a ciphertext and a sequence of N secret keys. It outputs the underlying message μ.*

The notion of a "fresh" ciphertext as defined above is a ciphertext that has not had evaluations computed on it. In other words, a ciphertext is fresh if it was generated from the encryption algorithm.

The MFHE scheme keeps the FHE algorithms SetUp, KeyGen, Encrypt, Eval, Decrypt and also introduces a new algorithm Expand that transforms a ciphertext encrypted under one key in the GSW FHE scheme to become encrypted under multiple keys in a modified GSW FHE scheme. We also modify Decrypt to attain a single round N-of-N threshold decrpytion scheme, where each of the N parties will output a partial decryption of an expanded ciphertext, such that combining the partial decryptions will result in the decrypted expanded ciphertext.

While C_1 may have been encrypted under the first party's key and C_2 may have been encrypted under the second party's key, we can expand the two ciphertexts to obtain $\hat{C}_1, \hat{C}_2$ such that joint homomorphic computation is possible on $\hat{C}_i$. Evaluation on the expanded ciphertexts will yield $\hat{C} = f(\hat{C}_1, \hat{C}_2 \ldots, \hat{C}_i)$ for a function f with i inputs. In order to decrypt $\hat{C}$ in a secure manner, all N parties will user their secret keys independently to output a partial decryption p_i, such that $\sum_i p_i$ will yield the plaintext of ciphertext $\hat{C}$.

To present the GSW Multi-Key FHE as constructed in [MW16], we first present auxiliary algorithms and definitions, namely the expanded gadget matrix and linear combination function, which allows for the construction of Multi-Key FHE.

3.2.1 Notation & Prelim for GSW Multi-Key FHE

To begin, let us define some notation. Define $\hat{sk} = \hat{t} = [t_1, t_2, \dots t_N] \in \mathbb{Z}_q^{mN}$ be the concatenation of all the secret keys. We define the expanded gadget matrix $\hat{G_N} \in \mathbb{Z}_q^{nN \times mN}$ as follows:

$$\hat{G_N} = \begin{bmatrix} G & & & \\ & G & & \\ & & \dots & \\ & & & G \end{bmatrix}$$

There is also an inverse function $\hat{G_N^{-1}}$ such that for any matrix V with nN rows, we see that $\hat{G_N^{-1}}(V) \in \{0,1\}^{mN \times mN}$ and that $\hat{G_N}\hat{G_N^{-1}}(V) = V$.

In order to define $\hat{C}$, let us work in the simplified case where $N = 2$. Assume that C_1, the encryption of μ_1, is encrypted under party 1's public key. We wish to construct $\widehat{C_1}$ such that

$$\hat{C_1} = \begin{bmatrix} C_1 & X \\ 0 & C_1 \end{bmatrix}$$

Such that $\hat{t}\hat{C_1} = \mu_1 \hat{t}\hat{G_2}$. We see that $\hat{t}\hat{C_1} = [t_1 C_1, t_1 X + t_2 C]$. Note that $t_2 = (-s_2, 1)$ and public key $A_1 = \begin{bmatrix} B \\ b_1 \end{bmatrix}$ where $b_2 = t_2 B + e$ so we see

$$t_2 C_1 = (-s_2, 1)(A_1 R + \mu_1 G) = -s_2 BR + b_1 R + \mu_1 t_2 G \approx (-b_2 + b_1)R + \mu_1 t_2 G$$

Thus, if we set X such that $t_1 X \approx (b_2 - b_1)R$, we see that $t_1 X + t_2 C \approx \mu t_2 G$ and thus we have attained our desired property for $\widehat{C_1}$. We can create X through two auxiliary functions, one that also encrypts *the randomness* used to encrypt μ_1, and one function that uses this subsequent information to generate X.

Firstly, define EncRand [MW16]. In the standard GSW FHE scheme, we encrypt a ciphertext $C = AR + \mu G$ for some $R \xleftarrow{\$} \{0,1\}^{m \times m}$.

Definition 16 (Encryption of Randomness)**.** *Given a binary matrix R,* $\mathsf{EncRand}(R,t)$ *outputs an array of m^2 ciphertexts $\mathcal{U}$ of all the bits of R, encrypted under a secret key t.*

In other words, if we index $\mathcal{U}$ as $\mathcal{U}_{i,j}$, where $i,j \in [0:m-1]$, we can think of $\mathcal{U}_{i,j} = AR' + R_{i,j}G$ where $R_{i,j}$ is the bit at the ith row and jth column of R. The randomness R' used to encrypt $R_{i,j}$ can be subsequently forgotten.

Now, we can use $\mathcal{U}$ to generate X. We define a "linear combination" function LComb to achieve such property:

Definition 17 (Linear Combination Function [MW16])**.** *Assume we have a binary $m \times m$ matrix R, such that $\mathcal{U} \leftarrow$* $\mathsf{EncRand}(R,t)$ *for some secret key t. Let v be a vector.* $\mathsf{LComb}(\mathcal{U},v)$ *which outputs C_{lc}, such that $tC_{lc} = vR + e$.*

To implement the function, we define m^2 matrices $Z_{i,j} \in \mathbb{Z}_q^{n\times m}$ where $i,j \in [1:m]$ as follows. The matrix $Z_{i,j}$ is mostly zeros, except at the last row and jth column where $Z_{i,j} = v[i]$. In other words, $Z_{i,j}[n,j] = v[i]$, and all other entries of $Z_{i,j}$ are 0.

With this, we can define $C_{lc} = \sum_{i=1,j=1}^{m,m} U_{i,j}G^{-1}(Z_{i,j})$. A full proof of correctness can be found in [MW16].

Now, if $\mathcal{U}$ is the encrypted randomness associated with ciphertext C under party 1's public key, if we run $\mathsf{LComb}(\mathcal{U},(b_2-b_1))$ to attain C_{lc}, we see that $t_1C_{lc} = (b_2-b_1)R + e$ for some error e, which is exactly what we desire to set X as.

3.2.2 GSW Multi-Key FHE

Given our preliminary auxiliary functions, we can now describe the following functions in a Multi-Key FHE, based off of the GSW FHE Scheme:

- $\mathsf{MFHE.SetUp}(1^\lambda, 1^d)$: We run $\mathsf{GSW.SetUp}$ to set up the parameters $\mathsf{params} = (q,n,m,\chi,B_\chi,\mathbf{B})$.

- $\mathsf{MFHE.KeyGen}(\mathsf{params})$: for each party, generate the (sk,pk) pair for a GSW scheme as follows:

– $sk = t \leftarrow \mathsf{GSW.SKGen}(\mathsf{params})$

– $pk = \mathbf{A} = \leftarrow \mathsf{GSW.PKGen}(\mathsf{params})$.

- $\mathsf{MFHE.Encrypt}(pk, \mu)$: The encryption scheme used here is a modification of the GSW encryption scheme. We output the pair $c = (C, \mathcal{U})$ defined as follows:

  – $C \leftarrow \mathsf{GSW.Enc}(pk, \mu) = AR + \mu G$

  – $\mathcal{U} \leftarrow \mathsf{EncRand}(R)$.

- $\mathsf{MFHE.Expand}((pk_1, \ldots pk_N), i, c)$. Given all N parties' public keys, and a *fresh* ciphertext [2] party i wishes to create an expanded ciphertext $\hat{C} \in \mathbb{Z}_q^{nN \times mN}$.

  – First, let $X_j \leftarrow \mathsf{LComb}(\mathcal{U}, (b_j - b_i))$ for $j \neq i$.

  – Now we construct $\hat{C}$. Imagine this matrix as a concatenation of N^2 sub-matrices, such that each $C_{a,b} \in \mathbb{Z}_q^{n \times m}$ for $a, b \in [1 : N]$ is defined as follows:

$$C_{a,b} = \begin{cases} C & \text{if } a = b \\ X_j & \text{if } b = j, a = i \neq j \\ 0 & \text{otherwise} \end{cases}$$

  In other words, a fresh expanded ciphertext $\widehat{C}$ will have the single-key ciphertext C along its diagonals, and the linear combination matrices X_j along the ith row.

- $\mathsf{MFHE.Eval}$: for the multi-key evaluation function, we can simply use the GSW addition and multiplication algorithms, except in an expanded dimension and using the expanded gadget matrices $\hat{G_N}, \hat{G_N^{-1}}$. In other words, for ciphertexts $\widehat{C_1}, \widehat{C_2}$

  – $\widehat{C^+} = \widehat{C_1} + \widehat{C_2}$

  – $\widehat{C^\times} = \widehat{C_1} \hat{G_N^{-1}}(\widehat{C_2})$.

[2]Note: we need $c = (C, \mathcal{U})$ to be fresh for this particular MFHE scheme because it is "single hop." For multi-hop FHE schemes, see [BP16, PS16]

- $\mathsf{MFHE.Decrypt}(\widehat{t}, \widehat{C})$: Given all N secret keys $\widehat{t}$ we can run the GSW decryption scheme in an expanded manner. First define expanded $\widehat{w} = [0, 0, \ldots 0, \lceil q/2 \rceil]$ of length nN. We can decrypt a ciphertext $\widehat{C}$ by running

$$v = \widehat{t}\widehat{C}\widehat{G_N^{-1}}(\widehat{w})$$

and outputting $\left| \left\lfloor \frac{v}{q/2} \right\rceil \right|$ as the decryption.

Note, however, that $\mathsf{MFHE.Decrypt}$ will not be called in a practical setting, because no single entity will (or should) possess all N secret keys. Instead, we run an interactive protocol such that each party i, using her secret key t_i, can output a partial decryption p_i of the ciphertext $\widehat{C}$, as follows:

- $\mathsf{MFHE.PartDec}(t_i, \widehat{C})$: Given a secret key t_i, we first parse $\widehat{C}$ to consist of N sub-matrices $\widehat{C}^{(i)} \in \mathbb{Z}_q^{n \times mN}$ such that

$$\widehat{C} = \begin{bmatrix} \widehat{C}^{(1)} \\ \vdots \\ \widehat{C}^{(N)} \end{bmatrix}$$

We define expanded $\widehat{w} = [0, 0, \ldots 0, \lceil q/2 \rceil]$ of length nN. We can partially decrypt a ciphertext $\widehat{C}$ by outputting

$$p_i = t_i \widehat{C}^{(i)} \widehat{G_N^{-1}}(\widehat{w}) + e_i^{sm}$$

where $e_i^{sm} \xleftarrow{\$} [-B^{dec}, B^{dec}]$ is some random smudging noise to mask t_i. We define $B^{dec} = 2^{d\lambda \log \lambda} B_\chi$.

- $\mathsf{MFHE.FinDec}(p_1, \ldots p_N)$: given N partial decryptions, compute the sum $p = \sum_i p_i$. We then output $\left| \left\lfloor \frac{p}{q/2} \right\rceil \right|$ as the decryption.

3.2.3 Noise Analysis

Intuitively, we can see that due to the $\mathsf{MFHE.Expand}$ procedure, a multi-key FHE ciphertext will contain more noise than a GSW ciphtertext. Assume that currently N parties have contributed ciphertexts; in otherwords $\widehat{C}$ has dimension $nN \times mN$. Define $n' = nN$ and $m' = mN$.

In order to preserve correctness, we will show that as long as a ciphertext $\widehat{C}$ has noise $\beta_{final} \leq q/(4m')$, decryption will correctly occur.

First, we note that a fresh expanded ciphertext $\widehat{C}$ –that is, a ciphertext of dimension $n' \times m'$ without having being evaluated –is a β_{init}-noisy ciphertext, where $\beta_{init} \leq (m^4 + m)B_\chi$ [MW16]. We see that $\beta_{init} = 2^{O(\log \lambda)} B_\chi$.

In order to show correctness of evaluation, let $\widehat{C}_1, \ldots, \widehat{C}_l$ be freshly expanded ciphertexts corresponding to bits $\mu_1, \ldots \mu_l$. Assume we are given a circuit $\mathcal{C}$ of depth d, such that $\mu = \mathcal{C}(\mu_1, \mu_2, \ldots \mu_l)$ and $\widehat{C} = \mathcal{C}(\widehat{C}_1, \widehat{C}_2, \ldots \widehat{C}_l)$. We know that $\hat{t}\hat{C} = \mu\hat{t}\hat{G} + e$ where $||e||_\infty \leq \beta_{init}(m'+1)^d$. If we define $\beta_{final} = \beta_{init}(m'+1)^d = (m^4+m)B_\chi(m'+1)^d$, we see that $\beta_{final} = 2^{O(d \log \lambda)} B_\chi$. As we chose $q = B_\chi 2^{\omega(d\lambda \log \lambda)}$, we see that $\beta_{final} \leq q/(4m')$, which satisfies correctness of decryption.

To show the correctness of partial decrytion, assume we are given N shares $p_1, \ldots p_N$ which correspond to the partial decryption of some ciphertext $\hat{C}$ for bit μ. We see that

$$p = \sum_i p_i = \sum_i t_i \hat{C}^{(i)} + \sum_i e_i^{sm} = \hat{t}\hat{C}\widehat{G_N^{-1}}(\widehat{w}) + \sum_i e_i^{sm}$$

We know that $\hat{t}\hat{C} = \mu\hat{t}\hat{G} + e$ where $||e||_\infty \leq \beta_{final}$. Thus, we see that $e\widehat{G_N^{-1}}(\widehat{w})$ has noise $\beta_{final} mN = 2^{O(d \log \lambda)} B_\chi$. We also see that $|| \sum_i e_i^{sm} ||_\infty \leq NB^{dec} = 2^{O(d\lambda \log \lambda)} B_\chi$. Thus, the noise from the decryption $e\widehat{G_N^{-1}}(\widehat{w})$ and $\sum_i e_i^{sm}$ when summed together is less than $q/4$, which shows correctness of our partial decryption.

3.2.4 NTRU Multi-Key FHE

In this subsection, we describe the extension of NTRU to create a multi-key FHE scheme, as designed by [LTV13].

We now show the extension of NTRU to the multi-key setting, first presented in [LTV13].

Compared to the multi-key construction by Mukherjee and Wichs [MW16] based on the GSW FHE Scheme [GSW13], the NTRU multi-key FHE scheme does not require public parameters for the multi-key setting, and the size of the ciphertext stays constant, while the size of the ciphertext grows with $O(N^2)$ in [MW16], where N is the number of parties participating in the multi-key FHE computation.

The multi-key extension of NTRU is presented as follows. We first present a somewhat-homomorphic encryption scheme. We can then use modulus reduction to create a leveled-fully homomorphic scheme, such that computing the decryption circuit on the ciphertext is possible to make the scheme fully homomorphic.

- $\mathsf{Keygen}(1^\kappa)$: Each party individually samples $f', g \leftarrow \chi$ and sets $f \overset{def}{=} 2f' + 1$ so $f \equiv 1(\mod 2)$ If f is not invertible in R_q, resample f', otherwise let f^{-1} be the inverse of f in R_q. We let the public key be

  $$pk \overset{def}{=} h = [2gf^{-1}]_q \in R_q$$

  and the secret key be

  $$sk \overset{def}{=} f \in R_q$$

  In addition, we sample two more noise vectors $\tilde{s}, \tilde{e} \leftarrow \chi^{\lceil \log q \rceil}$, and we compute the evaluation key as a vector

  $$ek \overset{def}{=} [h\tilde{s} + 2\tilde{e} + Pow(f)]_q \in R_q^{\lceil \log q \rceil}$$

  where the power function $Pow(f)$ is the bitwise representation of f multiplied by its corresponding index. In other words, if $f = f_{\log}q \dots f_1 f_0$ then $Pow(f) = 2^{\log q} f_{\log q}, \dots 2^1 f_1, 2^0 f_0$.

- $\mathsf{Enc}(pk, m)$: To encrypt $m \in \{0,1\}$ with public key $pk = h$, sample polynomials $s, e \leftarrow \chi$ and output ciphertext

$$c \overset{def}{=} [hs + 2e + m]_q \in R_q$$

- $\mathsf{Dec}(sk_1, \ldots sk_N, c)$: To decrypt ciphertext $c \in R_q$, assume that ciphertext c is a function of encrytions by a subset of parties $i \in F$ for some set F. In other words, $c = a(c_i, c_k, \ldots c_k)$ for $i, j, \ldots k \in F$. We can decrypt as follows by first computing

$$\mu \overset{def}{=} [(\Pi_{i \in F} f_i)c]_q \in R_q$$

and then outputting

$$m \overset{def}{=} \mu (\mod 2)$$

- $\mathsf{Eval}(C, (c_1, pk_1, ek_1), \ldots (c_l, pk_l, ek_l))$: Given a l-variate boolean circuit $C : \{0,1\}^l \rightarrow \{0,1\}$ of depth D, the Eval operation computes c that is the evaluation of the ciphertext. When given ciphertexts c_a, c_b, let K_a and K_b be the set of distinct public keys associated with each ciphertext respectively. The public key set of a fresh encryption c_i is simply $K_i = \{pk_i\}$. We now see how K and c are computed through addition and multiplication operations:

  - **Addition**: given ciphertexts c_a, c_b with corresponding public key sets K_a, K_b, output $c^+ = [c_a + c_b]_q \in R_q$ as an encryption of the *sum* of the underlying messages. We define $K^+ = K_a \cup K_b$.

  - **Multiplication**: given ciphertexts c_a, c_b and corresponding public key sets K_a, K_b, we first denote $c_0 = [c_a \cdot c_b]_q \in R_q$. If $K_a \cap K_b = \emptyset$, then output $c^* = c_0$ and $K^* = K_a \cup K_b$.

    Otherwise, let $K' = K_a \cap K_b = \{pk_{i_1}, \ldots pk_{i_t}\}$. For $j \in [t]$, compute $[\langle Bit(c_{j-1}), ek_{i,j} \rangle]_q$. This is the "relinearization" function that uses the evaluation key. At the end, let $c^* = c_t$ and $K^* = K_a \cup K_a$.

This preserves the property that $\Pi_{i \in K^*} f_i c^*$ can be correctly decrypted [3].

Correctness and security of this multi-key FHE scheme is shown in [LTV13].

Shortcomings One issue with the decryption process is that all secret keys must be used together at once to decrypt ciphertext c. In an ideal setting, each party, using his own secret key sk_i, should be able to output d_i such that $\sum k_i d_i = m$ for some coefficient k_i. The current decryption setup does not allow for this. In [LTV13], in order to obtain a multi-party protocol, in which parties cannot output sk_i in the clear for obvious security reasons, a MPC protocol for decryption is presented such that no party j will see sk_i, yet the decryption can still be computed. Doröz et al also provide extensions to the NTRU FHE scheme [DS16].

We present modifications to the NTRU multi-key FHE scheme in section 4.2 to create a threshold multi-key FHE scheme.

3.3 Threshold FHE

Firstly, we should define a threshold FHE scheme. A threshold Fully Homomorphic Scheme, first introduced in [BGG$^+$17], builds on top of a FHE scheme with a single public/secret key. Through a secret sharing scheme, the secret key sk can be distributed among N parties, with party i having their own secret share sk_i. Given a ciphertext C, as long as k-of-N parties correctly output their partial decryptions using sk_i, the plaintext bit from C can be correctly recovered.

A threshold FHE scheme uses a undelying fully homomorphic encryption scheme such that the decryption procedure has a linear property. Say that a FHE decryption scheme uses sk to decrypt a cipher text ct to obtain p. If we used two shares sk_1, sk_2 such that $sk_1 + sk_2 = sk$, and we applied used sk_1 and sk_2 as secret keys for the decryption scheme for ct to output p_1, p_2 respectively, then $p_1 + p_2 = p$. From a high level overview, a threshold FHE should have the following algorithms.

[3]without using the "relinearization" function, to decrypt c_0 we would need to compute $\Pi_{i \in K_a} f_i \Pi_{j \in K_b} f_j c_0 = \Pi_{i \in K^*} f_i \Pi_{j \in K_a \cap K_b} f_j c_0$ which reveals information about the circuit for which c was computed from.

- $\mathsf{tFHE.KeyGen}(\mathsf{params}) \to (pk, sk)$The key generation algorithm takes in the public parameters of the FHE scheme, and outputs a public and secret key pair.

- $\mathsf{tFHE.KeyShare}(sk, \mathbb{A}) \to (sk_1, \ldots sk_N)$ The key sharing algorithm takes in a secret key, an access structure $\mathbb{A}$ and outputs a set of N shares of the secret keys to distribute to the N parties.

- $\mathsf{tFHE.Enc}(pk, \mu) \to ct$: The encryption algorithm is a randomized algorithm that, given the public key and a plaintext bit μ, outputs the corresponding ciphertext ct.

- $\mathsf{tFHE.Eval}(\mathcal{C}, (ct_1, \ldots ct_l)) \to ct$: Given a set of ciphertexts $(ct_1, ct_2, \ldots ct_l)$, and a circuit $\mathcal{C}$, the evaluation algorithm will compute $ct \leftarrow \mathcal{C}(ct_1, ct_2, \ldots ct_k)$.

- $\mathsf{tFHE.PartDec}(sk_i, ct) \to p_i$: The partial decryption algorithm, given a party's secret key share and a ciphertext, will output a partial decryption p_i.

- $\mathsf{tFHE.FinDec}(\{p_i\}, ct) \to \mu$: The final decryption algorithm is a deterministic algorithm where, given a set of partial decryptions associated with a ciphertext, will output the plaintext associated with ct if the set of partial decryptions is a valid set, then. Otherwise, output $\perp$.

The GSW FHE scheme is a valid FHE scheme from which we can build a threshold FHE scheme. Thus, the Setup, KeyGen, Enc, and Eval algorithms for the tFHE scheme are simply the same algorithms as the GSW FHE scheme. However, the key sharing and partial decryption algorithms are modified.

In the following two subsections, we detail two ways to obtain a threshold fully homomorphic encryption scheme with a threshold access structure. In a k-of-n threshold access structure, any k out of n parties may use their secret keys to correctly decrypt a ciphertext.

One key insight we note for designing a threshold FHE scheme is that when the partial decryption algorithm is run, a smudging noise must be added to the partial decryption as the output of a party, in order for the party's secret key to not be

	{0,1}-LSSS	Shamir
Secret Shares	$O(n^{4.2})$	$O(n)$
Ciphertext Growth	$O(1)$	$O(n \log n)$

Table 3.1: Comparison of threshold FHE schemes [BGG+17].

learned. Thus, naively applying Shamir Secret Sharing Scheme may not work since the coefficients multiplied to the partial outputs may be arbitrarily large [Sha79], thus blowing up the noise. To compensate for this issue, we can limit the noise by either having the coefficients multiplied to the partial outputs be binary, or increase our underlying prime q such that the Lagrange coefficients can be bounded.

3.3.1 {0,1}-LSSS Threshold FHE

The {0,1}-LSSS Threshold FHE scheme was proposed in [BGG+17]. By using this secret sharing scheme, when combining the partial decryptions, since the reconstruction coefficients are binary, there is no contribution to the noise in the final decryption ciphertext from the secret sharing scheme. Since the underlying FHE scheme can tolerate an amount of noise, applying the {0,1}-LSSS Threshold secret sharing structure will preserve correctness.

- $\mathsf{tFHE.KeyGen}(\mathsf{params}) \to (pk, sk)$: Simply run $(pk, sk) \leftarrow \mathsf{GSW.KeyGen}(\mathsf{params})$.
- $\mathsf{tFHE.KeyShare}(sk, \mathbb{A}) \to (sk_1, \dots sk_N)$: Use the $\{0,1\}$-LSSS secret sharing method as described in figure 2.4.1 to generate N sets of shares, in which party i receives secret key share sk_i.
- $\mathsf{tFHE.Enc}(pk, \mu) \to ct$: The encryption algorithm simply runs $ct \leftarrow \mathsf{GSW.Enc}(pk, \mu)$
- $\mathsf{tFHE.Eval}(\mathcal{C}, (ct_1, \dots ct_l)) \to ct$: The evaluation algorithm simply runs $ct \leftarrow \mathsf{GSW.Eval}(\mathcal{C}, (ct_1, \dots ct_l))$
- $\mathsf{tFHE.PartDec}(sk_i, ct) \to p_i$: The partial decryption algorithm uses sk_i and computes $p_i \leftarrow \mathsf{GSW.Dec}(sk_i, ct)$. In other words, compute

$$p_i' = (sk_i)(ct)G^{-1}w$$

The algorithm then outputs $p_i = p_i' + e$ where $e \in \mathbb{Z}_q$ is sampled from error distribution χ.

- $\mathsf{tFHE.FinDec}(\{p_i\}, ct) \to \mu$: The final decryption algorithm, given a set of $\{p_i\}$ partial decryptions, chooses a set of $\{p_i\}$ such that the corresponding secret keys span the monotone span program as originally defined in the folklore secret sharing scheme. The partial decrpytion is computed as $p = \sum_i p_i$ for a set of p_i span the monotone span program. It then outputs $\mu = 0$ if $p \in [-\lfloor \frac{q}{4} \rceil, \lfloor \frac{q}{4} \rceil]$ and $\mu = 1$ otherwise.

Note that the secret key shares in this scheme consists of many vectors. As the folklore secret sharing scheme generates on the order of $O(N^{4.2})$ shares for each party, the partial decryption algorithm will output $O(N^{4.2})$ partial decryptions. There exists an efficient combining algorthm for tFHE.FinDec in [BGG+17].

The correctness of this scheme lies in the noise analysis. Due to the underlying correctness of the GSW FHE scheme, and the fact that the noise output from tFHE.PartDec is from the same distribution as the noise in the GSW FHE scheme, correctness is maintained applying the threshold scheme on top of GSW.

The security of this scheme can be argued via a hybrid argument.

- **Hybrid 0**: The threshold FHE scheme, as stated above.

- **Hybrid 1**: Same as the threshold FHE scheme, except the challenger simulates the partial decryptions. When an set of partial decryptions that is not within the access structure is given, this hybrid is computationally indistinguishable from the original tFHE scheme.

- **Hybrid 2**: Same as Hybrid 1, except the challenger now simulates secret key shares sk_i. From the information-theoretic security of the secret sharing scheme, these two hybrids are indistinguishable.

Chapter 4

Threshold Multi-Key Fully Homomorphic Encryption

In this section, we detail new constructions combining secret sharing techniques where the reconstruction coefficient is binary, with valid multi-key FHE schemes, to obtain a threshold multi-key FHE scheme.

In a threshold multi-key setting, we wish to thresholdize the decryption process. A threshold multi-key fully homomorphic encryption scheme is an extension of a multi-key FHE scheme. However, instead having a strict N-of-N threshold decryption requirement, we can use the thresholdizing process presented in [BGG$^+$17] to allow for a k-of-N threshold decryption process. All N parties will still encrypt their ciphertexts under their own generated public/secret key.

We denote a public-secret key pair as (pk_i, sk_i) in the multi-key FHE scheme, and party j's share of sk_i is denoted as $sk_i^{(j)}$.

Definition 18 (Threshold Multi-Key FHE)**.** *A threshold multi-key FHE scheme is a tuple of algorithms described as follows:*

- $\mathsf{Setup}(1^\lambda, 1^d) \rightarrow$ *params: Setup is a PPT algorithm given security parameter λ, circuit depth d, and outputs the system parameters, where all the other algorithms take in the parameters implicitly.*

- $\mathsf{Keygen}(\mathit{params}) \to (pk_i, sk_i)$: *The key generation algorithm outputs the public and secret keys.*

- $\mathsf{KeyShare}(sk_i, \mathbb{A}_i) \to (sk_i^{(1)}, \dots sk_i^{(N)})$ *The key sharing algorithm takes in a secret key, an access structure $\mathbb{A}_i$ for the i^{th} party and outputs a set of N shares of the secret keys to distribute to the N parties.*

- $\mathsf{Encrypt}(pk, \mu) \to ct$. *The encryption algorithm is a randomized algorithm that uses a public key to encrypt a message μ, outputting ciphertext ct.*

- $\mathsf{Expand}((pk_1, \dots, pk_N), i, ct) \to \widehat{ct}$. *The expansion algorithm is a deterministic algorithm that, given a sequence of N public-keys and a fresh ciphertext ct, output an expanded ciphertext $\widehat{ct}$.*

- $\mathsf{Eval}(params, \mathcal{C}, \widehat{ct_1}, \dots, \widehat{ct_l}) \to \widehat{ct}$. *The evaluation algorithm is a deterministic algorithm and is given a circuit $\mathcal{C}$ along with l expanded ciphertext, and outputs an evaluated ciphertext $\widehat{ct}$.*

- $\mathsf{PartDec}(sk_i, ct) \to p_i$: *The partial decryption algorithm, given a party's secret key share and a ciphertext, will output a partial decryption p_i.*

- $\mathsf{FinDec}(\{p_i\}, ct) \to \mu$: *The final decryption algorithm is a deterministic algorithm where, given a set of partial decryptions associated with a ciphertext, will output the plaintext associated with ct if the set of partial decryptions is a valid set, then. Otherwise, output $\perp$.*

4.1 Threshold Multi-Key FHE from GSW

In this section, we present a construction of threshold multi-key FHE from the GSW FHE scheme. This is a combination of [MW16] and [BGG+17], which was also in the recent work of [BJMS18].

- $\mathsf{Setup}(1^\lambda, 1^d) \to \mathsf{params}$: This algorithm runs $\mathsf{GSW.Setup}(1^\lambda, 1^d)$ to obtain random matrix $B \in \mathbb{Z}_q^{n-1\times m}$

- $\mathsf{Keygen}(\mathsf{params}) \to (pk_i, sk_i)$: The key generation algorithm runs $\mathsf{GSW.Setup}(1^\lambda, 1^d)$ to obtain public key $A \in \mathbb{Z}_q^{n\times m}$ and private key $t \in \mathbb{Z}_q^n$ for the i^{th} party in the multi-key FHE scheme.

- $\mathsf{KeyShare}(sk_i, \mathbb{A}_i) \to (sk_i^{(1)}, \dots sk_i^{(N)})$ The key sharing uses the {0,1}-LSSS secret sharing algorithm with input sk_i of the multi-key FHE scheme, and outputs shares $sk_i^{(j)})$ to distribute to the N parties.

- $\mathsf{Encrypt}(pk, \mu) \to ct$. The encryption algorithm runs $\mathsf{GSW.Enc}(pk, \mu)$.

- $\mathsf{Expand}((pk_1, \dots, pk_N), i, ct) \to \widehat{ct}$. The expansion algorithm, given a fresh GSW ciphertext ct, runs $\mathsf{MFHE.Expand}((pk_1, \dots, pk_N), i, ct)$ and outputs $\widehat{ct}$.

- $\mathsf{Eval}(params, \mathcal{C}, \widehat{ct_1}, \dots, \widehat{ct_l}) \to \widehat{ct}$. The evaluation algorithm runs

$$\mathsf{MFHE.Eval}(params, \mathcal{C}, \widehat{ct_1}, \dots, \widehat{ct_l})$$

  and outputs an evaluated ciphertext $\widehat{ct}$.

- $\mathsf{PartDec}(sk_i^{(j)}, ct) \to p_i$: The partial decryption algorithm, runs $\mathsf{MFHE.PartDec}(sk_i^{(j)}, ct)$ and outputs a partial decryption $p_i^{(j)}$.

- $\mathsf{FinDec}(\{p_i^{(j)}\}, ct) \to \mu$: The final decryption algorithm runs $\mathsf{tFHE.FinDec}(\{p_i^{(j)}\}, ct)$ and outputs μ if the set of shares is in the access structure.

We will now analyze the correctness and security of the protocol. From a high level, the correctness of the protocol is maintained when the noise from the FHE evaluation and partial decryptions do not affect the correctness of the final decryption. The security of the protocol is a series of hybrid arguments.

4.1.1 Correctness

Consider an expanded multi-key FHE which has been evaluated on a depth $< d$ circuit. We will now show that for set of partial decryptions that forms a valid set within the access structure, the final decryption algorithm will output the correct μ.

More formally, to decrypt $\widehat{C}$ we first parse it to consist of N sub-matrices $\widehat{C}^{(i)} \in \mathbb{Z}_q^{n\times mN}$ such that

$$\widehat{C} = \begin{bmatrix} \widehat{C}^{(1)} \\ \vdots \\ \widehat{C}^{(N)} \end{bmatrix}$$

.

Recall that the partial decryption in the threshold multi-key FHE scheme consists of

$$p_i = t_i \widehat{C}^{(i)} \widehat{G_N^{-1}}(\widehat{w}) + e_i^{sm}$$

In our threshold multi-key FHE scheme, a partial decryption of the i^{th} sub-matrix in the ciphertext, using party j's secret key share, is

$$p_i^{(j)} = t_i^{(j)} \widehat{C}^{(i)} \widehat{G_N^{-1}}(\widehat{w}) + e_i^{sm}$$

Where $t_i^{(j)}$ is the $sk_i^{(j)}$ and $e_i^{sm} \xleftarrow{\$} [-B^{dec}, B^{dec}]$ is some random smudging noise to mask $t_i^{(j)}$.

The final decryption in the threshold multi-key FHE scheme would be as follows. First, for every i, we wish to find a set of $p_i^{(j)}$ such that they form a valid access set for reconstructing $p_i = t_i \widehat{C}^{(i)} \widehat{G_N^{-1}}(\widehat{w}) + e'_i$. We can subsequently define

$$p_i = \sum_j p_i^{(j)}$$

as the reconstruction coefficient from the {0,1}-LSSS is binary. Finally, we define

$$p = \sum_i p_i$$

and output $\mu = 0$ if $p \in [-\lfloor \frac{q}{4} \rceil, \lfloor \frac{q}{4} \rceil]$ and $\mu = 1$ otherwise.

We see that

$$\begin{aligned}
\sum_{i\in[N]} p_i &= \sum_{i\in[N]}\sum_{j\in\mathbb{A}_i} p_i^{(j)} \\
&= \sum_{i\in[N]} t_i \widehat{C}^{(i)} \widehat{G_N^{-1}}(\widehat{w}) + \sum_{i\in[N]}\sum_{j\in\mathbb{A}_i} e_i^{(j)} \\
&= \widehat{t}\widehat{C}\widehat{G_N^{-1}}(\widehat{w}) + e_{sm} \\
&= \mu\lceil q/2 \rceil + e' + e_{sm}
\end{aligned}$$

Where $e_{sm} = \sum_{i\in[N]}\sum_{j\in\mathbb{A}_i} e_i^{(j)}$. From the correctness of multi-key FHE's partial decryption, shown in previous sections, we know that $|e_{sm}| \leq NB_{smdg}^{dec} = 2^{O(d\lambda\log\lambda)}B_\chi$ and $e' = e\widehat{G}^{-1}(\widehat{w}^T)$ has norm $|e'| \leq \beta'_{final} mN = 2^{O(d\log\lambda)}B_\chi$. Since we know that $q = 2^{\omega(d\lambda\log\lambda)}B_\chi$, we see that $|e' + e_{sm}| < q/4$ and correctness holds [MW16].

4.1.2 Security

The security of the threshold multi-key FHE scheme can be proved by the following hybrid arguments. Assume there exists a set of $\{p_i\}$ shares that do not form an authorized set. We wish to show that no adversary can learn any information about the underlying encryption from the partial shares, and no adversary learns information about the secret key shares either.

- **Hybrid 0**: This hybrid is the "real" threshold multi-key FHE scheme, with an unauthorized set of partial shares $\{p_i\}$.
- **Hybrid 1**: This hybrid is the same as hybrid 0, except the partial decryptions are sampled uniformly at random.
- **Hybrid 2**: This hybrid is the same as hybrid above, except the secret key shares that are generated are with respect to 0 rather than sk_i.
- **Hybrid 3**: This hybrid is the same as hybrid above, except the encryptions contain encryptions of 0 rather than m_i.

Hybrid 0 and Hybrid 1 are computationally indistinguishable from each other from the learning with errors (LWE) assumption. Specifically, we see that the following two distributions are computationally indistinguishable:

$$(\widehat{C}^{(i)}\widehat{G_N^{-1}}(\widehat{w}), p_i^{(j)} = t_i^{(j)}\widehat{C}^{(i)}\widehat{G_N^{-1}}(\widehat{w}) + e_i^{sm}) \approx_c (\widehat{C}^{(i)}\widehat{G_N^{-1}}(\widehat{w}), p'^{(i)} = u^{(i)})$$

Hybrid 1 and Hybrid 2 are statistically indistinguishable from the information theoretic security of the folklore secret sharing scheme. Due to having an unauthorized set of partial decryptions, the distribution of p_i shares are statistically indistinguishable from if the secret key shares were shares of 0.

Hybrid 2 and Hybrid 3 are computationally indistinguishable, due to the semantic security of the underlying encryption scheme. Since both the secret key shares and the partial decryptions are simulated, and the underlying FHE scheme is computationally indistinguishable, the adversary cannot distinguish between an encryption of 0 and an encryption of 1.

4.2 Threshold Multi-Key FHE from NTRU

In this section, we present a modification of the NTRU multi-key FHE scheme in [LTV13] such that we can now do a t-of-N partial decryption instead of the N-of-N decryption as in the original multi-key FHE scheme. While in the NTRU scheme, decryption requires a *product* of the secret keys multipled by the ciphertext, our modification involves running a two-round MPC protocol to obtain secret shares such that the *sum* of the partial decryptions (consisting of the products of the ciphertexts and the secret key shares) is able to be used to obtain the underlying message.

- $\mathsf{Keygen}(\mathsf{params}) \to (pk_i, sk_i)$: The key generation algorithm runs $\mathsf{NTRU.Setup}(1^\lambda, 1^d)$ to obtain public key $h = [2gf^{-1}]_q \in \mathbb{R}_q$ and private key $f \in R_q$ for the i^{th} party in the multi-key FHE scheme.

- $\mathsf{KeyShare}((sk_1, \dots sk_N), \mathbb{A}) \rightarrow (sk^{(1)}, \dots sk^{(N)})$ The key sharing uses the {0,1}-LSSS secret sharing algorithm with inputs $sk_1, \dots sk_N$ of the multi-key FHE scheme, and outputs shares $sk^{(j)})$ to distribute to the N parties. Since we wish to obtain secret key shares $sk^{(j)}$ such that $\sum_j sk^{(j)} = \prod_i sk_i$, we run a 2 Round MPC such as [GS18].

- $\mathsf{Encrypt}(pk_i, \mu) \rightarrow ct$. The encryption algorithm runs $\mathsf{NTRU.Enc}(pk_i, \mu)$.

- $\mathsf{Eval}(params, \mathcal{C}, \widehat{ct_1}, \dots, \widehat{ct_l}) \rightarrow \widehat{ct}$. The evaluation algorithm runs

$$\mathsf{NTRU.Eval}(params, \mathcal{C}, \widehat{ct_1}, \dots, \widehat{ct_l})$$

  and outputs an evaluated ciphertext $\widehat{ct}$.

- $\mathsf{PartDec}(sk^{(j)}, ct) \rightarrow p^{(j)}$: Given a secret key share $sk^{(j)}$, the partial decryption algorithm runs $\mathsf{NTRU.Dec}(sk^{(j)}, ct)$. In other words, we compute $p^{(j)} = [sk^{(j)} \cdot ct]_q + 2e$ where $e \leftarrow \chi$ is sampled from the error distribution.

- $\mathsf{FinDec}(\{p^{(j)}\}, ct) \rightarrow \mu$: The final decryption algorithm is given a set of partial shares $\{p^{(j)}\}$, and for the shares $p'^{(j)}$ that are in the access structure, let $p = \sum_j p'^{(j)}$. The algorithm then outputs $p \mod 2$.

We will now show correctness and security for the threshold multi-key NTRU scheme.

Correctness We can assume the correctness of the 2-round MPC protocol, such that with each party's input as sk_i, the MPC protocol outputs shares to each party j as $sk^{(j)}$ such that $\sum_j sk^{(j)} = \prod_i sk_i$.

Our protocol also assumes honest majority, and without loss of generality an odd number of parties participate. Assume we have a ciphertext ct that is the evaluation of ciphertexts $C(ct_1, \dots ct_N)$ for N parties. The final decryption algorithm is as follows:

$$\begin{aligned}
p &= \sum_j p^{(j)} \\
&= \sum_j [sk^{(j)} \cdot ct]_q + 2e^{(j)} \\
&= \prod_i [sk_i \cdot ct]_q + \sum_j 2e^{(j)} \\
&= 2e' + \prod_i sk_i \mu + 2e''
\end{aligned}$$

where $2e'$ is the noise from the ciphertext in the NTRU encryption and $2e''$ is the noise from the partial decryption. We see that $p \mod 2 = \mu$ as long as the noise incurred is minimal.

Security Security can be shown via a series of hybrid arguments, similar to the argument in threshold multi-key FHE from GSW. Assume there exists a set of $\{p^j\}$ shares that do not form an authorized set. We wish to show that no adversary can learn any information about the underlying encryption from the partial shares, and no adversary learns information about the secret key shares either.

- **Hybrid 0**: This hybrid is the "real" threshold multi-key FHE scheme, with an unauthorized set of partial shares $\{p_i\}$.
- **Hybrid 1**: This hybrid is the same as hybrid 0, except the partial decryptions are sampled uniformly at random.
- **Hybrid 2**: This hybrid is the same as hybrid above, except the secret key shares that are generated are with respect to 0 rather than sk_i.
- **Hybrid 3**: This hybrid is the same as hybrid above, except the encryptions contain encryptions of 0 rather than m_i.

Hybrid 0 and Hybrid 1 are computationally indistinguishable from each other from the ring learning with errors (LWE) assumption. Specifically, we see that the following two distributions are computationally indistinguishable:

$$(ct, [sk^j \cdot ct]_q + 2e^{(j)}) \approx_c (ct, u^{(j)})$$

Hybrid 1 and Hybrid 2 are statistically indistinguishable from the information theoretic security of the folklore secret sharing scheme, which can be generated from a two-round information-theoretic MPC. Due to having an unauthorized set of partial decryptions, the distribution of p_i shares are statistically indistinguishable from if the secret key shares were shares of 0.

Hybrid 2 and Hybrid 3 are computationally indistinguishable, due to the semantic security of the underlying encryption scheme. Since both the secret key shares and the partial decryptions are simulated, and the underlying FHE scheme is computationally indistinguishable, the adversary cannot distinguish between an encryption of 0 and an encryption of 1.

4.3 Decentralized FHE

In a decentralized fully homomorphic encryption scheme, n parties wish to encrypt data to a FHE scheme, such that a common public key is known to all, but no individual party knows the secret key. In order to decrypt a ciphertext, parties may perform a k-of-n decryption scheme as mentioned above in the threshold fully homomorphic encryption setting. We note that due to a *key homomorphic* property of secret and public keys in Lattice-based encryption schemes, we can realize this construction.

The high level idea is that in the GSW public key

$$\mathbf{A} := \begin{bmatrix} B \\ b \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$$

We see that if two parties generate $b_1 = s_1 B + e_1, b_2 = s_2 B + e_2 \in \mathbb{Z}1 \times m$, encrypting under

$$\mathbf{A}' := \begin{bmatrix} B \\ b_1 + b_2 \end{bmatrix} \in \mathbb{Z}_q^{n \times m}$$

would require the combined secret key $(-(s_1 + s_2), 1)$ to decrypt. However, outputting partial decryptions with respect to secret keys s_1, s_2 independently, and then combining the partial decryptions, would also yield a correct decryption of a ciphertext. This construction eliminates the ciphertext size dependency on the number of parties; however it requires another round of setup before encryption can happen since the public key, in which encryption is with respect to, is the dependent on the sum of all the secret keys.

4.4 Comparison of Constructions

	GSW MK-tFHE	NTRU MK-tFHE	Decentralized GSW FHE
Underlying Assumption	LWE	LWE	RLWE
Semi-Honest Security Round Complexity	2	3	3
Malicious Security Round Complexity	3	3	3
Public Parameters	Yes	No	Yes
Ciphertext Size	$O(1^\lambda, N^2)$	$O(1^\lambda, N \log N)$	$O(1^\lambda)$

Table 4.1: Comparison of multi-key threshold FHE schemes.

4.5 Applications to Multi Party Computation

Multi-party computation (MPC) is a cryptographic protocol in which n mutually distrustful parties compute a joint function of their private inputs without revealing anything more than the output to each other [Yao86, GMW87].

There are many security settings for MPC. The two most commonly considered setting is that of *semi-honest* security, where adversaries do not deviate from the protocol but will learn information when given, and that of *malicious* security where an adversary corrupts a subset of parties who can deviate arbitrarily from the protocol. We also consider the setting of *fairness*, where corrupted parties receive their output only if all honest parties receive output, and *guaranteed output delivery*, where corrupted parties cannot prevent honest parties from receiving their output [Cle86]. In the guaranteed output delivery setting, we may assume that parties may become offline (resulting from a network partition, for example) and are not available for the entire duration of the computation. This may happen either by a malicious adversary in the malicious setting, or non-maliciously in the semi-honest setting.

[GLS15] proved that there cannot be a two-round MPC with honest majority and guaranteed output delivery in the malicious setting. Assuming LWE, one can construct a three-round (round-optimal) MPC with honest majority and guaranteed output delivery in the malicious setting, if we use NIZKs to account for malicious adversaries. Concurrent work include [BJMS18, ACGJ18].

4.5.1 Low Round MPC

While feasibility results for multi party computation have existed since [BOGW88, GMW87], recent research in MPC has focused on low-round constructions.

[BL18] and [GS18] construct two-round MPCs from oblivious transfer. In the honest majority setting, [ABT18] construct information-theoretic two-round MPC in the semi-honest setting. Their construction is based on extending randomized polynomials with degree 3 [IK02], to a *multi party randomized polynomial* with *effective degree* 2.

Through threshold multi-key FHE (under the LWE assumption), we can construct a two round MPC with guaranteed output delivery in the semi-honest setting, and a three-round MPC with guaranteed output delivery in the malicious setting. In the malicious setting we will require the assumption of non-interactive zero knowledge proofs (NIZKs). There are currently no constructions of NIZKs from LWE, though there is recent progress in this direction [RSS18, CLW18].

Two-Round Guaranteed Output Delivery MPC with Semi-Honest adversaries

- **Round 1**: Each party P_i does the following:
  - Run $\mathsf{tmFHE.KeyGen}(1^\lambda)$ to obtain $(pk^{(i)}, sk^{(i)})$
  - Encrypt private input as $c^{(i)} \leftarrow \mathsf{tmFHE.Enc}(pk^{(i)}, x^{(i)})$.
  - Run $(sk_1^{(i)}, \ldots sk_n^{(i)} \leftarrow \mathsf{tmFHE.KeyShare}(sk_i, \mathbb{A})$ for access structure $\mathbb{A}$.
  - For each party j, send $sk_j^{(i)}, c^{(i)}$.
- **Round 2**: Each party P_i does the following:
  - After receiving $c^{(j)}$ and the secret key shares from the other parties, each party computes $c' \leftarrow \mathsf{tmFHE.Eval}(c^{(1)}, \ldots c^{(n)}, f)$
  - For the parties that participate in the threshold decryption process, let $p_1^{(i)}, \ldots p_k^{(i)}$ be the outputs of $\mathsf{tmFHE.PartDec}(sk_1^{(i)}, \ldots sk_k^{(i)}, c')$ for corresponding secret key shares.
  - Output $p_1^{(i)}, \ldots p_k^{(i)}$
- **Computation after Round 2**
  - Upon receiving the broadcasts of all $p_j^{(i)}$ partial decryptions, run $\mathsf{tmFHE.FinDec(B)}$ where $B = \{p_i^{(j)}\}$, to obtain μ if B is within the access structure, or $\perp$ otherwise.

Figure 4-1: A two round MPC with guaranteed output delivery.

4.5.2 Two Round MPC with Guaranteed Output Delivery in Semi-Honest Setting

In this subsection, we detail a construction of a two round MPC with Guaranteed Output Delivery, in which concurrent work was done by [BJMS18]. The input to an MPC consists of n parties, each with their private input x_i, and a function f that the parties wish to compute. The output of the MPC is $f(x_1, \ldots x_n)$ for all n parties. A two round MPC with guaranteed output delivery is presented in figure 4-1. Assuming that public parameters have been set up prior, we obtain a two round MPC; otherwise a third round would be needed prior to the MPC to set up the public parameters.

Correctness Correctness follows from the correctness of the underlying threshold multi-key FHE scheme derived from the GSW FHE. Because we see that $\mathsf{tmFHE.Eval}$ will compute $x = f(x_1, \ldots x_n)$ correctly, and that given a valid access structure $\mathsf{tmFHE.PartDec}$ and $\mathsf{tmFHE.FinDec}$ will decrypt x correctly.

Security From a high level, the security of this MPC also follows from the security of the underlying threshold multi-key FHE scheme derived from GSW FHE. Assume that after the second round of computation, there is no valid set of parties within the access structure who are still online. From the security of the threshold multi-key FHE scheme (and the underlying secret sharing scheme), it is not possible to decrypt c'. In other words, the information-theoretic security of the secret sharing scheme provides the security when the output shares are not within the access structure.

4.5.3 Three Round MPC with Guaranteed Output Delivery in Malicious Setting

In this subsection, we present a three round MPC with guaranteed output delivery with malicious adversaries. Our scheme relies on the LWE assumption to construct threshold multi-key FHE, NIZKs, and a honest majority of parties. The scheme is presented in figure 4-2.

We assume the familiarity with a NIZK scheme, which consists of algorithms $\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify}$. From a high level, $\mathsf{Gen} \rightarrow crs$ outputs a common reference string, $\mathsf{Prove}(crs, y_i, (x_i, r_i)) \rightarrow \pi_i$ takes in input x_i, randomness r_i, and statement y_i, to generate a proof π_i. $\mathsf{Verify}(crs, y_i, \pi_i) \rightarrow \{0, 1\}$ verifies whether the statement y_i is consistent with proof π_i.

Correctness Correctness follows from the correctness of the underlying threshold multi-key FHE scheme derived from the GSW FHE, and the proof is the same as in the semi-honest setting. Because we see that $\mathsf{tmFHE.Eval}$ will compute $x = f(x_1, \ldots x_n)$ correctly, and that given a valid access structure $\mathsf{tmFHE.PartDec}$ and $\mathsf{tmFHE.FinDec}$ will decrypt x correctly. Parties that compute a NIZK proof Π correctly will have their proof verified, and thus other parties will accept their shares and partial decryptions

Three-Round Guaranteed Output Delivery MPC with Malicious adversaries

- **Round 1**: Each party P_i does the following:
  - Run $\mathsf{tmFHE.Setup}(1^\lambda)$ jointly to generate the public parameters
  - Run $\mathsf{NIZK.Gen}(1^\lambda) \rightarrow crs_i$ to generate common reference string.
  - Output the public parameters and crs_i.
- **Round 2**: Each party P_i does the following:
  - Run $\mathsf{tmFHE.KeyGen}(1^\lambda)$ to obtain $(pk^{(i)}, sk^{(i)})$
  - Encrypt private input as $c^{(i)} \leftarrow \mathsf{tmFHE.Enc}(pk^{(i)}, x^{(i)})$.
  - Run $(sk_1^{(i)}, \ldots sk_n^{(i)} \leftarrow \mathsf{tmFHE.KeyShare}(sk_i, \mathbb{A})$ for access structure $\mathbb{A}$.
  - In addition, generate proofs $\mathsf{NIZK.Prove}(CRS, y, (x, r)) \rightarrow \pi_j^{(i)}$ showing that $c^{(i)}$ and secret shares $sk_j^{(i)}$ were generated correctly.
  - For each party j, send $sk_j^{(i)}, c^{(i)}, \pi_j^{(i)}$.
- **Round 3**: Each party P_i does the following:
  - After receiving $c^{(j)}$ and the secret key shares $sk_i^{(j)}$ from the other parties, first from $\pi_i^{(j)}$ verify that the ciphertexts and secret key shares were generated correctly. If $\mathsf{NIZK.Ver}(y_i, \pi_i) \rightarrow 1$, each party computes $c' \leftarrow \mathsf{tmFHE.Eval}(c^{(1)}, \ldots c^{(n)}, f)$
  - For the parties that participate in the threshold decryption process, let $p_1^{(i)}, \ldots p_k^{(i)}$ be the outputs of $\mathsf{tmFHE.PartDec}(sk_1^{(i)}, \ldots sk_k^{(i)}, c')$ for corresponding secret key shares.
  - In addition, generate proofs $\mathsf{NIZK.Prove}(CRS, y, (x, r)) \rightarrow \pi'^{(i)}$ showing that the partial decryptions were generated correctly.
  - Output $p_1^{(i)}, \ldots p_k^{(i)}, \pi'^{(i)}$
- **Computation after Round 3**
  - Upon receiving the broadcasts of all $p_j^{(i)}$ partial decryptions, let $B = \{p_i^{(j)}\}$ such that the corresponding $\mathsf{NIZK.Ver}(p_i^{(j)}, \pi'^{(j)}) = 1$.
  - Run $\mathsf{tmFHE.FinDec(B)}$ to obtain μ if B is within the access structure, or $\perp$ otherwise.

Figure 4-2: A three round MPC with guaranteed output delivery.

to output the correct μ.

Security From a high level, the security of this MPC also follows from the security of the underlying threshold multi-key FHE scheme derived from GSW FHE. Assume that after the second round of computation, there is no valid set of parties within the access structure who are still online. From the security of the threshold multi-key FHE scheme (and the underlying secret sharing scheme), it is not possible to decrypt c'. In other words, the information-theoretic security of the secret sharing scheme provides the security when the output shares are not within the access structure.

The security against malicious adversaries derives from the NIZK proofs. While malicious adversaries may deviate arbitrarily from the protocol, the verify algorithm of the NIZK will not pass if the ciphertext, secret share, or partial decryption was generated inconsistently.

Chapter 5

Predicate & Functional Encryption

A recently introduced paradigm is that of *functional encryption*, which is a generalization of classical definitions of encryption. Functional encryption can be thought of as a superclass of public-key encryption, such that there are restricted secret keys such that a keyholder may learn a specific function of the encrypted data, but nothing else [BSW11, ABB10, ABV^{+}12].

In attribute-based encryption, a ciphertext ct of μ is associated with a *public* set of attributes y. The secret key associated is associated with a predicate C, and decryption succeeds if $C(y) = 1$.

In predicate encryption, a ciphertext ct of μ is associated with a *private* set of attributes x. The secret key associated is associated with a predicate C, and decryption succeeds if $C(x) = 1$. There is the additional security requirement that nothing is learned about (x, μ) if the ciphertext cannot be decrypted.

In a way, we can think of attribute-based encryption (ABE) as a subclass of predicate encryption (PE), while predicate encryption is a subclass of functional encryption (FE).

In this section, we present the definitions for predicate encryption and also show a new construction of a threshold predicate encryption scheme. Our construction builds off of the predicate encryption scheme of [GVW15], which preserves compactness. However, it is not immediately clear how to convert this predicate encryption scheme to a functional encryption, because the predicate encryption is not *attribute-hiding*, in that a decryption of ct leaks information about x.

5.1 Partially-Hiding Predicate Encryption

In order to define the Predicate Encryption scheme in [GVW15], we refer to an intermediate construction known as *partially hiding predicate encryption.* This encryption scheme is a hybrid of attribute-based encryption, and predicate encryption, in that each ciphertext is associated with attributes (x, y), where x is private but y is publicly known. We present a definition and construction below:

Definition 19 (Partially-Hiding Predicate Encryption)**.** *A partially-hiding predicate encryption scheme is a series of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

- $\mathsf{Setup}(1^\lambda) \to (mpk, msk)$*: The setup algorithm takes the security parameter* λ *as input and outputs the public parameter* mpk *and master secret key* msk*.*
- $\mathsf{KeyGen}(msk, C) \to sk_C$*: The key generation algorithm takes in the master secret key* msk *as input, and a predicate* $C \in \mathcal{C}$*, and outputs a secret key* sk_C*.*
- $\mathsf{Enc}(mpk, ((x, y), \mu)) \to ct$*: The encryption algorithm takes in the master public key* mpk*, and a message* μ*, and outputs a ciphertext* ct *with public attributes* y *and public attributes* x*.*
- $\mathsf{Dec}(sk_C, ct) \to \mu$*: The decryption algorithm takes in secret key* sk_C *along with a public predicate* C*, and outputs either the decryption of* ct*,* μ*, or* $\perp$*.*

The intuition behind using a partially-hiding predicate encryption scheme is that the family of circuits allowed to be computed in this scheme consists of all circuits on public attribute y, and only the inner product on private attribute x. If we allow the public attribute to be a FHE encryption of attributes, and the private attribute to be the secret key of the FHE encryption, then because FHE decryption is an inner product operation, we can achieve predicate encryption for all circuits.

More specifically, we define the class of functions for the partially-hiding predicate encryption scheme as $C : \mathbb{Z} \times \{0,1\}^l \to \{0,1\}$ of the form $\hat{C} \circ \mathsf{IP}_\gamma$ where $\hat{C}$ is depth d, and

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

where $\mathsf{IP}_\gamma(x, z) = 1$ if and only if $\langle x, z \rangle = \gamma \mod q$.

The construction of a partially-hiding predicate encryption scheme, as defined in [GVW15] is as follows, where we assume the message space consists of a single bit, and the length of the secret attribute is t, the length of the public attribute is l, and the circuit depth bound is d:

- $\mathsf{PH.Setup}(1^\lambda, 1^t, 1^l, 1^d) \to (mpk, msk)$: The setup algorithm is as follows:
  - First, choose random matrices $A_i \in \mathbb{Z}_q^{n\times m}$ for $i \in [l]$, $B_i \in \mathbb{Z}_q^{n\times m}$ for $i \in [t]$, and $P \in \mathbb{Z}_q^{n\times m}$.
  - Sample a matrix with associated trapdoor $(A, T) \leftarrow \mathsf{TrapGen}(1^m, 1^n, q)$
  - Output the master public key $mpk = (\{A_i\}, \{B_i\}, A, P)$ and the master secret key is $msk = T$.
- $\mathsf{PH.KeyGen}(msk, C) \to sk_C$: The key generation algorithm takes in the master secret key msk as input, and a predicate $C \in \mathcal{C}$, and outputs a secret key $sk_C = R \in \mathbb{Z}_q^{2m\times m}$ as follows:
  - Let $A_{\hat{C}\circ\mathsf{IP}_\gamma} \leftarrow \mathsf{Eval_{pk}}(\{A_i\}, \{B_i\}, \hat{C} \circ \mathsf{IP})$
  - Sample $R \in \mathbb{Z}_q^{2m\times m}$ such that $[A|A_{\hat{C}\circ\mathsf{IP}} + \gamma \cdot G] \cdot R = P \mod q$, where we sample R via

$$R \leftarrow \mathsf{SampleLeft}(A, A_{\hat{C}\circ\mathsf{IP}_\gamma} + \gamma \cdot G, T, P, s)$$

    And output R as sk_C.
- $\mathsf{PH.Enc}(mpk, ((x, y), \mu)) \to ct$: The encryption algorithm is as follows:

- Choose a secret vector $s \leftarrow \chi^n$ and error terms $e, e' \leftarrow \chi^m$
- Let $b = [0, \dots 0, \lceil q/2 \rceil \mu]^T \in \mathbb{Z}_q^m$. Compute

$$\beta_0 = A^T s + e, \beta_1 = P^T s + e' + b$$

- For all $i \in [l]$, let $u_i = (A_i + y[i] \cdot G + R_i^T e)$ where $R_i \leftarrow \{-1, 1\}^{m \times m}$. This is to encode the public attributes.
- For all $i \in [t]$, let $v_i = (B_i + x[i] \cdot G + R_i'^T e)$ where $R_i' \leftarrow \{-1, 1\}^{m \times m}$. This is to encode the private attributes.
- Finally, output ciphertext

$$ct = \Big(\{u_i\}_{i \in [l]}, \{v_i\}_{i \in [t]}, \beta_0, \beta_1 \Big)$$

- $\mathsf{PH.Dec}(sk_C, ct) \rightarrow \mu$: The decryption algorithm is as follows:
  - First, compute

$$u_{\hat{C} \circ \mathsf{IP}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(\{A_i, u_i\}, \{B_i, v_i\}, \hat{C} \circ \mathsf{IP}, y)$$

  where we see $u_{\hat{C} \circ \mathsf{IP}} \approx (A_{\hat{C} \circ \mathsf{IP}} + \rho G)^T s + e$ for some $\rho \in \mathbb{Z}_q$
  - Next, compute

$$\eta = \beta_1 - R^T \cdot \begin{bmatrix} \beta_0 \\ u_{\hat{C} \circ \mathsf{IP}} \end{bmatrix} \in \mathbb{Z}_q^m$$

  If the first $m - 1$ terms in the vector η satisfy $|\eta[i]| < q/4$, then output $\mu = 0$ if $|\eta[m]| < q/4$, and $\mu = 1$ otherwise.

Correctness and security of this scheme is shown in [GVW15].

Refer to the auxiliary algorithms $\mathsf{Eval}_{\mathsf{pk}}$ and $\mathsf{Eval}_{\mathsf{ct}}$ in [GVW15]. From a high level, the auxiliary algorithms have the properties as follows:

- $\mathsf{Eval}_{\mathsf{pk}}$ takes in l matrices $A_1, \dots A_l \in \mathbb{Z}_q^{n \times m}$ and a predicate $C : \{0,1\}^l \to \{0,1\}$, and outputs a matrix $A_C \in \mathbb{Z}_q^{n \times m}$. We have the property that if $(A_1, \dots A_l) = (AR_1 - y[1] \cdot G, \dots, AR_l - y[l] \cdot G)$ where $R_1, \dots R_l$ are small-norm matrices, then

$$A_C = AR_C - C(y) \cdot G$$

  where R_C is a small norm-matrix with a n^{2d} multiplicative blow-up.

- $\mathsf{Eval}_{\mathsf{ct}}$ takes in l matrices $A_1, \dots A_l \in \mathbb{Z}_q^{n \times m}$ and a predicate $C : \{0,1\}^l \to \{0,1\}$ as above. In addition, it takes in public attribute $y \in \{0,1\}^l$ and l vectors $u_1, \dots u_l \in \mathbb{Z}_q^m$ and outputs a vector $u_C \in bbZ_q^m$. If the inputs satisfy $(u_1, \dots u_l) \approx ((A_1 - y[1] \cdot G)^T s, \dots, (A_l - y[l] \cdot G)^T s)$, then

$$u_C \approx (A_C + C(y) \cdot G)^T s$$

  .

5.2 Predicate Encryption

Having defined partially hiding predicate encryption, we have the building blocks to construct a predicate encryption scheme from LWE. From a high level, the predicate encryption scheme is constructed as follows:

- First, we are given a FHE scheme satisfying the definitions above $\mathcal{FHE} =$ (HE.KeyGen, HE.Enc, HE.Eval, HE.Dec) such that l is the size of the initial ciphertext encrypting k bit messages and t is the size of the FHE scret key.

- We are given a partially hiding predicate encryption scheme

$$\mathcal{PHPE} = (\mathsf{PH.Setup},\ \mathsf{PH.Keygen},\ \mathsf{PH.Enc},\ \mathsf{PH.Dec})$$

for the class of predicates $\mathcal{C}_{PHPE}$ boinded by some depth parameter $d' = \text{poly}(d, \lambda, \log q)$

- Combining these two schemes, we obtain a predicate encryption scheme where the public portion of the PHPE attribute can be a FHE encryption of the attribute, and the private portion of the PHPE attribute is the corresponding FHE secret key.

We present a formal definition of predicate encryption:

Definition 20 (Predicate Encryption). *A predicate encryption scheme is a series of PPT algorithms* $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *defined as follows:*

- $\mathsf{Setup}(1^\lambda) \to (mpk, msk)$*: The setup algorithm takes the security parameter* λ *as input and outputs the public parameter* mpk *and master secret key* msk*.*
- $\mathsf{KeyGen}(msk, C) \to sk_C$*: The key generation algorithm takes in the master secret key* msk *as input, and a predicate* $C \in \mathcal{C}$*, and outputs a secret key* sk_C*.*
- $\mathsf{Enc}(mpk, (x, \mu)) \to ct$*: The encryption algorithm takes in the master public key* mpk*, and a message* μ*, and outputs a ciphertext* ct *with attributes* x*.*
- $\mathsf{Dec}(sk_C, ct) \to \mu$*: The decryption algorithm takes in secret key* sk_C *along with a public predicate* C*, and outputs either the decryption of* ct*,* μ*, or* $\perp$*.*

We first define the construction of predicate encryption from [GVW15], which is as follows:

- $\mathsf{PE.Setup}(1^\lambda, 1^k, 1^d)$: The setup algorithm takes in a security parameter λ, attribute length k, predicate depth bound d, and generates a master public and secret key pair (mpk, msk) as follows:
  1. Run the partially-hiding predicate encryption scheme to obtain

$$(ph.mpk, ph.msk) \leftarrow PH.Setup(1^\lambda, 1^t, 1^l, 1^{d'})$$

where for k-bit messages and depth d circuits, t is the length of the FHE secret key, l is the bit-length of the initial FHE ciphertext, and d' is the bound on the FHE evaluation circuit.

2. Let $(mpk = ph.mpk, msk = ph.msk)$

- $\mathsf{PE.KeyGen}(msk, C)$: The key generation algorithm takes as input the master secret key, and a predicate C. It outputs the secret key sk_C associated with the predicate as follows:

  1. Let $\hat{C}(\cdot) = HE.Eval(\cdot, C)$, and let $(\hat{C} \circ \mathsf{IP}_\gamma)$ be the predicates for $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$ for error bound B.
  2. For all $\gamma = \lfloor q/2 \rfloor - B, \ldots, \lfloor q/2 \rfloor + B$, compute

  $$sk_{\hat{C}\cdot\mathsf{IP}_\gamma} \leftarrow \mathsf{PH.Keygen}(ph.msk, \hat{C} \cdot \mathsf{IP}_\gamma)$$

  3. Output the secret key set as $sk_C = (\{sk_{\hat{C}\cdot\mathsf{IP}_\gamma}\}_{\gamma=\lfloor q/2 \rfloor - B,\ldots,\lfloor q/2 \rfloor + B})$

- $\mathsf{PE.Enc}(mpk, (a, \mu))$ The encryption algorithm takes as input the master public key, and a message $\mu \in \{0, 1\}$ with input attribute vector $a \in \{0, 1\}^k$, and proceeds as follows:

  1. Sample a fresh FHE secret key $\mathsf{fhe.sk} \in \mathbb{Z}_q^t$ through FHE key generation.
  2. Encrypt the input attribute vector to obtain $\mathsf{fhe.ct} \leftarrow \mathsf{HE.Enc}(\mathsf{fhe.sk}, a) \in \{0, 1\}^l$.
  3. Compute the PHPE ciphertext $\mathsf{ct}_{\mathsf{fhe.ct}} \leftarrow \mathsf{PE.Enc}(\mathsf{mpk},\ (\mathsf{fhe.sk},\ \mathsf{fhe.ct}), \mu)$
  4. The output ciphertext is $\mathsf{ct} = (\mathsf{ct}_{\mathsf{fhe.ct}}, \mathsf{fhe.ct})$

- $\mathsf{PE.Dec}((sk_C, C), ct)$ The decryption algorithm is as follows. Given the secret key for a function sk_C and the corresponding predicate C, along with the ciphertext ct, if there exists a γ such that $\gamma \in \{\lfloor q/2 \rfloor - B, \lfloor q/2 \rfloor + B\}$ and

  $$\mathsf{PH.Dec}((sk_{\hat{C}\cdot\mathsf{IP}_\gamma}, \hat{C} \cdot \mathsf{IP}_\gamma), (\mathsf{ct}_{\mathsf{fhe.ct}}, \mathsf{fhe.ct})) = \mu \neq \perp$$

then output μ. Otherwise, output γ.

5.3 Threshold Predicate Encryption

As noted above, a predicate encryption scheme has four algorithms, $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$. In this section, we present a way to make the decryption algorithm a distributed procedure. In the following sections, we also present ideas to make setup and keygen a distributed procedure.

In this section, we modify the construction of predicate encryption to attain a threshold predicate encryption scheme. Assuming N parties are participating, with a k-of-N access structure, from any k parties outputting partial decryptions using their secret key shares, the final decryption can recover the underlying message from the ciphertext.

We define threshold predicate encryption as follows:

Definition 21 (Threshold Predicate Encryption)**.** *A predicate encryption scheme is a series of PPT algorithms* $\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec}$ *defined as follows:*

- $\mathsf{Setup}(1^\lambda) \to (mpk, msk)$*: The setup algorithm takes the security parameter* λ *as input and outputs the public parameter* mpk *and master secret key* msk*.*
- $\mathsf{KeyGen}(msk, C) \to sk_C$*: The key generation algorithm takes in the master secret key* msk *as input, and a predicate* $C \in \mathcal{C}$*, and outputs a secret key* sk_C*.*
- $\mathsf{KeyShare}(sk_C, \mathbb{A}) \to (sk_C^{(1)}, \dots sk_C^{(N)})$ *The key sharing algorithm takes in a secret key associated with a predicate, an access structure* $\mathbb{A}$ *and outputs a set of* N *shares of the secret keys to distribute to the* N *parties.*
- $\mathsf{Enc}(mpk, (x, \mu)) \to ct$*: The encryption algorithm takes in the master public key* mpk*, and a message* μ*, and outputs a ciphertext* ct *with attributes* x*.*
- $\mathsf{PartDec}(sk_C^{(i)}, ct) \to p_i$*: The partial decryption algorithm takes in secret key share* $sk_C^{(i)}$ *along with a public predicate* C*, and outputs a partial decryption* p_i

- $\mathsf{FinDec}(\{p_i\}) \to \mu$: *The final algorithm takes in partial decryptions, and outputs either the decryption of ct, μ, if the set of partial decryptions forms a valid set in the access structure, or $\perp$ otherwise.*

We now show a construction of threshold predicate encryption from [GVW15, BGG+17].

- $\mathsf{tPE.Setup}(1^\lambda) \to (mpk, msk)$: The setup algorithm runs $\mathsf{PE.Setup}(1^\lambda)$ to obtain (mpk, msk)

- $\mathsf{tPE.KeyGen}(msk, C) \to sk_C$: The keygen algorithm runs $\mathsf{PE.KeyGen}(msk, C)$ to obtain sk_C.

- $\mathsf{tPE.KeyShare}(sk_C, \mathbb{A}) \to (sk_C^{(1)}, \ldots sk_C^{(N)})$ The key sharing algorithm takes in $sk_C = R \in \mathbb{Z}_q^{2m \times m}$ is a matrix, and an t-of-N threshold access structure. Using the {0,1}-LSSS scheme, we can output shares $(sk_C^{(1)}, \ldots sk_C^{(N)})$ to each of the N parties.

- $\mathsf{tPE.Enc}(mpk, (x, \mu)) \to ct$: The encryption algorithm runs $\mathsf{PE.Enc}(mpk, (x, \mu))$ to obtain ct.

- $\mathsf{tPE.PartDec}(sk_C^{(i)}, ct) \to p_i$: The partial decryption algorithm runs $\mathsf{PE.Dec}(sk_C^{(i)}, ct)$ to obtain partial decryption p_i. We know the ciphertext is in the form $ct = \Big(\{u_i\}_{i\in[l]}, \{v_i\}_{i\in[t]}, \beta_0, \beta_1\Big)$. Where the $\{v_i\}$ consists of encodings of the FHE secret key, while the $\{u_i\}$ consists of the encodings of the FHE encryption of the attributes. First compute

$$u_{\hat{C}\circ\mathsf{IP}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(\{A_i, u_i\}, \{B_i, v_i\}, \hat{C} \circ \mathsf{IP}, y)$$

  Given that $sk_C^{(i)} = R^{(i)}$ is a matrix, sample $e^{(i)} \leftarrow \chi^m$ from error distribution and output

$$R^{(i)} \cdot \begin{bmatrix} \beta_0 \\ u_{\hat{C}\circ\mathsf{IP}} \end{bmatrix} + e^{(i)} \in \mathbb{Z}_q^m$$

- $\mathsf{tPE.FinDec}(\{p_i\}) \to \mu$: Given a set of partial decryptions $\{p_i\}$, we first find a set of partial decryptions that are part of the access structure. For such subset of p_i, define

$$\eta = \beta_1 - \sum_i p_i$$

  If the first $m-1$ terms in the vector η satisfy $|\eta[i]| < q/4$, then output $\mu = 0$ if $|\eta[m]| < q/4$, and $\mu = 1$ otherwise.

We now show correctness and security of the scheme. The correctness of the threshold predicate encryption scheme lies in the noise analysis when decrypting. The security of the scheme can be constructed via a series of hybrid arguments.

Correctness. The correctness of setup, key generation and encryption is preserved from the predicate encryption scheme of [GVW15]. We now only need to show that the partial decryption and final decryption algorithms preserve the correctness of the decryption of the ciphertext. Given a ciphertext ct, and partial decryptions p_i corresponding to the ciphertext ct, in the final decryption we see that

$$\begin{aligned}\eta &= \beta_1 - \sum_i p_i \\ &= \beta_1 - \sum_i \left(R^{(i)} \cdot \begin{bmatrix} \beta_0 \\ u_{\hat{C}\circ \mathsf{IP}} \end{bmatrix} + e^{(i)}\right) \\ &= \beta_1 - R \cdot \begin{bmatrix} \beta_0 \\ u_{\hat{C}\circ \mathsf{IP}} \end{bmatrix} + e'\end{aligned}$$

where $e' = \sum_i e^{(i)}$. We see that as long as e' remains small, then correctness is maintained by the folklore secret sharing scheme, in which the reconstruction coefficient is binary.

Security The security of the scheme can be shown via a series of hybrid arguments. Assume there exists a set of $\{p_i\}$ shares that do not form an authorized set. We wish to show that no adversary can learn any information about the underlying encryption from the partial shares, and no adversary learns information about the secret key shares either.

- **Hybrid 0** The original threshold predicate encryption scheme
- **Hybrid 1** Same as the previous hybrid, except the partial decryptions are sampled uniformly at random.
- **Hybrid 2** Same as the previous hybrid, except the secret key shares are generated with respect to a 0 matrix.
- **Hybrid 3** Same as the previous hybrid, except the encryption is of a message $\mu = 0$
- **Hybrid 4** Same as the previous hybrid, except the encryption consists of attributes $x = [0, \dots 0]$

Hybrid 0 and hybrid 1 are computationally indistinguishable due to the LWE assumption of the partial decryptions. In other words, let $p^{(i)} = R^{(i)} \cdot \begin{bmatrix} \beta_0 \\ u_{\hat{C} \circ \mathsf{IP}} \end{bmatrix}$ in Hybrid 0 be defined in the partial decryption algorithm, and let $u^{(i)} \leftarrow \mathbb{Z}_q^m$ in Hybrid 1. By LWE, we see

$$\left(\begin{bmatrix} \beta_0 \\ u_{\hat{C} \circ \mathsf{IP}} \end{bmatrix}, R^{(i)} \cdot \begin{bmatrix} \beta_0 \\ u_{\hat{C} \circ \mathsf{IP}} \end{bmatrix} + e^{(i)} \right) \approx_c \left(\begin{bmatrix} \beta_0 \\ u_{\hat{C} \circ \mathsf{IP}} \end{bmatrix}, u^{(i)} \right)$$

Hybrid 1 and Hybrid 2 are statistically indistinguishable from the information theoretic security of the folklore secret sharing scheme. Due to having an unauthorized set of partial decryptions, the distribution of p_i shares are statistically indistinguishable from if the secret key shares were shares of 0.

Hybrid 2 and Hybrid 3 are computationally indistinguishable, due to the semantic security of the underlying encryption scheme. Since both the secret key shares and the partial decryptions are simulated, and the underlying FHE scheme is computationally indistinguishable, the adversary cannot distinguish between an encryption of 0 and an encryption of 1.

Similarly, Hybrid 3 and Hybrid 4 are computationally indistinguishable by the semantic security of the predicate encryption scheme.

5.4 Decentralized Predicate Encryption

In this section, we show how to thresholdize the $\mathsf{Setup}, \mathsf{KeyGen}$ protocols such that they can be computed in a decentralized manner. Using [BKP13] to generate threshold constructions of the lattice algorithms $\mathsf{TrapGen}, \mathsf{SampleRight}, \mathsf{SampleD}$ we can distribute the Key Generation and Setup algorithms. This way, instead of having a central party own the master secret key and be in charge of key generation, we can split the master secret key into n shares where any t parties outputting partial secret keys will combine to form a sk_C for some predicate C. One limitation to this construction is that in [BKP13] there is a online-offline split, in that for the "offline" computations to occur, preprocessing online phase must occur after a bounded number of offline computations.

Chapter 6

Future Directions

In this section, we consider directions for future work.

6.1 Threshold FHE

One direction we are currently working on is seeing how threshold FHE can guarantee correctness in a malicious setting, where parties are no longer *semi-honest* and are guaranteed to follow the protocol. Verifiable Secret Sharing can guarantee that the dealer will share the secret correctly, and the shareholders will also be required to behave correctly. Fully homomorphic signatures and zero knowledge proofs with preprocessing (PZK) are also potential techniques that we can use to guarantee that the decryption process can proceed correctly.

Another extension for the threshold multi FHE scheme would be to look at whether it is possible to extend it to the multi hop scheme. One shortcoming of the multi-key FHE scheme provided in [MW16] is that the expand operation can only be done on freshly encrypted ciphertexts. Once an expanded ciphertext has been evaluated on, the encryptions of the noise $\mathcal{U}$ are no longer useful for expanding the cipher text. In other words, if some parties encrypt their plaintext into ciphertexts encrypted under their own key, and then expand the ciphertext to perform evaluations on the expanded ciphertext, it is not possible for another party to then encrypt plaintext under his own secret key to augment the expanded ciphertext in evaluations. The multi-hop multi-key FHE paradigm resolves this shortcoming [BP16, PS16], and it would be interesting to see how to provide a threshold decryption function to this protocol.

6.2 Homomorphic Secret Sharing

Homomorphic secret sharing [BGI15, BCG$^+$18] is a natural extension of secret sharing, in which addition and multiplication operations on the individual shares preserve the underlying structure of the data. In other words, the following computation should be satisfied. If a set of dealers with secrets $s, t, \ldots$ distributes secrets to $s_i, t_i, \ldots$ to party i, an individual party may compute a functions on his own shares $y_i = f(s_i, t_i, \ldots)$ such that the reconstruction phase with the evaluated y_i shares should reconstruct y. While HSS can be constructed from LWE based assumptions, it would be interesting to see whether multi-party HSS can be constructed from weaker assumptions.

We can build a multi-party hss scheme using a 2 round MPC in which we share the universal circuit.

6.3 Conclusion

Finally, I would like to thank my advisor, Prof. Yael Kalai, and Adam Sealfon for their help throughout the year.

Bibliography

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient Lattice (H)IBE in the Standard Model *. In *EUROCRYPT*, 2010.

[ABT18] Benny Applebaum, Zvika Brakerski, and Rotem Tsabary. Perfect Secure Computation in Two Rounds. In *TCC*, 2018.

[ABV$^+$12] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional Encryption for Threshold Functions (or Fuzzy IBE) from Lattices. *Public Key Cryptography*, 2012.

[ACGJ18] Prabhanjan Ananth, Arka Rai Choudhuri, Aarushi Goel, and Abhishek Jain. Round-Optimal Secure Multiparty Computation with Honest Majority. Technical report, 2018.

[BCG$^+$18] Elette Boyle, Geoffroy Couteau, Niv Gilboa, Yuval Ishai, and Michele Orrù. Homomorphic Secret Sharing: Optimizations and Applications. 2018.

[Bei] Amos Beimel. Secret-Sharing Schemes: A Survey.

[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 *. *EUROCRYPT*, 2014.

[BGG$^+$17] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M R Rasmussen, and Amit Sahai. Threshold Cryptosystems From Threshold Fully Homomorphic Encryption *. 2017.

[BGI15] Elette Boyle, Niv Gilboa, and Yuval Ishai. Breaking the circuit size barrier for secure computation under ddh. Cryptology ePrint Archive, Report 2016/585, 2015. `https://eprint.iacr.org/2016/585`.

[BJMS18] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure mpc: Laziness leads to god. *IACR Cryptology ePrint Archive*, 2018:580, 2018.

[BKP13] Rikke Bendlin, Sara Krehbiel, and Chris Peikert. How to Share a Lattice Trapdoor: Threshold Protocols for Signatures and (H)IBE. In Michael Jacobson, Michael Locasto, Payman Mohassel, and Reihaneh Safavi-Naini, editors, *Applied Cryptography and Network Security*, pages 218–236, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BL18] Fabrice Benhamouda and Huijia Lin. k-Round MPC from k-Round OT via Garbled Interactive Circuits. In *EUROCRYPT*, 2018.

[BLMR15] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. Cryptology ePrint Archive, Report 2015/220, 2015. `https://eprint.iacr.org/2015/220`.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, STOC '88, pages 1–10, New York, NY, USA, 1988. ACM.

[BP16] Zvika Brakerski and Renen Perlman. Lattice-Based Fully Dynamic Multi-Key FHE with Short Ciphertexts. 2016.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional Encryption: Definitions and Challenges. 2011.

[BV11] Z. Brakerski and V. Vaikuntanathan. Efficient fully homomorphic encryption from (standard) lwe. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*, pages 97–106, Oct 2011.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *STOC*, 1986.

[CLW18] Ran Canetti, Alex Lombardi, and Daniel Wichs. Non-interactive zero knowledge and correlation intractability from circular-secure fhe. Cryptology ePrint Archive, Report 2018/1248, 2018. `https://eprint.iacr.org/2018/1248`.

[DF89] Yvo Desmedt and Yair Frankel. Threshold Cryptosystems. *CRYPTO*, 1989.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Trans. Inf. Theor.*, 22(6):644–654, September 1976.

[DS16] YarkÄśn Doröz and Berk Sunar. Flattening NTRU for Evaluation Key Free Homomorphic Encryption. Technical report, 2016.

[Gen09] Craig Gentry. *A Fully Homomorphic Encryption Scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[GLS15] S Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-Round MPC with Fairness and Guarantee of Output Delivery. Technical report, 2015.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 218–229, New York, NY, USA, 1987. ACM.

[Gol14] Oded Goldreich. ValiantâĂŹs polynomial-size monotone formula for majority, 2014.

[GS18] Sanjam Garg and Akshayaram Srinivasan. Two-Round Multiparty Secure Computation from Minimal Assumptions *. In *EUROCRYPT*, 2018.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. 2013. `https://eprint.iacr.org/2013/340`.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate Encryption for Circuits from LWE. *CRYPTO*, 2015.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. Ntru: A ring-based public key cryptosystem. In *Lecture Notes in Computer Science*, pages 267–288. Springer-Verlag, 1998.

[IK02] Yuval Ishai and Eyal Kushilevitz. Perfect Constant-Round Secure Computation via Perfect Randomizing Polynomials. In *ICALP*, 2002.

[LPR12] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. Cryptology ePrint Archive, Report 2012/230, 2012. `https://eprint.iacr.org/2012/230`.

[LTV13] Adriana Lopez-Alt, Eran Tromer, and Vinod Vaikuntanathan. On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. Cryptology ePrint Archive, Report 2013/094, 2013. `https://eprint.iacr.org/2013/094`.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for Lattices: Simpler, Tighter, Faster, Smaller. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 700–718, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[MW16] Pratyay Mukherjee and Daniel Wichs. Two Round Multiparty Computation via Multi-Key FHE. 2016.

[PS16] Chris Peikert and Sina Shiehian. Multi-Key FHE from LWE, Revisited. 2016.

[Reg05] Oded Regev. On Lattices, learning with errors, random linear codes, and cryptography. *STOC*, 2005.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[RSS18] Ron D. Rothblum, Adam Sealfon, and Katerina Sotiraki. Towards non-interactive zero-knowledge for np from lwe. Cryptology ePrint Archive, Report 2018/240, 2018. `https://eprint.iacr.org/2018/240`.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[SS11] Damien Stehlé and Ron Steinfeld. Making ntru as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, pages 27–47, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Val84] Leslie G. Valiant. Short monotone formulae for the majority function. In *J. Algorithms*, 1984.

[VNS+03] V Vinod, Arvind Narayanan, K Srinathan, C Pandu Rangan, and Kwangjo Kim. On the Power of Computational Secret Sharing. In Thomas Johansson and Subhamoy Maitra, editors, *Progress in Cryptology - INDOCRYPT 2003*, pages 162–176, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[Weg87] Ingo Wegener. *The Complexity of Boolean Functions.* John Wiley & Sons, Inc., New York, NY, USA, 1987.

[Yao86] A. C. Yao. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, pages 162–167, Oct 1986.