Karina Mochetti de Magalhães

“Lattice-Based Predicate Encryption”

“*Encriptação com Predicados Baseada em Reticulados*”

CAMPINAS
2014

University of Campinas
Institute of Computing

Universidade Estadual de Campinas
Instituto de Computação

Karina Mochetti de Magalhães

“Lattice-Based Predicate Encryption”

Supervisor:
Orientador(a): **Prof. Dr. Ricardo Dahab**

Co-Supervisor:
Co-orientador(a): **Prof. Dr. Michel Abdalla (ENS - France)**

“Encriptação com Predicados Baseada em Reticulados”

PhD Thesis presented to the Post Graduate Program of the Institute of Computing of the University of Campinas to obtain a Doctor degree in Computer Science.

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Computação da Universidade Estadual de Campinas para obtenção do título de Doutora em Ciência da Computação.

This volume corresponds to the final version of the Thesis defended by Karina Mochetti de Magalhães, under the supervision of Prof. Dr. Ricardo Dahab.

Este exemplar corresponde à versão final da Tese defendida por Karina Mochetti de Magalhães, sob orientação de Prof. Dr. Ricardo Dahab.

Supervisor's signature / *Assinatura do Orientador(a)*

CAMPINAS
2014

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca do Instituto de Matemática, Estatística e Computação Científica
Ana Regina Machado - CRB 8/5467

M27L

Magalhães, Karina Mochetti de, 1982-
Lattice-based predicate encryption / Karina Mochetti de Magalhães. – Campinas, SP : [s.n.], 2014.

Orientador: Ricardo Dahab.
Coorientador: Michel Abdalla.
Tese (doutorado) – Universidade Estadual de Campinas, Instituto de Computação.

1. Criptografia. 2. Teoria dos reticulados. I. Dahab, Ricardo,1957-. II. Abdalla, Michel. III. Universidade Estadual de Campinas. Instituto de Computação. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Encriptação com predicados baseada em reticulados
Palavras-chave em inglês:
Cryptography
Lattice theory
Área de concentração: Ciência da Computação
Titulação: Doutora em Ciência da Computação
Banca examinadora:
Ricardo Dahab [Orientador]
Routo Terada
Marcus Vinicius Soledade Poggi de Aragão
Diego de Freitas Aranha
Sueli Irene Rodrigues Costa
Data de defesa: 27-11-2014
Programa de Pós-Graduação: Ciência da Computação

TERMO DE APROVAÇÃO

Defesa de Tese de Doutorado em Ciência da Computação, apresentada pelo(a) Doutorando(a) **Karina Mochetti de Magalhães**, aprovado(a) em **27 de novembro de 2014**, pela Banca examinadora composta pelos Professores Doutores:

Routo Terada

Prof(ª). Dr(ª). Routo Terada
Titular

Prof(ª). Dr(ª). Marcus Vinicius Soledade Poggi de Aragão
Titular

Diego de Freitas Aranha

Prof(ª). Dr(ª). Diego de Freitas Aranha
Titular

Prof(ª). Dr(ª). Sueli Irene Rodrigues Costa
Titular

Ricardo

Prof(ª). Dr(ª). Ricardo Dahab
Presidente(a)

Institute of Computing /*Instituto de Computação*
University of Campinas /*Universidade Estadual de Campinas*

Lattice-Based Predicate Encryption

Karina Mochetti de Magalhães[1]

November 27, 2014

Examiner Board/*Banca Examinadora*:

- Prof. Dr. Ricardo Dahab (Supervisor/*Orientador*)
- Prof. Dr. Diego Aranha
  Institute of Computing - UNICAMP
- Prof. Dr. Sueli Costa
  Institute of Mathematics, Statistics and Scientific Computation - UNICAMP
- Prof. Dr. Routo Terada
  Institute of Mathematics and Statistics - USP
- Prof. Dr. Marcus Poggi de Aragão
  Informatics Department - PUC-Rio
- Dr. Julio López
  Institute of Computing - UNICAMP (Substitute/*Suplente*)
- Prof. Dr. Eduardo Xavier
  Institute of Computing - UNICAMP (Substitute/*Suplente*)
- Prof. Dr. Jeroen van de Graaf
  Department of Computer Science - UFMG (Substitute/*Suplente*)

[1] Financial support:
CAPES scholarship mar/2009 - oct/2009
FAPESP scholarship (process 2009/11071-0) nov/2009 - nov/2011; jul/2012 - oct/2013
CAPES scholarship (process 4594-11-8) dec/2011 - jun/2012

Abstract

In a functional encryption system, an authority holding a master secret key can generate a key that enables the computation of some function on the encrypted data. Then, using the secret key the decryptor can compute the function from the ciphertext. Important examples of functional encryption are Identity-Based Encryption, Attribute-Based Encryption, Inner Product Encryption, Fuzzy Identity-Based Encryption, Hidden Vector Encryption, Certificate-Based Encryption, Public Key Encryption with Keyword Search and Identity-Based Encryption with Wildcards. Predicate encryption schemes are a specialization of functional encryption schemes, in which the function does not give information on the plaintext, but it determines whether the decryption should or should not work properly.

Lattice-Based Cryptography is an important alternative to the main cryptographic systems used today, since they are conjectured to be secure against quantum algorithms. Shor's algorithm is capable of solving the Integer Factorization Problem and the Discrete Logarithm Problem in polynomial time on a quantum computer, breaking the most used and important cryptosystems such as RSA, Diffie-Hellman and Elliptic Curve Cryptography.

In this work we focus on Lattice-Based Predicate Encryption. We study and describe the main lattice-based schemes found in the literature, extending them to hierarchical versions and showing how the use of ideal lattice affects their security proofs. For each scheme, a formal proof of security is detailed, analyses of complexity and variable sizes are givem, as well as the choice of parameters that ensures correct decryption.

Resumo

Em um sistema de criptografia funcional, uma autoridade de posse de uma chave mestra pode gerar uma chave secreta que permite o cálculo de uma função sobre a mensagem nos dados criptografados. Assim, é possível calcular tal função no texto cifrado usando somente a chave secreta. Exemplos importantes de criptografia funcional são Criptografia Baseada em Identidades, Criptografia Baseada em Atributos, Criptografia com Produto Escalar, Criptografia Difusa Baseada em Identidades, Criptografia de Vector Oculto, Criptografia Baseada em Certificados, Criptografia com Pesquisa de Palavra-Chave e Criptografia Baseada em Identidades com Curinga. Esquemas de criptografia com predicados são uma especialização de esquemas de criptografia funcionais, em que a função utilizada não fornece informações sobre a mensagem, mas determina se a decriptação deve ou não funcionar corretamente.

Criptografia baseada em reticulados é uma importante alternativa para os principais sistemas criptográficos utilizados atualmente, uma vez que elas são supostamente seguras contra algoritmos quânticos. O Algoritmo de Shor é capaz de resolver o Problema da Fatoração Inteira e o Problema do Logaritmo Discreto em tempo polinomial em um computador quântico, quebrando os sistemas criptográficos mais usados e importantes atualmente, como o RSA, o Diffie-Hellman e a Criptografia de Curvas Elípticas.

Neste trabalho nos concentramos em esquemas de criptografia com predicados baseados em reticulados. Nós estudamos e descrevemos os principais sistemas baseados em reticulados encontrados na literatura, estendendo-os a versões hierárquicas e mostrando como o uso de um reticulado com estrutura ideal afeta a prova de segurança. Para cada esquema, uma prova formal de segurança é detalhada, as análises de complexidade e do tamanho das variáveis são mostradas e a escolha dos parâmetros garantindo o funcionamento correto da decriptação é dada.

“Le principal fléau de l’humanité n’est pas l’ignorance, mais le refus de savoir.”

Simone de Beauvoir

Contents

List of Tables

List of Algorithms

List of Abbreviation

ABE	Atribute-Based Encryption
AH	Attribute Hiding Secure
CVP	Closest Vector Problem
FBE	Fuzzy Identity-Based Encryption
HFBE	Hierarchical Fuzzy Identity-Based Encryption
HHVE	Hierarchical Hidden Vector Encryption
HIBE	Hierarchical Identity-Based Encryption
HIPE	Hierarchical Inner Product Encryption
HVE	Hidden Vector Encryption
IBE	Identity-Based Encryption
IND-CCA2	Indistinguishable under Adaptive Chosen-Ciphertext Attack
IND-CCA	Indistinguishable under Chosen-Ciphertext Attack
IND-CPA	Indistinguishable under Chosen-Plaintext Attack
IPE	Inner Product Encryption
LWE	Learning With Errors
PH	Payload Hiding Secure
PKG	Public Key Generator
sAT	Selective Attribute Secure
SIVP	Shortest Independent Vector Problem

SVP Shortest Vector Problem

wAH Weakly Attribute Hiding Secure

WIBE Identity-Based Encryption with Wildcards

Chapter 1

Introduction

1.1 Motivation and Main Goals

Public-key cryptographic schemes are based on one-way functions, i.e., functions that are easy to compute, but hard to invert. For encryption and signing, it is also necessary that such functions possess a trapdoor, i.e., a shortcut that is kept secret, and that makes it possible for the secret holder to easily invert the function. Such functions include large integer factorization, taking discrete logarithms in certain cyclic groups, or finding shortest vectors in certain classes of lattices.

The first public-key cryptographic scheme was proposed by Diffie and Hellman [25] in 1976 and it was based on the Discrete Logarithm Problem. Two years later, Rivest, Shamir and Adleman published a public-key cryptographic scheme for encryption and signing, known as RSA [65], based on the perceived hardness of the Integer Factorization Problem. Both these problems are hard in general although there are no mathematical proofs of this fact. However, quantum computers are capable of solving both these problems in polynomial time using Shor's algorithm [71]. Moreover, Elliptic Curve Cryptography (ECC) is also known to be vulnerable to a modified Shor's algorithm for solving the discrete logarithm for any choice of parameters. Thus, the construction of quantum computers will undermine the security of the main cryptographic systems used today making the study of alternatives essential. Moreover, it is conjectured that there is no quantum algorithms for solving NP-hard problems based on lattice theory and codes in polynomial time. Therefore, systems such as McEliece [48] (based on Goppa codes) and NTRU [37] (based on lattices), along with other systems based on quadratic equations over finite fields and cryptographic hash functions, have been actively studied in the area that has been called post-quantum cryptography [12].

Functional encryption provides a system administrator with a fine-grained control over the decryption capabilities of its users. In particular, each secret key in the system will allow its holder to compute a particular function of the plaintext. Such ability makes functional encryption schemes appealing in many emerging applications such as cloud services where the notion of public-key encryption reveals its inadequacy. More specifically, in a functional encryption system, an authority holding a master secret key can generate a key that enables the computation of some function on the encrypted data. Then, using the secret key the decryptor can compute the function from the ciphertext. Important examples of functional encryption are Identity-Based Encryption, Attribute-Based Encryption, Inner Product Encryption, Fuzzy Identity-Based Encryption, Hidden Vector Encryption, Certificate-Based Encryption, Public Key Encryption with Keyword Search and Identity-Based Encryption with Wildcards. Predicate encryption schemes are a specialization of functional encryption schemes, in which the function does not give information on the plaintext, but it determines whether the decryption should or should not work properly.

More to our interests, there are few known lattice-based predicate encryption schemes, all of which are less efficient than classical factoring or discrete log-based schemes. However, since they have important, and sometimes unique, specific applications, improving them should enhance their deployability. It is important to note that post-quantum systems do not depend on the existence of quantum computers, and can be implemented in the classical model. This fact has a crucial role in maintaining a classical system resistant to quantum machines, since the coexistence of the two paradigms is inevitable once the quantum computer becomes feasible in practice (no new technology replaces the current paradigm immediately).

1.2 Notation

In this section we present the basic notation used in our work.

- We use capital letters (e.g. A) to denote matrices, bold lowercase letters (e.g. $\boldsymbol{v}$) to denote vectors and simple lowercase letters to denote numbers (e.g. x).

- The notation $A^\top$ denotes the transpose of the matrix A, i.e., the rows of matrix A are the columns of matrix $A^\top$.

- If A_1 is an $n \times m$ matrix and A_2 is an $n \times m'$ matrix, then we let $[A_1|A_2]$ denote the $n \times (m + m')$ matrix formed by concatenating A_1 and A_2. If $\boldsymbol{v}_1$ is a m-length vector and $\boldsymbol{v}_2$ is a m'-length vector, then we let $\left[\begin{smallmatrix}\boldsymbol{v}_1\\ \boldsymbol{v}_2\end{smallmatrix}\right]$ denote the $(m + m')$-length vector formed by concatenating $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$.

- The notation $[c, d]$, for $c < d$ denotes the set of positive integers $\{c, c+1, \ldots, d-1, d\}$.

- The notation $\lceil c \rfloor$ denotes rounding c to the nearest integer.

- An n-degree polynomial $f(x) = a_0 + a_1x^1 + \dots a_nx^n$ can be represented by an $(n+1)$-length vector $\boldsymbol{v}$, in which each position is a coefficient of $f(x)$, i.e., $\boldsymbol{v} = (a_0, a_1, \dots, a_n)$.

- The notation $\mathcal{I}_n$ denotes the identity matrix $n \times n$, i.e., a square matrix with ones on the main diagonal and zeros elsewhere. The notation $\mathcal{O}$ denotes the null matrix, i.e., a matrix with all its elements being zero.

- For the set $\mathbb{S}$ and the operation $\star$, the notation $(\mathbb{S}, \star)$ denotes an abelian group, i.e., the operation $\star$ on $\mathbb{S}$ that is associative, commutative, closed, has an identity element and every element has an inverse.

- Although most cryptography books and papers use the modular congruence notation $a \equiv b \pmod{q}$ we will use the notation that is more common to the lattice-based cryptography field: $a = b \mod q$.

- If $\mathbb{S}$ is a set and $s \in \mathbb{S}$, then $s \overset{\$}{\leftarrow} \mathbb{S}$ denotes a random sample s chosen from set $\mathbb{S}$.

- The big O notation $f(x) = O(g(x))$ denotes that:

  $$\exists k > 0, \exists n_0, \forall n > n_0 : |f(n)| \leq |g(n) \cdot k|.$$

  The soft-O notation $f(x) = \widetilde{O}(g(x))$ denotes $f(x) = O(g(x) \log^k g(x))$, for some k. The small omega notation $f(x) = \omega(g(x))$ denotes that:

  $$\forall k > 0, \exists n_0, \forall n > n_0 : |f(n)| \geq k \cdot |g(n)|.$$

- The notation $\|\boldsymbol{v}\|$ denotes the Euclidean Norm of vector $\boldsymbol{v}$, i.e., $\|\boldsymbol{v}\| = \sqrt{v_1^2 + \dots + v_n^2}$.

- The notation $\|\boldsymbol{v} - \boldsymbol{w}\|$ denotes the Euclidean Distance between the vectors $\boldsymbol{v}$ and $\boldsymbol{w}$.

- The notation $\langle \boldsymbol{v}, \boldsymbol{w} \rangle$ denotes the inner product of two vectors, i.e., $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = v_1w_1 + v_2w_2 + \dots + v_nw_n$.

- We say that a function $f(n)$ is *polynomial* if $f(x) = O(n^c)$ for some $c > 0$, and we use $\text{poly}(n)$ to denote a polynomial function of n.

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

- The notation $\Pr[X = a]$ denotes the probability of an event a in distribution X and the notation $\Delta(X, Y) = \frac{1}{2}\sum_a |\Pr[X = a] - \Pr[Y = a]|$ denotes the statistical difference between two distributions. We say two distributions are *statistically close* if $\Delta(X, Y)$ is negligible.

- The notation $\perp$, when used as a returned value for an algorithm means that the vale is empty, i.e., that no answer is returned in that case from this algorithm.

- For two rings $\hat{\boldsymbol{x}} = (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_k)$ and $\hat{\boldsymbol{y}} = (\boldsymbol{y}_1, \cdots, \boldsymbol{y}_k)$, the notation $\hat{\boldsymbol{x}} \otimes \hat{\boldsymbol{y}}$ denotes the multiplication of the two rings, given by the polynomial $\sum \boldsymbol{x}_i \boldsymbol{y}_i$.

- For a ring $\hat{\boldsymbol{r}} = (\boldsymbol{r}_1, \cdots, \boldsymbol{r}_k)$ and a polynomial $\boldsymbol{v}$, the notation $\hat{\boldsymbol{r}} \cdot \boldsymbol{v}$ denotes the multiplication of a polynomial and a ring, given by the ring $(\boldsymbol{v} \cdot \boldsymbol{r}_1, \cdots, \boldsymbol{v} \cdot \boldsymbol{r}_k)$.

1.3 Our Contributions

In this work we focus on Lattice-Based Predicate Encryption, describing and extending several types of schemes. We show several schemes along with their formal security proofs, using notions and standard techniques found in the literature. We highlight the following contributions:

- a hierarchical version of a known Inner Product Encryption (IPE) scheme, described in Section 5.2 and in a paper published on LatinCrypt 2012 [1];

- a hierarchical version of a known Hidden Vector Encryption (HVE) scheme, described in Section 6.2 and in a paper published on SBSeg 2014 [53];

- a hierarchical version of a known Fuzzy Identity-Based Encryption (FBE) scheme, described in Section 7.2;

- an ideal version of a known Identity-Based Encryption (IBE) scheme and its hierarchical version, described in Sections 8.1 and 8.2 and in a technical report published at IC/UNICAMP [54];

- an analysis of how the ideal and hierarchical features affect each scheme's security proof, complexity and key size, in Chapter 9.

1.4 Outline

Chapter 2 gives an overview of Lattice Theory and some lattice-based cryptosystems. Chapter 3 examines the literature for related work on Predicate Encryption and details some of the main known schemes. Chapter 4 contains the description of a lattice-based IBE scheme and its hierachical version. Chapter 5 presents the description of a lattice-based IPE scheme and its hierachical version. Chapter 6 consists of the description of a lattice-based HVE scheme and its hierachical version. Chapter 7 gives the description of a lattice-based FBE scheme and its hierachical version. Chapter 8 uses the schemes described in Chapter 4 to construct an ideal lattice-based scheme of the general and hierachical version of the IBE scheme. Chapter 9 gives a detailed analysis of all schemes defined in this work, comparing their complexity and key size and detailing how the main features of each scheme affect their security proof. Chapter 10 draws the conclusions of our work and gives a perspective for future research.

Chapter 2

Lattice-Based Cryptography

In this chapter we give basic definitions on lattices and their use in cryptographic schemes. In Section 2.1 we give basic concepts on lattices. In Section 2.2 we define Gaussian distributions over lattices and how to use them to generate trapdoor functions. In Section 2.3 we describe the Learning With Errors Problem and its relation to the Hard Lattice Problems. Finally, in Sections 2.4 and 2.5 two important cryptographic schemes based on lattices are described, GGH and NTRU.

2.1 Lattices

A *lattice* Λ is a set of points in $\mathbb{R}^n$ generated by integral linear combinations of vectors from a basis $B = [\boldsymbol{b}_1|\boldsymbol{b}_2|\cdots|\boldsymbol{b}_m]$, with $n, m \in \mathbb{Z}$, where the $\boldsymbol{b}_i$ are linearly independent vectors. Therefore,

$$\Lambda(B) = \{B\boldsymbol{x} : \boldsymbol{x} \in \mathbb{Z}^m\}.$$

The vector space generated by B has a similar definition:

$$\mathrm{span}(B) = \{B\boldsymbol{x} : \boldsymbol{x} \in \mathbb{R}^m\}.$$

If each $\boldsymbol{b}_i \in \mathbb{Z}^n$, then $\Lambda(B)$ is an *integer lattice*.

For a basis $B \in \mathbb{R}^{n\times m}$, the value n is called the *dimension* of the lattice and the value m is called the *rank* of the lattice. If $n = m$, then Λ is called a *full rank lattice*.

A lattice can have several bases; for two bases B and B', $\Lambda(B) = \Lambda(B')$ if and only if $B' = BU$, where U is a unimodular matrix, i.e., a square matrix with $\det(U) = \pm 1$.

Let B be a lattice basis; $G = B^\top B$ is called the *Gram matrix* and the *lattice determinant* is given by the square root of the determinant of G.

The following operations can be performed on a lattice basis, resulting in a new basis for the same lattice:

1. swap two columns: $\boldsymbol{b}_i \leftrightarrow \boldsymbol{b}_j$
2. multiply a column by -1: $\boldsymbol{b}_i \leftarrow -\boldsymbol{b}_i$
3. multiply a column by an integer α and add it to another column: $\boldsymbol{b}_j \leftarrow \boldsymbol{b}_j + \alpha \boldsymbol{b}_i$ $(i \neq j)$

Figure 2.1 shows some points of a 2-dimensional lattice for the following bases:

$$B = \begin{pmatrix} 1 & 1 \\ 2 & -1 \end{pmatrix} \qquad B' = \begin{pmatrix} 2 & 3 \\ 1 & 3 \end{pmatrix}$$

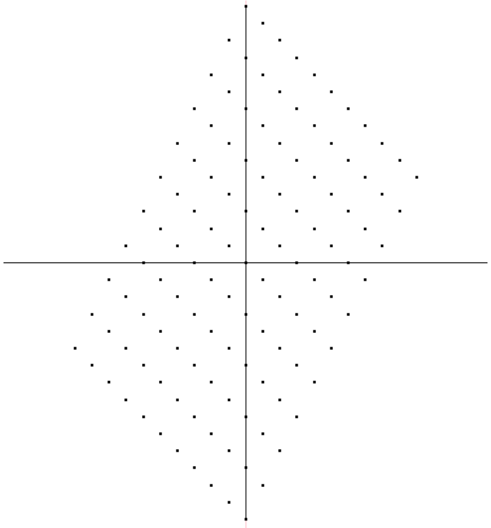

Figure 2.1: Some points of the lattice defined by basis B and B'.

A matrix B is in *Hermite Normal Form* (HNF) if B has the form $B = [H|\mathcal{O}]$, where H is a non-negative lower-triangular matrix (i.e., $h_{ij} = 0$ for $i < j$ and $h_{ij} \geq 0$ for $i \geq j$) and the maximum of each row is unique and located on the main diagonal.

A lattice is *modular* or *q-ary* if it is invariant under shifts by some fixed modulus $q \in \mathbb{Z}$ in each of the coordinates:

$$\begin{aligned} \Lambda_q^{\perp}(B) &= \{\boldsymbol{e} \in \mathbb{Z}^m : B \cdot \boldsymbol{e} = 0 \mod q\} \\ \Lambda_q^{\boldsymbol{u}}(B) &= \{\boldsymbol{e} \in \mathbb{Z}^m : B \cdot \boldsymbol{e} = \boldsymbol{u} \mod q\} \\ \Lambda_q(B) &= \{\boldsymbol{e} \in \mathbb{Z}^m : \exists\ \boldsymbol{s} \in \mathbb{Z}_q^m \text{ with } B^{\top} \cdot \boldsymbol{s} = \boldsymbol{e} \mod q\}. \end{aligned}$$

The lattice $\Lambda_q^{\boldsymbol{u}}(B)$ is a coset of $\Lambda_q^{\perp}(B)$; namely, $\Lambda_q^{\boldsymbol{u}}(B) = \Lambda_q^{\perp}(B) + \boldsymbol{t}$ for any $\boldsymbol{t}$ such that $B \cdot \boldsymbol{t} = \boldsymbol{u} \mod q$.

The *length of a lattice basis* B is given by the maximum length of all vectors in the basis:

$$\|B\| = \max \|\boldsymbol{b}\|, (\boldsymbol{b} \in B).$$

The *successive minima* of a lattice are defined as: for $i \in [1, m]$, $\lambda_i(\Lambda)$ is the radius of the smaller sphere with its center on the origin that contains i linearly independent vectors in Λ. Note that $\lambda_1(\Lambda)$ gives the length of the smallest vector in Λ.

2.1.1 Basis Reduction

As already stated, a lattice can have several bases, and it may be important to find a basis with short orthogonal vectors. A short basis allows some lattice information, such as the successive minima, to be found easily. All reduction algorithms known so far have a running time at least exponential in the dimension of the lattice for the optimal or exact solution or have a polynomial running time for an approximate solution. A short basis can be used as a trapdoor in a cryptographic scheme (see Section 2.2 for more details).

The Gram-Schmidt orthogonalization is a process for obtaining an orthogonal basis $\widetilde{B}$ of a vector space given by basis B. It is defined by the following iterative formula:

$$\widetilde{\boldsymbol{b}}_i = \boldsymbol{b}_i - \sum_{j=1}^{i-1} \frac{\langle \widetilde{\boldsymbol{b}}_j, \boldsymbol{b}_i \rangle}{\langle \widetilde{\boldsymbol{b}}_j, \widetilde{\boldsymbol{b}}_j \rangle} \widetilde{\boldsymbol{b}}_j, \text{ for } i \in [2, m].$$

Although this process always returns an orthogonal basis for a vector space, this is not true for lattices. Therefore, the lattice given by $\Lambda(\widetilde{B})$ is not always the same as given by $\Lambda(B)$, although $\operatorname{span}(B) = \operatorname{span}(\widetilde{B})$.

The Gram-Schmidt orthogonalization is an important step of the Lenstra-Lenstra-Lovász [43] reduction algorithm, or simply LLL. The LLL algorithm is a polynomial time algorithm that finds a δ-LLL reduced basis.

Definition 2.1. *A basis $B = [\boldsymbol{b}_1 | \cdots | \boldsymbol{b}_m]$ is a δ-LLL reduced basis, for $\delta \in \{0, \ldots, 1\}$, if for all $i > j$,*

$$\left| \frac{\langle \widetilde{\boldsymbol{b}}_j, \boldsymbol{b}_i \rangle}{\langle \widetilde{\boldsymbol{b}}_j, \widetilde{\boldsymbol{b}}_j \rangle} \right| \leq \frac{1}{2} \quad \textit{and} \quad \delta \|\pi_i(\boldsymbol{b}_i)\|^2 \leq \|\pi_i(\boldsymbol{b}_{i+1})\|^2$$

$$\textit{where } \pi_i(\boldsymbol{x}) = \sum_{j=i}^{m} \frac{\langle \widetilde{\boldsymbol{b}}_j, \boldsymbol{x} \rangle}{\langle \widetilde{\boldsymbol{b}}_j, \widetilde{\boldsymbol{b}}_j \rangle}.$$

The δ-LLL reduced basis is not the exact solution, it is an approximation solution by δ. The LLL algorithm has many applications as factoring polynomials or finding integer relations. In lattices, it can be used to solve an important hard lattice problem, the γ-SVP, for $\gamma = (2/\sqrt{3})^m$. See Section 2.1.3 for more details on Hard Lattice Problems and Section 2.1.4 for more details on the LLL Algorithm.

2.1.2 Ideal Lattices

A set $\mathbb{S}$ is a *ring* if it has two operations $+$ and $\star$, in which $(\mathbb{S}, +)$ and $(\mathbb{S}, \star)$ are abelian groups. We denote by $\mathbb{S}[x]$ the set of polynomials with coefficients in $\mathbb{S}$. Clearly, $\mathbb{S}[x]$ is a ring. For a polynomial $f(x)$ on $\mathbb{S}[x]$, we denote $\mathbb{S}[x]/\langle f(x)\rangle$ the ring in which the two operations are taken modulo $f(x)$. A subset I of a ring $\mathcal{R} = \mathbb{Z}[x]/\langle f(x)\rangle$ is an *ideal* if:

1. $(I, +)$ is a subset of $(\mathcal{R}, +)$
2. $\forall x \in I$ and $\forall r \in \mathcal{R}$, $x \cdot r \in I$
3. $\forall x \in I$ and $\forall r \in \mathcal{R}$, $r \cdot x \in I$

Let I be an ideal of the ring $\mathcal{R} = \mathbb{Z}[x]/\langle f(x)\rangle$; then I is a sublattice of $\mathbb{Z}^n$, called *ideal lattice*.

We can represent a polynomial $g(x) \in \mathcal{R}$ as a vector $\boldsymbol{g} \in \mathbb{Z}^n$ where, for each $i \in [0, n-1]$, g_i is the coefficient of $g(x)$ for x^i. We can define the basis of the ideal lattice $\Lambda_q^{\perp}(B)$, where each row i of B is given by the coefficients of $x^i g(x) \mod f(x)$ for $i \in [0, n-1]$. The function that, given a vector $\boldsymbol{g}$, creates the matrix B that is a basis of an ideal lattice is called rot:

$$B = \mathsf{rot}_f(\boldsymbol{g}) \in \mathbb{Z}_q^{n\times n},$$

Note that if $f(x) = x^n + 1$, then the matrix $A = \mathsf{rot}_f(\boldsymbol{g}) \in \mathbb{Z}_q^{n\times n}$ is an anti-circulant matrix, as shown in Figure C.3, and if $f(x) = x^n - 1$, we have that the matrix $A = \mathsf{rot}_f(\boldsymbol{g}) \in \mathbb{Z}_q^{n\times n}$ is a circulant matrix, as shown in Figure C.2. A *circulant matrix* is a matrix where each row is rotated one element to the right relative to the preceding row. An *anti-circulant matrix* is a circulant matrix in which after the rotation, the first element of the row vector has its sign changed. See Appendix C for more details on Toeplitz and circulant matrices. The lattices generated by these matrices are called *cyclic lattices* and they are a special class of ideal lattices.

$$A = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \\ -a_n & a_0 & \cdots & a_{n-2} & a_{n-1} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ -a_2 & -a_3 & \cdots & a_0 & a_1 \\ -a_1 & -a_2 & \cdots & -a_n & a_0 \end{pmatrix}$$

Figure 2.2: An anti-circulant matrix A.

$$A = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \\ a_n & a_0 & \cdots & a_{n-2} & a_{n-1} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ a_2 & a_3 & \cdots & a_0 & a_1 \\ a_1 & a_2 & \cdots & -a_n & a_0 \end{pmatrix}$$

Figure 2.3: A circulant matrix A.

For a ring $\mathcal{R} = \mathbb{Z}[x]/\langle f(x) \rangle$, we have that $\hat{\boldsymbol{g}} \in \mathcal{R}^k$ is the vector resulting of the concatenation of k polynomials in $\mathcal{R}$. We define $A = \mathsf{Rot}_f(\hat{\boldsymbol{g}}) \in \mathbb{Z}_q^{n \times nk}$ as the concatenation of every matrix $A_i = \mathsf{rot}_f(\boldsymbol{g}_i) \in \mathbb{Z}_q^{n \times n}$, where $\boldsymbol{g}_i$ is a polynomial in $\hat{\boldsymbol{g}}$ for $i \in [0, k-1]$. Figure 2.4 shows the construction of matrix $A = \mathsf{Rot}_f(\hat{\boldsymbol{g}}) \in \mathbb{Z}_q^{n \times nk}$.

$$A = \left(\begin{array}{c|c|c|c} \boldsymbol{g}_0 x^0 \mod f(x) & \boldsymbol{g}_1 x^0 \mod f(x) & & \boldsymbol{g}_{k-1} x^0 \mod f(x) \\ \boldsymbol{g}_0 x^1 \mod f(x) & \boldsymbol{g}_1 x^1 \mod f(x) & & \boldsymbol{g}_{k-1} x^1 \mod f(x) \\ \vdots & \vdots & \cdots & \vdots \\ \boldsymbol{g}_0 x^{n-2} \mod f(x) & \boldsymbol{g}_1 x^{n-2} \mod f(x) & & \boldsymbol{g}_{k-1} x^{n-2} \mod f(x) \\ \boldsymbol{g}_0 x^{n-1} \mod f(x) & \boldsymbol{g}_1 x^{n-1} \mod f(x) & & \boldsymbol{g}_{k-1} x^{n-1} \mod f(x) \end{array}\right)$$

Figure 2.4: Construction of matrix $A = \mathsf{Rot}_f(\hat{\boldsymbol{g}}) \in \mathbb{Z}_q^{n \times nk}$.

There are two main advantages of using ideal lattices: more efficient multiplication and smaller parameters to define a lattice. The basis of an ideal lattice consists of the concatenation of k Toeplitz $n \times n$ matrices and the multiplication of a Toeplitz matrix by a vector can be done in a more efficient way [59]. Although these basis are $n \times kn$ matrices, they can be described as a polynomial vector of size kn, decreasing the parameters size.

2.1.3 Hard Lattice Problems

Lattices can be used in the construction of cryptographic systems, as shown by Ajtai [8]. The security of these cryptosystems is usually linked to a reduction of hard problems in lattices, i.e., an algorithm can be considered secure if breaking it implies in efficiently solving one of the following lattice problems: Shortest Vector Problem (SVP), Closest Vector Problem (CVP) and Shortest Independent Vector Problem (SIVP).

The Shortest Vector Problem (SVP) consists of finding a non-zero vector in a lattice that has the smallest length. The Approximation Shortest Vector Problem (γ-SVP) is the same problem, but instead of finding the exact shortest vector, it is enough to find a vector whose length is sufficiently close to the shortest one.

Definition 2.2. ***The* SVP *Problem***: *Given a lattice $\Lambda(B)$, find the vector $\boldsymbol{v} \in \Lambda(B)$ such that $\|\boldsymbol{v}\| \neq 0$ and $\|\boldsymbol{v}\| = \lambda_1(\Lambda)$.*

Definition 2.3. ***The* γ-SVP *Problem***: *Given a lattice $\Lambda(B)$ and an approximation factor $\gamma \geq 1$, find the vector $\boldsymbol{v} \in \Lambda(B)$ such that $\|\boldsymbol{v}\| \neq 0$ and $\|\boldsymbol{v}\| \leq \gamma\lambda_1(\Lambda)$.*

The Gap Shortest Vector Problem (GapSVP) is a decision version of the Shortest Vector Problem, in which one should decide whether or not the shortest vector is inside a sphere of radius r and center on the origin. The approximation version defines that the vector must be inside within an approximation factor.

Definition 2.4. ***The* GapSVP *Problem***: *Given a lattice $\Lambda(B)$ and a radius $r > 0$, return YES if $\lambda_1(\Lambda) \leq r$ or NO if $\lambda_1(\Lambda) > r$.*

Definition 2.5. ***The* γ-GapSVP *Problem***: *Given a lattice $\Lambda(B)$, an approximation factor $\gamma \geq 1$ and a radius $r > 0$, return YES if $\lambda_1(\Lambda) \leq r$ or NO if $\lambda_1(\Lambda) > \gamma r$. In any other cases, return arbitrarily.*

The exact version of SVP was conjectured to be NP-hard in 1981 [74], but it was only proved in 2001 [50], where it was also proved that the approximation version γ-SVP with $\gamma = \sqrt{2}$ is NP-hard. It is currently known that γ-SVP is NP-hard for factors $\gamma = 2^{(\log n)^{1/2-\epsilon}}$, where $\epsilon > 0$ is a small constant [42].

The Closest Vector Problem (CVP) consists in finding a nonzero vector in the lattice that is closest to a given vector. The Approximation Closest Vector Problem (γ-CVP) is the approximation version of the problem, but instead of finding the exact closest vector, it is enough to find one close to an approximation value.

Definition 2.6. ***The* CVP *Problem***: *Given a lattice $\Lambda(B)$ and a vector $\boldsymbol{t} \in \mathbb{R}^n$, find the vector $\boldsymbol{v} \in \Lambda(B)$ such that $\|\boldsymbol{v} - \boldsymbol{t}\|$ is minimum.*

Definition 2.7. ***The γ-CVP Problem****: Given a lattice $\Lambda(B)$, a vector $\boldsymbol{t} \in \mathbb{R}^n$ and an approximation factor $\gamma \geq 1$, find the vector $\boldsymbol{v} \in \Lambda(B)$ such that $\|\boldsymbol{v} - \boldsymbol{t}\| \leq \gamma d$, where d is the smallest distance between $\boldsymbol{t}$ and any other vector in $\Lambda(B)$.*

The Gap Closest Vector Problem (GapCVP) is a decision version of the Closest Vector Problem, in which one should decide if, given a vector $\boldsymbol{t}$, there is a closest vector in the lattice within a distance d of $\boldsymbol{t}$ or if every vector in the lattice is at a distance greater than d from $\boldsymbol{t}$. The approximation version defines that every vector in the lattice must be at a distance greater than d within an approximation factor.

Definition 2.8. ***The GapCVP Problem****: Given a lattice $\Lambda(B)$, a vector $\boldsymbol{t} \in \mathbb{R}^n$ and a distance $d > 0$, return YES if $\exists \boldsymbol{v} : \|\boldsymbol{v} - \boldsymbol{t}\| \leq d$ or NO if $\forall \boldsymbol{v} : \|\boldsymbol{v} - \boldsymbol{t}\| > d$.*

Definition 2.9. ***The γ-GapCVP Problem****: Given a lattice $\Lambda(B)$, a vector $\boldsymbol{t} \in \mathbb{R}^n$, an approximation factor $\gamma \geq 1$ and a distance $d > 0$, return YES if $\exists \boldsymbol{v} : \|\boldsymbol{v} - \boldsymbol{t}\| \leq d$ or NO if $\forall \boldsymbol{v} : \|\boldsymbol{v} - \boldsymbol{t}\| > \gamma d$. In any other cases, return arbitrarily.*

The exact version of CVP was proved to be NP-hard in 1981 [74]. It is also known that γ-GapCVP is hard for a sub-polynomial factor $\gamma = n^{O(1/\log\log n)}$ [26].

It is possible to reduce SVP to CVP, therefore, an oracle that solves CVP can be used to solve SVP in the same dimension and within the same approximation factor [32]. The reduction goes as follows:

Given a lattice basis $B = [\boldsymbol{b}_1|\cdots|\boldsymbol{b}_m]$, define bases $B^i = [\boldsymbol{b}_1|\cdots|2\boldsymbol{b}_i|\cdots|\boldsymbol{b}_m]$. Use a CVP-oracle to find the closest vector to $\boldsymbol{b}_i$ in B^i for all $i \in [1, m]$. The oracle will return vector $\boldsymbol{v}_i$. The shortest vector of the lattice given by basis B is the shortest of $\{\boldsymbol{v}_1 - \boldsymbol{b}_1, ..., \boldsymbol{v}_m - \boldsymbol{b}_m\}$.

The Shortest Independent Vector Problem (SIVP) consists in finding m linearly independent vectors, with the maximum length of these vectors restricted by $\lambda_m(\Lambda)$, where m is the rank of the lattice. The resulting matrix $S = (\boldsymbol{s}_1, \cdots, \boldsymbol{s}_m)$ does not need to form a basis of Λ nor does it need to be the successive minima, only the maximum length vector is bounded by the successive minima. The approximation version restricts the largest vector of S by an approximation factor of $\lambda_m(\Lambda)$.

Definition 2.10. ***The SIVP Problem****: Given a lattice $\Lambda(B)$ with rank m, find m linearly independent vectors $S = [\boldsymbol{s}_1|\cdots|\boldsymbol{s}_m]$, such that $\max \|\boldsymbol{s}_i\| = \lambda_m(\Lambda)$.*

Definition 2.11. ***The γ-SIVP Problem****: Given a lattice $\Lambda(B)$ with rank m, find m linearly independent vectors $S = [\boldsymbol{s}_1|\cdots|\boldsymbol{s}_m]$, such that $\max \|\boldsymbol{s}_i\| \leq \gamma\lambda_m(\Lambda)$.*

It is known that the SIVP Problem is NP-hard, as well as its approximation version, γ-SIVP, for $\gamma = n^{1/\log\log n}$ [13].

2.1.4 The LLL and Babai's Algorithm

The two most important algorithms that present approximation solutions for hard lattice problems are the Lenstra-Lenstra-Lovász Algorithm, or LLL, and Babai's Nearest Plane Algorithm. The LLL Algorithm [43] was developed in 1982 and solves the γ-SVP Problem for $\gamma = (\frac{2}{\sqrt{3}})^m$. The Babai's Nearest Plane Algorithm [10] was developed in 1986 and it solves the γ-CVP Problem for $\gamma = 2(\frac{2}{\sqrt{3}})^m$.

The main goal of the LLL Algorithm is to find a new basis $B' = [\boldsymbol{b}'_1|\cdots|\boldsymbol{b}'_m]$ for the lattice $\Lambda(B)$, with $B = [\boldsymbol{b}_1|\cdots|\boldsymbol{b}_m]$, such that the length of vector $\boldsymbol{b}'_1$ is sufficiently close to λ_1. That can be achieved by reducing the basis to a δ-LLL reduced basis with $\delta = (1/4) + (3/4)^{m/(m-1)}$ (see Definition 2.1):

Lemma 2.1. *If $B = [\boldsymbol{b}_1|\cdots|\boldsymbol{b}_m] \in \mathbb{R}^{n\times m}$ is a δ-LLL reduced basis with $\delta = (1/4) + (3/4)^{m/(m-1)}$, then $\|\boldsymbol{b}_1\| \leq (2/\sqrt{3})^m\lambda_1$.*

The LLL Algorithm has two main steps, the reduction step and the swap step, each taking care of one property of the δLLL reduced basis definition. The reduction step redefines the value of each vector $\boldsymbol{b}_i$ so that $\left|\frac{\langle\widetilde{\boldsymbol{b}}_j,\boldsymbol{b}_i\rangle}{\langle\widetilde{\boldsymbol{b}}_j,\widetilde{\boldsymbol{b}}_j\rangle}\right| \leq \frac{1}{2}$. The swap step swaps the vectors $\boldsymbol{b}_i$ and $\boldsymbol{b}_{i+1}$ so that $\delta\|\pi_i(\boldsymbol{b}_i)\|^2 \leq \|\pi_i(\boldsymbol{b}_{i+1})\|^2$. The LLL Algorithm is described in 2.1.

Algorithm 2.1 LLL(): Lenstra-Lenstra-Lovász Algorithm.

```
Input: lattice basis $B$, approximation factor $\delta$
Output: $\delta LLL$ reduced basis $B'$
   while $end$ not 1
      for $i \leftarrow 2$ to $m$
         for $j \leftarrow i-1$ downto 1
            $\boldsymbol{b}_i \leftarrow \boldsymbol{b}_i - \lceil\langle\boldsymbol{b}_i,\widetilde{\boldsymbol{b}}_j\rangle/\langle\widetilde{\boldsymbol{b}}_j,\widetilde{\boldsymbol{b}}_j\rangle\boldsymbol{b}_j\rfloor$
      $end \leftarrow 1$
      for $i \leftarrow 1$ to $m-1$
         if $\delta\|\pi_i(\boldsymbol{b}_i)\|^2 > \|\pi_i(\boldsymbol{b}_{i+1})\|^2$
            $\boldsymbol{b}_i \leftrightarrow \boldsymbol{b}_{i+1}$
            $end \leftarrow 0$
            $i \leftarrow m$
   output $B' = (\boldsymbol{b}_1,\cdots,\boldsymbol{b}_m)$
```

The goal of Babai's Algorithm is to find a vector that is close to the target vector $\boldsymbol{t}$. To achieve this, Babai's Nearest Plane Algorithm uses the LLL reduction algorithm, then performs a step that is essentially the same as the first step of the LLL Algorithm. Babai's Algorithm is described in 2.2.

We may consider this algorithm from a geometrical point of view, which explains the name of the algorithm: First find the vector $\boldsymbol{s}$ that is a projection of $\boldsymbol{t}$ on $\mathrm{span}(B)$ and $c \in \mathbb{Z}$ such that the hyperplane $c\widetilde{\boldsymbol{b}}_m + \mathrm{span}(B')$, with $B' = [\boldsymbol{b}_1|\cdots|\boldsymbol{b}_{m-1}]$, is as close as possible to $\boldsymbol{s}$. Then, call the algorithm recursively for basis B' and vector $\boldsymbol{s} - c\boldsymbol{b}_m$, and for a return vector $\boldsymbol{x}$, output $\boldsymbol{x} - c\boldsymbol{b}_m$ as an answer.

Algorithm 2.2 Babai(): Babai's Nearest Plane Algorithm.

```
Input: lattice basis $B$, target vector $\boldsymbol{t}$
Output: vector $\boldsymbol{v}$ close to target vector $\boldsymbol{t}$
  $B \leftarrow \mathsf{LLL}(B, 3/4)$
  $\boldsymbol{b} \leftarrow \boldsymbol{t}$
  for $j \leftarrow m$ downto $1$
    $\boldsymbol{b} \leftarrow \boldsymbol{b} - \lceil \langle \boldsymbol{b}, \widetilde{\boldsymbol{b}}_j \rangle / \langle \widetilde{\boldsymbol{b}}_j, \widetilde{\boldsymbol{b}}_j \rangle \boldsymbol{b}_j \rfloor$
  output $\boldsymbol{v} \leftarrow \boldsymbol{t} - \boldsymbol{b}$
```

2.2 Trapdoor Generation and Gaussian Samples

The Gaussian function $f(x)$ in one dimension for $x \in \mathbb{R}$ is given by:

$$f(x) = ae^{-(x-b)^2/2c^2};$$

for $a, b, c \in \mathbb{R}$ and e is Euler's number. The *Gaussian*, or *normal* distribution, is the probability distribution in $\mathbb{R}$, with $a = 1/\sigma\sqrt{2\pi}$, $b = \mu$ and $c = \sigma$, with μ being the mean and σ being the standard deviation:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}.$$

The *Standard Spherical Gaussian* is the distribution in $\mathbb{R}^n$, with $a = 1$, $b = \boldsymbol{c}$ and $c = \sqrt{2\pi}\sigma$, for $\boldsymbol{c} \in \mathbb{R}^n$ and real $\sigma > 0$:

$$\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x}) = e^{-\pi\|\boldsymbol{x}-\boldsymbol{c}\|^2/\sigma^2}.$$

The *Elliptical Gaussian* distribution in $\mathbb{R}^n$ is given by a vector of standard deviation $\boldsymbol{\sigma}$ as follows:

$$\rho'_{\boldsymbol{\sigma},\boldsymbol{c}}(\boldsymbol{x}) = e^{-\sum[\pi(x_i-c_i)^2/\sigma_i^2]}.$$

The *Discrete Gaussian* distribution $D_{\Lambda,\sigma,\boldsymbol{c}}$ over a lattice is given by:

$$\mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}(\boldsymbol{x}) = \frac{\rho_{\sigma,\boldsymbol{c}}(\boldsymbol{x})}{\sum_{y\in\Lambda} \rho_{\sigma,\boldsymbol{c}}(\boldsymbol{y})}, \text{ if } \boldsymbol{x} \in \Lambda \text{ and } 0 \text{ elsewhere.}$$

The following lemma captures standard properties of these distributions.

Lemma 2.2. *Let $q \geq 2$ and let A be a matrix in $\mathbb{Z}_q^{n\times m}$ with $m > n$. Let S be a basis for $\Lambda_q^{\perp}(A)$ and $\sigma \geq \|S\| \cdot \omega(\sqrt{\log m})$. Then for $\boldsymbol{c} \in \mathbb{R}^m$ and $\boldsymbol{u} \in \mathbb{Z}_q^n$:*
$\Pr\left[\|\boldsymbol{x} - \boldsymbol{c}\| > \sigma\sqrt{m} \ : \ \boldsymbol{x} \xleftarrow{\$} \mathcal{D}_{\Lambda,\sigma,\boldsymbol{c}}\right] \leq \mathrm{negl}(n)$

Here we define two important distributions in our context:

Definition 2.12. *For $\alpha \in \{0, \ldots, 1\}$ and an integer $q > 2$, let $\overline{\Psi}_\alpha$ denote the probability distribution over $\mathbb{Z}_q$ obtained by choosing $x \in \mathbb{R}$ according to $\rho_{\sigma,\boldsymbol{c}}$ with $\boldsymbol{c} = 0$ and $\sigma = \alpha/\sqrt{2\pi}$ and outputting $\lfloor qx \rceil$.*

Definition 2.13. *For $\alpha \in \{0, \ldots, 1\}$ and an integer n, let Υ_α denote the probability distribution over $\mathbb{Z}_q$ obtained by choosing $x \in \mathbb{R}$ according to $\rho'_{\boldsymbol{\sigma},\boldsymbol{c}}$ with $\boldsymbol{c} = 0$ and $\sigma_i^2 = \sigma_{i+n/2}^2 = \alpha^2(1 + \sqrt{n}x_i)$, for $x_i \in \overline{\Psi}_\alpha$.*

2.2.1 Generating a Trapdoor

Public-key cryptographic schemes are based on one-way functions, i.e., functions that are easy to compute, but hard to invert. For encryption and signing, it is also necessary that such functions possess a trapdoor, i.e., a shortcut that is kept secret, and that makes it possible for the secret holder to easily invert the function. For lattices, the trapdoor is usually a short basis, as described in Section 2.1.1.

Ajtai [8] and later Alwen and Peikert [9] showed how to generate an essentially uniform matrix $A \in \mathbb{Z}_q^{n\times m}$ along with a basis S of $\Lambda_q^{\perp}(A)$ with low Gram-Schmidt norm.

Theorem 2.1 ([9]). *Let q, n, m be positive integers with $q \geq 2$ and $m \geq 6n \lg q$. Then, there is a probabilistic polynomial-time algorithm* $\mathsf{TrapGen}(q, n, m)$ *that outputs a pair $(A \in \mathbb{Z}_q^{n\times m}, S \in \mathbb{Z}^{m\times m})$ such that A is statistically close to uniform in $\mathbb{Z}_q^{n\times m}$ and S is a basis for $\Lambda_q^{\perp}(A)$, satisfying $\|\widetilde{S}\| \leq O(\sqrt{n \log q})$ and $\|S\| \leq O(n \log q)$ with overwhelming probability in n.*

Stehlé et al. [73] showed an adaptation of Ajtai's trapdoor key generation algorithm for ideal lattices, generating an essentially uniform vector $\hat{\boldsymbol{a}} \in (\mathbb{Z}_q[x]/\langle f(x)\rangle)^k$ along with a basis S of $\Lambda_q^{\perp}(\mathsf{Rot}_f(\hat{\boldsymbol{a}}))$.

Theorem 2.2 ([73]). *Let n, σ, q, k be positive integers with $q \equiv 3 \mod 8$, $k \geq \lceil \log q + 1 \rceil$ and n being a power of 2 and let $f(x)$ be a n-degree polynomial in $\mathbb{Z}[x]$, with $f(x) = x^n + 1$.*

There is a probabilistic polynomial-time algorithm $\mathsf{IdealTrapGen}(q, n, k, \sigma, f)$ *that outputs a pair* $(\hat{\boldsymbol{a}} \in (\mathbb{Z}_q[x]/\langle f(x)\rangle)^k, S \in \mathbb{Z}^{kn \times kn})$ *such that* $\hat{\boldsymbol{a}}$ *is statistically close to uniform in* $\hat{\boldsymbol{a}} \in (\mathbb{Z}_q[x]/\langle f(x)\rangle)^k$ *and* S *is a basis for* $\Lambda_q^{\perp}(A)$*, for* $A = \mathsf{Rot}_f(\hat{\boldsymbol{a}})$*, satisfying* $|S| = O(n \log q \sqrt{\omega(\log n)})$ *with overwhelming probability in* n*.*

Later, Micciancio and Peikert [52] improved the TrapGen algorithms by using a simpler and faster method based only on the multiplication of random matrices to generate the lattice basis, without involving any expensive Hermite Normal Form or matrix inversion computations. This construction is faster and simpler and it also improves the quality of the trapdoor generated.

2.2.2 Sample Algorithms

Using a short basis and a Gaussian distribution it is possible to sample lattice points. This section describes the sample algorithms used on most of the lattice-based schemes as shown in [29], [61] and [23].

The foundation for all the sample algorithms is the SampleInt() described in Algorithm 2.3. It samples from a discrete Gaussian distribution over $\mathbb{Z}$, i.e., it samples over a particular one-dimensional lattice. On input of the Gaussian parameters c and σ, it outputs an integer e statistically close to $\mathcal{D}_{\mathbb{Z},\sigma,c}$.

Algorithm 2.3 SampleInt(): Algorithm to sample from a Gaussian distribution over $\mathbb{Z}$.

```
Input: Gaussian parameters $\sigma$ and $c$ and integer $n$
Output: integer $e$ statistically close to $\mathcal{D}_{\mathbb{Z},\sigma,c}$
    choose $t(n) \geq \omega(\sqrt{\log n})$
    $x \xleftarrow{\$} \mathbb{Z} \cap [c - \sigma \cdot t, c + \sigma \cdot t]$
    output $e$ with probability $\rho_{\sigma,0}(x - c)$
```

Theorem 2.3 ([29])**.** *Let* σ, c *be Gaussian parameters and let* n *be an integer. Then there is a probabilistic polynomial algorithm* $\mathsf{SampleInt}(\sigma, c, n)$ *that outputs an integer* $e \in \mathbb{Z}$ *statistically close to* $\mathcal{D}_{\mathbb{Z},\sigma,c}$*.*

Using the algorithm that samples integers from a discrete Gaussian distribution over $\mathbb{Z}$ we can construct SampleLattice(), described in Algorithm 2.4, that samples from a discrete Gaussian over any lattice. The algorithm works like Babai's Nearest Plane Algorithm, described on Section 2.1.4, but it chooses a plane at random with a probability given by a discrete Gaussian over $\mathbb{Z}$.

Algorithm 2.4 SampleLattice(): Algorithm to sample from a Gaussian distribution over $\Lambda(A)$.

```
Input: lattice basis $A$, with rank $m$, and Gaussian parameters $\sigma$ and $c$
Output: vector $\boldsymbol{e} \in \mathbb{Z}^m$ statistically close to $\mathcal{D}_{\Lambda(A),\sigma,\boldsymbol{c}}$
    if $m = 0$, return 0
    compute the Gram-Schmidt vector $\widetilde{\boldsymbol{a}}_m$ from $A$
    vector $\boldsymbol{t}$ is the projection of $\boldsymbol{c}$ onto span$(A)$
    integer $t \leftarrow \langle \boldsymbol{t}, \widetilde{\boldsymbol{a}}_m \rangle / \langle \widetilde{\boldsymbol{a}}_m, \widetilde{\boldsymbol{a}}_m \rangle$
    integer $z \leftarrow \mathsf{SampleInt}(\sigma / \|\widetilde{\boldsymbol{a}}_m\|, t)$
    matrix $A' = [\boldsymbol{a}_1|...|\boldsymbol{a}_{m-1}]$
    output $\boldsymbol{e} \leftarrow z \cdot \boldsymbol{a}_m + \mathsf{SampleLattice}(A', \sigma, \boldsymbol{t} - z \cdot \boldsymbol{a}_m)$
```

Theorem 2.4 ([29])**.** *Let $\sigma, \boldsymbol{c}$ be Gaussian parameters, let $A \in \mathbb{Z}^{n \times m}$ be a matrix and let n, m be integers such that $\sigma \geq \|A\| \cdot \omega(\sqrt{\log n})$. Then there is a probabilistic polynomial algorithm* SampleLattice$(A, \sigma, \boldsymbol{c})$ *that outputs a vector $\boldsymbol{e} \in \mathbb{Z}^m$ statistically close to $\mathcal{D}_{\Lambda(A),\sigma,\boldsymbol{c}}$.*

The SampleLattice$(A, \sigma, \boldsymbol{c})$ allows us to sample over any $\Lambda(A)$, but it restricts the basis length. It is usual, however, that the basis of the lattice has an arbitrary length, but we have a short set S of linearly independent lattice vectors of $\Lambda_q^\perp(A)$. In this case, it is possible to sample from a discrete Gaussian distribution over $\Lambda_q^\perp(A)$ using SampleGaussian(), described in Algorithm 2.5.

Algorithm 2.5 SampleGaussian(): Algorithm to sample from a Gaussian distribution over $\Lambda_q^\perp(A)$.

```
Input: lattice basis $A$, short set of linearly independent lattice vectors $S$ and Gaussian
parameters $\sigma$ and $c$
Output: vector $\boldsymbol{e} \in \mathbb{Z}^m$ statistically close to $\mathcal{D}_{\Lambda_q^\perp(A),\sigma,\boldsymbol{c}}$
    choose $\boldsymbol{v} \in \Lambda(A)$, such that $\boldsymbol{v} \mod \Lambda(S)$ is distributed uniformly in $\Lambda(A)/\Lambda(S)$
    $\boldsymbol{y} \leftarrow \mathsf{SampleLattice}(S, \sigma, \boldsymbol{c} - \boldsymbol{v})$
    output $\boldsymbol{e} \leftarrow \boldsymbol{y} + \boldsymbol{v}$
```

Theorem 2.5 ([29])**.** *Let $A \in \mathbb{Z}_q^{n \times m}$ be a full rank matrix, let S be a short basis of $\Lambda_q^\perp(A)$ and let $\sigma, \boldsymbol{c}$ be Gaussian parameters. Let q, m, n be integers such that $q > 2$ and $m > n$ and let $\sigma > \|S\| \cdot \omega(\sqrt{\log m})$. Then there is a probabilistic polynomial algorithm* SampleGaussian$(A, S, \sigma, \boldsymbol{c})$ *that outputs a vector $\boldsymbol{e} \in \mathbb{Z}^m$ statistically close to $\mathcal{D}_{\Lambda_q^\perp(A),\sigma,\boldsymbol{c}}$.*

Note that $\Lambda_q^{\boldsymbol{u}}(A) = \boldsymbol{t} + \Lambda_q^\perp(A)$ for some $\boldsymbol{t} \in \Lambda_q^{\boldsymbol{u}}(A)$. Therefore, to sample from a distribution in $\Lambda_q^{\boldsymbol{u}}(A)$, we simply call SampleGaussian with $\boldsymbol{c} = \boldsymbol{t}$ and subtract $\boldsymbol{t}$ from the result. This algorithm, called SamplePre, is described in Algorithm 2.6.

Algorithm 2.6 SamplePre(): Algorithm to sample from a Gaussian distribution over $\Lambda_q^{\boldsymbol{u}}(A)$.

```
Input: lattice basis $A$, short set of linearly independent lattice vectors $S$, Gaussian parameter $\sigma$ and vector $\boldsymbol{u}$
Output: vector $\boldsymbol{e} \in \mathbb{Z}^m$ statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(A),\sigma,\boldsymbol{c}}$
    $\boldsymbol{t} \xleftarrow{\$} \mathbb{Z}^m$, such that $A\boldsymbol{t} = \boldsymbol{u} \bmod q$
    $\boldsymbol{x} \leftarrow \mathsf{SampleGaussian}(A, S, \sigma, \boldsymbol{t})$
    output $\boldsymbol{e} \leftarrow \boldsymbol{x} - \boldsymbol{t}$
```

Theorem 2.6 ([29]). *Let $A \in \mathbb{Z}_q^{n\times m}$ be a full rank matrix, let S be a short basis of $\Lambda_q^{\perp}(A)$ and let $\sigma, \boldsymbol{c}$ be Gaussian parameters such that $\boldsymbol{c} = \boldsymbol{0}$. Let q, m, n be integers such that $q > 2$ and $m > n$ and let $\sigma > \|S\| \cdot \omega(\sqrt{\log m})$. Then there is a probabilistic polynomial algorithm* SamplePre$(A, S, \boldsymbol{u}, \sigma)$ *that outputs a vector $\boldsymbol{e} \in \mathbb{Z}^m$ statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(A),\sigma,\boldsymbol{c}}$.*

It may be important for some schemes to sample from a discrete Gaussian distribution over $\Lambda_q^{\boldsymbol{u}}(A|B)$, for $A \in \mathbb{Z}_q^{n\times m}$ and $B \in \mathbb{Z}_q^{n\times m_1}$, when we only have the short basis for $\Lambda_q^{\perp}(A)$. SampleLeft$(A, B, S, \boldsymbol{u}, \sigma)$, described in Algorithm 2.7, accomplishes that, using a sample over $\mathbb{Z}^{m_1}$ and the SamplePre algorithm.

Algorithm 2.7 SampleLeft(): Algorithm to sample from a Gaussian distribution over $\Lambda_q^{\boldsymbol{u}}(A|B)$.

```
Input: lattice basis $A$, short set of linearly independent lattice vectors $S$, matrix $B$, Gaussian parameter $\sigma$ and vector $\boldsymbol{u}$
Output: $\boldsymbol{e} \in \mathbb{Z}^{m+m_1}$ statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(F),\sigma,\boldsymbol{c}}$, with $F = (A|B)$
    matrix $Z$ is a basis for $\mathbb{Z}^{m_1}$
    $\boldsymbol{e_2} \leftarrow \mathsf{SampleLattice}(Z, \sigma, \boldsymbol{0})$
    $\boldsymbol{y} \leftarrow \boldsymbol{u} - B\boldsymbol{e_2}$
    $\boldsymbol{e_1} \leftarrow \mathsf{SamplePre}(A, S, \sigma, \boldsymbol{y})$
    output $\boldsymbol{e} \leftarrow [\boldsymbol{e_1}|\boldsymbol{e_2}]$
```

Theorem 2.7 ([4]). *Let $A \in \mathbb{Z}_q^{n\times m}$ be a full rank matrix, let S be a short basis of $\Lambda_q^{\perp}(A)$, let $B \in \mathbb{Z}_q^{n\times m_1}$ be a matrix, let $\boldsymbol{u} \in \mathbb{Z}_q^n$ be a vector and $\sigma, \boldsymbol{c}$ be Gaussian parameters such that $\boldsymbol{c} = \boldsymbol{0}$. Let q, m, n be integers such that $q > 2$ and $m > 2n \log q$ and let $\sigma > \|S\| \cdot \omega(\sqrt{\log(m + m_1)})$. Then there is a probabilistic polynomial algorithm* SampleLeft$(A, B, S, \boldsymbol{u}, \sigma)$ *that outputs a vector $\boldsymbol{e} \in \mathbb{Z}^{m+m_1}$ statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(F),\sigma,\boldsymbol{c}}$, with $F = (A|B)$.*

The last algorithm samples a vector over $\Lambda_q^{\boldsymbol{u}}(F)$, with $F = [A|AR + B]$, and only a short basis for $\Lambda_q^{\perp}(B)$ is known. In this case, we find a matrix $T \in \mathbb{Z}^{n\times 2m}$ with linearly independent vectors in $\Lambda_q^{\perp}(F)$, construct a short basis S_F for the lattice using Lemma 2.3 and call $\mathsf{SamplePre}()$ to sample from a discrete Gaussian distribution over $\Lambda_q^{\boldsymbol{u}}(F)$. Algorithm 2.8 describes $\mathsf{SampleRight}()$.

Lemma 2.3. *There is a deterministic polynomial-time algorithm* $\mathsf{BasisConstruction}(A, T)$ *that, given an arbitrary basis* A *of* Λ *and a full-rank set* T *in* Λ*, returns a basis* S *of* Λ *satisfying* $\|\widetilde{T}\| \le \|\widetilde{S}\|$ *and* $\|T\| \le \|S\|\sqrt{m}/2$*.*

Algorithm 2.8 $\mathsf{SampleRight}()$: Algorithm to sample from a Gaussian distribution over $\Lambda_q^{\perp}(A|AR + B)$.

```
Input: lattice basis $B$, short set of linearly independent lattice vectors $S$ matrices $A$ and $R$, Gaussian parameter $\sigma$ and vector $\boldsymbol{u}$
Output: vector $\boldsymbol{e} \in \mathbb{Z}^{m+m_1}$ statistically close to $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(F),\sigma,\boldsymbol{c}}$, with $F = (A|AR + B)$
  $F = [A|AR + B]$
  for $i \leftarrow 1$ to $m$
    $\boldsymbol{t}_i \leftarrow \begin{bmatrix} -R\boldsymbol{s}_i \\ \boldsymbol{s}_i \end{bmatrix}^\top$
  choose $U$, such that $A\mathcal{I} + BU = \mathcal{O} \bmod q$
  for $i \leftarrow m + 1$ to $2m$
    $\boldsymbol{t}_i \leftarrow \begin{bmatrix} \boldsymbol{w}_i - R\boldsymbol{u}_i \\ \boldsymbol{u}_i \end{bmatrix}^\top$, $\boldsymbol{w}_i \in \mathcal{I}$
  $S_F \leftarrow \mathsf{BasisConstruction}(A, T)$
  output $\boldsymbol{e} \leftarrow \mathsf{SamplePre}(F, S_F, \boldsymbol{u}, \sigma)$
```

Theorem 2.8 ([4])**.** *Let* $A \in \mathbb{Z}_q^{n\times m}$ *and* $B \in \mathbb{Z}_q^{n\times m_1}$ *be a full rank matrices, let* S *be a short basis of* $\Lambda_q^{\perp}(B)$*, let* $R \in \{-1, 1\}^{m\times m_1}$ *be a uniform random matrix, let* $\boldsymbol{u} \in \mathbb{Z}_q^n$ *be a vector and let* $\sigma, \boldsymbol{c}$ *be Gaussian parameters such that* $\boldsymbol{c} = \boldsymbol{0}$*. Let* q, m, n *be integers such that* $q > 2$ *and* $m > n$ *and let* $\sigma > \|S\| \cdot \sqrt{m} \cdot \omega(\sqrt{\log m})$*. Then there is a probabilistic polynomial algorithm* $\mathsf{SampleRight}(A, B, R, S, \boldsymbol{u}, \sigma)$ *that outputs a vector* $\boldsymbol{e} \in \mathbb{Z}^{m+m_1}$ *statistically close to* $\mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(F),\sigma,\boldsymbol{c}}$*, with* $F = (A|AR + B)$*.*

Besides sampling vectors from lattices, we may need to sample random basis from lattices.

The first sample basis algorithm, $\mathsf{SampleBasis}()$, just generates another basis for the same lattice. It uses algorithm $\mathsf{SampleLattice}()$ to sample vectors from the lattice and keeps the vectors that are linearly independent. Using the linearly independent vectors, it is possible to build a new lattice basis using algorithm $\mathsf{BasisConstruction}()$. Algorithm 2.9 describes $\mathsf{SampleBasis}()$.

Algorithm 2.9 SampleBasis(): Algorithm to sample a basis for lattice $\Lambda(A)$.

```
Input: lattice basis $A$ and Gaussian parameter $\sigma$
Output: lattice basis $T$
    for $i \leftarrow 1$ to $m$
        while $\boldsymbol{v}$ is not linearly independent of $V = (\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{i-1})$
            $\boldsymbol{v} \leftarrow \mathsf{SampleLattice}(A, \sigma, 0)$
        $\boldsymbol{v}_i \leftarrow \boldsymbol{v}$
    $T \leftarrow \mathsf{BasisConstruction}(\mathsf{HNF}(A), V)$
    output $T$
```

Theorem 2.9 ([23]). *Let $A \in \mathbb{Z}_q^{n\times m}$ be a full rank matrix, let σ be Gaussian parameter and let n, m be integers such that $\sigma \geq \|A\| \cdot \omega(\sqrt{\log n})$. Then there is a probabilistic polynomial algorithm* SampleBasis(A, σ, c) *that outputs a new basis $T \in \mathbb{Z}_q^{n\times m}$ for the lattice $\Lambda(A)$.*

SampleBasisLeft(), described in Algorithm 2.10, is similar to algorithm SampleBasis(), but instead of finding a basis for lattice $\Lambda(A)$ it finds a new basis for lattice $\Lambda_q^{\perp}(A|B)$, given that A is a basis for $\Lambda_q^{\perp}(A)$, S is a short set of linearly independent lattice vectors and M is a random matrix. It accomplishes that by replacing the SampleLattice() algorithm with the SampleLeft() in the main loop.

Algorithm 2.10 SampleBasisLeft(): Algorithm to sample a basis for lattice $\Lambda_q^{\perp}(A|B)$.

```
Input: lattice basis $A$, short set of linearly independent lattice vectors $S$, matrix $B$ and Gaussian parameter $\sigma$
Output: lattice basis $T$
    for $i \leftarrow 1$ to $m$
        while $\boldsymbol{v}$ is not linearly independent of $V = (\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{i-1})$
            $\boldsymbol{v} \leftarrow \mathsf{SampleLeft}(A, S, B, \sigma, 0)$
        $\boldsymbol{v}_i \leftarrow \boldsymbol{v}$
    $T \leftarrow \mathsf{BasisConstruction}(\mathsf{HNF}(A), V)$
    output $T$
```

Theorem 2.10 ([4]). *Let $A \in \mathbb{Z}_q^{n\times m}$ be a full rank matrix, let S be a short basis of $\Lambda_q^{\perp}(A)$, let $B \in \mathbb{Z}_q^{n\times m_1}$ be a matrix and let σ be a Gaussian parameter. Let q, m, n be integers such that $q > 2$ and $m > 2n \log q$ and let $\sigma > \|S\| \cdot \omega(\sqrt{\log(m + m_1)})$. Then there is a probabilistic polynomial algorithm* SampleBasisLeft(A, B, S, σ) *that outputs a new basis $T \in \mathbb{Z}^{n\times m+m_1}$ for the lattice $\Lambda_q^{\perp}(F)$, with $F = (A|B)$.*

As the two previous basis sample algorithms, the SampleBasisRight() algorithm finds a basis for the lattice $\Lambda_q^{\perp}(A|AR + B)$ by building a linearly independent vector set V and using the BasisConstruction() algorithm to construct a basis from it. The vector set V is now build using the SampleRight() algorithm to sample from lattice $\Lambda_q^{\perp}(A|AR+B)$. Algorithm 2.11 describes SampleBasisRight().

Algorithm 2.11 SampleBasisRight(): Algorithm to sample a basis for lattice $\Lambda_q^{\perp}(A|AR+B)$.

```
Input: lattice basis $B$, short set of linearly independent lattice vectors $S$, matrices $A$ and $R$, Gaussian parameter $\sigma$
Output: lattice basis $T$
  for $i \leftarrow 1$ to $m$
    while $\boldsymbol{v}$ is not linearly independent of $V = (\boldsymbol{v}_1, \cdots, \boldsymbol{v}_{i-1})$
      $\boldsymbol{v} \leftarrow \mathsf{SampleRight}(A, B, R, S, 0, \sigma)$
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}$
  $T \leftarrow \mathsf{BasisConstruction}(\mathsf{HNF}(A), V)$
  output $T$
```

Theorem 2.11 ([4]). *Let $A \in \mathbb{Z}_q^{n\times m}$ and $B \in \mathbb{Z}_q^{n\times m_1}$ be a full rank matrices, let S be a short basis of $\Lambda_q^{\perp}(B)$, let $R \in \{-1,1\}^{m\times m1}$ be a uniform random matrix and let σ be a Gaussian parameter. Let q, m, n be integers such that $q > 2$ and $m > n$ and let $\sigma > \|S\| \cdot \sqrt{m} \cdot \omega(\sqrt{\log m})$. Then,* SampleBasisRight(A, B, R, S, σ)*, is a probabilistic polynomial algorithm that outputs a new basis $T \in \mathbb{Z}^{n\times m+m_1}$ for the lattice $\Lambda_q^{\perp}(F)$, with $F = (A|AR+B)$.*

2.3 The Learning With Errors Problem

The Learning with Errors Problem, or LWE, is not a hard lattice problem, but it has been used as the security base for several lattice-based cryptosystems. This is due to the proximity of the LWE problem to lattices and to the reductions from hard lattice problems to LWE.

The LWE problem basically states that given samples of the form $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{r}\rangle + e)$ one must find $\boldsymbol{r}$. In its decision version, samples from a distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ are given and one must decide if these samples are from a uniform distribution or of the form $(\boldsymbol{a}, \langle \boldsymbol{a}, \boldsymbol{r}\rangle + e)$.

Definition 2.14. ***The* LWE *Problem***: *Let $n \geq 1$ and $q \geq 2$ be integers, and let χ be a probability distribution on $\mathbb{Z}_q$. For $\boldsymbol{r} \in \mathbb{Z}_q^n$, let $A_{\boldsymbol{r},\chi}$ be the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $\boldsymbol{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in$*

$\mathbb{Z}_q$ according to χ, and outputting $(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{r} \rangle + e)$. Given an arbitrary number of independent samples from the distribution $A_{\boldsymbol{r},\chi}$, find $\boldsymbol{r}$.

Definition 2.15. *The decision-*LWE *Problem*: *Let $n \geq 1$ and $q \geq 2$ be integers, and let χ be a probability distribution on $\mathbb{Z}_q$. For $\boldsymbol{r} \in \mathbb{Z}_q^n$, let $A_{\boldsymbol{r},\chi}$ be the probability distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$ obtained by choosing a vector $\boldsymbol{a} \in \mathbb{Z}_q^n$ uniformly at random, choosing $e \in \mathbb{Z}_q$ according to χ, and outputting $(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{r} \rangle + e)$. Given an arbitrary number of independent samples from $\mathbb{Z}_q^n \times \mathbb{Z}_q$, return YES if the samples come from the distribution $A_{\boldsymbol{r},\chi}$ and NO if they come from the uniform distribution on $\mathbb{Z}_q^n \times \mathbb{Z}_q$.*

The two versions of the LWE problem above are equivalent, i.e., the decision version of LWE is as hard as the regular (or search) version, and the search version is as hard as the decision version. We give a sketch of the proof of this equivalence.

It is easy to notice that if we have an oracle for the LWE problem, then we can solve the decision version. Given samples from $\mathbb{Z}_q^n \times \mathbb{Z}_q$, use the search oracle to find a vector $\boldsymbol{r} \in \mathbb{Z}_q^n$ and check if it produces the pair $(\boldsymbol{a}, b = \langle \boldsymbol{a}, \boldsymbol{r} \rangle + e)$ on the samples.

Given an oracle to the *decision*-LWE problem, it is possible to solve the search version one coordinate at a time. For the i-th coordinate of $\boldsymbol{r}$ choose a random number $l \in \mathbb{Z}_q$ and one guess k for r_i. Call the decision oracle for the sample $(\boldsymbol{a} + (l, 0, \ldots, 0), b + (lk)/q)$; if the guess k was not correct, the decision will choose the uniform distribution, if it was correct it will choose the distribution $A_{\boldsymbol{r},\chi}$.

Regev [62] showed a reduction from γ-GapSVP and γ-SIVP to LWE, proving that the LWE problem is as hard as the two lattice problems. This is a quantum reduction, i.e., it uses quantum algorithms and it is only valid within quantum machines. Later, Peikert [60] improved Regev's work, by showing a classical reduction from γ-GapSVP to LWE. For a simpler notation, we use $\mathsf{LWE}_{q,n,\chi}$ to denote the LWE for inputs q, n and χ.

Theorem 2.12 ([62])**.** *Let n, q be integers and $\alpha \in \{0, \ldots, 1\}$ such that $q = \mathrm{poly}(n)$ and $\alpha q > 2\sqrt{n}$. If there exists an efficient (possibly quantum) algorithm that solves decision-$\mathsf{LWE}_{q,n,\overline{\Psi}_\alpha}$, then there exists an efficient quantum algorithm that solves γ-SIVP and γ-GapSVP for $\gamma = \widetilde{O}(n/\alpha)$ in the worst case.*

Theorem 2.13 ([60])**.** *Let n, q be integers and $\alpha \in \{0, \ldots, 1\}$, such that $q = \sum_i q_i \leq 2^{n/2}$, where the q_i are distinct primes satisfying $\omega(\log n)/\alpha \leq q_i \leq \mathrm{poly}(n)$. If there exists an efficient (classical) algorithm that solves decision-$\mathsf{LWE}_{q,n,\overline{\Psi}_\alpha}$, then there exists an efficient (classical) algorithm that solves γ-GapSVP for $\gamma = \widetilde{O}(n/\alpha)$ in the worst case.*

2.3.1 LWE Problem for Rings and Ideal Lattices

As already stated, one important class of lattices is the ideal lattices. They are the basis for several cryptosystems and in proving the security of these cryptosystems it is crucial to show that the LWE Problem is still hard for these specific lattices. The Ring-LWE Problem is a special case, in which the inner product is replaced by a product of polynomials and the probability distribution is on a larger range. The Ideal-LWE Problem is a lot similar to the general LWE Problem, except for a change in the form in which the samples are chosen and the fact that it specifies the number of samples.

Definition 2.16. ***The* Ring-LWE *Problem*:** *Let n and $q = 1 \mod 2n$ be integers, with n a power of 2, let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ be a ring and let χ be a probability distribution on $\mathcal{R}_q$. For $\boldsymbol{r} \in \mathcal{R}_q$, let $A_{\boldsymbol{r},\chi}$ be the probability distribution on $\mathcal{R}_q \times \mathcal{R}_q$ obtained by choosing a vector $\boldsymbol{a} \in \mathcal{R}_q$ uniformly at random, choosing $\boldsymbol{e} \in \mathcal{R}_q$ according to χ, and outputting $(\boldsymbol{a}, \boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{r} + \boldsymbol{e})$. Given an arbitrary number of independent samples from the distribution $A_{\boldsymbol{r},\chi}$, find $\boldsymbol{r}$.*

Definition 2.17. ***The decision*-Ring-LWE *Problem*:** *Let n and $q = 1 \mod 2n$ be integers, with n a power of 2, let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ be a ring and let χ be a probability distribution on $\mathcal{R}_q$. For $\boldsymbol{r} \in \mathcal{R}_q$, let $A_{\boldsymbol{r},\chi}$ be the probability distribution on $\mathcal{R}_q \times \mathcal{R}_q$ obtained by choosing a vector $\boldsymbol{a} \in \mathcal{R}_q$ uniformly at random, choosing $\boldsymbol{e} \in \mathcal{R}_q$ according to χ, and outputting $(\boldsymbol{a}, \boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{r} + \boldsymbol{e})$. Given an arbitrary number of independent samples from $\mathcal{R}_q \times \mathcal{R}_q$, return YES if the samples come from the distribution $A_{\boldsymbol{r},\chi}$ and NO if they come from the uniform distribution on $\mathcal{R}_q \times \mathcal{R}_q$.*

The search-to-decision reduction shown for the LWE Problem does not work for the Ring-LWE Problem. This happens because the distribution is now on $\mathcal{R}_q \times \mathcal{R}_q$ instead of $\mathbb{Z}_q^n \times \mathbb{Z}_q$ and it is not possible to isolate each coordinate of $\boldsymbol{r}$ on the Ring-LWE Problem and we must guess all of them correct at a time. A more sophisticated reduction for only cyclotomic rings is shown by Lyubashevsky, Peikert and Regev [47].

The hardness of the Ring-LWE Problem was shown by Lyubashevsky et al. [47]. The proof is a lot similar to the one shown by Regev [62] for the general LWE Problem, the quantum reduction is used almost without any adaptation. Note, however, that the proof only applies if the error distribution is chosen from a certain distribution on non-spherical Gaussian variables.

Theorem 2.14 ([47])**.** *Let n, q be integers and $\alpha > 0$ such that $q \geq 2$, $q = 1 \mod m$ and q be a $\mathrm{poly}(n)$-bounded prime such that $\alpha q \geq \omega(\sqrt{\log n})$. If there exists an efficient (possibly quantum) algorithm that solves decision-Ring-LWE$_{q,n,\Upsilon_\alpha}$, then there exists an efficient quantum algorithm that solves γ-SIVP and γ-SVP for $\gamma = \widetilde{O}(n/\alpha)$ in the worst case.*

The Ideal-LWE Problem [73] does not use polynomial multiplication, it uses only the vector and matrix representation of polynomials, i.e., the function $\mathsf{Rot}_f()$. That is possible, because for $\boldsymbol{a}, \boldsymbol{r} \in \mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$, we have that the polynomial multiplication of $\boldsymbol{a}$

and $\boldsymbol{r}$ gives a polynomial that has its vector representation equals to $\mathsf{rot}_f(\boldsymbol{a})^\top \boldsymbol{r} = \boldsymbol{a} \cdot \boldsymbol{r}$, by the Lemma C.1. So, by limiting the number of samples by k, we can replace the polynomial multiplication for a matrix by vector multiplication.

Definition 2.18. ***The* Ideal-LWE *Problem***: *Let n, k and q be integers, with n a power of 2, let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ be a ring and let χ be a probability distribution on $\mathcal{R}_q$. For $\boldsymbol{r} \in \mathbb{Z}_q^n$, let $A_{\boldsymbol{r},\chi}$ be the probability distribution on $\mathcal{R}_q^k \times \mathbb{Z}_q^{kn}$ obtained by choosing a vector $\hat{\boldsymbol{a}} \in \mathcal{R}_q^k$ uniformly at random, choosing $\hat{\boldsymbol{e}} \in \mathbb{Z}_q^{kn}$ according to χ, and outputting $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}} = \mathsf{Rot}_f(\hat{\boldsymbol{a}})^\top \boldsymbol{r} + \hat{\boldsymbol{e}})$. Given an arbitrary number of independent samples from the distribution $A_{\boldsymbol{r},\chi}$, find $\boldsymbol{r}$.*

Definition 2.19. ***The decision*-Ideal-LWE *Problem***: *Let n, k and q be integers, with n a power of 2, let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n + 1\rangle$ be a ring and let χ be a probability distribution on $\mathcal{R}_q$. For $\boldsymbol{r} \in \mathbb{Z}_q^n$, let $A_{\boldsymbol{r},\chi}$ be the probability distribution on $\mathcal{R}_q^k \times \mathbb{Z}_q^{kn}$ obtained by choosing a vector $\hat{\boldsymbol{a}} \in \mathcal{R}_q^k$ uniformly at random, choosing $\hat{\boldsymbol{e}} \in \mathbb{Z}_q^{kn}$ according to χ, and outputting $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{b}} = \mathsf{Rot}_f(\hat{\boldsymbol{a}})^\top \boldsymbol{r} + \hat{\boldsymbol{e}})$. Given an arbitrary number of independent samples from $\mathcal{R}_q^k \times \mathcal{R}_q^k$, return YES if the samples come from the distribution $A_{\boldsymbol{r},\chi}$ and NO if they come from the uniform distribution on $\mathcal{R}_q^k \times \mathbb{Z}_q^{kn}$.*

Since the multiplication of polynomials is replaced by a matrix-vector multiplication, the search-to-decision reduction used for the general LWE can be used for the Ideal-LWE: for each line of $\mathsf{Rot}_f(\hat{\boldsymbol{a}})^\top$ and each element of $\hat{\boldsymbol{e}}$ and $\hat{\boldsymbol{b}}$, just treat it like a general LWE.

Unlike the Ring-LWE Problem, the error here can be spherical, but it depends on the number of samples k. The reduction, although not to the decision problem, is also quantum and it is from an important problem called the Small Integer Solution (SIS), whose hardness is well know even for its ideal version [46].

Definition 2.20. ***The* SIS *Problem***: *Given an integer q, a matrix $A \in \mathbb{Z}^{n \times k}$ and a real number β, find a nonzero integer vector $\boldsymbol{e} \in \mathbb{Z}^k$ such that $A\boldsymbol{e} = 0 \mod q$ and $\|\boldsymbol{e}\| \leq \beta$. The* Ideal-SIS *Problem is exactly the same, but with $A = \mathsf{Rot}_f(\hat{\boldsymbol{a}})$, for $\hat{\boldsymbol{a}} \in (\mathbb{Z}_q[x]/\langle f(x)\rangle)^k$.*

Theorem 2.15 ([73])**.** *Let k, n, q be integers and $\alpha \in \{0, 1\}$, with $k \geq 41 \log q$ and $\alpha < \frac{1}{10\sqrt{\log(kn)}}$. If there exists an efficient (possibly quantum) algorithm that solves* $\mathsf{Ideal\text{-}LWE}_{q,n,\overline{\Psi}_\alpha}$*, then there exists an efficient (quantum) algorithm that solves* Ideal-SIS*, for $f(x) = x^n + 1$, with n a power of 2 and $q \equiv 3 \mod 8$.*

2.4 GGH

The GGH scheme [31], due to Goldreich, Goldwasser and Halevi, is the first and simplest lattice-based cryptosystem created. It uses a short "good" basis (which makes the CVP problem easy to solve) as private key and a "bad" basis (which makes the CVP problem hard to solve) as a public key. Its security was based on the CVP problem, but it was never formally proved. The scheme was later broken by Nguyen [55].

The GGH scheme consists of three algorithms: key generation KeyGen(), encryption Enc() and decryption Dec(). The KeyGen() algorithm chooses the lattice bases that will be used as keys. The Enc() algorithm uses the message as a vector to find a point in the lattice, then adds to it a short error vector in $\{-\sigma, \sigma\}^m$, with $\sigma << n$ so that the ciphertext is a point close to the lattice. The Dec() algorithm finds the original message using the short basis in the secret key with Babai's Nearest Plane Algorithm to solve the CVP Problem, i.e., find the closest point in the lattice. Since that point was calculated using the message as a vector, we only need to invert the basis to recover the message. Algorithms 2.12, 2.13 and 2.14 describe the GGH scheme.

Algorithm 2.12 GGH-KeyGen(): Key Generation Algorithm for the GGH Scheme.

```
Input: security parameter $n$
Output: public key $PK$ and secret key $SK$
    choose an orthogonal basis $B \in \mathbb{Z}^{n \times m}$ of lattice $\Lambda(B)$
    choose a unimodular matrix $U \in \mathbb{Z}^{n \times n}$
    $A = UB$
    secret key $SK = B$
    public key $PK = A$
```

Algorithm 2.13 GGH-Enc(): Encryption Algorithm for the GGH Scheme.

```
Input: message $M$ and public key $PK$
Output: ciphertext $CT$
    $M = \boldsymbol{m} \in \mathbb{Z}^n$
    $\boldsymbol{r} \xleftarrow{\$} \{-\sigma, \sigma\}^m$
    $\boldsymbol{c} \leftarrow \boldsymbol{m}A + \boldsymbol{r}$
    ciphertext $CT = \boldsymbol{c}$
```

Algorithm 2.14 GGH-Dec(): Decryption Algorithm for the GGH Scheme.

```
Input: public key $PK$ secret key $SK$ ciphertext $CT$
Output: message $M'$
    $\boldsymbol{v} \leftarrow \mathsf{Babai}(B, \boldsymbol{c})$
    $\boldsymbol{m'} = \boldsymbol{v}A^{-1}$
    $M' = \boldsymbol{m'}$
```

One important step of this algorithm is to change the short and orthogonal basis of Λ that is part of the secret key, into a random non-orthogonal matrix that is still a basis of Λ to compound the public key. In the original work many elementary matrices of different forms are multiplied to generate a random unimodular transformation U. Later, Micciancio [49] proposed to use the Hermite Normal Form.

Although it was based on the CVP Problem, the security of the GGH scheme was never formally presented. In 1999, Nguyen [55] broke the scheme showing two main flaws: the ciphertext leaks information of the message and it is possible to reduce the decrypting ciphertext problem to a special closest vector problem much easier than the general one. These problems can be avoided by increasing the security parameters, i.e., the size of the lattice. This, however, makes the GGH scheme impractical.

2.5 NTRU

The NTRU scheme [37] can be described as a ring-based or a lattice-based cryptosystem and its security is based on the CVP Problem. The name NTRU is a short for n-th degree truncated polynomial ring, i.e., the ring $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n - 1\rangle$. It is specified by three integer parameters: the degree n, the small modulus p and the big modulus q. The moduli q and p are coprime and the degree n is prime.

Let $\mathcal{R}_q = \mathbb{Z}_q[x]/\langle x^n - 1\rangle$ be a ring and let $\mathcal{R}_{d_1,d_2}$ be the set of all polynomials in $\mathcal{R}$ with d_1 coefficients equal to 1, d_2 coefficients equal to -1 and the rest equal to 0. Let $\boldsymbol{f} \in \mathcal{R}$, if $\boldsymbol{f}$ is invertible in $\mathcal{R}_q$, then there exists $\boldsymbol{f}_q^{-1}$ such that $\boldsymbol{f} \cdot \boldsymbol{f}_q^{-1} = 1 \bmod q$.

The secret key consists of two polynomials $\boldsymbol{f}$ and $\boldsymbol{g}$ in $\mathcal{R}_{d_1,d_2}$, with $\boldsymbol{f}$ invertible in $\mathcal{R}_q$ and $\mathcal{R}_p$. The public key is the resulting polynomial of the multiplication of the inverse of $\boldsymbol{f}$ and $\boldsymbol{g}$. The encryption of a message in $\mathcal{R}_p$ will be the polynomial in the public key multiplied by a random polynomial and added to the message. The decryption uses $\boldsymbol{f}$ and its inverse in $\mathcal{R}_p$ to recover the message. Algorithms 2.15, 2.16 and 2.17 describe the NTRU scheme.

Algorithm 2.15 NTRU-KeyGen(): Key Generation Algorithm for the NTRU Scheme.

```
Input:: security parameter $n$, public integers $p$ and $q$
Output:: public key $PK$ and secret key $SK$
    $\boldsymbol{f} \in \mathcal{R}_{d+1,d}$, invertible in $\mathcal{R}_p$ and $\mathcal{R}_q$
    $\boldsymbol{g} \in \mathcal{R}_{d,d}$
    $\boldsymbol{h} = \boldsymbol{f}_q^{-1} \cdot \boldsymbol{g}$
    secret key $SK = (\boldsymbol{f}, \boldsymbol{g})$
    public key $PK = \boldsymbol{h}$
```

Algorithm 2.16 NTRU-Enc(): Encryption Algorithm for the NTRU Scheme.

```
Input:: message $M$ and public key $PK$
Output:: ciphertext $CT$
    $M = \boldsymbol{m} \in \mathcal{R}_p$
    $r \xleftarrow{\$} \mathcal{R}_{d,d}$
    $\boldsymbol{c} \leftarrow p\boldsymbol{r} \cdot \boldsymbol{h} + \boldsymbol{m}$
    ciphertext $CT = \boldsymbol{c}$
```

Algorithm 2.17 NTRU-Dec(): Decryption Algorithm for the NTRU Scheme.

```
Input:: public key $PK$ secret key $SK$ ciphertext $CT$
Output:: message $M'$
    $\boldsymbol{a} \leftarrow \boldsymbol{f} \cdot \boldsymbol{c} \bmod q$
    $\boldsymbol{m'} \leftarrow \boldsymbol{f}_p^{-1} \cdot \boldsymbol{a} \bmod p$
    $M' = \boldsymbol{m'}$
```

The NTRU scheme can be described as a ring-based scheme, as seen above, or as a lattice-based scheme. For the lattice-based view we can see the two polynomials $\boldsymbol{f}$ and $\boldsymbol{g}$ as the generators for the lattice $\Lambda_q(F)$, with $F = [\mathsf{rot}_{x^n-1}(\boldsymbol{f})|\mathsf{rot}_{x^n-1}(\boldsymbol{g})]^\top$. The public key is a "hard" basis for this lattice, given by the Hermite Normal Form H that can be described using the polynomial $\boldsymbol{h}$:

$$H = \begin{pmatrix} \mathcal{I} & \mathcal{O} \\ \mathsf{rot}_{x^n-1}(\boldsymbol{h}) & \mathcal{I} \end{pmatrix}$$

The encryption, as in the GGH scheme, can be seen as a point close to a lattice point, but unlike GGH, the message is part of the error and not the lattice point. Decryption is possible because of the "good" basis $\mathsf{rot}_{x^n-1}(\boldsymbol{f})$. The use of ideal lattices makes it possible to decrease the key sizes, since to describe the lattice basis we only need a vector.

Like GGH, no proof of security of the NTRU scheme is known and all current attacks can be prevented by choosing the right parameters set [36]. As always, there is a trade-off between efficiency and security, but even with the large parameters, the NTRU scheme is still practical and one of the most important lattice-based public key cryptosystems. Other than the usual attacks, such as brute force, meet-in-the-middle and chosen-ciphertext attack, lattice reduction algorithms, such as the LLL can also be used to break the system.

In the same year of its publication, a company called NTRU Cryptosystems, Inc. [40] was founded by the developers and a patent on the cryptosystem was registered.

Chapter 3

Predicate Cryptography

In this chapter we give the basic definition of Predicate Encryption and details of the main schemes. In Section 3.1 we give basic concepts on predicate encryption, its security definitions and a description of the hierarchical encryption. In the rest of the chapter we describe the main predicate encryption schemes known: Identity-Based Encryption (IBE) in Section 3.2, Fuzzy Identity-Based Encryption (FBE) in Section 3.3, Attribute-Based Encryption (ABE) in Section 3.4, Identity-Based Encryption with Wildcards (WIBE) in Section 3.5, Inner Product Encryption (IPE) in Section 3.6 and Hidden Vector Encryption (HVE) in Section 3.7.

3.1 Basic Concepts

Functional encryption has become quite popular in the last few years because it provides users with a much finer control of decryption capabilities. More specifically, in a functional encryption system, secret keys allow users to learn functions of encrypted data.

The definition of *functional encryption* given by Boneh, Sahai and Waters [17] is that a user with a master key MK can generate a secret key SK using an attribute A that enables the computation of a function $f(\cdot, \cdot)$ on the encrypted data, i.e., for a plaintext M and an attribute A, the decryptor can compute $f(A, M)$ without learning anything else about the plaintext.

Predicate encryption [41] is a sub-class of functional encryption, in which the decryption is only possible if the function $f(A)$ is 1. Note that in a predicate encryption the function only depends on the attribute A, i.e., it does not use the plaintext M. A Predicate Encryption Scheme is defined by four algorithms:

- $\mathsf{Setup}(1^n)$: takes as input the security parameter n and outputs the public key PK and the master secret key MK.

- $\mathsf{KeyGen}(PK, MK, f)$: generates a secret key SK using the public key, the master key and the predicate f.
- $\mathsf{Enc}(PK, M, A)$: returns the ciphertext CT of message M using the public key and the attribute A.
- $\mathsf{Dec}(PK, SK, CT)$: decrypts the ciphertext CT into message M only if $f(A) = 1$.

In such schemes a trusted third party, called *Private Key Generator* (PKG), is needed to generate all the keys. Since the PKG has all the keys, it can encrypt and decrypt any message and, therefore, it can be more difficult to prove the integrity and origin of a message. Besides, if the PKG is compromised the whole system is compromised, so the PKG can be a good target for adversaries. Finally, a secure channel between each user and the PKG is needed for the transmission of secret keys.

3.1.1 Security

Predicate encryption schemes usually have their security defined through a game against an adversary:

1. The challenger $\mathcal{C}$ runs the Setup algorithm to generate the public key PK, which it gives to the adversary $\mathcal{A}$, and the master key, which it retains.
2. The adversary $\mathcal{A}$ is given oracle access to KeyGen.
3. The adversary $\mathcal{A}$ gives a pair of messages (M_0, M_1) and a pair of attributes (A_0, A_1) to the challenger $\mathcal{C}$. Then the challenger $\mathcal{C}$ chooses random bit $b \in \{0, 1\}$, encrypts M_b under A_b and sends the resulting ciphertext to the adversary $\mathcal{A}$.
4. The adversary $\mathcal{A}$ may continue to request keys for additional predicates.
5. The adversary $\mathcal{A}$ outputs a bit b', and succeeds if $b' = b$.

An encryption scheme is *indistinguishable under chosen-plaintext attack* (IND-CPA) if no adversary has a non-negligible advantage of winning the above game. If on item 2, the adversary $\mathcal{A}$ can run the decryption algorithm, besides the key generation algorithm, the scheme is *indistinguishable under chosen-ciphertext attack* (IND-CCA). Moreover, if the adversary can run the decryption algorithm on item 2 and on item 4, then the scheme is *indistinguishable under adaptive chosen-ciphertext attack* (IND-CCA2). There is also a weaker security notion, the *selective attribute* (sAT, also called sID if the attribute is an identity), in which the adversary must commit the challenge attributes at the beginning of the game, i.e., before item 1, instead of item 3. Finally, we have that a *attribute hiding*

(AH) scheme requires that the ciphertext conceal the attribute and the message, while *payload hiding* (PH) requires that the ciphertext conceal only the message. Note that all predicate encryption schemes must be attribute hiding, while not all functional encryption schemes must.

3.1.2 Hierarchical Encryption

For cryptosystems that use a trusted third party, as predicate schemes, it is convenient to have a hierarchy of certificate authorities, that is, the root certificate authority can issue certificates for other certificate authorities, which can issue certificates for users. For predicate schemes; the PKGs have to compute private keys only to the entities immediately below them, this scheme is called *hierarchical*, as introduced in [35]. In a hierarchical scheme, a user in level t can use his/her secret key to derive a secret key for a user at level $t + 1$.

Hanaoka et al. introduced the concept of hierarchical encryption [35] and Horwitz and Lynn [38] presented the first hierarchical predicate scheme, a hierarchical identity-based scheme (HIBE), but their scheme has only 2 levels. Most hierarchical schemes are from IBE schemes [30, 14, 15, 63, 68], but there are also hierarchical ABE schemes [44, 75, 76] and hierarchical IPE schemes [56, 57, 58].

Cash et. el. proposed the first lattice-based hierarchical scheme, on their seminal work about Bonsai Trees [22]. Most of lattice-based IBE schemes known so far were presented along with its hierarchical version [4, 72]. Besides IBE, a lattice-based IPE scheme [7] by Abdalla et al. [1] and a lattice-based HVE scheme by Mochetti et al. [53] also have hierarchical versions and are contributions of this work.

An *Hierarchical Scheme* is defined by the following tuple of four algorithms:

- $\mathsf{Setup}(1^n, 1^\mu)$: takes as input the security parameter n and hierarchical format μ and outputs the public key PK and the master secret key, i.e., the secret key for level 0, $SK_0 = MK$.

- $\mathsf{KeyDerive}(PK, SK_{t-1}, f_t)$: generates a secret key for level t, i.e., for the predicates $(f_1, \ldots, f_t)$, using: the public key, the secret key SK_{t-1} for level $t - 1$ and the predicate f_t for level t.

- $\mathsf{Enc}(PK, M, A)$: returns the ciphertext CT of message M using the public key and the attribute list $A = (a_1, \ldots, a_t)$.

- $\mathsf{Dec}(PK, SK_t, CT)$: decrypts the ciphertext CT into message M only if $f(A) = 1$.

3.2 Identity-Based Encryption (IBE)

In *Identity-Based Encryption Schemes* (IBE) the secret key is based on a user's unique information, such as his email address, and the decryption is only possible if the information used during the encryption is exactly the same as the one associated with the key. IBE schemes do not need a public key distribution infrastructure, since the authenticity is guaranteed implicitly.

IBE schemes were proposed by Shamir [70] in 1984, but remained as an open problem until 2001 when Boneh and Franklin [16] and Cocks [24] constructed the first schemes. Boneh and Franklin's identity-based encryption scheme is based on bilinear pairings, while Cock's IBE scheme's security is based on the quadratic residuousity problem.

Both schemes have their security based on the random oracle model. Other schemes are proven secure in the standard model, but under selective security, a weaker notion of security [20, 14] or are proven adaptively secure [77, 28].

The first lattice-based IBE scheme was proposed by Gentry et al. [29] and its security is based on the LWE problem in the random oracle model. Another lattice-based IBE scheme, but with hierarchical property, was proposed by Agrawal et al. [4] and it is based on the Bonsai Trees concept. This scheme's security does not use random oracles, but it is also based on the LWE problem.

An IBE Scheme is defined by the following tuple of four algorithms:

- $\mathsf{Setup}(1^n)$: takes as input the security parameter n and outputs the public key PK and the master secret key MK.
- $\mathsf{KeyGen}(PK, MK, \boldsymbol{id})$: generates a secret key SK using the public key, the master key and the user's identity $\boldsymbol{id}$.
- $\mathsf{Enc}(PK, M, \boldsymbol{id})$: returns the ciphertext CT of message M using the public key and the user's identity $\boldsymbol{id}$.
- $\mathsf{Dec}(PK, SK, CT)$: decrypts the ciphertext CT into message M only if the identity used during the key generation is the same use in th encryption.

To ensure the correctness of the scheme the following must be true:

$$\forall M, \boldsymbol{id} : \mathsf{Dec}(PK, \mathsf{KeyGen}(PK, MK, \boldsymbol{id}), \mathsf{Enc}(PK, M, \boldsymbol{id})) = M.$$

3.3 Fuzzy Identity-Based Encryption (FBE)

A *Fuzzy Identity-Based Encryption Scheme* (FBE) uses a list of attributes $\boldsymbol{\omega}$ associated with the secret key and a list of attributes $\boldsymbol{\omega}'$ associated with the encryption, and it is only possible to decrypt if the lists have at least k elements in common. Since a message can be encrypted defining the attributes of the users allowed to decrypt it, FBE schemes are often used on broadcast encryption to decrease the number of keys used and in biometric systems.

The first scheme was proposed by Sahai and Waters [66] and it uses the selective model to prove the security of the scheme. The Sahai and Waters' scheme uses Shamir's method of secret sharing [69] and a group for which an efficient bilinear map exists, but for which the Computational Diffie-Hellman Problem is assumed to be hard. Baek et al. improved the first scheme's efficiency by employing random oracles [11] and Ren et al. presented the first fully secure scheme in the standard model [64].

The only lattice-based FBE scheme known was proposed by Agrawal et al. [6]. It is similar to the previous IBE scheme [4] and it is secure against selective-identity attacks in the standard model, based on the hardness of the LWE problem.

A FBE scheme is defined by the following tuple of four algorithms:

- $\mathsf{Setup}(1^n)$: takes as input the security parameter n and outputs the public key PK and the master secret key MK.
- $\mathsf{KeyGen}(PK, MK, \boldsymbol{\omega})$: generates a secret key SK using the public key, the master key and the user's attributes $\boldsymbol{\omega}$.
- $\mathsf{Enc}(PK, M, \boldsymbol{\omega}')$: returns the ciphertext CT of message M using the public key and the user's attributes $\boldsymbol{\omega}'$.
- $\mathsf{Dec}(PK, SK, CT)$: decrypts the ciphertext CT into message M only if $|\boldsymbol{\omega} \cap \boldsymbol{\omega}'| \geq k$.

To ensure the correctness of the scheme the following must be true:

$$\forall M, \boldsymbol{\omega}, \boldsymbol{\omega}', \text{ with } |\boldsymbol{\omega} \cap \boldsymbol{\omega}'| \geq k : \mathsf{Dec}(PK, \mathsf{KeyGen}(PK, MK, \boldsymbol{\omega}, k), \mathsf{Enc}(PK, M, \boldsymbol{\omega}')) = M.$$

3.4 Attribute-Based Encryption (ABE)

In *Attribute-Based Encryption Schemes* (ABE), the secret key is based on the user's attributes $\boldsymbol{\omega}$ and the decryption is only possible if the attributes $\boldsymbol{\omega}'$ used during the encryption are close enough to $\boldsymbol{\omega}$, i.e. if $\boldsymbol{\omega}$ and $\boldsymbol{\omega}'$ are within a certain distance of each other as judged by some metric.

It is important to notice the difference between IBE, ABE and FBE schemes. In an IBE scheme, the user's information associated with the key and the encryption are unique and must be exactly the same for the decryption to occur properly. In ABE and FBE schemes the secret key and the encryption are associated with a list of attributes that do not need to match exactly for the decryption to occur properly. FBE schemes are a particular type of ABE, where the metric used to compare each list of attributes is only the number of attributes that are present in both lists, while in ABE the metric can be more general, i.e., it can be an arbitrary boolean formula.

The initial approach to ABE schemes was made by Sahai and Waters [66] with the FBE scheme, but the first ABE scheme, i.e., the first scheme in which the metric used was a boolean formula, was proposed by Goyal et al. [34], also based on secret sharing and bilinear maps. Recently, Waters [78] improved the ABE schemes to support functionality for regular languages. This scheme is also based on bilinear maps and is secure in the selective model under the Bilinear Diffie-Hellman Exponent assumption.

Zhang et al. proposed the first lattice-based ABE scheme [81]. It uses Shamir secret sharing and is secure against chosen plaintext attack in the selective model under the Learning With Errors (LWE) assumption. Later, Boyen [19] used a basis splicing technique to propose a lattice-based ABE scheme that has its security in the selective sense reduced to the LWE problem in the standard model. Two similar and concurrent works [27, 33] are based on multilinear maps, but have a translation to lattices and can have security proofs reduced to the LWE Problem.

An ABE Scheme is defined by the following tuple of four algorithms:

- $\mathsf{Setup}(1^n)$: takes as input the security parameter n and outputs the public key PK and the master secret key MK.
- $\mathsf{KeyGen}(PK, MK, \boldsymbol{\omega})$: generates a secret key SK using the public key, the master key and the user's attributes $\boldsymbol{\omega}$.
- $\mathsf{Enc}(PK, M, \boldsymbol{\omega}')$: returns the ciphertext CT of message M using the public key and the user's attributes $\boldsymbol{\omega}'$.
- $\mathsf{Dec}(PK, SK, CT)$: decrypts the ciphertext CT into message M only if $\boldsymbol{\omega} \sim \boldsymbol{\omega}'$.

To ensure the correctness of the scheme the following must be true:

$$\forall M, \boldsymbol{\omega}, \boldsymbol{\omega}',\ \text{with } \boldsymbol{\omega} \sim \boldsymbol{\omega}' : \mathsf{Dec}(PK, \mathsf{KeyGen}(PK, MK, \boldsymbol{\omega}), \mathsf{Enc}(PK, M, \boldsymbol{\omega}')) = M.$$

3.5 Identity-Based Encryption with Wildcards (WIBE)

Identity-Based Encryption with Wildcards (WIBE) is a generalisation of HIBE schemes, in which one can encrypt messages to a range of users simultaneously, whose identities match a certain pattern. A pattern $\boldsymbol{p} = \{p_0, \cdots, p_n\}$ is a set containing identities or "don't care" elements (represented by $\star$). For an identity $\boldsymbol{id} = \{id_0, \cdots, id_n\}$, the encryption is possible only if $id_i = p_i, \forall i$ such that $p_i \neq \star$.

Abdalla et al. [2] proposed the three first WIBE schemes based on the following HIBE schemes: Water's HIBE scheme [77], Boneh-Boyen's HIBE scheme [14] and Boneh-Boyen-Goh's HIBE scheme [15].

Later, Abdalla et al. [3] explored the relations between HIBE schemes and WIBE schemes. They presented a generic transformation from any selective-secure WIBE to a fully-secure HIBE, depending only on the notion of admissible hash functions; and identified the conditions under which it is possible to get a generic transformation from a selective-secure HIBE scheme into a selective-secure WIBE scheme.

They also proposed the first lattice-based WIBE scheme using the constructions and HIBE scheme presented by Cash et al. [23]. The HIPE scheme (see Section 3.6) presented by Abdalla, De Caro and Mochetti [1] was also converted into a lattice-based WIBE scheme. Both schemes are selective-secure based on the Learning With Errors Problem (LWE).

A WIBE Scheme is defined by the following tuple of four algorithms:

- $\mathsf{Setup}(1^n)$: takes as input the security parameter n and outputs the public key PK and the master secret key MK.
- $\mathsf{KeyGen}(PK, MK, \boldsymbol{id})$: generates a secret key SK using the public key, the master key and the user's identity $\boldsymbol{id}$.
- $\mathsf{Enc}(PK, M, \boldsymbol{p})$: returns the ciphertext CT of message M using the public key and the pattern $\boldsymbol{p}$.
- $\mathsf{Dec}(PK, SK, CT)$: decrypts the ciphertext CT into message M only if $id_i = p_i, \forall i$, such that $p_i \neq \star$.

To ensure the correctness of the scheme the following must be true:

$$\forall M, \boldsymbol{id}, \boldsymbol{p}, \text{ with } id_i = p_i, \forall i \text{ where } p_i \neq \star :$$

$$\mathsf{Dec}(PK, \mathsf{KeyGen}(PK, MK, \boldsymbol{id}), \mathsf{Enc}(PK, M, \boldsymbol{p})) = M.$$

3.6 Inner Product Encryption (IPE)

Inner-Product Encryption Schemes (IPE) associate the ciphertext with a vector $\boldsymbol{w}$ and the secret key with a vector $\boldsymbol{v}$ and the decryption is only possible if the vector $\boldsymbol{v}$ and $\boldsymbol{w}$ are orthogonal to each other, i.e., if their inner product is zero. IPE schemes can be used to support conjunction, subset and range queries on encrypted data as well as disjunctions, polynomial evaluation, and CNF and DNF formulas.

IPE schemes were introduced by Katz, Sahai, and Waters [41] as a generalization of IBE schemes. Since its introduction, there has been an extensive amount of work on the construction of IPE schemes, most of them based on bilinear groups. The first IPE scheme proposed by Katz et al. was based on bilinear groups and has its security proved in the selective model. Later on, Okamoto et al. [56] were able to move to prime order bilinear groups, but maintaining the security in the selective model.

In the lattice-based setting, there is only one know construction, by Agrawal et al. [7], and its hierarchical version by Abdalla et al. [1]. Both are shown to be weak selective secure based on the difficulty of the learning with errors (LWE) problem. Later, a more efficient version of this scheme was showed by Xagawa [79].

An IPE Scheme is defined by the following tuple of four algorithms:

- $\mathsf{Setup}(1^n)$: takes as input the security parameter n and outputs the public key PK and the master secret key MK.
- $\mathsf{KeyGen}(PK, MK, \boldsymbol{v})$: generates a secret key SK using the public key, the master key and the vector $\boldsymbol{v}$.
- $\mathsf{Enc}(PK, M, \boldsymbol{w})$: returns the ciphertext CT of message M using the public key and the vector $\boldsymbol{w}$.
- $\mathsf{Dec}(PK, SK, CT)$: decrypts the ciphertext CT into message M only if $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = 0$.

To ensure the correctness of the scheme the following must be true:

$$\forall M, \boldsymbol{v}, \boldsymbol{w}, \text{ with } \langle \boldsymbol{v}, \boldsymbol{w} \rangle = 0 : \mathsf{Dec}(PK, \mathsf{KeyGen}(PK, MK, \boldsymbol{v}), \mathsf{Enc}(PK, M, \boldsymbol{w})) = M.$$

3.7 Hidden Vector Encryption (HVE)

Hidden Vector Encryption Schemes (HVE) associate the ciphertext with a binary vector $\boldsymbol{w}$ and the secret key with a binary vector $\boldsymbol{v}$, which can have special entries called "don't care" represented by $\star$. The decryption is only possible if the vector $\boldsymbol{v}$ and $\boldsymbol{w}$ are exactly the same, except for the "don't care" elements on vector $\boldsymbol{v}$, i.e., $v_i = w_i, \forall i$ such that $v_i \neq \star$. HVE schemes are usually used on conjunctive, comparison and range queries.

HVE schemes were proposed by Boneh, and Waters [18] as a generalization of IBE schemes. This scheme was based on bilinear groups and proved secure under the selective model. Other schemes, also based on bilinear groups, were also developed: Iovino et al. [39] scheme used bilinear groups of prime order while De Caro et al. [21] scheme is the first fully secure construction known.

HVE schemes were improved by Sedghi et al. [67] to create a cryptographic scheme that can search keywords with wildcards on encrypted data. This scheme is based on bilinear groups of prime order, supports vectors over any alphabet, not only binary vectors, and it is proved secure in a selective model, under the decision linear assumption.

Any IPE scheme can be used to build an HVE scheme, as showed by Boneh, and Waters [18]; therefore the lattice-based IPE scheme of Agrawal et al. [7] and its hierarchical version by Abdalla et al. [1] can be enhanced to create a lattice-based HVE scheme. In both schemes, the resultant HVE scheme will be weak selective secure based on the difficulty of the Learning With Errors Problem (LWE). Another lattice-based HVE scheme can be achieved by simple changes in the FBE scheme proposed by Agrawal et al. [6], as described in details by Mochetti et al. [53].

An HVE Scheme is defined by the following tuple of four algorithms:

- $\mathsf{Setup}(1^n)$: takes as input the security parameter n and outputs the public key PK and the master secret key MK.

- $\mathsf{KeyGen}(PK, MK, \boldsymbol{v})$: generates a secret key SK using the public key, the master key and the vector $\boldsymbol{v}$.

- $\mathsf{Enc}(PK, M, \boldsymbol{w})$: returns the ciphertext CT of message M using the public key and the vector $\boldsymbol{w}$.

- $\mathsf{Dec}(PK, SK, CT)$: decrypts the ciphertext CT into message M only if $v_i = w_i, \forall i$ such that $v_i \neq \star$.

To ensure the correctness of the scheme the following must be true:

$$\forall M, \boldsymbol{v}, \boldsymbol{w}, \text{ with } v_i = w_i, \forall i \text{ where } v_i \neq \star :$$

$$\mathsf{Dec}(PK, \mathsf{KeyGen}(PK, MK, \boldsymbol{v}), \mathsf{Enc}(PK, M, \boldsymbol{w})) = M.$$

3.7.1 HVE Scheme from an IPE Scheme

It is possible to build an HVE scheme using an IPE scheme [41]. In this section we show this construction. Given $\boldsymbol{w} \in \{0,1\}^n$ and $\boldsymbol{v} \in \{0,1,\star\}^n$, we want a scheme that can be decrypted only if $v_i = w_i, \forall i$ such that $v_i \neq \star$. We can build two vectors $\boldsymbol{a} \in \{0,1,\star\}^{2n}$ and $\boldsymbol{b} \in \{0,1\}^{2n}$ such that:

$$v_i = w_i, \forall i \text{ such that } v_i \neq \star \Rightarrow \langle \boldsymbol{a}, \boldsymbol{b} \rangle = 0 \tag{3.1}$$

and

$$\exists i, v_i \neq w_i \text{ and } v_i \neq \star \Rightarrow Pr[\langle \boldsymbol{a}, \boldsymbol{b} \rangle = 0] \text{ is negligible} \tag{3.2}$$

The vector $\boldsymbol{a}$, used during the **KeyGen** algorithm is built from the vector $\boldsymbol{v}$ as follows:

if $v_i \neq \star : a_{2i-1} \leftarrow 1$ and $a_{2i} \leftarrow v_i$
if $v_i = \star : a_{2i-1} \leftarrow 0$ and $a_{2i} \leftarrow 0$

Given a random vector $\boldsymbol{r} \in \mathbb{Z}_N^{2n}$, the vector $\boldsymbol{b}$, used during the Enc algorithm is built from the vector $\boldsymbol{w}$ as follows:

$b_{2i-i} \leftarrow -r_i w_i$ and $b_{2i} \leftarrow r_i$

Now we have to prove that equations 3.1 and 3.2 are valid for our construction of vectors $\boldsymbol{a}$ and $\boldsymbol{b}$. First notice that:

$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \sum_{j=1}^{2n} a_j b_j = \sum_{i=1}^{n} a_{2i-1} b_{2i-1} + a_{2i} b_{2i}$$

We have to analyse the sum term $a_{2i-1}b_{2i-1} + a_{2i}b_{2i}$. There are three main cases:

- $v_i = \star$

  We have that the terms a_{2i-1} and a_{2i} are 0, and therefore the whole sum term $a_{2i-1}b_{2i-1} + a_{2i}b_{2i}$ is also 0, for any values of b_{2i-1} and b_{2i}.

- $v_i = w_i$ and $v_i \neq \star$

  We have that the term a_{2i-1} is 1 and the term a_{2i} is v_i. Beside that, we have that the term b_{2i-1} is $-r_i w_i$ and b_{2i} is r_i. Remember that $v_i = w_i$, so by replacing the values on the sum term $a_{2i-1}b_{2i-1} + a_{2i}b_{2i}$ we have:

  $a_{2i-1}b_{2i-1} + a_{2i}b_{2i} = 1(-r_i w_i) + v_i r_i = r_i(v_i - w_i) = 0$

- $v_i \neq w_i$ and $v_i \neq \star$

  As before, we have that the term a_{2i-1} is 1 and the term a_{2i} is v_i. Beside that, we have that the term b_{2i-1} is $-r_i w_i$ and b_{2i} is r_i. Now $v_i \neq w_i$, so replacing the values on the sum term $a_{2i-1}b_{2i-1} + a_{2i}b_{2i}$ will give us:

  $a_{2i-1}b_{2i-1} + a_{2i}b_{2i} = 1(-r_i w_i) + v_i r_i = r_i(v_i - w_i)$

For equation 3.1, we only have terms on two first items; therefore, all the terms in the sum are 0 and $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = 0$.

For equation 3.2, we can have terms from all the items above; all of them are 0 except the third item, which will be $r_i(v_i - w_i)$. In this case we have to analyse $\sum r_i(v_i - w_i), \forall i$, where $v_i \neq w_i$. Assuming $\gcd(v_i - w_i, N) = 1$ we have that the probability of this sum be 0 is $1/N$. For N big enough, this probability is negligible [41].

Therefore, if $v_i = w_i, \forall i$ such that $v_i \neq \star$ then we have $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = 0$ and if $\exists i, v_i \neq w_i$ such that $v_i \neq \star$ then the probability of $\langle \boldsymbol{a}, \boldsymbol{b} \rangle = 0$ is negligible.

Chapter 4

Identity-Based Encryption

In this chapter we describe the lattice-based IBE scheme proposed by Agrawal, Boneh and Boyen [4]. Section 4.1 reviews the general scheme and Section 4.2 gives the hierarchical version, both described in the same work.

4.1 Lattice-Based Identity-Based Encryption

This Section reviews the IBE scheme proposed by Agrawal, Boneh and Boyen [4]. Section 4.1.1 describes the four algorithms that comprise the IBE scheme, Sections 4.1.2, 4.1.3, 4.1.4 and 4.1.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

4.1.1 Description

As described in Section 3.2, an Identity-Based Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n)$, $\mathsf{KeyGen}(PK, MK, \boldsymbol{id})$, $\mathsf{Enc}(PK, M, \boldsymbol{id})$ and $\mathsf{Dec}(PK, SK, CT)$. In this section we describe each algorithm as presented by Agrawal et al. [4].

IBE-SetUp creates a general lattice and chooses at random the matrices and vector that will form the public and master keys. IBE-KeyGen generates the secret key by encoding the identity $\boldsymbol{id}$ into a matrix using the $\mathsf{rot}_f()$ function and concatenating it to the lattice basis. The secret key is a vector $\boldsymbol{e}$ created by the SampleLeft algorithm (described in Section 2.2), using the matrix concatenation as the lattice basis; therefore $\boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(A_{\boldsymbol{id}})$.

IBE-Enc uses the message M, the identity $\boldsymbol{id}$ and the matrices in the public key to create an integer c' and vectors $\boldsymbol{c}_0$ and $\boldsymbol{c}_1$ that will compose the ciphertext for one bit. Finally, IBE-Dec can recover the message from the ciphertext only if the identity used during the key generation is the same as the one used during the encryption.

Let n be the security parameter and σ be the Gaussian parameter. Algorithms 4.1, 4.2, 4.3 and 4.4 describe the IBE scheme.

Algorithm 4.1 IBE-SetUp(): Setup Algorithm for the IBE Scheme

```
Input: security parameter $1^n$
Output: Public key $PK$ and master key $MK$
    $A, S \leftarrow \mathsf{TrapGen}(q, n, m)$
    $A_0, B \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$
    $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$
    public key $PK = (A, A_0, B, \boldsymbol{u})$
    master key $MK = S$
```

Algorithm 4.2 IBE-KeyGen(): Key Generation Algorithm for the IBE Scheme

```
Input: Public key $PK$, master key $MK$ and identity $\boldsymbol{id}$
Output: Secret key $SK$
    $C = A_0 + \mathsf{rot}_f(\boldsymbol{id})B$
    $A_{\boldsymbol{id}} = [A|C] \in \mathbb{Z}^{n \times 2m}$
    $\boldsymbol{e} \leftarrow \mathsf{SampleLeft}(A, C, S, \boldsymbol{u}, \sigma)$
    secret key $SK = \boldsymbol{e} \in \mathbb{Z}_q^{2m}$
```

Algorithm 4.3 IBE-Enc(): Encryption Algorithm for the IBE Scheme

```
Input: Public key $PK$, message $M$ and identity $\boldsymbol{id}$
Output: Ciphertext $CT$
    $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$
    $\boldsymbol{x} \in \overline{\Psi}_\alpha^m$ and $x \in \overline{\Psi}_\alpha$
    $R \xleftarrow{\$} \{-1, 1\}^{m \times m}$
    $C = A_0 + \mathsf{rot}_f(\boldsymbol{id})B$
    $\boldsymbol{c}_0 = A^\top \boldsymbol{s} + \boldsymbol{x} \in \mathbb{Z}_q^m$
    $\boldsymbol{c}_1 = C^\top \boldsymbol{s} + R^\top \boldsymbol{x} \in \mathbb{Z}_q^m$
    $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\boldsymbol{c}_0, \boldsymbol{c}_1, c')$
```

Algorithm 4.4 IBE-Dec(): Decryption Algorithm for the IBE Scheme

```
Input: Public key $PK$, secret key $SK$ and ciphertext $CT$
Output: message $M$
    $z = c' - \boldsymbol{e}^\top \left[\begin{smallmatrix}\boldsymbol{c}_0\\ \boldsymbol{c}_1\end{smallmatrix}\right] \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

4.1.2 Correctness

The correctness is straightforward. First we just substitute the values of $\boldsymbol{c}_0$, $\boldsymbol{c}_1$ and c' in z. If the identity used during the key generation is the same as the one used during the encryption, then $A_{\boldsymbol{id}}$ is $[A|C]$ and, therefore, $\boldsymbol{u}^\top = \boldsymbol{e}^\top[A|C]^\top$. Finally, we cancel the terms $\boldsymbol{u}^\top\boldsymbol{s}$, identify all terms that refer to the "noise" and get the right value of M in z.

$$\begin{aligned}
z &= c' - \boldsymbol{e}^\top \begin{bmatrix}\boldsymbol{c}_0\\ \boldsymbol{c}_1\end{bmatrix} \mod q\\
&= c' - \boldsymbol{e}^\top \begin{bmatrix}A^\top\boldsymbol{s}+\boldsymbol{x}\\ C^\top\boldsymbol{s}+R^\top\boldsymbol{x}\end{bmatrix} \mod q\\
&= c' - \boldsymbol{e}^\top \begin{bmatrix}A\\ C\end{bmatrix}^\top \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix}\boldsymbol{x}\\ R^\top\boldsymbol{x}\end{bmatrix} \mod q\\
&= \boldsymbol{u}^\top\boldsymbol{s} + x + M\lfloor q/2\rfloor - \boldsymbol{u}^\top\boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix}\boldsymbol{x}\\ R^\top\boldsymbol{x}\end{bmatrix} \mod q\\
&= x + M\lfloor q/2\rfloor - \boldsymbol{e}^\top \begin{bmatrix}\boldsymbol{x}\\ R^\top\boldsymbol{x}\end{bmatrix} \mod q\\
&= M\lfloor q/2\rfloor + err \mod q
\end{aligned}$$

Note that for the correct decryption the error term must be less than $q/4$.

4.1.3 Security

In this section we prove the following theorem.

Theorem 4.1. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 4.1.1 is IND-AH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.IBE-SetUp$(1^n, \boldsymbol{id}^\star)$: The algorithm chooses random matrices $A \in \mathbb{Z}_q^{n\times m}$ and $R^\star \in \{-1,1\}^{m\times m}$ and vector $\boldsymbol{u} \in \mathbb{Z}_q^n$ and it uses TrapGen(q,n,m) to generate $B^\star \in \mathbb{Z}_q^{n\times m}$

and the basis $S^\star \in \mathbb{Z}^{m\times m}$ for $\Lambda_q^\perp(B^\star)$. It then defines $A_0 \leftarrow AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star$ and outputs $PK = (A, A_0, \boldsymbol{u})$ and $MK = (R^\star, B^\star, S^\star)$.

$\mathsf{Sim.IBE\text{-}KeyGen}$ $(PK, MK, \boldsymbol{id})$: Secret keys are now sampled by the $\mathsf{SampleRight}$ algorithm, using the trapdoor $S^\star$. It outputs
$SK = \boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(A|AR^\star + (\mathsf{rot}_f(\boldsymbol{id}) - \mathsf{rot}_f(\boldsymbol{id}^\star))B^\star)$, where
$\boldsymbol{e} \leftarrow \mathsf{SampleRight}(A, (\mathsf{rot}_f(\boldsymbol{id}) - \mathsf{rot}_f(\boldsymbol{id}^\star))B^\star, R^\star, S^\star, \boldsymbol{u}, \sigma)$.
Note that we must have $\boldsymbol{id} \neq \boldsymbol{id}^\star$ for the algorithm $\mathsf{SampleRight}$ to work properly.

$\mathsf{Sim.IBE\text{-}Enc}$ $(PK, M, \boldsymbol{id}^\star)$: The algorithm differs from $\mathsf{IBE\text{-}Enc}$ in the sense that it uses matrices $R^\star$ and $B^\star$ instead of matrices R and B.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs $\mathsf{IBE\text{-}SetUp}$, answers $\mathcal{A}$'s secret key queries using the $\mathsf{IBE\text{-}KeyGen}$ algorithm, and generates the challenge ciphertext using the $\mathsf{IBE\text{-}Enc}$ with identity $\boldsymbol{id}^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs $\mathsf{Sim.IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.IBE\text{-}KeyGen}$, and generates the challenge ciphertext using the $\mathsf{Sim.IBE\text{-}Enc}$ algorithm with $\boldsymbol{id}^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs $\mathsf{Sim.IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.IBE\text{-}KeyGen}$, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs $\mathsf{Sim.IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.IBE\text{-}KeyGen}$, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs $\mathsf{Sim.IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.IBE\text{-}KeyGen}$, and generates the challenge ciphertext using the $\mathsf{Sim.IBE\text{-}Enc}$ algorithm with $\boldsymbol{id}^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs $\mathsf{IBE\text{-}SetUp}$, answers $\mathcal{A}$'s secret key queries using the $\mathsf{IBE\text{-}KeyGen}$ algorithm, and generates the challenge ciphertext using the $\mathsf{IBE\text{-}Enc}$ with identity $\boldsymbol{id}^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 4.1. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, matrix A is generated by $\mathsf{TrapGen}$ and matrix A_0 is uniformly random in $\mathbb{Z}_q^{n\times m}$. Instead, in Game 1, A is chosen uniformly at random and we have $A_0 \leftarrow AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star$, where $B^\star$ is generated by $\mathsf{TrapGen}$ and the matrix $R^\star$ is uniformly and independently chosen at random in $\{-1,1\}^{m\times m}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathbb{Z}_q^n$.

Secret keys In Game 0, the secret key for identity $\boldsymbol{id}$ is a vector $\boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(A_{\boldsymbol{id}})$, sampled using the $\mathsf{SampleLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext In both games the challenge ciphertext components c' and $\boldsymbol{c}_0$ are computed the same way but, in Game 0, the challenge ciphertext component $\boldsymbol{c}_1$ is computed as follows:

$$\boldsymbol{c}_1 = (A_0 + \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x} \in \mathbb{Z}_q^m \ .$$

On the other hand, in Game 1, we have:

$$\begin{aligned}\boldsymbol{c}_1 &= (A_0 + \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x} \\ &= (AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star + \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x} \ . \\ &= (AR)^{\star\top}\boldsymbol{s} + R^{\star\top}\boldsymbol{x} \in \mathbb{Z}_q^m\end{aligned}$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(A, A_0, \boldsymbol{c}_1)$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemmas A.4 and A.5 we have that the following two distributions are statistically indistinguishable for every fixed matrix $B^\star$, every $\boldsymbol{id}^\star$ and every vector $\boldsymbol{x} \in \mathbb{Z}_q^m$:

$$\left(A, A_0, R^{\star\top}\boldsymbol{x}\right) \approx_s \left(A, AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star, R^{\star\top}\boldsymbol{x}\right) \ .$$

Since $(AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star)^\top\boldsymbol{s}$ is statistically close to $A_0^\top\boldsymbol{s}$, it is possible to add each term to each side of the equation:

$$\begin{gathered}\left(A, A_0, A_0^\top\boldsymbol{s} + R^{\star\top}\boldsymbol{x}\right) \approx_s \\ \left(A, AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star, (AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star)^\top\boldsymbol{s} + R^{\star\top}\boldsymbol{x}\right)\end{gathered} \ .$$

Then, we add $(\mathsf{rot}_f(\boldsymbol{id}^\star)B^\star)^\top\boldsymbol{s}$ to each side of the equation:

$$\begin{gathered}\left(A, A_0, (A_0 + \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star)^\top\boldsymbol{s} + R^{\star\top}\boldsymbol{x}\right) \approx_s \\ \left(A, AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star, (AR^\star)^\top\boldsymbol{s} + R^{\star\top}\boldsymbol{x}\right)\end{gathered} \ .$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 4.2. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE.

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $m+1$ LWE challenge pairs $(\boldsymbol{a}_j, y_j) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where either $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s} \rangle + x_j$ for a random $\boldsymbol{s} \in \mathbb{Z}_q^n$ and a noise term $x_j \leftarrow \Psi_\alpha$, or y_j is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_j, y_j)$. The i-th column of matrix A will be the vector $\boldsymbol{a}_i$, for $1 \leq i \leq m$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrix A_0 is still calculated as in $\mathsf{Sim.IBE\text{-}SetUp}$, i.e., $A_0 \leftarrow AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star$.

Secret keys All private-key extraction queries are answered using $\mathsf{Sim.IBE\text{-}KeyGen}$.

Challenge Ciphertext The ciphertext $CT = (\boldsymbol{c}_0^\star, \boldsymbol{c}_1^\star, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_j, y_j)$, with $\boldsymbol{c}_0^\star = (y_1, \ldots, y_m)$, $\boldsymbol{c}_1^\star = R^{\star\top}\boldsymbol{c}_0^\star$ and $c'^\star = y_0 + M\lfloor q/2 \rceil$. If we have $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s} \rangle + x_j$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_j is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2 . If $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s} \rangle + x_j$, then

$(y_1, \ldots, y_m) = (\langle \boldsymbol{a}_1, \boldsymbol{s} \rangle + x_1, \ldots, \langle \boldsymbol{a}_m, \boldsymbol{s} \rangle + x_m) = A^\top \boldsymbol{s} + \boldsymbol{x}$.

Therefore, for Game 1 we have

$$\begin{aligned} \boldsymbol{c}_0^\star &= A^\top \boldsymbol{s} + \boldsymbol{x} \\ &= (y_1, \ldots, y_m) \ , \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{c}_1^\star &= C^\top \boldsymbol{s} + R^\top \boldsymbol{x} \\ &= (A_0 + \mathsf{rot}_f(\boldsymbol{id})B)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x} \\ &= (AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star + \mathsf{rot}_f(\boldsymbol{id})B)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x} \\ &= (AR^\star)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x} \\ &= R^{\star\top}(A^\top \boldsymbol{s} + \boldsymbol{x}) \\ &= R^{\star\top}\boldsymbol{c}_0^\star \ . \end{aligned}$$

If y_j is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_j is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s} \rangle + x_j$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 4.3. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. A and $\boldsymbol{u}$ are random and $A_0 \leftarrow AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star$, with $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 0}$ for Game 2 and $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 1}$ for Game 3.

Secret keys All private-key extraction queries are answered using Sim.IBE-KeyGen. The only difference is, again, that for Game 2, $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 0}$ and, for Game 3, $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrix A_0. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of A_0. From Lemmas A.4 and A.5 we can prove that A_0 for each game is statistically close to a uniformly random matrix, because

$A_0 \leftarrow AR^\star - \mathsf{rot}_f(\boldsymbol{id}^\star)B^\star$.

□

4.1.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma\sqrt{2m}$ from Lemma A.1 and $\|R\boldsymbol{e}\| \leq 12\sqrt{2m}\|\boldsymbol{e}\|$ from Lemma A.2; therefore, for $\boldsymbol{e} = \begin{bmatrix} \boldsymbol{e}_1 \\ \boldsymbol{e}_2 \end{bmatrix}$:

$$\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq (\sigma\sqrt{2m} + 12\sqrt{2m}\sigma\sqrt{2m}) \qquad \text{so}$$
$$\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq O(\sigma m) \ .$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x} \rangle \leq \|\boldsymbol{y}\| q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x} \rangle \leq O(\sigma m) q\alpha w(\sqrt{\log 2m}) + O(\sigma m)\sqrt{2m}/2$$
$$\langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x} \rangle \leq \widetilde{O}(\sigma m q\alpha) + O(\sigma m^{3/2}) \ .$$

To ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned}
&x - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} < q/4 \\
&x - \boldsymbol{e}_1^\top \boldsymbol{x} - \boldsymbol{e}_2^\top R^\top \boldsymbol{x} < q/4 \\
&x - (\boldsymbol{e}_1 + R\boldsymbol{e}_2)^\top \boldsymbol{x} < q/4 \\
&x - \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x} \rangle < q/4 \\
&\widetilde{O}(\sigma m q \alpha) + O(\sigma m^{3/2}) < q/4
\end{aligned}$$

To ensure that σ is sufficiently large for SampleLeft and SampleRight (Theorems 2.7 and 2.8), we have

$$\sigma > \|S\| \sqrt{2m} \omega(\sqrt{\log 2m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned}
&m \geq 6n \log q \qquad \text{and} \\
&\|S\| \leq O(n \log q) \ .
\end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha \ .$$

Therefore, we need to set the parameters as

$$\begin{aligned}
&m = 6n^{\delta+1}, \\
&q = m^{2.5} \omega(\sqrt{\log n}), \\
&\alpha = (m^2 \omega(\sqrt{\log n}))^{-1}, \\
&\sigma = m \omega(\sqrt{\log n}),
\end{aligned}$$

with δ such that $n^\delta = O(\log q)$.

4.1.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 4.1.1. Note that for security parameter n and modulus q, we have $m = O(n \log q)$ (see Section 4.1.4).

The master key MK is just an $m \times m$ matrix, therefore its size is m^2. The public key PK is comprised of a vector of length n and three $n \times m$ matrices; therefore its size is $n + 3nm$, which is $O(mn)$. The secret key is just a vector of length $2m$; therefore its size is $2m$, which is $O(m)$. Finally, the ciphertext is comprised of an integer and two vectors of length m; therefore its size is $1 + 2m$, which is $O(m)$.

The complexity of IBE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, therefore, IBE-SetUp is also polynomial. The complexity of IBE-KeyGen is based on the complexity of the SampleLeft algorithm plus one matrix-matrix multiplication ($O(n^2m)$) and a matrix addition ($O(nm)$). As before, we have that the SampleLeft algorithm is polynomial, by Theorem 2.7.

The IBE-Enc algorithm does one matrix-matrix multiplication ($O(n^2m)$), three matrix-vector multiplications ($O(nm+nm+m^2)$), one inner product ($O(n)$), one matrix addition ($O(nm)$), two vector additions ($O(m)$, each) and two simple additions $O(1)$. Therefore, the complexity of IBE-Enc is based on the matrix-matrix multiplication operation. The IBE-Dec algorithm does only the inner product between two vectors and a simple addition. Since the vectors are of length $2m$, we have that the complexity of IBE-Dec is $O(m)$.

Table 4.1 summarises the size of the main variables and Table 4.2 summarises the complexity of the four algorithms of the scheme described in Section 4.1.1.

Variable	**Size**
Public Key PK	$O(n^2 \log q)$
Master Key MK	$O(n^2 \log^2 q)$
Secret Key SK	$O(n \log q)$
Ciphertext CT	$O(n \log q)$

Table 4.1: Key Sizes of the general IBE Scheme

Algorithm	**Complexity**
SetUp	$O(\text{poly}(n))$
KeyGen	$O(\text{poly}(n) + n^3 \log q)$
Enc	$O(n^3 \log q)$
Dec	$O(n \log q)$

Table 4.2: Complexity of the general IBE Scheme

4.2 Lattice-Based Hierarchical Identity-Based Encryption

This Section reviews the hierarchical IBE scheme proposed by Agrawal, Boneh and Boyen [4]. Section 4.2.1 describes the four algorithms that comprise the HIBE scheme, Sections 4.2.2, 4.2.3, 4.2.4 and 4.2.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

4.2.1 Description

As described in Section 3.1, an Hierarchical Identity-Based Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n, \mu)$, $\mathsf{KeyDerive}(PK, SK_{t-1}, \boldsymbol{id}_1, \cdots, \boldsymbol{id}_t)$, $\mathsf{Enc}(PK, M, \boldsymbol{id}_1, \cdots, \boldsymbol{id}_t)$ and $\mathsf{Dec}(PK, SK_t, CT)$. In this section we describe each algorithm as presented by Agrawal et al. [4]. The hierarchy is described by parameter μ and has maximum depth d.

HIBE-SetUp creates a general lattice and chooses at random $d+1$ matrices and a vector that will form the public and master keys. HIBE-KeyDerive generates the secret key by encoding each identity $\boldsymbol{id}_i$ into a matrix using the $\mathsf{rot}_f()$ function and concatenating it to the lattice basis. Now, the secret key is a short basis for the lattice generate by this concatenation, using the algorithm SampleBasisLeft, described in Section 2.2. Note that $SK_0 = MK$.

HIBE-Enc uses the message M, the identities $\boldsymbol{id}_i$ and the matrices in the public key to create an integer c', vector $\boldsymbol{c}_0$ and t vectors $\boldsymbol{c}_i$, one for each level, that will compose the ciphertext for one bit. Finally, HIBE-Dec uses the algorithm SamplePre with the basis that comprise SK_t to find a vector $\boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(A|C_1|\cdots|C_t)$ and then recover the message from the ciphertext only if all the identities $\boldsymbol{id}_j$ used are the same, for all $j \in [1, t]$.

Let n be the security parameter, μ be the hierarchical parameter and σ_i (for $i \in [1, d]$) be the Gaussian parameters. Algorithms 4.5, 4.6, 4.7 and 4.8 describe the HIBE scheme.

Algorithm 4.5 HIBE-SetUp(): Setup Algorithm for the HIBE Scheme

```
Input: security parameter $1^n$ and hierarchical parameter $1^\mu$
Output: Public key $PK$ and master key $MK$
    $A, S \leftarrow \mathsf{TrapGen}(q, n, m)$
    $A_i \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, for $i \in [1, d]$
    $B \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$
    $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$
    public key $PK = (A, \{A_i\}, B, \boldsymbol{u})$
    master key $MK = S$
```

Algorithm 4.6 HIBE-KeyDerive(): Key Generation Algorithm for the HIBE Scheme

```
Input: Public key $PK$, secret key $SK_{t-1}$ and identities $\boldsymbol{id}_1, \cdots, \boldsymbol{id}_t$
Output: Secret key $SK_t$
    $C_i = A_i + \mathsf{rot}_f(\boldsymbol{id}_i)B$, for $i \in [1, t]$
    $C = [C_1| \cdots |C_{t-1}]$
    $S_t \leftarrow \mathsf{SampleBasisLeft}([A|C], C_t, S_{t-1}, \sigma_t)$
    secret key $SK_t = S_t \in \mathbb{Z}_q^{n \times (t+1)m}$
```

Algorithm 4.7 HIBE-Enc(): Encryption Algorithm for the HIBE Scheme

```
Input: Public key $PK$, message $M$ and identities $\boldsymbol{id}_1, \cdots, \boldsymbol{id}_t$
Output: Ciphertext $CT$
    $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$
    $\boldsymbol{x} \in \overline{\Psi}_\alpha^m$ and $x \in \overline{\Psi}_\alpha$
    $R_i \xleftarrow{\$} \{-1, 1\}^{m \times m}$
    $C_i = A_i + \mathsf{rot}_f(\boldsymbol{id}_i)B$
    $\boldsymbol{c}_0 = A^\top \boldsymbol{s} + \boldsymbol{x} \in \mathbb{Z}_q^m$
    $\boldsymbol{c}_i = C_i^\top \boldsymbol{s} + R_i^\top \boldsymbol{x} \in \mathbb{Z}_q^m$
    $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\boldsymbol{c}_0, \{\boldsymbol{c}_i\}, c')$
```

Algorithm 4.8 HIBE-Dec(): Decryption Algorithm for the HIBE Scheme

```
Input: Public key $PK$, secret key $SK_t$ and ciphertext $CT$
Output: message $M$
    $C_i = A_i + \mathsf{rot}_f(\boldsymbol{id}_i)B$, for $i \in [1, t]$
    $C = [C_1| \cdots |C_t]$
    $\sigma = \sigma_t \sqrt{m(t+1)}\omega(\sqrt{\log(tm)})$
    $\boldsymbol{e} = \mathsf{SamplePre}([A|C], S_t, \boldsymbol{u}, \sigma)$
    $z = c' - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_t \end{bmatrix} \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

4.2.2 Correctness

The correctness is straightforward, exactly like the IBE scheme. First we just substitute the values of $\boldsymbol{c}_0$, $\boldsymbol{c}_i$ and c' in z. If the identities used during the key generation are the same as the ones used during the encryption, then $\boldsymbol{u}^\top = \boldsymbol{e}^\top [A|C]^\top$, with $C = [C_1|\cdots|C_t]$. Finally, we cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get the right value of M in z.

$$\begin{aligned}
z &= c' - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_t \end{bmatrix} \mod q \\
&= c' - \boldsymbol{e}^\top \begin{bmatrix} A^\top \boldsymbol{s} + \boldsymbol{x} \\ C_1^\top \boldsymbol{s} + R_1^\top \boldsymbol{x} \\ \vdots \\ C_t^\top \boldsymbol{s} + R_t^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= c' - \boldsymbol{e}^\top \begin{bmatrix} A^\top \\ C^\top \end{bmatrix} \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= c' - \boldsymbol{e}^\top [A|C]^\top \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= x + M\lfloor q/2 \rfloor - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}$$

Note that for the correct decryption the error term must be less than $q/4$.

4.2.3 Security

In this section we prove the following theorem.

Theorem 4.2. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 4.2.1 is IND-AH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.HIBE-SetUp$(1^n, 1^\mu, \boldsymbol{id}_1^\star, \cdots, \boldsymbol{id}_d^\star)$: The algorithm chooses random $A \in \mathbb{Z}_q^{n\times m}$, $R_i^\star \in \{-1,1\}^{m\times m}$ and $\boldsymbol{u} \in \mathbb{Z}_q^n$ and it uses $\mathsf{TrapGen}(q,n,m)$ to generate $B^\star \in \mathbb{Z}_q^{n\times m}$ and the basis $S^\star \in \mathbb{Z}^{m\times m}$ for $\Lambda_q^\perp(B^\star)$. It then defines $A_i \leftarrow AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star$, for $i \in [1,d]$ and outputs $PK = (A, \{A_i\}, \boldsymbol{u})$ and $MK = (R^\star, B^\star, S^\star)$.

Sim.HIBE-KeyDerive$(PK, MK, \boldsymbol{id}_1, \cdots, \boldsymbol{id}_t)$: Secret keys are now sampled by the SampleBasisRight algorithm, using the trapdoor $S^\star$. It outputs $SK_t = S_t$ which is a basis for lattice $\Lambda_q^\perp(A|AR_1^\star - \mathsf{rot}_f(\boldsymbol{id}_1^\star)B^\star|\cdots|AR_t^\star - \mathsf{rot}_f(\boldsymbol{id}_t^\star)B^\star)$:
$S \leftarrow \mathsf{SampleBasisRight}(A, B^\star_{\boldsymbol{id}}, R^\star, S^\star, \sigma_t)$,
with $R^\star = [R_1^\star|\cdots|R_t^\star]$
and $B^\star_{\boldsymbol{id}} = [(\mathsf{rot}_f(\boldsymbol{id}_1) - \mathsf{rot}_f(\boldsymbol{id}_1^\star))B^\star|\cdots|(\mathsf{rot}_f(\boldsymbol{id}_t) - \mathsf{rot}_f(\boldsymbol{id}_t^\star))B^\star]$.
Note that we must have $\boldsymbol{id}_i \neq \boldsymbol{id}_i^\star$, for all $i \in [1,t]$, for the algorithm SampleRight to work properly.

Sim.HIBE-Enc$(PK, M, \boldsymbol{id}_1^\star, \cdots, \boldsymbol{id}_t^\star)$: The algorithm differs from HIBE-Enc in the sense that it uses matrices $R_i^\star$ and $B^\star$ instead of matrices R_i and B.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs HIBE-SetUp, answers $\mathcal{A}$'s secret key queries using the HIBE-KeyDerive algorithm, and generates the challenge ciphertext using the HIBE-Enc with identities $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_t^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.HIBE-SetUp with identities $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_d^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HIBE-KeyDerive, and generates the challenge ciphertext using the Sim.HIBE-Enc algorithm with $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_t^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.HIBE-SetUp with identity identities $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_d^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HIBE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.HIBE-SetUp with identity identities $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_d^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HIBE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.HIBE-SetUp with identities $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_d^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HIBE-KeyDerive, and generates the challenge ciphertext using the Sim.HIBE-Enc algorithm with $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_t^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs HIBE-SetUp, answers $\mathcal{A}$'s secret key queries using the HIBE-KeyDerive algorithm, and generates the challenge ciphertext using the HIBE-Enc with identities $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_t^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

For simplicity reasons, assume that every time we refer to matrices A_i and $R_i^\star$ and vectors $\boldsymbol{c}_i$ we are referring to all matrices or vector for $i \in [1,d]$.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 4.4. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, matrix A is generated by $\mathsf{TrapGen}$ and matrices A_i are uniformly random in $\mathbb{Z}_q^{n\times m}$. Instead, in Game 1, A is chosen uniformly at random and we have $A_i \leftarrow AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star$, where $B^\star$ is generated by $\mathsf{TrapGen}$ and the matrices $R_i^\star$ are uniformly and independently chosen at random in $\{-1,1\}^{m\times m}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathbb{Z}_q^n$.

Secret keys In Game 0, the secret key for identity $\boldsymbol{id}$ is a basis of $\Lambda_q^\perp(A|C)$, with $C = [A_1 + \mathsf{rot}_f(\boldsymbol{id}_1)B|\dots|A_t + \mathsf{rot}_f(\boldsymbol{id}_t)B]$ sampled using the $\mathsf{SampleBasisLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleBasisRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext In both games the challenge ciphertext components c' and $\boldsymbol{c}_0$ are computed the same way but, in Game 0, the challenge ciphertext components $\boldsymbol{c}_i$ are computed as follows:

$$\begin{aligned}\boldsymbol{c}_i &= C_i^\top \boldsymbol{s} + R_i^{\star\top}\boldsymbol{x}\\ \boldsymbol{c}_i &= (A_i + \mathsf{rot}_f(\boldsymbol{id}_i)B^\star)^\top \boldsymbol{s} + R_i^\top \boldsymbol{x} \in \mathbb{Z}_q^m\end{aligned} \; .$$

On the other hand, in Game 1, we have:

$$\begin{aligned}\boldsymbol{c}_i &= C_i^\top \boldsymbol{s} + R_i^{\star\top}\boldsymbol{x}\\ &= (A_i + \mathsf{rot}_f(\boldsymbol{id}_i)B^\star)^\top \boldsymbol{s} + R_i^{\star\top}\boldsymbol{x} \in \mathbb{Z}_q^m\\ &= (AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star + \mathsf{rot}_f(\boldsymbol{id}_i)B^\star)^\top \boldsymbol{s} + R_i^{\star\top}\boldsymbol{x} \in \mathbb{Z}_q^m\\ &= (AR_i^\star)^\top \boldsymbol{s} + R_i^\top \boldsymbol{x} \in \mathbb{Z}_q^m\end{aligned} \; .$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(A, \{A_i\}, \{\boldsymbol{c}_i\})$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemmas A.4 and A.5 we have that the following two distributions are statistically indistinguishable for every fixed matrix $B^\star$, every $\boldsymbol{id}_i^\star$ and every vector $\boldsymbol{x} \in \mathbb{Z}_q^m$:

$$\left(A, A_i, R_i^{\star\top}\boldsymbol{x}\right) \approx_s \left(A, AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star, R_i^{\star\top}\boldsymbol{x}\right) \; .$$

Since each $R_i^\star$ is chosen independently for every i, then the joint distribution of them are statistically close:

$$\big(A, \{A_i\}, \{R_i^{\star\top}\boldsymbol{x}\}\big) \approx_s \big(A, \{AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star\}, \{R_i^{\star\top}\boldsymbol{x}\}\big)\ .$$

Since $(AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star)^\top\boldsymbol{s}$ is statistically close to $A_i^\top\boldsymbol{s}$, it is possible to add each term to each side of the equation:

$$\begin{aligned} &\big(A, \{A_i\}, \{A_i^\top\boldsymbol{s} + R_i^{\star\top}\boldsymbol{x}\}\big) \approx_s \\ \big(A, \{AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star\}, &\{(AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star)^\top\boldsymbol{s} + R_i^{\star\top}\boldsymbol{x}\}\big) \end{aligned}\ .$$

Then, we add $(\mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star)^\top\boldsymbol{s}$ to each side of the equation:

$$\begin{aligned} \big(A, \{A_i\}, \{(A_i + \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star)^\top\boldsymbol{s} + R_i^{\star\top}\boldsymbol{x}\}\big) \approx_s \\ \big(A, \{AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star\}, \{(AR_i^\star)^\top\boldsymbol{s} + R_i^{\star\top}\boldsymbol{x}\}\big) \end{aligned}\ .$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 4.5. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE*.*

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $m+1$ LWE challenge pairs $(\boldsymbol{a}_j, y_j) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where either $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$ for a random $\boldsymbol{s} \in \mathbb{Z}_q^n$ and a noise term $x_j \leftarrow \Psi_\alpha$, or y_j is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_j, y_j)$. The i-th column of matrix A will be the vector $\boldsymbol{a}_i$, for $1 \le i \le m$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrices A_i are still calculated as in Sim.HIBE-SetUp, i.e., $A_i \leftarrow AR_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star)B^\star$.

Secret keys All private-key extraction queries are answered using Sim.HIBE-KeyDerive.

Challenge Ciphertext The ciphertext $CT = (\boldsymbol{c}_0^\star, \{\boldsymbol{c}_i^\star\}, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_j, y_j)$, with $\boldsymbol{c}_0^\star = (y_1, \ldots, y_m)$, $c'^\star = y_0 + M\lfloor q/2\rceil$ and $\boldsymbol{c}_i^\star = R_i^{\star\top}\boldsymbol{c}_0^\star$. If we have $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_j is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2. If $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$, then

$(y_1, \ldots, y_m) = (\langle \boldsymbol{a}_1, \boldsymbol{s}\rangle + x_1, \ldots, \langle \boldsymbol{a}_m, \boldsymbol{s}\rangle + x_m) = A^\top\boldsymbol{s} + \boldsymbol{x}$.

Therefore, for Game 1 we have

$$\begin{aligned} \boldsymbol{c}_0^\star &= A^\top \boldsymbol{s} + \boldsymbol{x} \\ &= (y_1, \ldots, y_m) \ , \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{c}_i^\star &= C_i^\top \boldsymbol{s} + R_i^{\star\top} \boldsymbol{x} \\ &= (A_i + \mathsf{rot}_f(\boldsymbol{id}_i) B^\star)^\top \boldsymbol{s} + R_i^{\star\top} \boldsymbol{x} \\ &= (A R_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star) B^\star + \mathsf{rot}_f(\boldsymbol{id}_i) B^\star)^\top \boldsymbol{s} + R_i^{\star\top} \boldsymbol{x} \\ &= (A R_i^\star)^\top \boldsymbol{s} + R_i^{\star\top} \boldsymbol{x} \\ &= R_i^{\star\top} (A^\top \boldsymbol{s} + \boldsymbol{x}) \\ &= R_i^{\star\top} \boldsymbol{c}_0^\star \ . \end{aligned}$$

If y_j is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_j is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s} \rangle + x_j$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 4.6. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. A and $\boldsymbol{u}$ are random and $A_i \leftarrow A R_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star) B^\star$, with $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 0}$ for Game 2 and $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 1}$ for Game 3.

Secret keys All private-key extraction queries are answered using $\mathsf{Sim.HIBE\text{-}KeyDerive}$. The only difference is, again, that for Game 2, $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 0}$ and, for Game 3, $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrices A_i. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of A_i. From Lemmas A.4 and A.5 we can prove that each A_i for each game is statistically close to a uniformly random matrix, because

$A_i \leftarrow A R_i^\star - \mathsf{rot}_f(\boldsymbol{id}_i^\star) B^\star$.

□

4.2.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma_t\sqrt{(t+1)m}$ from Lemma A.1 and $\|R\boldsymbol{e}\| \leq 12\sqrt{tm+m}\|\boldsymbol{e}\|$ from Lemma A.2, for $R = [R_1|\cdots|R_t]$, with $\max(t) = d$; therefore, for $\boldsymbol{e} = \begin{bmatrix}\boldsymbol{e}_1\\ \boldsymbol{e}_2\end{bmatrix}$:

$$\begin{aligned}&\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq \sigma_t\sqrt{(d+1)m} + d12\sqrt{(d+1)m}\sigma_t\sqrt{(d+1)m} \qquad \text{so}\\ &\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq O(\sigma_t d^2 m)\ .\end{aligned}$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \leq \|\boldsymbol{y}\|q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\begin{aligned}&\langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle \leq O(\sigma_t d^2 m)q\alpha_t\omega(\sqrt{\log m}) + O(\sigma_t d^2 m)\sqrt{m}/2\\ &\langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle \leq \widetilde{O}(\sigma_t d^2 m q\alpha_t) + O(\sigma_t d^2 m^{3/2})\ .\end{aligned}$$

To ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned}&x - \boldsymbol{e}^\top\begin{bmatrix}\boldsymbol{x}\\ R^\top\boldsymbol{x}\end{bmatrix} < q/4\\ &x - \boldsymbol{e}_1^\top\boldsymbol{x} - \boldsymbol{e}_2^\top R^\top\boldsymbol{x} < q/4\\ &x - (\boldsymbol{e}_1 + R\boldsymbol{e}_2)^\top\boldsymbol{x} < q/4\\ &x - \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle < q/4\\ &\widetilde{O}(\sigma_t d^2 m q\alpha) + O(\sigma_t d^2 m^{3/2}) < q/4\end{aligned}$$

To ensure that σ_t is sufficiently large for SampleBasisLeft and SampleBasisRight (Theorems 2.10 and 2.11), we have

$$\sigma_t > \|S\|\sqrt{m}\omega(\sqrt{\log m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned}&m \geq 6n\log q \qquad \text{and}\\ &\|S\| \leq O(n\log q)\ .\end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha_t\ .$$

Therefore, we need to set the parameters as

$$\begin{aligned} m &= 6n^{\delta+1}, \\ q &= m^{2.5}\omega(\sqrt{\log n}), \\ \alpha_t &= (m^2\omega(\sqrt{\log n}))^{-1}, \\ \sigma_t &= m\omega(\sqrt{\log n}), \end{aligned}$$

with δ such that $n^\delta = O(\log q)$.

4.2.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 4.2.1. Note that for security parameter n, hierarchy's maximum depth d and modulus q, we have $m = O(n \log q)$ (see Section 4.2.4).

The master key MK is just an $m \times m$ matrix, therefore its size is m^2. The public key PK is comprised of a vector of length n and $d+2$ matrices of size $n \times m$; therefore its size is $n + (d+2)nm$, which is $O(dmn)$. The secret key is now a matrix, not a vector, of size $n \times (t+1)m$, with $\max(t) = d$; therefore its size is at most $n(d+1)m$, which is $O(dnm)$. Finally, the ciphertext is comprised of an integer and $t+1$ vectors of length m, with $\max(t) = d$; therefore its size is at most $1 + m + dm$, which is $O(dm)$.

The complexity of HIBE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, therefore, HIBE-SetUp is also polynomial. The complexity of HIBE-KeyDerive is based on the complexity of the SampleBasisLeft algorithm plus t matrix-matrix multiplications ($O(n^2m)$, each) and t matrix addition ($O(nm)$, each). As before, we have that the SampleBasisLeft algorithm is polynomial, by Theorem 2.10.

The HIBE-Enc algorithm does t matrix-matrix multiplication ($O(n^2m)$, each), t matrix additions ($O(nm)$, each), $2t+1$ matrix-vector multiplications ($O(nm + tnm + tm^2)$), one inner product ($O(n)$), $1+t$ vector additions ($O(m+tm)$) and two simple additions $O(1)$. Therefore, the complexity of HIBE-Enc is based on the matrix-matrix multiplication operations for $t = d$. The HIBE-Dec algorithm does t matrix-matrix multiplications ($O(n^2m)$, each), t matrix-matrix addition operations ($O(nm)$, each), one inner product between two vectors of length tm, a few simple additions and multiplications and calls the SamplePre algorithm. We have, by Theorem 2.6, that SamplePre is polynomial, therefore the complexity of the HIBE-Dec algorithm is based on SamplePre and the matrix-matrix multiplications.

Table 4.3 summarises the size of the main variables and Table 4.4 summarises the complexity of the four algorithms of the scheme described in Section 4.2.1.

Variable	**Size**
Public Key PK	$O(dn^2 \log q)$
Master Key MK	$O(n^2 \log^2 q)$
Secret Key SK	$O(dn^2 \log q)$
Ciphertext CT	$O(dn \log q)$

Table 4.3: Key Sizes of the HIBE Scheme

Algorithm	**Complexity**
SetUp	$O(\text{poly}(n))$
KeyDerive	$O(\text{poly}(n) + dn^3 \log q)$
Enc	$O(dn^3 \log q)$
Dec	$O(\text{poly}(n) + dn^3 \log q)$

Table 4.4: Complexity of the HIBE Scheme

Chapter 5

Inner Product Encryption

In this chapter we describe the lattice-based IPE scheme proposed by Agrawal, Freeman and Vaikuntanathan [7]. Section 5.1 describes the general scheme as described by Agrawal et al. [7] and Section 5.2 describes our contribution, a hierarchical version of the same scheme as decsribed by Abdalla et al. [1].

5.1 Lattice-Based Inner Product Encryption

This Section reviews the IPE scheme proposed by Agrawal, Freeman and Vaikuntanathan [7]. Section 5.1.1 describes the four algorithms that comprise the IPE scheme, Sections 5.1.2, 5.1.3, 5.1.4 and 5.1.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

5.1.1 Description

As described in Section 3.6, an Inner Product Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n)$, $\mathsf{KeyGen}(PK, MK, \boldsymbol{v})$, $\mathsf{Enc}(PK, M, \boldsymbol{w})$ and $\mathsf{Dec}(PK, SK, CT)$. In this section we describe each algorithm as presented by Agrawal et al. [7].

This scheme is similar to the IBE scheme described in Section 4.1, but the attributes are vectors, instead of identities, and the decryption is only possible if the inner product of these two vectors is zero. These vectors have length l and their elements are integers in $[0, q-1]$. For a vector $\boldsymbol{v} \in \mathbb{Z}_q^l$, each element $v_j \in \boldsymbol{v}$ can be decomposed in k integers as follows:

$$v_j = \sum_{\gamma=0}^{k} v_{j,\gamma} r^{\gamma}.$$

As before, $\mathsf{IPE\text{-}SetUp}$ creates a general lattice and chooses at random the matrices and a vector that will form the public and master keys. But now it will create $l(1+k)$

matrices, instead of only two. IPE-KeyGen generates the secret key by multiplying each element of $\boldsymbol{v}$, decomposed in k elements, with a matrix, adding all the resulting matrices and concatenating it to the lattice basis. The secret key is the vector $\boldsymbol{e}$ created by the SampleLeft algorithm as described in Section 2.2, using the matrix concatenation as the lattice basis; therefore $\boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(A_{\boldsymbol{v}})$.

IPE-Enc uses the message M, the elements of vector $\boldsymbol{w}$ and the matrices in the public key to create an integer c', a vector $\boldsymbol{c}_0$ and several vectors $\boldsymbol{c}_{j,\gamma}$ that will compose the ciphertext for one bit. Finally, IPE-Dec can recover the message from the ciphertext only if $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = 0$.

Let n be the security parameter and σ be the Gaussian parameter. Algorithms 5.1, 5.2, 5.3 and 5.4 describe the IPE scheme.

Algorithm 5.1 IPE-SetUp(): Setup Algorithm for the IPE Scheme

```
Input: security parameter $1^n$
Output: Public key $PK$ and master key $MK$
    $A, S \leftarrow \mathsf{TrapGen}(q, n, m)$
    $A_{j,\gamma} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, for $j \in [1, l]$ and $\gamma \in [0, k]$
    $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$
    public key $PK = (A, \{A_{j,\gamma}\}, \boldsymbol{u})$
    master key $MK = S$
```

Algorithm 5.2 IPE-KeyGen(): Key Generation Algorithm for the IPE Scheme

```
Input: Public key $PK$, master key $MK$ and vector $\boldsymbol{v}$
Output: Secret key $SK$
    $C = \sum_{j=1}^{l} \sum_{\gamma=0}^{k} v_{j,\gamma} A_{j,\gamma}$
    $A_{\boldsymbol{v}} = [A|C] \in \mathbb{Z}^{n \times 2m}$
    $\boldsymbol{e} \leftarrow \mathsf{SampleLeft}(A, C, S, \boldsymbol{u}, \sigma)$
    secret key $SK = \boldsymbol{e} \in \mathbb{Z}_q^{2m}$
```

Algorithm 5.3 IPE-Enc(): Encryption Algorithm for the IPE Scheme

```
Input: Public key $PK$, message $M$ and vector $\boldsymbol{w}$
Output: Ciphertext $CT$
    $B \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$
    $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^{n}$
    $\boldsymbol{x} \in \overline{\Psi}_\alpha^m$ and $x \in \overline{\Psi}_\alpha$
    $R \xleftarrow{\$} \{-1,1\}^{m\times m}$
    $\boldsymbol{c}_0 = A^\top \boldsymbol{s} + \boldsymbol{x} \in \mathbb{Z}_q^m$
    $\boldsymbol{c}_{j,\gamma} = (A_{j,\gamma} + r^\gamma w_j B)^\top \boldsymbol{s} + R^\top \boldsymbol{x} \in \mathbb{Z}_q^m$
    $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\boldsymbol{c}_0, \{\boldsymbol{c}_{j,\gamma}\}, c')$
```

Algorithm 5.4 IPE-Dec(): Decryption Algorithm for the IPE Scheme

```
Input: Public key $PK$, secret key $SK$ and ciphertext $CT$
Output: message $M$
    $\boldsymbol{c}_{\boldsymbol{v}} = \sum_{j=1}^{l} \sum_{\gamma=0}^{k} v_{j,\gamma} \boldsymbol{c}_{j,\gamma}$
    $z = c' - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_{\boldsymbol{v}} \end{bmatrix} \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

5.1.2 Correctness

On the first step of the correctness we substitute the values of all vectors $\boldsymbol{c}_{j,\gamma}$ in $\boldsymbol{c}_{\boldsymbol{v}}$. In this step we recover the value with the inner product of $\boldsymbol{v}$ and $\boldsymbol{w}$ and we only can cancel the term multiplying B if $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = 0$. Then, we substitute the value of $\boldsymbol{c}_{\boldsymbol{v}}$ together with $\boldsymbol{c}_0$ and c' in z. Finally, we cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get the right value of M in z.

$$\begin{aligned}
\boldsymbol{c_v} &= \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}\boldsymbol{c}_{j,\gamma} \\
&= \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}[(A_{j,\gamma} + r^{\gamma}w_i B)^\top \boldsymbol{s} + R^\top \boldsymbol{x}] \\
&= \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}A_{j,\gamma}^\top \boldsymbol{s} + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}r^{\gamma}w_i B^\top \boldsymbol{s} + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\top \boldsymbol{x} \\
&= C^\top \boldsymbol{s} + \langle \boldsymbol{v}, \boldsymbol{w}\rangle B^\top \boldsymbol{s} + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\top \boldsymbol{x} \\
&= C^\top \boldsymbol{s} + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\top \boldsymbol{x}
\end{aligned}$$

$$\begin{aligned}
z &= c' - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c_v} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{e}^\top \begin{bmatrix} A^\top \boldsymbol{s} + \boldsymbol{x} \\ C^\top \boldsymbol{s} + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{e}^\top \begin{bmatrix} A^\top \\ C^\top \end{bmatrix} \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= x + M\lfloor q/2 \rfloor - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}$$

Note that for the correct decryption the error term must be less than $q/4$.

5.1.3 Security

In this section we prove the following theorem.

Theorem 5.1. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 5.1.1 is IND-wAH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.IPE-SetUp$(1^n, \boldsymbol{w}^\star)$: The algorithm chooses random $A \in \mathbb{Z}_q^{n\times m}$, $R^\star_{j,\gamma} \in \{-1,1\}^{m\times m}$ and $\boldsymbol{u} \in \mathbb{Z}_q^n$ and it uses TrapGen(q,n,m) to generate $B^\star \in \mathbb{Z}_q^{n\times m}$ and the basis $S^\star \in \mathbb{Z}^{m\times m}$ for $\Lambda_q^\perp(B^\star)$. It then defines, for $j \in [1,l]$ and $\gamma \in [0,k]$, $A_{j,\gamma} \leftarrow AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star$ and outputs $PK = (A, \{A_{j,\gamma}\}, \boldsymbol{u})$ and $MK = (\{R^\star_{j,\gamma}\}, B^\star, S^\star)$.

Sim.IPE-KeyGen$(PK, MK, \boldsymbol{v})$: Secret keys are now sampled by the SampleRight algorithm, using the trapdoor $S^\star$. It outputs
$SK = \boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(A|A\sum\sum v_{j,\gamma}R^\star_{j,\gamma} - \langle \boldsymbol{v}, \boldsymbol{w}\rangle B^\star)$, where

$$\boldsymbol{e} \leftarrow \mathsf{SampleRight}\left(A, -\langle \boldsymbol{v}, \boldsymbol{w}\rangle B^\star, \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R^\star_{j,\gamma}, S^\star, \boldsymbol{u}, \sigma\right).$$

Note that $\langle \boldsymbol{v}, \boldsymbol{w}\rangle$ must be non-zero for the algorithm SampleRight to work properly.

Sim.IPE-Enc$(PK, M, \boldsymbol{w}^\star)$: The algorithm differs from IPE-Enc in the sense that it uses matrices $R^\star_{j,\gamma}$ and $B^\star$ instead of matrices $R_{j,\gamma}$ and B.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs IPE-SetUp, answers $\mathcal{A}$'s secret key queries using the IPE-KeyGen algorithm, and generates the challenge ciphertext using the IPE-Enc with vector $\boldsymbol{w}^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.IPE-SetUp with vector $\boldsymbol{w}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.IPE-KeyGen, and generates the challenge ciphertext using the Sim.IPE-Enc algorithm with $\boldsymbol{w}^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.IPE-SetUp with vector $\boldsymbol{w}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.IPE-KeyGen, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.IPE-SetUp with vector $\boldsymbol{w}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.IPE-KeyGen, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.IPE-SetUp with vector $\boldsymbol{w}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.IPE-KeyGen, and generates the challenge ciphertext using the Sim.IPE-Enc algorithm with $\boldsymbol{w}^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs IPE-SetUp, answers $\mathcal{A}$'s secret key queries using the IPE-KeyGen algorithm, and generates the challenge ciphertext using the IPE-Enc with vector $\boldsymbol{w}^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

For simplicity reasons, assume that every time we refer to matrices $A_{j,\gamma}$ and $R^\star_{j,\gamma}$ and vectors $\boldsymbol{c}_{j,\gamma}$ we are referring to all matrices or vector for $j \in [1,l]$ and $\gamma \in [0,k]$.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 5.1. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, matrix A is generated by $\mathsf{TrapGen}$ and matrices $A_{j,\gamma}$ are uniformly random in $\mathbb{Z}_q^{n\times m}$. Instead, in Game 1, A is chosen uniformly at random and we have $A_{j,\gamma} \leftarrow AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star$, where $B^\star$ is generated by $\mathsf{TrapGen}$ and the matrices $R^\star_{j,\gamma}$ are uniformly and independently chosen at random in $\{-1,1\}^{m\times m}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathbb{Z}_q^n$.

Secret keys In Game 0, the secret key for vector $\boldsymbol{v}$ is a vector $\boldsymbol{e} \in \Lambda_q^{\boldsymbol{u}}(A_{\boldsymbol{v}})$, sampled using the $\mathsf{SampleLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext In both games the challenge ciphertext components c' and $\boldsymbol{c}_0$ are computed the same way but, in Game 0, the challenge ciphertext component $\boldsymbol{c}_{j,\gamma}$ is computed as follows:

$$\boldsymbol{c}_{j,\gamma} = (A_{j,\gamma} + r^\gamma w^\star_j B^\star)^\top \boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x} \in \mathbb{Z}_q^m \ .$$

On the other hand, in Game 1, we have:

$$\begin{aligned} \boldsymbol{c}_{j,\gamma} &= (A_{j,\gamma} + r^\gamma w^\star_j B^\star)^\top \boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x} \\ &= (AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star + r^\gamma w^\star_j B^\star)^\top \boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x} \ . \\ &= (AR^\star_{j,\gamma})^\top \boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x} \in \mathbb{Z}_q^m \end{aligned}$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(A, A_{j,\gamma}, \boldsymbol{c}_{j,\gamma})$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemmas A.4 and A.5 we have that the following two distributions are statistically indistinguishable for every fixed matrix $B^\star$, every $\boldsymbol{w}^\star$ and every vector $\boldsymbol{x} \in \mathbb{Z}_q^m$:

$$\left(A, A_{j,\gamma}, R^{\star\top}_{j,\gamma}\boldsymbol{x}\right) \approx_s \left(A, AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star, R^{\star\top}_{j,\gamma}\boldsymbol{x}\right) \ .$$

Matrices $R^\star_{j,\gamma}$ are chosen independently for every j, γ, therefore the joint distributions of these quantities are also statistically close:

$$\big(A, \{A_{j,\gamma}\}, \{R^{\star\top}_{j,\gamma}\boldsymbol{x}\}\big) \approx_s \big(A, \{AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star\}, \{R^{\star\top}_{j,\gamma}\boldsymbol{x}\}\big) \ .$$

Since $(AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star)^\top \boldsymbol{s}$ is statistically close to $A^\top_{j,\gamma}\boldsymbol{s}$, it is possible to add each term to each side of the equation:

$$\begin{array}{r} \big(A, \{A_{j,\gamma}\}, \{A^\top_{j,\gamma}\boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x}\}\big) \approx_s \\ \big(A, \{AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star\}, \{(AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star)^\top \boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x}\}\big) \end{array} .$$

Then, we add $(r^\gamma w^\star_j B^\star)^\top \boldsymbol{s}$ to each side of the equation:

$$\begin{array}{r} \big(A, \{A_{j,\gamma}\}, \{(A_{j,\gamma} + r^\gamma w^\star_j B^\star)^\top \boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x}\}\big) \approx_s \\ \big(A, \{AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star\}, \{(AR^\star_{j,\gamma})^\top \boldsymbol{s} + R^{\star\top}_{j,\gamma}\boldsymbol{x}\}\big) \end{array} .$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 5.2. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE.

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $m+1$ LWE challenge pairs $(\boldsymbol{a}_j, y_j) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where either $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$ for a random $\boldsymbol{s} \in \mathbb{Z}_q^n$ and a noise term $x_j \leftarrow \Psi_\alpha$, or y_j is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_j, y_j)$. The i-th column of matrix A will be the vector $\boldsymbol{a}_i$, for $1 \le i \le m$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrices $A_{j,\gamma}$ are still calculated as in Sim.IPE-SetUp, i.e., $A_{j,\gamma} \leftarrow AR^\star_{j,\gamma} - r^\gamma w^\star_j B^\star$.

Secret keys All private-key extraction queries are answered using Sim.IPE-KeyGen.

Challenge Ciphertext The ciphertext $CT = (\boldsymbol{c}^\star_0, \boldsymbol{c}^\star_{j,\gamma}, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_j, y_j)$, with $\boldsymbol{c}^\star_0 = (y_1, \ldots, y_m)$, $\boldsymbol{c}^\star_{j,\gamma} = R^{\star\top}_{j,\gamma}\boldsymbol{c}^\star_0$ and $c'^\star = y_0 + M\lfloor q/2 \rceil$. If we have $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_j is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2 . If $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$, then

$(y_1, \ldots, y_m) = (\langle \boldsymbol{a}_1, \boldsymbol{s}\rangle + x_1, \ldots, \langle \boldsymbol{a}_m, \boldsymbol{s}\rangle + x_m) = A^\top \boldsymbol{s} + \boldsymbol{x}$.
Therefore, for Game 1 we have

$$\begin{aligned} \boldsymbol{c}_0^\star &= A^\top \boldsymbol{s} + \boldsymbol{x} \\ &= (y_1, \ldots, y_m) \ , \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{c}_{j,\gamma}^\star &= (AR_{j,\gamma}^\star)^\top \boldsymbol{s} + R_{j,\gamma}^{\star\top} \boldsymbol{x} \\ &= R_{j,\gamma}^{\star\top}(A^\top \boldsymbol{s} + \boldsymbol{x}) \\ &= R_{j,\gamma}^{\star\top} \boldsymbol{c}_0^\star \ . \end{aligned}$$

If y_j is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_j is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 5.3. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. A and $\boldsymbol{u}$ are random and $A_{j,\gamma} \leftarrow AR_{j,\gamma}^\star - r^\gamma w_j^\star B^\star$, with $\boldsymbol{w}^\star = \boldsymbol{w}^{\star 0}$ for Game 2 and $\boldsymbol{w}^\star = \boldsymbol{w}^{\star 1}$ for Game 3.

Secret keys All private-key extraction queries are answered using Sim.IPE-KeyGen. The only difference is, again, that for Game 2, $\boldsymbol{w}^\star = \boldsymbol{w}^{\star 0}$ and, for Game 3, $\boldsymbol{w}^\star = \boldsymbol{w}^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrix $A_{j,\gamma}$. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of $A_{j,\gamma}$. From Lemmas A.4 and A.5 we can prove that each $A_{j,\gamma}$ for each game is statistically close to a uniformly random matrix, because

$A_{j,\gamma} \leftarrow AR_{j,\gamma}^\star - r^\gamma w_j^\star B^\star$.

□

5.1.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma\sqrt{2m}$ from Lemma A.1 and $\|R_{j,\gamma}\boldsymbol{e}\| \leq 12\sqrt{2m}\|\boldsymbol{e}\|$ from Lemma A.2. We also know $v_{j,\gamma} \in [0, r-1]$ because of the decomposition, therefore, for $\boldsymbol{e} = \begin{bmatrix}\boldsymbol{e}_1\\ \boldsymbol{e}_2\end{bmatrix}$:

$$\|\boldsymbol{e}_1 + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}\boldsymbol{e}_2\| \leq (\sigma\sqrt{2m} + l(k+1)r12\sqrt{2m}\sigma\sqrt{2m}) \qquad \text{so}$$
$$\|\boldsymbol{e}_1 + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}\boldsymbol{e}_2\| \leq O(lkr\sigma m)\ .$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \leq \|\boldsymbol{y}\|q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\left\langle \boldsymbol{e}_1 + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}\boldsymbol{e}_2, \boldsymbol{x}\right\rangle \leq O(lkr\sigma m)q\alpha w(\sqrt{\log 2m}) + O(lkr\sigma m)\sqrt{2m}/2$$
$$\left\langle \boldsymbol{e}_1 + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}\boldsymbol{e}_2, \boldsymbol{x}\right\rangle \leq \widetilde{O}(lkr\sigma mq\alpha) + O(lkr\sigma m^{3/2})\ .$$

To ensure that the error term is less than $q/4$, we need the following:

$$x - \boldsymbol{e}^{\top}\begin{bmatrix}\boldsymbol{x}\\ \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}^{\top}\boldsymbol{x}\end{bmatrix} < q/4$$
$$x - \boldsymbol{e}_1^{\top}\boldsymbol{x} - \boldsymbol{e}_2^{\top}\sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}^{\top}\boldsymbol{x} < q/4$$
$$x - \left(\boldsymbol{e}_1 + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}\boldsymbol{e}_2\right)^{\top}\boldsymbol{x} < q/4$$
$$x - \left\langle \boldsymbol{e}_1 + \sum_{i=1}^{l}\sum_{\gamma=0}^{k} v_{j,\gamma}R_{j,\gamma}\boldsymbol{e}_2, \boldsymbol{x}\right\rangle < q/4$$
$$\widetilde{O}(lkr\sigma mq\alpha) + O(lkr\sigma m^{3/2}) < q/4$$

To ensure that σ is sufficiently large for SampleLeft and SampleRight (Theorems 2.7 and 2.8), we have

$$\sigma > \|S\|\sqrt{2m}\omega(\sqrt{\log 2m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned} &m \geq 6n\log q \qquad \text{and} \\ &\|S\| \leq O(n\log q)\ . \end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha\ .$$

Therefore, we need to set the parameters as

$$\begin{aligned} &r = 2, \\ &k = \log q, \\ &m = n^{\delta+1}, \\ &\sigma = m\omega(\sqrt{\log n}), \\ &q = lm^{2.5}\omega(\sqrt{\log n}\log q), \\ &\alpha = (lm^2\omega(\sqrt{\log n})\log q)^{-1}, \end{aligned}$$

with δ such that $n^\delta = O(\log q)$.

5.1.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 5.1.1. Note that for security parameter n, vector length l and modulus q, we have $m = O(n\log q)$ and $k = \log q$ (see Section 5.1.4).

The master key MK is just an $m \times m$ matrix, therefore its size is m^2. The public key PK is comprised of a vector of length n and $l(k+1)+1$ matrices of size $n \times m$; therefore its size is $n + l(k+1)nm + nm$, which is $O(lkmn)$. The secret key is just a vector of length $2m$; therefore its size is $2m$, which is $O(m)$. Finally, the ciphertext is comprised of an integer and $(1 + l(k+1))$ vectors of length m; therefore its size is $1 + m + l(k+1)m$, which is $O(lkm)$.

The complexity of IPE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, therefore, IPE-SetUp is also polynomial. The complexity of IPE-KeyGen is based on the complexity of the SampleLeft algorithm plus $l(k+1)$ constant-matrix multiplications ($O(lknm)$) and $l(k+$

1) matrix additions ($O(lknm)$). As before, we have that the SampleLeft algorithm is polynomial, by Theorem 2.7.

The IPE-Enc algorithm does $1 + 2l(k + 1)$ matrix-vector multiplications ($mn + l(k + 1)nm + l(k + 1)m^2$ operations, which is $O(lkm^2)$), $l(k + 1)$ matrix addictions ($O(lkmn)$), $1+l(k+1)$ vector additions ($O(lkm)$), one inner product ($O(n)$) and two simple additions $O(1)$. Therefore, the complexity of IPE-Enc is based on the several matrix-vector multiplications. The IPE-Dec algorithm does $l(k + 1)$ constant-vector multiplication, $l(k + 1)$ vector additions, one inner product between two vectors and a simple addition. Since the vectors are of length $2m$, we have that the complexity of IPE-Dec is $O(lkm)$.

Table 5.1 summarises the size of the main variables and Table 5.2 summarises the complexity of the four algorithms of the scheme described in Section 5.1.1.

Variable	**Size**
Public Key PK	$O(ln^2 \log^2 q)$
Master Key MK	$O(n^2 \log^2 q)$
Secret Key SK	$O(n \log q)$
Ciphertext CT	$O(ln \log^2 q)$

Table 5.1: Key Sizes of the general IPE Scheme

Algorithm	**Complexity**
SetUp	$O(\text{poly}(n))$
KeyGen	$O(\text{poly}(n) + ln^2 \log^2 q)$
Enc	$O(ln^2 \log^3 q)$
Dec	$O(ln \log^2 q)$

Table 5.2: Complexity of the general IPE Scheme

5.2 Lattice-Based Hierarchical Inner Product Encryption

This Section describes the Hierarchical IPE scheme as proposed by Abdalla, De Caro and Mochetti [1]. Section 5.2.1 describes the four algorithms that comprise the HIPE scheme, Sections 5.2.2, 5.2.3, 5.2.4 and 5.2.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

5.2.1 Description

As described in Section 3.1, an Hierarchical Inner Product Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n, 1^\mu)$, $\mathsf{KeyDerive}(PK, SK_{t-1}, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_t)$, $\mathsf{Enc}(PK, M, \boldsymbol{w}_1, \cdots, \boldsymbol{w}_t)$ and $\mathsf{Dec}(PK, SK_t, CT)$. In this section we describe each algorithm as presented by Abdalla et al. [1]. The hierarchy is described by parameter μ and has maximum depth d.

As in the IPE scheme described in Section 5.1, the attributes are vectors instead of identities and the decryption is only possible if the inner product of the pairs of these vectors are zero, i.e., $\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle = 0$, for all $i \in [1, d]$. Each vector has length l and its elements are integers in $[0, q-1]$. For each vector $\boldsymbol{v}_i \in \mathbb{Z}_q^l$, each element $v_{i,j} \in \boldsymbol{v}_i$ can be decomposed in k integers as follows:

$$v_{i,j} = \sum_{\gamma=0}^{k} v_{i,j,\gamma} r^\gamma.$$

As before, $\mathsf{HIPE\text{-}SetUp}$ creates a general lattice and chooses at random the matrices and a vector that will form the public and master keys. But now it will create $ld(1+k)$ random $n \times m$ matrices. $\mathsf{HIPE\text{-}KeyDerive}$ generates the secret key by multiplying each element of each $\boldsymbol{v}_i$, decomposed in k elements, with a matrix, adding all the resulting matrices and concatenating it to the lattice basis. Now, the secret key is the basis for the lattice generate by this concatenation, using the algorithm $\mathsf{SampleBasisLeft}$ described in Section 2.2. Note that $SK_0 = MK$.

$\mathsf{HIPE\text{-}Enc}$ uses the message M, the elements of vectors $\boldsymbol{w}_i$ and the matrices in the public key to create an integer c', a vector $\boldsymbol{c}_0$ and several vectors $\boldsymbol{c}_{i,j,\gamma}$ that will compose the ciphertext for one bit. Finally, $\mathsf{HIPE\text{-}Dec}$ can recover the message from the ciphertext only if $\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle = 0$, for all $i \in [1, t]$.

Let n be the security parameter, μ be the hierarchical parameter, l be the vectors lenght and σ_i (for $i \in [1, d]$) be the Gaussian parameters. Algorithms 5.5, 5.6, 5.7 and 5.8 describe the HIPE scheme.

Algorithm 5.5 HIPE-SetUp(): Setup Algorithm for the HIPE Scheme

```
Input: security parameter $1^n$ and hierarchical parameter $1^\mu$
Output: Public key $PK$ and master key $MK$
    $A, S \leftarrow \mathsf{TrapGen}(q, n, m)$
    $A_{i,j,\gamma} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, for $i \in [1, d]$, $j \in [1, l]$ and $\gamma \in [0, k]$
    $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$
    public key $PK = (A, \{A_{i,j,\gamma}\}, \boldsymbol{u})$
    master key $MK = S$
```

Algorithm 5.6 HIPE-KeyDerive(): Key Generation Algorithm for the HIPE Scheme

```
Input: Public key $PK$, secret key $SK_{t-1}$ and vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_t$
Output: Secret key $SK_t$
    $C_i = \sum_{j=1}^{l} \sum_{\gamma=0}^{k} v_{i,j,\gamma} A_{i,j,\gamma}$
    $C = [C_1|\cdots|C_{t-1}] \in \mathbb{Z}^{n\times(t-1)m}$
    $S_t \leftarrow \mathsf{SampleBasisLeft}([A|C], C_t, S_{t-1}, \sigma_t)$
    secret key $SK_t = S_t \in \mathbb{Z}_q^{n\times(t+1)m}$
```

Algorithm 5.7 HIPE-Enc(): Encryption Algorithm for the HIPE Scheme

```
Input: Public key $PK$, message $M$ and vectors $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_t$
Output: Ciphertext $CT$
    $B \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$
    $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$
    $\boldsymbol{x} \in \overline{\Psi}_\alpha^m$ and $x \in \overline{\Psi}_\alpha$
    $R_{i,j,\gamma} \xleftarrow{\$} \{-1, 1\}^{m\times m}$
    $\boldsymbol{c}_0 = A^\top \boldsymbol{s} + \boldsymbol{x} \in \mathbb{Z}_q^m$
    $\boldsymbol{c}_{i,j,\gamma} = (A_{i,j,\gamma} + r^\gamma w_{i,j} B)^\top \boldsymbol{s} + R_{i,j,\gamma}^\top \boldsymbol{x} \in \mathbb{Z}_q^m$
    $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\boldsymbol{c}_0, \{\boldsymbol{c}_{i,j,\gamma}\}, c')$
```

Algorithm 5.8 HIPE-Dec(): Decryption Algorithm for the HIPE Scheme

```
Input: Public key $PK$, secret key $SK_t$ and ciphertext $CT$
Output: message $M$
    $\boldsymbol{c}_i = \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}\boldsymbol{c}_{i,j,\gamma}$
    $C_i = \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}A_{i,j,\gamma}$
    $C = [C_1|\cdots|C_t]$
    $\sigma = \sigma_t\sqrt{m(t+1)}\omega(\sqrt{\log(tm)})$
    $\boldsymbol{e} = \mathsf{SamplePre}([A|C], S_t, \boldsymbol{u}, \sigma)$
    $z = c' - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{c}_0 \\ \vdots \\ \boldsymbol{c}_t \end{bmatrix} \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

5.2.2 Correctness

On the first step of the correctness we substitute the values of all the vectors $\boldsymbol{c}_{i,j,\gamma}$ in each $\boldsymbol{c}_i$. In this step we recover the value with the inner product of $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$ and we are only able to cancel the term multiplying B if $\langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle = 0$, for all $i \in [1,t]$. Then, we substitute the value of each $\boldsymbol{c}_i$ together with $\boldsymbol{c}_0$ and c' in z. Finally, we cancel the terms $\boldsymbol{u}^\top\boldsymbol{s}$, identify all terms that refer to the "noise" and get the right value of M in z.

Note that $R_i = \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}R_{i,j,\gamma}$ and $R = [R_1|\cdots|R_t]$.

$$\begin{aligned}
\boldsymbol{c}_i &= \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}\boldsymbol{c}_{i,j,\gamma} \\
&= \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}((A_{i,j,\gamma} + r^\gamma w_{i,j}B)^\top \boldsymbol{s} + R_{i,j,\gamma}^\top \boldsymbol{x}) \\
&= \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}A_{i,j,\gamma}^\top \boldsymbol{s} + \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}(r^\gamma w_{i,j}B)^\top \boldsymbol{s} + \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}R_{i,j,\gamma}^\top \boldsymbol{x} \\
&= C_i^\top \boldsymbol{s} + \langle \boldsymbol{v}_i, \boldsymbol{w}_i \rangle B^\top \boldsymbol{s} + R_i^\top \boldsymbol{x} \\
&= C_i^\top \boldsymbol{s} + R_i^\top \boldsymbol{x}
\end{aligned}$$

$$
\begin{aligned}
z &= c' - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{c}_0 \\ \boldsymbol{c}_1 \\ \vdots \\ \boldsymbol{c}_t \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 - \boldsymbol{e}^\top \begin{bmatrix} A^\top \boldsymbol{s} + \boldsymbol{x} \\ C_1^\top \boldsymbol{s} + R_1^\top \boldsymbol{x} \\ \vdots \\ C_t^\top \boldsymbol{s} + R_t^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 - \boldsymbol{e}^\top \begin{bmatrix} A^\top \boldsymbol{s} + \boldsymbol{x} \\ C^\top \boldsymbol{s} + R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{e}^\top [A|C]^\top \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= x + M\lfloor q/2 \rfloor - \boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}
$$

Note that for the correct decryption the error term must be less than $q/4$.

5.2.3 Security

In this section we prove the following theorem.

Theorem 5.2. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 5.2.1 is IND-wAH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.HIPE-SetUp$(1^n, 1^\mu, \boldsymbol{w}_1^\star, \cdots, \boldsymbol{w}_d^\star)$: The algorithm chooses random $A \in \mathbb{Z}_q^{n \times m}$, $R_{i,j,\gamma}^\star \in \{-1, 1\}^{m \times m}$ and $\boldsymbol{u} \in \mathbb{Z}_q^n$ and it uses $\mathsf{TrapGen}(q, n, m)$ to generate $B^\star \in \mathbb{Z}_q^{n \times m}$ and the basis $S^\star \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^\perp(B^\star)$. It then defines, for $i \in [1, d]$, $j \in [1, l]$ and $\gamma \in [0, k]$, $A_{i,j,\gamma} \leftarrow AR_{i,j,\gamma}^\star - r^\gamma w_{i,j}^\star B^\star$ and outputs $PK = (A, \{A_{i,j,\gamma}\}, \boldsymbol{u})$ and $MK = (\{R_{i,j,\gamma}^\star\}, B^\star, S^\star)$.

Sim.HIPE-KeyDerive$(PK, MK, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_t)$: Secret keys are now created using the trapdoor $S^\star$, sampled by the $\mathsf{SampleBasisRight}$ algorithm. It outputs $SK_t = S$, a basis of lattice $\Lambda_q^\perp(A|AR_1^\star - \langle \boldsymbol{v}_1, \boldsymbol{w}_1^\star \rangle B^\star| \cdots |AR_t^\star - \langle \boldsymbol{v}_t, \boldsymbol{w}_t^\star \rangle B^\star)$, using

$S \leftarrow \mathsf{SampleBasisRight}(A, B_{\boldsymbol{v}}^\star, R^\star, S^\star, \sigma_t)$,

with $R^\star = [R_1^\star|\cdots|R_t^\star\ ]$, for $R_i^\star = \sum_{j=1}^{l}\sum_{\gamma=0}^{k} v_{i,j,\gamma}R_{i,j,\gamma}^\star$
and $B_{\boldsymbol{v}}^\star = [\langle \boldsymbol{v}_1, \boldsymbol{w}_1^\star\rangle B^\star|\cdots|\langle \boldsymbol{v}_t, \boldsymbol{w}_t^\star\rangle B^\star]$.

Note that $\langle \boldsymbol{v}_i, \boldsymbol{w}_i\rangle$ must be non-zero, for all $i \in [1,t]$, for the algorithm SampleBasisRight to work properly.

Sim.HIPE-Enc$(PK, M, \boldsymbol{w}_1^\star, \cdots, \boldsymbol{w}_t^\star)$: The algorithm differs from HIPE-Enc in the sense that it uses matrices $R_{i,j,\gamma}^\star$ and $B^\star$ instead of matrices $R_{i,j,\gamma}$ and B.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs HIPE-SetUp, answers $\mathcal{A}$'s secret key queries using the HIPE-KeyDerive algorithm, and generates the challenge ciphertext using the HIPE-Enc with vectors $\boldsymbol{w}_1^{\star 0}, \cdots$, $\boldsymbol{w}_t^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.HIPE-SetUp with vectors $\boldsymbol{w}_1^{\star 0}, \cdots, \boldsymbol{w}_d^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HIPE-KeyDerive, and generates the challenge ciphertext using the Sim.HIPE-Enc algorithm with $\boldsymbol{w}_1^{\star 0}, \cdots, \boldsymbol{w}_t^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.HIPE-SetUp with vectors $\boldsymbol{w}_1^{\star 0}, \cdots, \boldsymbol{w}_d^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HIPE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.HIPE-SetUp with vectors $\boldsymbol{w}_1^{\star 1}, \cdots, \boldsymbol{w}_d^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HIPE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.HIPE-SetUp with vectors $\boldsymbol{w}_1^{\star 1}, \cdots, \boldsymbol{w}_d^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HIPE-KeyDerive, and generates the challenge ciphertext using the Sim.HIPE-Enc algorithm with $\boldsymbol{w}_1^{\star 1}, \cdots, \boldsymbol{w}_t^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs HIPE-SetUp, answers $\mathcal{A}$'s secret key queries using the HIPE-KeyDerive algorithm, and generates the challenge ciphertext using the HIPE-Enc with vectors $\boldsymbol{w}_1^{\star 1}, \cdots$, $\boldsymbol{w}_t^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

For simplicity reasons, assume that every time we refer to matrices $A_{i,j,\gamma}$ and $R_{i,j,\gamma}^\star$ and vectors $\boldsymbol{c}_{i,j,\gamma}$ we are referring to all matrices for $i \in [1,d]$, $j \in [1,l]$ and $\gamma \in [0,k]$.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 5.4. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, matrix A is generated by $\mathsf{TrapGen}$ and matrices $A_{i,j,\gamma}$ are uniformly random in $\mathbb{Z}_q^{n\times m}$. Instead, in Game 1, A is chosen uniformly at random and we have $A_{i,j,\gamma} \leftarrow AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star$, where $B^\star$ is generated by $\mathsf{TrapGen}$ and the matrices $R^\star_{i,j,\gamma}$ are uniformly and independently chosen at random in $\{-1,1\}^{m\times m}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathbb{Z}_q^n$.

Secret keys In Game 0, the secret key for level t is a basis of lattice given by $\Lambda_q^\perp\left(A|C_1 + r^\gamma w_{1,j} B|\dots|C_t + r^\gamma w_{t,j} B\right)$ sampled using the $\mathsf{SampleBasisLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleBasisRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext In both games the challenge ciphertext components c' and $\boldsymbol{c}_0$ are computed the same way but, in Game 0, the challenge ciphertext component $\boldsymbol{c}_{i,j,\gamma}$ is computed as follows:

$$\boldsymbol{c}_{i,j,\gamma} = (A_{i,j,\gamma} + r^\gamma w^\star_{i,j} B^\star)^\top \boldsymbol{s} + R^{\star\top}_{i,j,\gamma}\boldsymbol{x} \in \mathbb{Z}_q^m \ .$$

On the other hand, in Game 1, we have:

$$\begin{aligned}\boldsymbol{c}_{i,j,\gamma} &= (A_{i,j,\gamma} + r^\gamma w^\star_{i,j} B^\star)^\top \boldsymbol{s} + R^{\star\top}_{i,j,\gamma}\boldsymbol{x} \\ &= (AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star + r^\gamma w^\star_{i,j} B^\star)^\top \boldsymbol{s} + R^{\star\top}_{i,j,\gamma}\boldsymbol{x} \ . \\ &= (AR^\star_{i,j,\gamma})^\top \boldsymbol{s} + R^{\star\top}_{i,j,\gamma}\boldsymbol{x} \in \mathbb{Z}_q^m\end{aligned}$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(A, \{A_{i,j,\gamma}\}, \{\boldsymbol{c}_{i,j,\gamma}\})$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemmas A.4 and A.5 we have that the following two distributions are statistically indistinguishable for every fixed matrix $B^\star$, every $\boldsymbol{w}^\star_i$ and every vector $\boldsymbol{x} \in \mathbb{Z}_q^m$:

$$\left(A, A_{i,j,\gamma}, R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\right) \approx_s \left(A, AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star, R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\right) \ .$$

Matrices $R^\star_{i,j,\gamma}$ are chosen independently for every i, j, γ, therefore the joint distributions of these quantities are also statistically close:

$$\left(A, \{A_{i,j,\gamma}\}, \{R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\}\right) \approx_s \left(A, \{AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star\}, \{R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\}\right) \ .$$

Since $(AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star)^\top \boldsymbol{s}$ is statistically close to $A^\top_{i,j,\gamma}\boldsymbol{s}$, it is possible to add each term to each side of the equation:

$$\begin{aligned}\left(A, \{A_{i,j,\gamma}\}, \{A^\top_{i,j,\gamma}\boldsymbol{s} + R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\}\right) \approx_s \\ \left(A, \{AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star\}, \{(AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star)^\top \boldsymbol{s} + R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\}\right)\end{aligned} \ .$$

Then, we add $(r^\gamma w^\star_{i,j} B^\star)^\top \boldsymbol{s}$ to each side of the equation:

$$\begin{array}{c}\big(A,\{A_{i,j,\gamma}\},\{(A_{i,j,\gamma}+r^{\gamma}w^{\star}_{i,j}B^{\star})^{\top}\boldsymbol{s}+R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\}\big)\approx_s \\ \big(A,\{AR^{\star}_{i,j,\gamma}-r^{\gamma}w^{\star}_{i,j}B^{\star}\},\{(AR^{\star}_{i,j,\gamma})^{\top}\boldsymbol{s}+R^{\star\top}_{i,j,\gamma}\boldsymbol{x}\}\big)\end{array}.$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 5.5. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE*.*

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $m+1$ LWE challenge pairs $(\boldsymbol{a}_j,y_j)\in\mathbb{Z}_q^n\times\mathbb{Z}_q$, where either $y_j=\langle\boldsymbol{a}_j,\boldsymbol{s}\rangle+x_j$ for a random $\boldsymbol{s}\in\mathbb{Z}_q^n$ and a noise term $x_j\leftarrow\Psi_\alpha$, or y_j is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_j,y_j)$. The i-th column of matrix A will be the vector $\boldsymbol{a}_i$, for $1\le i\le m$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrices $A_{i,j,\gamma}$ are still calculated as in Sim.HIPE-SetUp, i.e., $A_{i,j,\gamma}\leftarrow AR^{\star}_{i,j,\gamma}-r^{\gamma}w^{\star}_{i,j}B^{\star}$.

Secret keys All private-key extraction queries are answered using Sim.HIPE-KeyDerive.

Challenge Ciphertext The ciphertext $CT=(\boldsymbol{c}^{\star}_0,\boldsymbol{c}^{\star}_{i,j,\gamma},c'^{\star})$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_j,y_j)$, with $\boldsymbol{c}^{\star}_0=(y_1,\ldots,y_m)$, $c'^{\star}=y_0+M\lfloor q/2\rceil$ and $c^{\star}_{i,j,\gamma}=R^{\star\top}_{i,j,\gamma}\boldsymbol{c}^{\star}_0$. If we have $y_j=\langle\boldsymbol{a}_j,\boldsymbol{s}\rangle+x_j$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_j is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2 . If $y_j=\langle\boldsymbol{a}_j,\boldsymbol{s}\rangle+x_j$, then

$(y_1,\ldots,y_m)=(\langle\boldsymbol{a}_1,\boldsymbol{s}\rangle+x_1,\ldots,\langle\boldsymbol{a}_m,\boldsymbol{s}\rangle+x_m)=A^{\top}\boldsymbol{s}+\boldsymbol{x}$.

Therefore, for Game 1 we have

$$\begin{aligned}\boldsymbol{c}^{\star}_0&=A^{\top}\boldsymbol{s}+\boldsymbol{x}\\&=(y_1,\ldots,y_m)\ ,\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{c}^\star_{i,j,\gamma} &= (A_{i,j,\gamma} + r^\gamma w_{i,j} B)^\top \boldsymbol{s} + R^\top_{i,j,\gamma}\boldsymbol{x} \\
&= (AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star + r^\gamma w_{i,j} B)^\top \boldsymbol{s} + R^\top_{i,j,\gamma}\boldsymbol{x} \\
&= (AR^\star_{i,j,\gamma})^\top \boldsymbol{s} + R^\top_{i,j,\gamma}\boldsymbol{x} \\
&= R^{\star\top}_{i,j,\gamma}(A^\top \boldsymbol{s} + \boldsymbol{x}) \\
&= R^{\star\top}_{i,j,\gamma}\boldsymbol{c}^\star_0 \ .
\end{aligned}$$

If y_j is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_j is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_j = \langle \boldsymbol{a}_j, \boldsymbol{s}\rangle + x_j$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 5.6. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. A and $\boldsymbol{u}$ are random and $A_{i,j,\gamma} \leftarrow AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star$, with $\boldsymbol{w}^\star_i = \boldsymbol{w}^{\star 0}_i$ for Game 2 and $\boldsymbol{w}^\star_i = \boldsymbol{w}^{\star 1}_i$ for Game 3.

Secret keys All private-key extraction queries are answered using Sim.HIPE-KeyDerive. The only difference is, again, that for Game 2, $\boldsymbol{w}^\star_i = \boldsymbol{w}^{\star 0}_i$ and, for Game 3, $\boldsymbol{w}^\star_i = \boldsymbol{w}^{\star 1}_i$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrix $A_{i,j,\gamma}$. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of $A_{i,j,\gamma}$. From Lemmas A.4 and A.5 we can prove that each $A_{i,j,\gamma}$ for each game is statistically close to a uniformly random matrix, because

$A_{i,j,\gamma} \leftarrow AR^\star_{i,j,\gamma} - r^\gamma w^\star_{i,j} B^\star$.

□

5.2.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma_t\sqrt{(t+1)m}$ from Lemma A.1 and $\|R_{i,j,\gamma}\boldsymbol{e}\| \leq 12\sqrt{tm+m}\|\boldsymbol{e}\|$ from Lemma A.2. We also know $v_{i,j,\gamma} \in [0, r-1]$ because of the decomposition therefore, for $\max(t) = d$, $\boldsymbol{e} = \begin{bmatrix} \boldsymbol{e}_1 \\ \boldsymbol{e}_2 \end{bmatrix}$, $R = [R_1| \cdots |R_t]$ and $R_i = \sum \sum v_{i,j,\gamma} R_{i,j,\gamma}$:

$$\begin{aligned} &\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq \sigma_t \sqrt{(t+1)m} + dl(k+1)r12\sqrt{(t+1)m}\sigma_t\sqrt{(t+1)m} \qquad \text{so} \\ &\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq O(d^2 lkr\sigma_t m) \ . \end{aligned}$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x} \rangle \leq \|\boldsymbol{y}\| q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\begin{aligned} \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x} \rangle &\leq O(d^2 lkr\sigma_t m) q\alpha_t \omega(\sqrt{\log m}) + O(d^2 lkr\sigma_t m)\sqrt{m}/2 \\ \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x} \rangle &\leq \widetilde{O}(d^2 lkr\sigma_t mq\alpha) + O(d^2 lkr\sigma_t m^{3/2}) \ . \end{aligned}$$

To ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned} &\boldsymbol{e}^\top \begin{bmatrix} \boldsymbol{x} \\ R^\top \boldsymbol{x} \end{bmatrix} < q/4 \\ &x - \boldsymbol{e}_1^\top \boldsymbol{x} - \boldsymbol{e}_2^\top R^\top \boldsymbol{x} < q/4 \\ &x - (\boldsymbol{e}_1 + R\boldsymbol{e}_2)^\top \boldsymbol{x} < q/4 \\ &x - \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x} \rangle < q/4 \\ &\widetilde{O}(d^2 lkr\sigma_t mq\alpha) + O(d^2 lkr\sigma_t m^{3/2}) < q/4 \end{aligned}$$

To ensure that σ_t is sufficiently large for SampleBasisLeft and SampleBasisRight (Theorems 2.10 and 2.11), we have

$$\sigma_t > \|S\|\sqrt{m}\omega(\sqrt{\log m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned} &m \geq 6n \log q \qquad \text{and} \\ &\|S\| \leq O(n \log q) \ . \end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha \ .$$

Therefore, we need to set the parameters as

$$\begin{aligned} &r = 2, \\ &k = \log q, \\ &m = n^{\delta+1}, \\ &\sigma = m\omega(\sqrt{\log n}), \\ &q = lm^{2.5}\omega(\sqrt{\log n} \log q), \\ &\alpha = (lm^2\omega(\sqrt{\log n}) \log q)^{-1}, \end{aligned}$$

with δ such that $n^{\delta} = O(\log q)$.

5.2.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 5.2.1. Note that for security parameter n, vector length l, hierarchy's maximum depth d and modulus q, we have $m = O(n \log q)$ and $k = \log q$ (see Section 5.2.4).

As always, the master key MK is just an $m \times m$ matrix, therefore its size is m^2. The public key PK is comprised of a vector of length n and $dl(k+1)+1$ matrices of size $n \times m$; therefore its size is $n + dl(k+1)nm + nm$, which is $O(dlkmn)$. The secret key is a matrix of size $n \times (t+1)m$, with $\max(t) = d$; therefore its size is $O(dnm)$. Finally, the ciphertext is comprised of an integer and $(1 + ld(k+1))$ vectors of length m; therefore its size is $1 + m + ld(k+1)m$, which is $O(dlkm)$.

The complexity of HIPE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, therefore, HIPE-SetUp is also polynomial. The complexity of HIPE-KeyDerive is based on the complexity of the SampleBasesLeft algorithm plus $tl(k+1)$ constant-matrix multiplications $(O(dlknm))$ and $tl(k+1)$ matrix additions $(O(dlknm))$. As before, we have that the SampleBasisLeft algorithm is polynomial, by Theorem 2.10.

The HIPE-Enc algorithm does $1 + 2tl(k+1)$ matrix-vector multiplications $(O(nm + dlknm + dlkm^2))$, $tl(k+1)$ matrix addictions $(O(dlkmn))$, $1 + tl(k+1)$ vector additions $(O(dlkm))$, one inner product $(O(n))$ and two simple additions $O(1)$. Therefore, the complexity of HIPE-Enc is based on the several matrix-vector multiplications. The HIPE-Dec algorithm does $tl(k+1)$ constant-vector multiplications $(O(dlkm))$, $tl(k+1)$ vector additions $(O(dlkm))$, $tl(k+1)$ constant-matrix multiplications $(O(dlkmn))$, $tl(k+1)$ matrix additions $(O(dlkmn))$, one inner product between two vectors $(O(dm))$, a few simple additions and multiplications and calls the SamplePre algorithm. We have, by Theorem 2.6, that SamplePre is polynomial, therefore the complexity of the HIPE-Dec algorithm is based on SamplePre and the constant-matrix multiplications. Note that $\max(t) = d$.

Table 5.3 summarises the size of the main variables and Table 5.4 summarises the complexity of the four algorithms of the scheme described in Section 5.2.1.

Variable	**Size**
Public Key PK	$O(dln^2 \log^2 q)$
Master Key MK	$O(n^2 \log^2 q)$
Secret Key SK	$O(dn^2 \log q)$
Ciphertext CT	$O(dln \log^2 q)$

Table 5.3: Key Sizes of the general HIPE Scheme

Algorithm	**Complexity**
SetUp	$O(\text{poly}(n))$
KeyDerive	$O(\text{poly}(n) + dln^2 \log^2 q)$
Enc	$O(dln^2 \log^3 q)$
Dec	$O(\text{poly}(n) + dln^2 \log^2 q)$

Table 5.4: Complexity of the general HIPE Scheme

Chapter 6

Hidden Vector Encryption

In this chapter we describe the lattice-based HVE scheme based on the FBE scheme proposed by Agrawal, Boyen, Vaikuntanathan, Voulgaris and Wee [6]. Section 6.1 describes the general HVE scheme and Section 6.2 describes our contribution, a hierarchical version of the same scheme, as described by Mochetti et al. [53].

6.1 Lattice-Based Hidden Vector Encryption

This Section reviews the HVE scheme based on the FBE scheme proposed by Agrawal, Boyen, Vaikuntanathan, Voulgaris and Wee [6]. Section 6.1.1 describes the four algorithms that comprise the hidden vector scheme, Sections 6.1.2, 6.1.3, 6.1.4 and 6.1.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

6.1.1 Description

As described in Section 3.7, an Hidden Vector Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n)$, $\mathsf{KeyGen}(PK, MK, \boldsymbol{v})$, $\mathsf{Enc}(PK, M, \boldsymbol{w})$ and $\mathsf{Dec}(PK, SK, CT)$. In this section we describe each algorithm similar to the one as presented by Agrawal et al. [6].

This scheme is similar to the IBE scheme described in Section 4.1, but now the attributes are vectors $\boldsymbol{v} \in \{0, 1, \star\}^l$ and $\boldsymbol{w} \in \{0, 1\}^l$ and the decryption is only possible if $v_j = w_j, \forall i$ such that $v_j \neq \star$. To make that possible, Shamir' Secret Sharing Scheme, described in Appendix B, is used. The Secret Sharing Scheme used is composed of two algorithms: $\mathsf{SplitVector}$, that divides a data vector into n vector pieces, and $\mathsf{FindLagrangianCoef}$, that find the lagrangian coefficients so it is possible to recover the vector data using $k \leq n$ pieces.

HVE-SetUp creates several general lattices, two for each coefficient of the attribute vector and chooses at random a vector that will form the public key. HVE-KeyGen generates the secret key by dividing each element of vector $\boldsymbol{u}$ using the SplitVector algorithm and creating vectors $\boldsymbol{u}_j$. The secret key will be the l vectors $\boldsymbol{e}_j$ created by the SamplePre algorithm, described in Section 2.2, using the correspondent matrices as the lattice basis; therefore $\boldsymbol{e}_j \in \Lambda_q^{\boldsymbol{u}_j}(A_{j,v_j})$, i.e., $A_{j,v_j}\boldsymbol{e}_j = \boldsymbol{u}_j$, for $j \in [1, l]$.

HVE-Enc uses the message M, the attribute vector $\boldsymbol{w}$ and the matrices in the public key to create an integer c' and several vectors $\boldsymbol{c}_j$ that will compose the ciphertext for one bit. Finally, HVE-Dec can recover the message from the ciphertext only if $v_j = w_j, \forall j$ such that $v_j \neq \star$, calculating the Lagrangian coefficients such as the Join algorithm does.

Let n be the security parameter, σ be the Gaussian parameter and k be the threshold. Algorithms 6.1, 6.2, 6.3 and 6.4 describe the HVE scheme.

Algorithm 6.1 HVE-SetUp(): Setup Algorithm for the HVE Scheme

```
Input: Security parameter $1^n$
Output: Public key $PK$ and master key $MK$
  $A_{j,b}, S_{j,b} \leftarrow \mathsf{TrapGen}(q, n, m)$ for $j \in [1, l]$ and $b \in \{0, 1\}$
  $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$
  public key $PK = (\{A_{j,b}\}, \boldsymbol{u})$
  master key $MK = (\{S_{j,b}\})$
```

Algorithm 6.2 HVE-KeyGen(): Key Generation Algorithm for the HVE Scheme

```
Input: Public key $PK$, master key $MK$, vector $\boldsymbol{v}$
Output: Secret key $SK$
  $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_l \leftarrow \mathsf{SplitVector}(\boldsymbol{u}, k)$
  if  $(v_j = \star)$: $b \xleftarrow{\$} \{0, 1\}$
  else: $b \leftarrow v_j$
  $\boldsymbol{e}_j \leftarrow \mathsf{SamplePre}(A_{j,b}, S_{j,b}, \boldsymbol{u}_j, \sigma) \in \mathbb{Z}_q^m$
  secret key $SK = (\{\boldsymbol{e}_j\})$
```

Algorithm 6.3 HVE-Enc(): Encryption Algorithm for the HVE Scheme

```
Input: Public key $PK$, message $M$ and vector $\boldsymbol{w}$
Output: Ciphertext $CT$
    $\beta \leftarrow (l!)^2$
    $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$
    $\boldsymbol{x}_j \in \overline{\Psi}_\alpha^m$, for $j \in [1, l]$ and $x \in \overline{\Psi}_\alpha$
    $\boldsymbol{c}_j \leftarrow A_{j,w_j}^\top \boldsymbol{s} + \beta \boldsymbol{x}_j \in \mathbb{Z}_q^m$
    $c' \leftarrow \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\{\boldsymbol{c}_j\}, c')$
```

Algorithm 6.4 HVE-Dec(): Decryption Algorithm for the HVE Scheme

```
Input: Public key $PK$, secret key $SK$ and ciphertext $CT$
Output: message $M$
    if $v_j \neq \star$: $\mathbb{G} \leftarrow \mathbb{G} \cup \{j\}$
    $\boldsymbol{l} \leftarrow \mathsf{FindLagrangianCoef}(\mathbb{G})$
    $z \leftarrow c' - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \boldsymbol{c}_\rho \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

6.1.2 Correctness

First recall that, based on Shamir's Secret Sharing Scheme (see Appendix B), we have

$$\sum_{\rho \in \mathbb{G}} l_\rho A_{\rho, v_\rho} \boldsymbol{e}_\rho = \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{u}_\rho = \boldsymbol{u}$$

Now the correctness is straightforward. We just substitute the values of $\boldsymbol{c}_j$ and c' in z and based on the Shamir's Secret Sharing Scheme we can recover the vector $\boldsymbol{u}$. Therefore, it is possible to cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get

the right value of M in z.

$$\begin{aligned}
z &= c' - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \boldsymbol{c}_\rho \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top (A_{\rho,w_\rho}^\top \boldsymbol{s} + \beta \boldsymbol{x}_\rho) \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \sum_{\rho \in \mathbb{G}} (l_\rho A_{\rho,w_\rho} \boldsymbol{e}_\rho)^\top \boldsymbol{s} - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \beta \boldsymbol{x}_\rho \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \beta \boldsymbol{x}_\rho \mod q \\
&= M\lfloor q/2 \rfloor + \beta x - \beta \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \boldsymbol{x}_\rho \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}$$

Note that for the correct decryption the error term must be less than $q/4$.

6.1.3 Security

In this section we prove the following theorem.

Theorem 6.1. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 6.1.1 is IND-wAH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.HVE-SetUp$(1^n, \boldsymbol{w}^\star)$: The l matrices $A_{j,w_j^\star}$, i.e., matrices that refer to the bits that belong to $\boldsymbol{w}^\star$ are chosen randomly in $\mathbb{Z}_q^{n\times m}$, and the algorithm $\mathsf{TrapGen}(q,n,m)$ is used to generate the l matrices $A_{j,\overline{w_j^\star}}$ and $S_{j,\overline{w_j^\star}}$, i.e., matrices that refer to the bits that do not belong to $\boldsymbol{w}^\star$. The vector $\boldsymbol{u} \in \mathbb{Z}_q^n$ is still chosen randomly.

Sim.HVE-KeyGen$(PK, MK, \boldsymbol{v})$: The secret keys are now created using two algorithms, SamplePre and SampleLattice. The l shares of $\boldsymbol{u}$ are now represented as

$$\boldsymbol{u}_j = \sum_{\gamma=0}^{k-1} \boldsymbol{a}_\gamma j^\gamma,$$

for $j \in [1, l]$, with $\boldsymbol{a}_0 = \boldsymbol{u}$.

First choose random values in $\{0,1\}$ for all values where $v_j = \star$. Assume without lot of generality that the first $\gamma < k$ bits of $\boldsymbol{v}$ are now equal to $\boldsymbol{w}$.

For $j \in [1, \gamma]$, such that $v_j = w_j^\star$, we have $\boldsymbol{e}_j \leftarrow \mathsf{SampleLattice}(A_{j,w_j}, \sigma)$, and $\boldsymbol{u}_j = A_{j,v_j} \boldsymbol{e}_j$.

For $j \in [\gamma+1, k-1]$, we randomly choose $\boldsymbol{u}_j$, determining all coefficients $\boldsymbol{a}_j$, and therefore, calculating all l shares $\boldsymbol{u}_j$.

Finally, for $j \in [k+1, l]$ we have

$\boldsymbol{e}_j \leftarrow \mathsf{SamplePre}(A_{j,\overline{w_j^\star}}, S_{j,\overline{w_j^\star}}, \boldsymbol{u}_j, \sigma)$.

Note that we must have less than k bits of $\boldsymbol{v}$ equal to $\boldsymbol{w}$ and different of $\star$ for the algorithm to work properly.

Sim.HVE-Enc$(PK, M, \boldsymbol{w}^\star)$: The algorithm differs from HVE-Enc in the way that it uses constant β to calculate the ciphertext. Now we have

$c'^\star = \beta(\boldsymbol{u}^\top \boldsymbol{s} + x) + M\lfloor q/2 \rfloor$

and

$\boldsymbol{c}_j^\star = \beta(A_{j,w_j^\star}^\top \boldsymbol{s} + \boldsymbol{x}_j)$.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs HVE-SetUp, answers $\mathcal{A}$'s secret key queries using the HVE-KeyGen algorithm, and generates the challenge ciphertext using the HVE-Enc with vector $\boldsymbol{w}^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.HVE-SetUp with vector $\boldsymbol{w}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HVE-KeyGen, and generates the challenge ciphertext using the Sim.HVE-Enc algorithm with $\boldsymbol{w}^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.HVE-SetUp with vector $\boldsymbol{w}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HVE-KeyGen, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.HVE-SetUp with vector $\boldsymbol{w}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HVE-KeyGen, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.HVE-SetUp with vector $\boldsymbol{w}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HVE-KeyGen, and generates the challenge ciphertext using the Sim.HVE-Enc algorithm with $\boldsymbol{w}^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs HVE-SetUp, answers $\mathcal{A}$'s secret key queries using the HVE-KeyGen algorithm, and generates the challenge ciphertext using the HVE-Enc with vector $\boldsymbol{w}^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 6.1. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp The public parameters are generated almost in the same way in both games. The vector $\boldsymbol{u}$ in both games is random. The matrices $A_{j,b}$ are chosen by the algorithm $\mathsf{TrapGen}$ in Game 0 and are chosen either by random or by the same algorithm according to the value of $\boldsymbol{w}_0^\star$ in Game 1.

Secret keys In Game 0, the secret key for vector $\boldsymbol{v}$ is a set of vectors $\boldsymbol{e}_j \in \Lambda_q^{\boldsymbol{u}}(A_{j,b})$, where b is chosen according to each value of v_j. Each vector is sampled using the $\mathsf{SamplePre}$ algorithm. The same happens in Game 1. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

The public parameters are generated almost in the same way in both games. The vector $\boldsymbol{u}$ in both games are random. The matrices $A_{j,b}$ are chosen by the algorithm $\mathsf{TrapGen}$ in Game 0 and are chosen either by random or by the same algorithm according to the value of $\boldsymbol{w}_0^\star$ in Game 1.

Since the matrices generated by $\mathsf{TrapGen}$ are indistinguishable for random, the public parameters are also indistinguishable in both games and we have that the indistinguishability of the two games depends on the ciphertext. For Game 0 we have

$\boldsymbol{c}_j = A^\top_{j,w^\star_j}\boldsymbol{s} + \beta\boldsymbol{x}_j$ and $c' = \boldsymbol{u}^\top\boldsymbol{s} + \beta x + M\lfloor q/2\rfloor$;

and for Game 1 we have

$\boldsymbol{c}^\star_j = \beta(A^\top_{j,w^\star_j}\boldsymbol{s} + \boldsymbol{x}_j)$ and $c'^\star = \beta(\boldsymbol{u}^\top\boldsymbol{s} + x) + M\lfloor q/2\rfloor$.

We know that vectors $\boldsymbol{s}$ and $\boldsymbol{u}$ are random and matrices $A_{j,b}$ are indistinguishable from random, therefore by Lemma A.7 we have that the two games are indistinguishable from each other.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 6.2. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE.

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $lm + 1$ LWE challenge pairs $(\boldsymbol{a}_k, y_k) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where either $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s}\rangle + x_k$ for a random $\boldsymbol{s} \in \mathbb{Z}_q^n$ and a noise term $x_k \leftarrow \Psi_\alpha$, or y_k is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_k, y_k)$. The i-th column of matrix $A_{j,w^\star_j}$ will be the vector $\boldsymbol{a}_{(j-1)m+i+1}$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrices $A_{j,\overline{w^\star_j}}$ and shares of $\boldsymbol{u}$ are calculated as in $\mathsf{Sim.HVE\text{-}SetUp}$.

Secret keys All private-key extraction queries are answered using $\mathsf{Sim.HVE\text{-}KeyGen}$.

Challenge Ciphertext The ciphertext $CT = (\boldsymbol{c}_j^\star, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_k, y_k)$, with $\boldsymbol{c}_j^\star = \beta(y_{(j-1)m+1}, \ldots, y_{jm+1})$, for $j \in [1, l]$, and $c'^\star = \beta y_0 + M\lfloor q/2 \rceil$. If we have $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s} \rangle + x_k$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_k is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2. If $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s} \rangle + x_k$, then

$$\begin{aligned}\beta(y_{(j-1)m+1}, \ldots, y_{jm+1}) &= (\beta\langle \boldsymbol{a}_{(j-1)m+1}, \boldsymbol{s}\rangle + \beta x_{(j-1)m+1}, \ldots, \beta\langle \boldsymbol{a}_{jm+1}, \boldsymbol{s}\rangle + \beta x_{jm+1}) \\ &= \beta A_{j,w_j^\star}^\top \boldsymbol{s} + \beta \boldsymbol{x}_j \\ &= \beta(A_{j,w_j^\star}^\top \boldsymbol{s} + \boldsymbol{x}_j)\end{aligned}$$

Therefore, for Game 1 we have

$$\begin{aligned}\boldsymbol{c}_j^\star &= \beta(A_{j,w_j^\star}^\top \boldsymbol{s} + \boldsymbol{x}) \\ &= \beta(y_{(j-1)m+1}, \ldots, y_{jm} + 1)\ ,\end{aligned}$$

If all y_k is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_k is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s} \rangle + x_k$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 6.3. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated almost in the same way in both games. The vector $\boldsymbol{u}$ in both games are random and the matrices $A_{j,b}$ are chosen at random or by the algorithm $\mathsf{TrapGen}$ according to the value of the bits in $\boldsymbol{w}^{\star 0}$ for Game 2 and $\boldsymbol{w}^{\star 1}$ for Game 3.

Secret keys All private-key extraction queries are answered using $\mathsf{Sim.HVE\text{-}KeyGen}$. The only difference is, again, that for Game 2, $\boldsymbol{w}^\star = \boldsymbol{w}^{\star 0}$ and, for Game 3, $\boldsymbol{w}^\star = \boldsymbol{w}^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated or generated by algorithm $\mathsf{TrapGen}$ in both games. Since the matrices generated by $\mathsf{TrapGen}$ are indistinguishable from random, we have that the two games are indistinguishable from each other.

□

6.1.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma\sqrt{m}$ from Lemma A.1, $|x| \leq 2m$ from Lemma A.6, $|\beta l_\rho| \leq \beta^2 \leq (l!)^4$ from Lemma B.1 and $\beta = (l!)^2$ from construction. Therefore, to ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned}
&|\beta x - \sum_{\rho\in\mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \beta \boldsymbol{x}_\rho| < q/4 \\
&\beta|x| + \sum_{\rho\in\mathbb{G}} l_\rho \beta |\boldsymbol{e}_\rho^\top \boldsymbol{x}_\rho| < q/4 \\
&\beta 2m + \sum_{\rho\in\mathbb{G}} \beta^2 2m\sigma\sqrt{m} < q/4 \\
&(l!)^2 2m + l(l!)^4 2m\sigma\sqrt{m} < q/4 \\
&2l(l!)^4 2m\sigma\sqrt{m} < q/4 \\
&4l(l!)^4 m^{1.5}\sigma < q/4 \\
&2^{5l} m^{1.5}\sigma < q/4
\end{aligned}$$

Note that $(l!)^4 \leq l^{4l} \leq 2^{5l}$. To ensure that σ is sufficiently large for SamplePre and SampleLattice (Theorems 2.6 and 2.4), we have

$$\sigma > \|S\|\omega(\sqrt{\log m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned}
&m \geq 6n\log q \qquad \text{and} \\
&\|S\| \leq O(n\log q)\ .
\end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha\ .$$

Therefore, we need to set the parameters as

$$\begin{aligned}
&m = n^{1.5} \\
&\sigma = m\log m, \\
&q = 2^{5l} m^3 \log m, \\
&\alpha = (2^{5l}\mathrm{poly}(n))^{-1}.
\end{aligned}$$

6.1.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 6.1.1. Note that for security parameter n, attribute vector of length l and modulus q, we have $m = O(n \log q)$ (see Section 6.1.4).

The master key MK is comprised of $2l$ matrices of size $m \times m$, therefore its size is $2lm^2$. The public key PK is comprised of $2l$ matrices of size $n \times m$ and a vector of length n; therefore its size is $2lmn + n$, which is $O(lmn)$. The secret key is comprised of l vectors of length m; therefore its size is $O(lm)$. Finally, the ciphertext is comprised of an integer and l vectors of length m; therefore its size is $1 + ml$, which is also $O(lm)$.

The complexity of HVE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, since it is executed $2l$ times, the complexity will be $2l \cdot \text{poly}(n)$. The complexity of HVE-KeyGen is based on the complexity of the SamplePre algorithm that are executed l times and the SplitVetor algorithm. As before, we have that the SamplePre algorithm is polynomial, by Theorem 2.6 and the SplitVetor algorithm is linear, therefre the complexity is given by the SamplePre algorithm.

The HVE-Enc algorithm does l matrix-vector multiplications ($O(lnm)$), l constant-vector multiplications ($O(lm)$), l vector additions ($O(lm)$), one inner product ($O(n)$) and two simple additions $O(1)$. Therefore, the complexity of HVE-Enc is based on the several matrix-vector multiplications. The HVE-Dec algorithm calls the FindLagrangianCoef which is $O(n)$, does at most l inner product between two vectors of lenght m and a simple addiction. So, we have that the complexity of HVE-Dec is $O(lm)$.

Table 6.1 summarises the size of the main variables and Table 6.2 summarises the complexity of the four algorithms of the scheme described in Section 6.1.1.

Variable	**Size**
Public Key PK	$O(ln^2 \log q)$
Master Key MK	$O(ln^2 \log^2 q)$
Secret Key SK	$O(ln \log q)$
Ciphertext CT	$O(ln \log q)$

Table 6.1: Key Sizes of the general HVE Scheme

Algorithm	Complexity
SetUp	$O(l \cdot \text{poly}(n))$
KeyGen	$O(l \cdot \text{poly}(n))$
Enc	$O(ln^2 \log q)$
Dec	$O(ln \log q)$

Table 6.2: Complexity of the general HVE Scheme

6.2 Lattice-Based Hierarchical Hidden Vector Encryption

This Section describes the Hierarchical HVE scheme as proposed by Mochetti and Dahab [53]. Section 6.2.1 describes the four algorithms that comprise the HHVE scheme, Sections 6.2.2, 6.2.3, 6.2.4 and 6.2.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

6.2.1 Description

As described in Section 3.1, an Hierarchical Hidden Vector Encryption Scheme consists of the following algorithms: SetUp$(1^n, 1^\mu)$, KeyDerive$(PK, SK_{t-1}, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_t)$, Enc$(PK, M, \boldsymbol{w}_1, \cdots, \boldsymbol{w}_t)$ and Dec(PK, SK_t, CT). In this section we describe each algorithm as presented by Mochetti et al. [53]. The hierarchy is described by parameter μ and has maximum depth d.

As in the HVE scheme described in Section 6.1, the attributes are vectors $\boldsymbol{v}_i \in \{0, 1, \star\}^l$ and $\boldsymbol{w}_i \in \{0, 1\}^l$ and the decryption is only possible if $v_{i,j} = w_{i,j}, \forall j$ such that $v_{i,j} \neq \star$ for all vectors $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$, with $i \in [1, t]$. Again, Shamir' Secret Sharing Scheme, described in Appendix B, is used. The Secret Sharing Scheme used is composed of two algorithms: SplitVector, that divide a data vector into n vector pieces, and FindLagrangianCoef, that find the lagrangian coefficients so it is possible to recover the vector data using $k \leq n$ pieces.

HHVE-SetUp creates l lattices' basis, several general random matrices and a random vector $\boldsymbol{u}$. HHVE-KeyDerive generates the secret key by choosing the correspondent matrices based on values of $\boldsymbol{v}_i$ and concatenating it to the lattice basis. Now, the secret key is the basis for the lattices generated by these concatenations, using the algorithm SampleBasisLeft described in Section 2.2. Note that $SK_0 = MK$.

HHVE-Enc uses the message M, the vectors $\boldsymbol{w}_j$ and the matrices in the public key to create an integer c' and several vectors $\boldsymbol{c}_{i,j}$ that will compose the ciphertext for one bit. Here, it splits a random vector $\boldsymbol{s}$ using the SplitVector algorithm. Finally, HHVE-Dec calculates the lagrangian coefficients so that it can recover the message from the ciphertext only if $v_{i,j} = w_{i,j}, \forall j$ such that $v_{i,j} \neq \star$ for all vectors $\boldsymbol{v}_i$ and $\boldsymbol{w}_i$, with $i \in [1, t]$, .

Let n be the security parameter, μ be the hierarchical parameter, l be the vectors lenght, σ_i (for $i \in [1, d]$) be the Gaussian parameters and k be the threshold. Algorithms 6.5, 6.6, 6.7 and 6.8 describe the HHVE scheme.

Algorithm 6.5 HHVE-SetUp(): Setup Algorithm for the HHVE Scheme

```
Input: security parameter $1^n$ and hierarchical parameter $1^\mu$
Output: Public key $PK$ and master key $MK$
    $A_j, T_j \leftarrow \mathsf{TrapGen}(q, n, m)$ for $j \in [1, l]$
    $\boldsymbol{u} \overset{\$}{\leftarrow} \mathbb{Z}_q^n$
    $A_{i,j,b} \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}$ for $i \in [1, d]$, $j \in [1, l]$ and $b \in \{0, 1\}$
    $B_j \overset{\$}{\leftarrow} \mathbb{Z}_q^{n \times m}$ for $j \in [1, l]$
    public key $PK = (\{A_j\}, \boldsymbol{u}, \{A_{i,j,b}\}, \{B_j\})$
    master key $MK = (\{T_j\})$
```

Algorithm 6.6 HHVE-KeyDerive(): Key Generation Algorithm for the HHVE Scheme

```
Input: Public key $PK$, secret key $SK_{t-1}$ and vectors $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_t$
Output: Secret key $SK_t$
    if $(v_{i,j} = \star)$: $b_i \overset{\$}{\leftarrow} \{0, 1\}$
    else: $b_i \leftarrow v_{i,j}$
    $C_j \leftarrow [A_{1,j,b_1} + B_j | \cdots | A_{t-1,j,b_{t-1}} + B_j]$
    $S_j \leftarrow \mathsf{SampleBasisLeft}([A_j | C_j], A_{t,j,b_t} + B_j, S'_j, \sigma_t)$, where $S'_j \in SK_{t-1}$
    secret key $SK_t = (\{S_j\})$, with $S_j \in \mathbb{Z}_q^{n \times (t+1)m}$
```

Algorithm 6.7 HHVE-Enc(): Encryption Algorithm for the HHVE Scheme

```
Input: Public key $PK$, message $M$ and identity $\boldsymbol{w}_1, \cdots, \boldsymbol{w}_t$
Output: Ciphertext $CT$
    $\beta \leftarrow (l!)^2$
    $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$
    $\boldsymbol{x}_j \in \overline{\Psi}_\alpha^m$, for $j \in [1, l]$ and $x \in \overline{\Psi}_\alpha$
    $R_{i,j} \xleftarrow{\$} \{-1, 1\}^{m \times m}$, for $i \in [1, d]$ and $j \in [1, l]$
    $\boldsymbol{s}_1, \cdots, \boldsymbol{s}_l \leftarrow \mathsf{SplitVector}(\boldsymbol{s}, k)$
    $\boldsymbol{c}_{0,j} \leftarrow A_j^\top \boldsymbol{s}_j + \beta \boldsymbol{x}_j \in \mathbb{Z}_q^m$
    $\boldsymbol{c}_{i,j} \leftarrow [A_{i,j,w_{i,j}} + B_j]^\top \boldsymbol{s}_j + \beta R_{i,j}^\top \boldsymbol{x}_j \in \mathbb{Z}_q^m$
    $c' \leftarrow \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M \lfloor q/2 \rfloor$
    ciphertext $CT = (\{\boldsymbol{c}_{i,j}\}, c')$
```

Algorithm 6.8 HHVE-Dec(): Decryption Algorithm for the HHVE Scheme

```
Input: Public key $PK$, secret key $SK_t$ and ciphertext $CT$
Output: message $M$
    if $v_{i,j} \neq \star$ for all $i \in [1, d]$: $\mathbb{G} \leftarrow \mathbb{G} \cup \{j\}$
    $\boldsymbol{l} \leftarrow \mathsf{FindLagrangianCoef}(\mathbb{G})$
    $C_j \leftarrow [A_{1,j,v_{1,j}} + B_j | \cdots | A_{t,j,v_{t,j}} + B_j]$
    $\sigma = \sigma_t \sqrt{m(t+1)}\omega(\sqrt{\log(tm)})$
    $\boldsymbol{e}_j \leftarrow \mathsf{SamplePre}([A_j|C_j], S_j, l_j \boldsymbol{u}, \sigma)$, where $S_j \in SK_t$
    $z \leftarrow c' - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{c}_{0,\rho} \\ \vdots \\ \boldsymbol{c}_{t,\rho} \end{bmatrix} \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

6.2.2 Correctness

First recall that, based on Shamir's Secret Sharing Scheme (see Appendix B) and since $\boldsymbol{u}^\top \boldsymbol{s} = \boldsymbol{s}^\top \boldsymbol{u}$, we have

$$\sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{u}^\top \boldsymbol{s}_\rho = \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{s}_\rho^\top \boldsymbol{u} = \boldsymbol{s}^\top \boldsymbol{u} = \boldsymbol{u}^\top \boldsymbol{s}$$

Now the correctness is straightforward. We just substitute the values of $\boldsymbol{c}_{i,j}$ and c' in z and based on the Shamir's Secret Sharing Scheme we can recover the vector $\boldsymbol{s}$. Therefore, it is possible to cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get

the right value of M in z.

$$
\begin{aligned}
z &= c' - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{c}_{0,\rho} \\ \boldsymbol{c}_{1,\rho} \\ \vdots \\ \boldsymbol{c}_{t,\rho} \end{bmatrix} \mod q \\
&= c' - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} A_\rho^\top \boldsymbol{s}_\rho + \beta \boldsymbol{x}_\rho \\ (A_{1,\rho,w_{1,\rho}} + B_j)^\top \boldsymbol{s}_\rho + \beta R_{1,\rho}^\top \boldsymbol{x}_\rho \\ \vdots \\ (A_{t,\rho,w_{t,\rho}} + B_j)^\top \boldsymbol{s}_\rho + \beta R_{t,\rho}^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= c' - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top [A_\rho | C_\rho]^\top \boldsymbol{s}_\rho - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= c' - \sum_{\rho\in\mathbb{G}} l_\rho \boldsymbol{u}^\top \boldsymbol{s}_\rho - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + \beta x - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}
$$

Note that for the correct decryption the error term must be less than $q/4$.

6.2.3 Security

In this section we prove the following theorem.

Theorem 6.2. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 6.2.1 is IND-wAH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.HHVE-SetUp$(1^n, 1^\mu, \boldsymbol{w}_1^\star, \cdots, \boldsymbol{w}_d^\star)$: The algorithm chooses randomly the l matrices A_j in $\mathbb{Z}_q^{n\times m}$ and a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, and it uses the algorithm $\mathsf{TrapGen}(q, n, m)$ to generate the l matrices $B_j^\star$ and $T_j^\star$. It defines the matrices that refer to the bits that belong to $\boldsymbol{w}_i^\star$ as $A_{i,j,w_{i,j}} \leftarrow A_j R_{i,j}^\star - B_j^\star$, for $j \in [1, l]$, where all matrices $R_{i,j}^\star$ are randomly chosen in $\{-1, 1\}^{m\times m}$. The l matrices $A_{j,\overline{w_{i,j}^\star}}$, i.e., matrices that refer to the bits that do not belong to $\boldsymbol{w}_i^\star$ are chosen randomly in $\mathbb{Z}_q^{n\times m}$. The algorithm also chooses $\boldsymbol{s}'^\star \in \mathbb{Z}_q^n$ at random and calculates $\boldsymbol{s}^\star$ such that all shares $\boldsymbol{s}_j^\star = \boldsymbol{s}'^\star$, i.e., it solves the equation $\boldsymbol{s}_j^\star = \boldsymbol{s} + \sum \boldsymbol{a}_i j^i$ for $\boldsymbol{s}_j^\star = \boldsymbol{s}'^\star$, $j \in [1, l]$ and $i \in [0, k-1]$. Finally, it outputs $PK = (\{A_j\}, \{A_{i,j,b}\}, \boldsymbol{u})$ and $MK = (\{R_{i,j}^\star\}, \{B_j^\star\}, \{T_j^\star\}, \boldsymbol{s}^\star, \boldsymbol{s}'^\star)$.

Sim.HHVE-KeyDerive$(PK, MK, \boldsymbol{v}_1, \cdots, \boldsymbol{v}_t)$: Secret keys are now created using the trapdoors $T_j^\star$, sampled by the SampleBasisRight algorithm. It outputs $SK_t = \{S_j\}$, a basis of lattice $\Lambda_q^\perp\left(A_j | A_j R_{1,j}^\star - B_j^\star | \cdots | A_j R_{t,j}^\star - B_j^\star\right)$, using
$S_j \leftarrow \mathsf{SampleBasisRight}(A_j, B_j^\star, R_j^\star, T_j^\star, \sigma_t)$,
with $R_j^\star = [R_{1,j}^\star | \cdots | R_{t,j}^\star\ ]$.

Sim.HHVE-Enc$(PK, M, \boldsymbol{w}_1^\star, \cdots, \boldsymbol{w}_t^\star)$: The algorithm differs from HHVE-Enc in the sense that it uses matrices $R_{i,j}^\star$ and $B_j^\star$ instead of matrices $R_{i,j}$ and B_j and vectors $\boldsymbol{s}^\star$ and $\boldsymbol{s}'^\star$ instead of vectors $\boldsymbol{s}$ and $\boldsymbol{s}_j$. It also uses constant β to calculate the ciphertext. Now we have
$\boldsymbol{c}_{0,j} = \beta(A_j^\top \boldsymbol{s}'^\star + \boldsymbol{x}_j)$,
$\boldsymbol{c}_{i,j} = \beta([A_{i,j,w_{i,j}} + B_j^\star]^\top \boldsymbol{s}'^\star + R_{i,j}^{\star\top} \boldsymbol{x}_j)$ and
$c' = \beta(\boldsymbol{u}^\top \boldsymbol{s}^\star + x) + M\lfloor q/2 \rfloor$.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs HHVE-SetUp, answers $\mathcal{A}$'s secret key queries using algorithm HHVE-KeyDerive, and generates the challenge ciphertext using the HHVE-Enc with vectors $\boldsymbol{w}_1^{\star 0}, \cdots, \boldsymbol{w}_t^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.HHVE-SetUp with vectors $\boldsymbol{w}_1^{\star 0}, \cdots, \boldsymbol{w}_t^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HHVE-KeyDerive, and generates the challenge ciphertext using the Sim.HHVE-Enc algorithm with $\boldsymbol{w}_1^{\star 0}, \cdots, \boldsymbol{w}_t^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.HHVE-SetUp with vectors $\boldsymbol{w}_1^{\star 0}, \cdots, \boldsymbol{w}_d^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HHVE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.HHVE-SetUp with vectors $\boldsymbol{w}_1^{\star 1}, \cdots, \boldsymbol{w}_d^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HHVE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.HHVE-SetUp with vectors $\boldsymbol{w}_1^{\star 1}, \cdots, \boldsymbol{w}_t^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HHVE-KeyDerive, and generates the challenge ciphertext using the Sim.HHVE-Enc algorithm with $\boldsymbol{w}_1^{\star 1}, \cdots, \boldsymbol{w}_t^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs HHVE-SetUp, answers $\mathcal{A}$'s secret key queries using algorithm HHVE-KeyDerive, and generates the challenge ciphertext using the HHVE-Enc with vectors $\boldsymbol{w}_1^{\star 1}, \cdots, \boldsymbol{w}_t^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 6.4. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, matrices A_j are generated by $\mathsf{TrapGen}$ and matrices $A_{j,b}$ are uniformly random in $\mathbb{Z}_q^{n\times m}$. Instead, in Game 1, A_j are chosen uniformly at random and we have $A_{i,j,w_{i,j}} \leftarrow A_j R^\star_{i,j} - B^\star_j$ and randomly chosen $A_{j,\overline{w^\star_{i,j}}}$, where $B^\star$ is generated by $\mathsf{TrapGen}$ and the matrices $R^\star_{i,j}$ are uniformly and independently chosen at random in $\{-1,1\}^{m\times m}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathbb{Z}_q^n$.

Secret keys In Game 0, the secret key for vectors $\boldsymbol{v}_j$ is a set of basis of lattices $\Lambda_q^\perp(A_j|C_j)$, with $C_j = [A_{1,v_{1,j},j} + B_j|\dots|A_{t,j,v_{t,j}} + B_j]$ sampled using the $\mathsf{SampleBasisLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleBasisRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext The challenge ciphertext components c' and $\boldsymbol{c}_{0,j}$ in both games are computed almost in the same way and are clearly indistinguishable by Lemma A.7. But, in Game 0, the challenge ciphertext components $\boldsymbol{c}_{i,j}$, for $i \in [1,t]$, are computed as follows:

$$\boldsymbol{c}_{i,j} = [A_{i,j,w_{i,j}} + B_j]^\top \boldsymbol{s}'^\star + \beta R_{i,j}^\top \boldsymbol{x}_j \ .$$

On the other hand, in Game 1, we have:

$$\begin{aligned}
\boldsymbol{c}_{i,j} &= \beta([A_{i,j,w_{i,j}} + B_j]^\top \boldsymbol{s}'^\star + R_{i,j}^{\star\top} \boldsymbol{x}_j) \\
&= \beta([A_j R^\star_{i,j} - B^\star_j + B^\star_j]^\top \boldsymbol{s}'^\star + R_{i,j}^{\star\top} \boldsymbol{x}_j) \ . \\
&= \beta([A_j R^\star_{i,j}]^\top \boldsymbol{s}'^\star + R_{i,j}^{\star\top} \boldsymbol{x}_j)
\end{aligned}$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(\{A_j\},\{A_{i,j,b}\},\{\boldsymbol{c}_{i,j}\})$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemmas A.4, A.5 and A.7 we have that the following two distributions are statistically indistinguishable for every fixed matrix $B^\star_j$ and every vector $\boldsymbol{x}_j \in \mathbb{Z}_q^m$:

$$\left(A_j, A_{i,j,b}, \beta R_{i,j}^{\star\top} \boldsymbol{x}_j\right) \approx_s \left(A_j, A_j R^\star_{i,j} - B^\star_j, R_{i,j}^{\star\top} \boldsymbol{x}_j\right) \ .$$

Since each $R^\star_{i,j}$ is chosen independently for every $i \in [1,d]$ and $j \in [1,l]$, then the joint distribution of them are statistically close:

$$\left(\{A_j\},\{A_{i,j,b}\},\{\beta R_{i,j}^{\star\top} \boldsymbol{x}_j\}\right) \approx_s \left(\{A_j\},\{A_j R^\star_{i,j} - B^\star_j\},\{R_{i,j}^{\star\top} \boldsymbol{x}_j\}\right) \ .$$

Since each $(A_jR^\star_{i,j} - B^\star_j)^\top \boldsymbol{s}'^\star$ is statistically close to $A^\top_{i,j,b}\boldsymbol{s}'^\star$, it is possible to add each term to each side of the equation:

$$\begin{array}{r}\big(\{A_j\},\{A_{i,j,b}\},\{A^\top_{i,j,b}\boldsymbol{s}'^\star + \beta R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big) \approx_s \\ \big(\{A_j\},\{A_jR^\star_{i,j} - B^\star_j\},\{[A_jR^\star_{i,j} - B^\star_j]^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big)\end{array}.$$

Then, we add $B^{\star\top}_j\boldsymbol{s}'^\star$ to each side of the equation:

$$\begin{array}{r}\big(\{A_j\},\{A_{i,j,b}\},\{[A_{i,j,b} + B^\star_j]^\top \boldsymbol{s}'^\star + \beta R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big) \approx_s \\ \big(\{A_j\},\{A_jR^\star_{i,j} - B^\star_j\},\{[A_jR^\star_{i,j}]^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big)\end{array}.$$

Finally, we can multiply β in one side of the equation by Lemma A.7:

$$\begin{array}{r}\big(\{A_j\},\{A_{i,j,b}\},\{[A_{i,j,b} + B^\star_j]^\top \boldsymbol{s}'^\star + \beta R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big) \approx_s \\ \big(\{A_j\},\{A_jR^\star_{i,j} - B^\star_j\},\{\beta([A_jR^\star_{i,j}]^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j)\}\big)\end{array}.$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 6.5. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE.

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $lm + 1$ LWE challenge pairs $(\boldsymbol{a}_k, y_k) \in \mathbb{Z}^m_q \times \mathbb{Z}_q$, where either $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s}\rangle + x_k$ for a random $\boldsymbol{s} \in \mathbb{Z}^n_q$ and a noise term $x_k \leftarrow \Psi_\alpha$, or y_k is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_k, y_k)$. The i-th column of matrix A_j will be the vector $\boldsymbol{a}_{(j-1)m+i+1}$, and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrices $A_{i,j,b}$ are still calculated as in Sim.HHVE-SetUp, i.e., $A_{i,j,w_{i,j}} \leftarrow A_jR^\star_{i,j} - B^\star_j$.

Secret keys All private-key extraction queries are answered using Sim.HHVE-KeyDerive.

Challenge Ciphertext The ciphertext $CT = (\boldsymbol{c}^\star_{i,j}, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_k, y_k)$, with $\boldsymbol{c}^\star_{0,j} = (y_{(j-1)n+1}, \cdots, y_{jn+1})$, $c'^\star = y_0 + M\lfloor q/2 \rceil$ and $\boldsymbol{c}^\star_{i,j} = R^{\star\top}_{i,j}\boldsymbol{c}_{0,j}$. If we have $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s}'^\star\rangle + x_k$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_k is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2. If $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s}'^\star\rangle + x_k$, then

$(y_{(j-1)m+1}, \ldots, y_{jm+1}) = (\langle \boldsymbol{a}_{(j-1)m+1}, \boldsymbol{s}'^{\star} \rangle + x_{(j-1)m+1}, \ldots, \langle \boldsymbol{a}_{jm+1}, \boldsymbol{s}'^{\star} \rangle + x_{jm+1})$.
Therefore, for Game 1 we have

$$\begin{aligned} \boldsymbol{c}^{\star}_{0,j} &= \beta(A_j^{\top} \boldsymbol{s}'^{\star} + \boldsymbol{x}_j) \\ &= \beta(y_{(j-1)m+1}, \ldots, y_{jm+1}) \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{c}^{\star}_{i,j} &= \beta((A_{i,j,w_{i,j}} + B^{\star}_j)^{\top} \boldsymbol{s}'^{\star} + R^{\star\top}_{i,j} \boldsymbol{x}_j) \\ &= \beta(A_j R^{\star}_{i,j} - B^{\star}_j + B^{\star}_j)^{\top} \boldsymbol{s}'^{\star} + R^{\star\top}_{i,j} \boldsymbol{x}_j) \\ &= \beta((A_j R^{\star}_{i,j})^{\top} \boldsymbol{s}'^{\star} + R^{\star\top}_{i,j} \boldsymbol{x}_j) \\ &= \beta R^{\star\top}_{i,j} (A_j^{\top} \boldsymbol{s}'^{\star} + \boldsymbol{x}_j) \\ &= \beta R^{\star\top}_{i,j} \boldsymbol{c}_{0,j} \ . \end{aligned}$$

If all y_k is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_k is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s} \rangle + x_k$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 6.6. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated almost in the same way in both games. The vector $\boldsymbol{u}$ in both games are random and the matrices $A_{i,j,w_{i,j}}$ are chosen at random or by the algorithm TrapGen according to the value of $w^{\star}_0$ for Game 2 and $w^{\star}_1$ for Game 3.

Secret keys All private-key extraction queries are answered using Sim.HHVE-KeyDerive. The only difference is, again, that for Game 2, $\boldsymbol{w}^{\star} = \boldsymbol{w}^{\star 0}$ and, for Game 3, $\boldsymbol{w}^{\star} = \boldsymbol{w}^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrix $A_{i,j,w_{i,j}}$. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of $A_{i,j,w_{i,j}}$. From Lemmas A.4 and A.5 we can prove that $A_{i,j,w_{i,j}}$ for each game is statistically close to a uniformly random matrix, because

$A_{i,j,w_{i,j}} \leftarrow A_j R^{\star}_{i,j} - B^{\star}_j$.

□

6.2.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma\sqrt{2m}$ from Lemma A.1 and $\|R_\rho \boldsymbol{e}\| \leq 12\sqrt{tm+m}\|\boldsymbol{e}\|$ from Lemma A.2, for $R_\rho = [R_{1,rho}|\cdots|R_{t,\rho}]$, with $\max(t) = d$; therefore, for $\boldsymbol{e} = \begin{bmatrix}\boldsymbol{e}_1 \\ \boldsymbol{e}_2\end{bmatrix}$:

$$\begin{aligned}
&\|\boldsymbol{e}_1 + R_\rho \boldsymbol{e}_2\| \leq (\sigma_t\sqrt{m(d+1)} + 12\sqrt{m(d+1)}\sigma_t\sqrt{m(d+1)}) \qquad \text{so} \\
&\|\boldsymbol{e}_1 + R_\rho \boldsymbol{e}_2\| \leq O(d^2\sigma_t m)\ .
\end{aligned}$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \leq \|\boldsymbol{y}\| q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\begin{aligned}
\langle \boldsymbol{e}_1 + R_\rho \boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq O(d^2\sigma_t m) q\alpha_t w(\sqrt{\log 2m}) + O(d^2\sigma_t m)\sqrt{2m}/2 \\
\langle \boldsymbol{e}_1 + R_\rho \boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq \widetilde{O}(\sigma_t d^2 m q\alpha_t) + O(\sigma_t d^2 m^{3/2})\ .
\end{aligned}$$

We also know that $|x| \leq 2m$ from Lemma A.6 and $\beta = (l!)^2$ from construction. Therefore, to ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned}
&\left|\beta x - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix}\beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho\end{bmatrix}\right| < q/4 \\
&\beta|x| - \sum_{\rho\in\mathbb{G}} \beta \boldsymbol{e}_\rho^\top \begin{bmatrix}\boldsymbol{x}_\rho \\ R_\rho^\top \boldsymbol{x}_\rho\end{bmatrix} < q/4 \\
&\beta 2m - l\beta \boldsymbol{e}_\rho^\top \begin{bmatrix}\boldsymbol{x}_\rho \\ R_\rho^\top \boldsymbol{x}_\rho\end{bmatrix} < q/4 \\
&\beta 2m - l\beta(\boldsymbol{e}_{1,\rho}^\top \boldsymbol{x}_\rho + \boldsymbol{e}_{2,\rho}^\top R_\rho^\top \boldsymbol{x}_\rho) < q/4 \\
&\beta 2m - l\beta(\boldsymbol{e}_{1,\rho} + R\boldsymbol{e}_{2,\rho})^\top \boldsymbol{x}_\rho < q/4 \\
&\beta 2m + l\beta\langle \boldsymbol{e}_{1,\rho} + R\boldsymbol{e}_{2,\rho}, \boldsymbol{x}_\rho\rangle < q/4 \\
&\beta 2m + l\beta(\widetilde{O}(\sigma_t d^2 m q\alpha_t) + O(\sigma_t d^2 m^{3/2})) < q/4 \\
&\widetilde{O}(l\beta\sigma_t d^2 m q\alpha_t) + O(l\beta\sigma_t d^2 m^{3/2})) < q/4 \\
&\widetilde{O}(l(l!)^2\sigma_t d^2 m q\alpha_t) + O(l(l!)^2\sigma_t d^2 m^{3/2})) < q/4 \\
&\widetilde{O}(2^{3l}\sigma_t d^2 m q\alpha_t) + O(2^{3l}\sigma_t d^2 m^{3/2})) < q/4
\end{aligned}$$

Note that $l(l!)^2 \leq l^{2l+1} \leq 2^{3l}$. To ensure that σ_t is sufficiently large for SampleBasisLeft and SampleBasisRight (Theorems 2.10 and 2.11), we have

$$\sigma_t > \|S\|\sqrt{m}\omega(\sqrt{\log m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned} m &\geq 6n \log q \qquad \text{and} \\ \|S\| &\leq O(n \log q) \ . \end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha \ .$$

Therefore, we need to set the parameters as

$$\begin{aligned} m &= 6n^{\delta+1}, \\ q &= 2^{3l} m^{2.5} \omega(\sqrt{\log n}), \\ \alpha_t &= (2^{3l} m^2 \omega(\sqrt{\log n}))^{-1}, \\ \sigma_t &= m\omega(\sqrt{\log n}), \end{aligned}$$

with δ such that $n^\delta = O(\log q)$.

6.2.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 6.2.1. Note that for security parameter n, vector length l, hierarchy's maximum depth d and modulus q, we have $m = O(n \log q)$ (see Section 6.2.4).

The master key MK is comprised of l matrices of size $m \times m$, therefore its size is lm^2. The public key PK is comprised of $2l(d+1)$ matrices of size $n \times m$ and a vector of length n; therefore its size is $2lmn + 2ldmn + n$, which is $O(ldmn)$. The secret key is comprised of l matrices of size $n \times (t+1)m$, with $\max(t) = d$; therefore its size is $ln(d+1)m$, which is $O(ldnm)$. Finally, the ciphertext is comprised of an integer and $l(t+1)$ vectors of length m, with $\max(t) = d$; therefore its size is $1 + l(d+1)m$, which is $O(ldm)$.

The complexity of HHVE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, since it is executed l times on HHVE-SetUp, the complexity will be $l \cdot \text{poly}(n)$. The complexity of HHVE-KeyDerive is based on the complexity of the SampleBasisLeft algorithm, that is executed l times, and on the lt matrices additions of size $n \times m$. Therefore, we have that the complexity of HHVE-KeyDerive is $O(ldnm + l \cdot \text{poly}(n))$.

The HHVE-Enc algorithm calls the SpliVector algorithm which is linear, does lt matrix additions ($O(nm)$ each), $2l(t+1)$ matrix-vector multiplications ($O(nm)$ each), lt matrix-vector multiplications ($O(m^2)$ each), $l(t+1)$ constant-vector multiplications ($O(m)$), $l(t+1)$ vector additions ($O(m)$), one inner product ($O(n)$) and two simple additions $O(1)$, always with $\max(t) = d$. Therefore, the complexity of HHVE-Enc is based on the several multiplications. The HHVE-Dec algorithm does lt matrix additions ($O(nm)$), l inner products, a simple addiction, calls the SamplePre algorithm l times and calls the FindLagrangianCoef algorithm one time (which is $O(n)$). We have, by Theorem 2.6, that SamplePre is polynomial, therefore the complexity of the HHVE-Dec algorithm is based on SamplePre and the matrix additions.

Table 6.3 summarises the size of the main variables and Table 6.4 summarises the complexity of the four algorithms of the scheme described in Section 6.2.1.

Variable	**Size**
Public Key PK	$O(dln^2 \log q)$
Master Key MK	$O(ln^2 \log^2 q)$
Secret Key SK	$O(dln^2 \log q)$
Ciphertext CT	$O(dln \log q)$

Table 6.3: Key Sizes of the general HHVE Scheme

Algorithm	**Complexity**
SetUp	$O(l \cdot \mathrm{poly}(n))$
KeyDerive	$O(ldn^2 \log q + l \cdot \mathrm{poly}(n))$
Enc	$O(dln^2 \log q)$
Dec	$O(l \cdot \mathrm{poly}(n) + dln^2 \log q)$

Table 6.4: Complexity of the general HHVE Scheme

Chapter 7

Fuzzy Identity-Based Encryption

In this chapter we describe the full FBE lattice-based scheme proposed by Agrawal, Boyen, Vaikuntanathan, Voulgaris and Wee [5]. Section 7.1 reviews the full original scheme and Section 7.2 describes our contribution, a hierarchical version of the same scheme.

7.1 Lattice-Based Fuzzy Identity-Based Encryption

This Section reviews the full FBE scheme proposed by Agrawal, Boyen, Vaikuntanathan, Voulgaris and Wee [5], given in the appendix of their work. Section 7.1.1 describes the four algorithms that comprise the FBE scheme, Sections 7.1.2, 7.1.3, 7.1.4 and 7.1.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

7.1.1 Description

As described in Section 3.3, an Fuzzy Identity-Based Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n)$, $\mathsf{KeyGen}(PK, MK, \boldsymbol{\omega})$, $\mathsf{Enc}(PK, M, \boldsymbol{\omega}')$ and $\mathsf{Dec}(PK, SK, CT)$. In this section we describe each algorithm as presented by Agrawal et al. [5].

As in the HVE scheme described in Section 6.1, Shamir' Secret Sharing Scheme, described in Appendix B, is used. The Secret Sharing Scheme used is composed of two algorithms: SplitVector, that divides a data vector into n vector pieces, and FindLagrangianCoef, that find the lagrangian coefficients so it is possible to recover the vector data using $k \leq n$ pieces.

FBE-SetUp creates l general lattices and chooses at random the matrices and vector that will form the public and master keys. FBE-KeyGen generates the secret key by dividing each element of vector $\boldsymbol{u}$ using the SplitVector algorithm and creating vectors

$\boldsymbol{u}_j$. Than, it encodes each identity $\boldsymbol{id}_j \in \boldsymbol{\omega}$ into a matrix using the $\mathsf{rot}_f()$ function and concatenates it to each corresponding lattice basis. The secret key will be the l vectors $\boldsymbol{e}_j$ created by the SampleLeft algorithm, described in Section 2.2; therefore $\boldsymbol{e}_j \in \Lambda_q^{\boldsymbol{u}_j}(A_{\boldsymbol{id}_j})$.

FBE-Enc uses the message M, the identity vector $\boldsymbol{\omega}'$ and the matrices in the public key to create an integer c' and several vectors $\boldsymbol{c}_j$ that will compose the ciphertext for one bit. Finally, FBE-Dec can recover the message from the ciphertext only if $|\boldsymbol{\omega} \cap \boldsymbol{\omega}'| \geq k$, where $\boldsymbol{\omega}$ and $\boldsymbol{\omega}'$ are identities vectors.

Let n be the security parameter, l be the vectors lenght, σ be the Gaussian parameter and k be the threshold. Algorithms 7.1, 7.2, 7.3 and 7.4 describe the IBE scheme.

Algorithm 7.1 FBE-SetUp(): Setup Algorithm for the FBE Scheme

```
Input: security parameter $1^n$
Output: Public key $PK$ and master key $MK$
  $A_j, S_j \leftarrow \mathsf{TrapGen}(q, n, m)$ for $j \in [1, l]$
  $A'_j, B_j \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ for $j \in [1, l]$
  $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$
  public key $PK = (\{A_j\}, \{A'_j\}, \{B_j\}, \boldsymbol{u})$
  master key $MK = (\{S_j\})$
```

Algorithm 7.2 FBE-KeyGen(): Key Generation Algorithm for the FBE Scheme

```
Input: Public key $PK$, master key $MK$ and identities vector $\boldsymbol{\omega} = [\boldsymbol{id}_1, \cdots, \boldsymbol{id}_l]$
Output: Secret key $SK$
  $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_l \leftarrow \mathsf{SplitVector}(\boldsymbol{u}, k)$
  $C_j = A'_j + \mathsf{rot}_f(\boldsymbol{id}_j)B_j$
  $\boldsymbol{e}_j \leftarrow \mathsf{SampleLeft}(A_j, C_j, S_j, \boldsymbol{u}_j, \sigma) \in \mathbb{Z}_q^{2m}$
  secret key $SK = (\{\boldsymbol{e}_j\})$
```

Algorithm 7.3 FBE-Enc(): Encryption Algorithm for the FBE Scheme

```
Input: Public key $PK$, message $M$ and identities vector $\boldsymbol{\omega}' = [\boldsymbol{id}'_1, \cdots, \boldsymbol{id}'_l]$
Output: Ciphertext $CT$
    $\beta \leftarrow (l!)^2$
    $\boldsymbol{s} \overset{\$}{\leftarrow} \mathbb{Z}_q^n$
    $R \overset{\$}{\leftarrow} \{-1, 1\}^{m \times m}$
    $\boldsymbol{x}_j \in \overline{\Psi}_\alpha^m$, for $j \in [1, l]$ and $x \in \overline{\Psi}_\alpha$
    $C_j = A'_j + \mathsf{rot}_f(\boldsymbol{id}'_j)B_j$
    $\boldsymbol{c}'_j \leftarrow A_j^\top \boldsymbol{s} + \beta \boldsymbol{x}_j \in \mathbb{Z}_q^m$
    $\boldsymbol{c}_j \leftarrow C_j^\top \boldsymbol{s} + \beta R^\top \boldsymbol{x}_j \in \mathbb{Z}_q^m$
    $c' \leftarrow \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\{\boldsymbol{c}_j\}, \{\boldsymbol{c}'_j\}, c')$
```

Algorithm 7.4 FBE-Dec(): Decryption Algorithm for the FBE Scheme

```
Input: Public key $PK$, secret key $SK$ and ciphertext $CT$
Output: message $M$
    if $\boldsymbol{id}_j = \boldsymbol{id}'_j$: $\mathbb{G} \leftarrow \mathbb{G} \cup \{j\}$
    $\boldsymbol{l} \leftarrow \mathsf{FindLagrangianCoef}(\mathbb{G})$
    $z \leftarrow c' - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{c}'_\rho \\ \boldsymbol{c}_\rho \end{bmatrix} \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

7.1.2 Correctness

First recall that, based on Shamir's Secret Sharing Scheme (see Appendix B) we have

$$\sum_{\rho \in \mathbb{G}} l_\rho [A_j | C_j] \boldsymbol{e}_\rho = \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{u}_\rho = \boldsymbol{u}$$

Now the correctness is straightforward. We just substitute the values of $\boldsymbol{c}_j$ and c' in z and based on the Shamir's Secret Sharing Scheme we can recover the vector $\boldsymbol{u}$. Therefore, it is possible to cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get

the right value of M in z.

$$
\begin{aligned}
z &= c' - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{c}'_\rho \\ \boldsymbol{c}_\rho \end{bmatrix} \mod q \\
&= c' - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \begin{bmatrix} A_\rho^\top \boldsymbol{s} + \beta \boldsymbol{x}_\rho \\ C_\rho^\top \boldsymbol{s} + \beta R^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= c' - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \begin{bmatrix} A_\rho \\ C_\rho \end{bmatrix}^\top \boldsymbol{s} + \beta \begin{bmatrix} \boldsymbol{x}_\rho \\ R^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \left([A_j | C_j]^\top \boldsymbol{s} + \beta \begin{bmatrix} \boldsymbol{x}_\rho \\ R^\top \boldsymbol{x}_\rho \end{bmatrix} \right) \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \sum_{\rho \in \mathbb{G}} (l_\rho [A_j | C_j] \boldsymbol{e}_\rho)^\top \boldsymbol{s} - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \beta \begin{bmatrix} \boldsymbol{x}_\rho \\ R^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{e}_\rho^\top \beta \begin{bmatrix} \boldsymbol{x}_\rho \\ R^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + \beta x - \sum_{\rho \in \mathbb{G}} \beta l_\rho \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{x}_\rho \\ R^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}
$$

Note that for the correct decryption the error term must be less than $q/4$.

7.1.3 Security

In this section we prove the following theorem.

Theorem 7.1. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 7.1.1 is IND-wAH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.FBE-SetUp$(1^n, \boldsymbol{\omega}'^\star)$: The algorithm chooses random matrices $A_j \in \mathbb{Z}_q^{n \times m}$ and $R^\star \in \{-1, 1\}^{m \times m}$ and vector $\boldsymbol{u} \in \mathbb{Z}_q^n$ and it uses $\mathsf{TrapGen}(q, n, m)$ to generate basis $B_j^\star \in \mathbb{Z}_q^{n \times m}$ and $S_j^\star \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^\perp(B_j^\star)$. It then defines $A'_j \leftarrow A_j R^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j) B_j^\star$ and outputs $PK = (\{A_j\}, \{A'_j\}, \boldsymbol{u})$ and $MK = (R^\star, \{B_j^\star\}, \{S_j^\star\})$.

Sim.FBE-KeyGen$(PK, MK, \boldsymbol{\omega})$: Secret keys are now sampled by the $\mathsf{SampleRight}$ algorithm, using the trapdoor $S_j^\star$. It outputs

$SK = \boldsymbol{e}_j \in \Lambda_q^{\boldsymbol{u}_j}(A_j | A_j R^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j) B_j^\star)$, where

$\boldsymbol{e}_j \leftarrow \mathsf{SampleRight}(A_j, (\mathsf{rot}_f(\boldsymbol{id}_j) - \mathsf{rot}_f(\boldsymbol{id}'^{\star}_j))B^{\star}_j, R^{\star}, S^{\star}_j, \boldsymbol{u}_j, \sigma)$.

Note that we must have $\boldsymbol{id}_j \neq \boldsymbol{id}'^{\star}_j$, for $j \in [1, l]$, for the algorithm SampleRight to work properly.

Sim.FBE-Enc$(PK, M, \boldsymbol{\omega}'^{\star})$: The algorithm differs from FBE-Enc in the sense that it uses matrices $R^{\star}$ and $B^{\star}_j$ instead of matrices R and B_j and it uses constant β to calculate the ciphertext. Now we have
$\boldsymbol{c}'^{\star}_j = \beta(A_j^{\top}\boldsymbol{s} + \boldsymbol{x}_j)$,
$\boldsymbol{c}^{\star}_j = \beta(C_j^{\top}\boldsymbol{s} + R^{\top}\boldsymbol{x}_j)$ and
$c'^{\star} = \beta(\boldsymbol{u}^{\top}\boldsymbol{s} + x) + M\lfloor q/2 \rfloor$.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs FBE-SetUp, answers $\mathcal{A}$'s secret key queries using the FBE-KeyGen algorithm, and generates the challenge ciphertext using the FBE-Enc with identities vector $\boldsymbol{\omega}'^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.FBE-SetUp with identities vector $\boldsymbol{\omega}'^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.FBE-KeyGen, and generates the challenge ciphertext using the Sim.FBE-Enc algorithm with $\boldsymbol{\omega}'^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.FBE-SetUp with identities vector $\boldsymbol{\omega}'^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.FBE-KeyGen, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.FBE-SetUp with identities vector $\boldsymbol{\omega}'^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.FBE-KeyGen, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.FBE-SetUp with identities vector $\boldsymbol{\omega}'^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.FBE-KeyGen, and generates the challenge ciphertext using the Sim.FBE-Enc algorithm with $\boldsymbol{\omega}'^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs FBE-SetUp, answers $\mathcal{A}$'s secret key queries using the FBE-KeyGen algorithm, and generates the challenge ciphertext using the FBE-Enc with identities vector $\boldsymbol{\omega}'^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 7.1. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, matrices A_j are generated by $\mathsf{TrapGen}$ and matrices A'_j are uniformly random in $\mathbb{Z}_q^{n\times m}$. Instead, in Game 1, A_j are chosen uniformly at random and we have $A'_j \leftarrow A_j R^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j$, where matrices $B^\star_j$ are generated by $\mathsf{TrapGen}$ and the matrix $R^\star$ is uniformly and independently chosen at random in $\{-1,1\}^{m\times m}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathbb{Z}_q^n$.

Secret keys In Game 0, the secret key for identities vector $\boldsymbol{\omega}$ is a set of vectors $\boldsymbol{e}_j \in \Lambda_q^{\boldsymbol{u}_j}(A_j|C_j)$, sampled using the $\mathsf{SampleLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext The challenge ciphertext components c' and $\boldsymbol{c}'_j$ in both games are computed almost in the same way and are clearly indistinguishable by Lemma A.7. But, in Game 0, the challenge ciphertext component $\boldsymbol{c}_j$ is computed as follows:

$$\begin{aligned}\boldsymbol{c}_j &= C_j^\top \boldsymbol{s} + \beta R^{\star\top}\boldsymbol{x}_j \\ &= (A'_j + \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j)^\top \boldsymbol{s} + \beta R^{\star\top}\boldsymbol{x}_j \in \mathbb{Z}_q^m\end{aligned} .$$

On the other hand, in Game 1, we have:

$$\begin{aligned}\boldsymbol{c}_j &= \beta(C_j^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x}_j) \\ &= \beta((A'_j + \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x}_j) \\ &= \beta((A_jR^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j + \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x}_j) \\ &= \beta((A_jR^\star)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x}_j) \in \mathbb{Z}_q^m\end{aligned} .$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(\{A_j\},\{A'_j\},\{\boldsymbol{c}_j\})$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemmas A.4, A.5 and A.7 we have that the following two distributions are statistically indistinguishable for every fixed matrix $B^\star_j$, every $\boldsymbol{id}'^\star_j$ and every vector $\boldsymbol{x}_j \in \mathbb{Z}_q^m$:

$$\left(\{A_j\},\{A'_j\},\{\beta R^{\star\top}\boldsymbol{x}_j\}\right) \approx_s \left(\{A_j\},\{A_jR^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j\},\{R^{\star\top}\boldsymbol{x}_j\}\right) .$$

Since $(A_jR^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j)^\top \boldsymbol{s}$ is statistically close to $A'^\top_j \boldsymbol{s}$, it is possible to add each term to each side of the equation:

$$\begin{aligned}&\left(\{A_j\},\{A'_j\},\{A'^\top_j \boldsymbol{s} + \beta R^{\star\top}\boldsymbol{x}_j\}\right) \approx_s \\ \left(\{A_j\},\{A_jR^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j\},\right.&\left.\{(A_jR^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x}_j\}\right)\end{aligned} .$$

Then, we add $(\mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j)^\top \boldsymbol{s}$ to each side of the equation:

$$\begin{aligned}\left(\{A_j\},\{A'_j\},\{(A'_j + \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j)^\top \boldsymbol{s} + \beta R^{\star\top}\boldsymbol{x}_j\}\right) \approx_s \\ \left(\{A_j\},\{A_jR^\star - \mathsf{rot}_f(\boldsymbol{id}'^\star_j)B^\star_j\},\{(A_jR^\star)^\top \boldsymbol{s} + R^{\star\top}\boldsymbol{x}_j\}\right)\end{aligned} .$$

Finally, we can multiply β in one side of the equation by Lemma A.7:

$$\begin{gathered}\big(\{A_j\},\{A'_j\},\{(A'_j+\mathsf{rot}_f(\boldsymbol{id}'^{\star}_j)B^{\star}_j)^{\top}\boldsymbol{s}+\beta R^{\star\top}\boldsymbol{x}_j\}\big)\approx_s\\ \big(\{A_j\},\{A_jR^{\star}-\mathsf{rot}_f(\boldsymbol{id}'^{\star}_j)B^{\star}_j\},\{\beta((A_jR^{\star})^{\top}\boldsymbol{s}+R^{\star\top}\boldsymbol{x}_j)\}\big)\end{gathered}\;.$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 7.2. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE.

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $lm+1$ LWE challenge pairs $(\boldsymbol{a}_k, y_k)\in\mathbb{Z}_q^n\times\mathbb{Z}_q$, where either $y_k=\langle\boldsymbol{a}_k,\boldsymbol{s}\rangle+x_k$ for a random $\boldsymbol{s}\in\mathbb{Z}_q^n$ and a noise term $x_k\leftarrow\Psi_\alpha$, or y_k is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_k, y_k)$. The i-th column of matrix A_j will be the vector $\boldsymbol{a}_{(j-1)m+i+1}$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrices A'_j are calculated as in Sim.FBE-SetUp.

Secret keys All private-key extraction queries are answered using Sim.FBE-KeyGen.

Challenge Ciphertext The ciphertext $CT=(\boldsymbol{c}'^{\star}_j,\boldsymbol{c}^{\star}_j,c'^{\star})$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_k,y_k)$, with $\boldsymbol{c}'^{\star}_j=\beta(y_{(j-1)m+1},\ldots,y_{jm+1})$, $\boldsymbol{c}^{\star}_j=\beta\boldsymbol{c}'^{\star}_j$ and $c'^{\star}=\beta y_0+M\lfloor q/2\rceil$. If we have $y_j=\langle\boldsymbol{a}_j,\boldsymbol{s}\rangle+x_j$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_j is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2. If $y_k=\langle\boldsymbol{a}_k,\boldsymbol{s}\rangle+x_k$, then

$$\begin{aligned}\beta(y_{(j-1)m+1},\ldots,y_{jm+1})&=(\beta\langle\boldsymbol{a}_{(j-1)m+1},\boldsymbol{s}\rangle+\beta x_{(j-1)m+1},\ldots,\beta\langle\boldsymbol{a}_{jm+1},\boldsymbol{s}\rangle+\beta x_{jm+1})\\&=\beta A^{\top}_{j,w^{\star}_j}\boldsymbol{s}+\beta\boldsymbol{x}_j\\&=\beta(A^{\top}_{j,w^{\star}_j}\boldsymbol{s}+\boldsymbol{x}_j)\end{aligned}$$

Therefore, for Game 1 we have

$$\begin{aligned}\boldsymbol{c}'^{\star}_j&=\beta(A^{\top}_j\boldsymbol{s}+\boldsymbol{x}_j)\\&=\beta(y_{(j-1)m+1},\ldots,y_{jm+1})\;,\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{c}_j^\star &= \beta(C_j^\top \boldsymbol{s} + R^\top \boldsymbol{x}_j) \\
&= \beta(A'_j + \mathsf{rot}_f(\boldsymbol{id}_j'^\star)B_j^\star)^\top \boldsymbol{s} + R^\top \boldsymbol{x}_j) \\
&= \beta(A_j R^\star - \mathsf{rot}_f(\boldsymbol{id}_j'^\star)B_j^\star + \mathsf{rot}_f(\boldsymbol{id}_j'^\star)B_j^\star)^\top \boldsymbol{s} + R^\top \boldsymbol{x}_j) \\
&= \beta(A_j R^\star)^\top \boldsymbol{s} + R^\top \boldsymbol{x}_j) \\
&= R^{\star\top} \beta(A_j^\top \boldsymbol{s} + \boldsymbol{x}_j) \\
&= R^{\star\top} \boldsymbol{c}_j'^\star \ .
\end{aligned}$$

If all y_k is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_k is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s} \rangle + x_k$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 7.3. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. A_j and $\boldsymbol{u}$ are random and $A'_j \leftarrow A_j R^\star - \mathsf{rot}_f(\boldsymbol{id}_j'^\star)B_j^\star$, with $\boldsymbol{\omega}'^\star = \boldsymbol{\omega}'^{\star 0}$ for Game 2 and $\boldsymbol{\omega}'^\star = \boldsymbol{\omega}'^{\star 1}$ for Game 3.

Secret keys All private-key extraction queries are answered using Sim.FBE-KeyGen. The only difference is, again, that for Game 2, $\boldsymbol{\omega}'^\star = \boldsymbol{\omega}'^{\star 0}$ and, for Game 3, $\boldsymbol{\omega}'^\star = \boldsymbol{\omega}'^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrices A'_j. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of A'_j. From Lemmas A.4 and A.5 we can prove that A'_j for each game is statistically close to a uniformly random matrix, because

$A'_j \leftarrow A_j R^\star - \mathsf{rot}_f(\boldsymbol{id}_j'^\star)B_j^\star$.

□

7.1.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma\sqrt{2m}$ from Lemma A.1 and $\|R\boldsymbol{e}\| \leq 12\sqrt{2m}\|\boldsymbol{e}\|$ from Lemma A.2; therefore, for $\boldsymbol{e} = \begin{bmatrix}\boldsymbol{e}_1 \\ \boldsymbol{e}_2\end{bmatrix}$:

$$\begin{aligned} \|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| &\leq (\sigma\sqrt{2m} + 12\sqrt{2m}\sigma\sqrt{2m}) \qquad \text{so} \\ \|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| &\leq O(\sigma m) \ . \end{aligned}$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \leq \|\boldsymbol{y}\|q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\begin{aligned} \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq O(\sigma m)q\alpha w(\sqrt{\log 2m}) + O(\sigma m)\sqrt{2m}/2 \\ \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq \widetilde{O}(\sigma m q\alpha) + O(\sigma m^{3/2}) \ . \end{aligned}$$

We also know that $|x| \leq 2m$ from Lemma A.6, $|\beta l_\rho| \leq \beta^2 \leq (l!)^4$ from Lemma B.1 and $\beta = (l!)^2$ from construction. Therefore, to ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned} &\left|\beta x - \sum_{\rho\in\mathbb{G}} \beta l_\rho \boldsymbol{e}_\rho^\top \begin{bmatrix}\boldsymbol{x}_\rho \\ R^\top\boldsymbol{x}_\rho\end{bmatrix}\right| < q/4 \\ &\beta|x| - \sum_{\rho\in\mathbb{G}} (l!)^4 \boldsymbol{e}_\rho^\top \begin{bmatrix}\boldsymbol{x}_\rho \\ R^\top\boldsymbol{x}_\rho\end{bmatrix} < q/4 \\ &(l!)^2 2m - l(l!)^4 \boldsymbol{e}_\rho^\top \begin{bmatrix}\boldsymbol{x}_\rho \\ R^\top\boldsymbol{x}_\rho\end{bmatrix} < q/4 \\ &(l!)^2 2m - l(l!)^4(\boldsymbol{e}_{1,\rho}^\top\boldsymbol{x}_\rho + \boldsymbol{e}_{2,\rho}^\top R^\top\boldsymbol{x}_\rho) < q/4 \\ &(l!)^2 2m - l(l!)^4(\boldsymbol{e}_{1,\rho} + R\boldsymbol{e}_{2,\rho})^\top\boldsymbol{x}_\rho < q/4 \\ &(l!)^2 2m + l(l!)^4\langle \boldsymbol{e}_{1,\rho} + R\boldsymbol{e}_{2,\rho}, \boldsymbol{x}_\rho\rangle < q/4 \\ &(l!)^2 2m + l(l!)^4(\widetilde{O}(\sigma m q\alpha) + O(\sigma m^{3/2})) < q/4 \\ &\widetilde{O}(l(l!)^4\sigma m q\alpha) + O(l(l!)^4\sigma m^{3/2})) < q/4 \\ &\widetilde{O}(l2^{5l}\sigma m q\alpha) + O(l2^{5l}\sigma m^{3/2})) < q/4 \end{aligned}$$

Note that $(l!)^4 \leq l^{4l} \leq 2^{5l}$. To ensure that σ_t is sufficiently large for SampleLeft and SampleRight (Theorems 2.7 and 2.8), we have

$$\sigma_t > \|S\|\sqrt{m}\omega(\sqrt{\log m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned} m &\geq 6n\log q \qquad \text{and} \\ \|S\| &\leq O(n\log q) \ . \end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha \ .$$

Therefore, we need to set the parameters as

$$\begin{aligned} m &= 6n^{\delta+1}, \\ q &= l2^{5l}m^{2.5}\omega(\sqrt{\log n}), \\ \alpha &= (l2^{5l}m^{2}\omega(\sqrt{\log n}))^{-1}, \\ \sigma &= m\omega(\sqrt{\log n}), \end{aligned}$$

with δ such that $n^{\delta} = O(\log q)$.

7.1.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 7.1.1. Note that for security parameter n, identities vector length l and modulus q, we have $m = O(n \log q)$ (see Section 7.1.4).

The master key MK comprised of l matrices of size $m \times m$ matrix, therefore its size is lm^2. The public key PK is comprised of a vector of length n and three $n \times m$ matrices; therefore its size is $n + 3nm$, which is $O(mn)$. The secret key is comprised of l vectors of length $2m$; therefore its size is $2lm$, which is $O(lm)$. Finally, the ciphertext is comprised of an integer and $2l$ vectors of length m; therefore its size is $1 + 2lm$, which is $O(lm)$.

The complexity of FBE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, since it is executed l times on FBE-SetUp, the complexity will be $l \cdot \text{poly}(n)$. The complexity of FBE-KeyGen is based on the complexity of the SampleLeft algorithm, that is executed l times, and on the l matrix-matrix multiplications of size ($O(n^2m)$ each). Therefore, we have that the complexity of FBE-KeyGen is $O(lmn^2 + l \cdot \text{poly}(n))$.

The FBE-Enc algorithm does l matrix-matrix multiplications ($O(n^2m)$, each), $3l$ matrix-vector multiplications ($O(lnm + lnm + lm^2)$ total), one inner product ($O(n)$), l matrix addition ($O(nm)$), $2l$ vector additions ($O(m)$ each) and two simple additions $O(1)$. Therefore, the complexity of FBE-Enc is based on the matrix-matrix multiplication operations. The FBE-Dec algorithm does at most l vector additions ($O(2m)$), l inner products, a simple addiction, and calls the FindLagrangianCoef algorithm one time (which is $O(n)$). We have, therefore, that the complexity of the FBE-Dec algorithm is based on the vector additions and inner products.

Table 7.1 summarises the size of the main variables and Table 7.2 summarises the complexity of the four algorithms of the scheme described in Section 7.1.1.

Variable	**Size**
Public Key PK	$O(ln^2 \log q)$
Master Key MK	$O(ln^2 \log^2 q)$
Secret Key SK	$O(ln \log q)$
Ciphertext CT	$O(ln \log q)$

Table 7.1: Key Sizes of the general FBE Scheme

Algorithm	**Complexity**
SetUp	$O(l \cdot \text{poly}(n))$
KeyGen	$O(l \cdot \text{poly}(n) + ln^3 \log q)$
Enc	$O(ln^3 \log q)$
Dec	$O(ln \log q)$

Table 7.2: Complexity of the general FBE Scheme

7.2 Lattice-Based Hierarchical Fuzzy Identity-Based Encryption

This Section reviews the HFBE scheme, one of the contributions of this work. Section 7.2.1 describes the four algorithms that comprise the HFBE scheme, Sections 7.2.2, 7.2.3, 7.2.4 and 7.2.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

7.2.1 Description

As described in Section 3.1, an Hierarchical Fuzzy Identity-Based Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n, 1^\mu)$, $\mathsf{KeyDerive}(PK, SK_{t-1}, \boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_t)$, $\mathsf{Enc}(PK, M, \boldsymbol{\omega}'_1, \cdots, \boldsymbol{\omega}'_t)$ and $\mathsf{Dec}(PK, SK_t, CT)$.

As in the HVE scheme described in Section 6.1, Shamir' Secret Sharing Scheme, described in Appendix B, is used. The Secret Sharing Scheme used is composed of two algorithms: SplitVector, that divides a data vector into n vector pieces, and FindLagrangianCoef, that find the lagrangian coefficients so it is possible to recover the vector data using $k \leq n$ pieces.

HFBE-SetUp creates l general lattices and chooses at random the matrices and vector that will form the public and master keys. HFBE-KeyDerive generates the secret key by choosing the correspondent matrices based on values of each $\boldsymbol{id}_{i,j}$ and concatenating it to the lattice basis. Now, the secret key is the basis for the lattices generated by these concatenations, using the algorithm SampleBasisLeft described in Section 2.2. Note that $SK_0 = MK$.

HFBE-Enc uses the message M, the identities vectors $\boldsymbol{\omega}'$ and the matrices in the public key to create an integer c' and several vectors $\boldsymbol{c}_{i,j}$ that will compose the ciphertext for one bit. Here, it splits a random vector $\boldsymbol{s}$ using the SplitVector algorithm. Finally, HFBE-Dec calculates the lagrangian coefficients so that it can recover the message from the ciphertext only if $|\boldsymbol{\omega}_i \cap \boldsymbol{\omega}'_i| \geq k$, for all $i \in [1, d]$ where each $\boldsymbol{\omega}_i$ and $\boldsymbol{\omega}'_i$ are identities vectors.

Let n be the security parameter, μ be the hierarchical parameter, l be the vectors lenght, σ_i (for $i \in [1, d]$) be the Gaussian parameter and k be the threshold. Algorithms 7.5, 7.6, 7.7 and 7.8 describe the HFBE scheme.

Algorithm 7.5 HFBE-SetUp(): Setup Algorithm for the HFBE Scheme

```
Input: security parameter $1^n$ and hierarchical parameter $1^\mu$
Output: Public key $PK$ and master key $MK$
    $A_j, T_j \leftarrow \mathsf{TrapGen}(q, n, m)$ for $j \in [1, l]$
    $A_{i,j}, B_j \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ for $i \in [1, d]$ and $j \in [1, l]$
    $\boldsymbol{u} \xleftarrow{\$} \mathbb{Z}_q^n$
    public key $PK = (\{A_j\}, \{A_{i,j}\}, \{B_j\}, \boldsymbol{u})$
    master key $MK = (\{T_j\})$
```

Algorithm 7.6 HFBE-KeyDerive(): Key Generation Algorithm for the HFBE Scheme

```
Input: Public key $PK$, secret key $SK_{t-1}$ and identities vectors $\boldsymbol{\omega} = \boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_t$
Output: Secret key $SK$
    $C_j = [A_{1,j} + \mathsf{rot}_f(\boldsymbol{id}_{1,j})B_j | \cdots | A_{t-1,j} + \mathsf{rot}_f(\boldsymbol{id}_{t-1,j})B_j]$
    $S_j \leftarrow \mathsf{SampleBasisLeft}([A_j|C_j], A_{t,j} + \mathsf{rot}_f(\boldsymbol{id}_{t,j})B_j, S'_j, \sigma_t)$, where $S'_j \in SK_{t-1}$
    secret key $SK_t = (\{S_j\})$, with $S_j \in \mathbb{Z}_q^{n \times (t+1)m}$
```

Algorithm 7.7 **HFBE-Enc()**: Encryption Algorithm for the HFBE Scheme

```
Input: Public key $PK$, message $M$ and identities vectors $\boldsymbol{\omega}' = \boldsymbol{\omega}'_1, \cdots, \boldsymbol{\omega}'_t$
Output: Ciphertext $CT$
    $\beta \leftarrow (l!)^2$
    $\boldsymbol{s} \xleftarrow{\$} \mathbb{Z}_q^n$
    $R_{i,j} \xleftarrow{\$} \{-1,1\}^{m \times m}$, for $i \in [1,d]$ and $j \in [1,l]$
    $\boldsymbol{x}_j \in \overline{\Psi}_\alpha^m$, for $j \in [1,l]$ and $x \in \overline{\Psi}_\alpha$
    $\boldsymbol{s}_1, \cdots, \boldsymbol{s}_l \leftarrow \mathsf{SplitVector}(\boldsymbol{s}, k)$
    $\boldsymbol{c}_{0,j} \leftarrow A_j^\top \boldsymbol{s}_j + \beta \boldsymbol{x}_j \in \mathbb{Z}_q^m$
    $\boldsymbol{c}_{i,j} \leftarrow [A_{i,j} + \mathsf{rot}_f(\boldsymbol{id}_{i,j})B_j]^\top \boldsymbol{s}_j + \beta R_{i,j}^\top \boldsymbol{x}_j \in \mathbb{Z}_q^m$
    $c' \leftarrow \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\{\boldsymbol{c}_{i,j}\}, c')$
```

Algorithm 7.8 **HFBE-Dec()**: Decryption Algorithm for the HFBE Scheme

```
Input: Public key $PK$, secret key $SK_t$ and ciphertext $CT$
Output: message $M$
    if $\boldsymbol{id}_{i,j} = \boldsymbol{id}'_{i,j}$ for all $i \in [1,d]$: $\mathbb{G} \leftarrow \mathbb{G} \cup \{j\}$
    $\boldsymbol{l} \leftarrow \mathsf{FindLagrangianCoef}(\mathbb{G})$
    $C_j \leftarrow [A_{1,j} + \mathsf{rot}_f(\boldsymbol{id}_{1,j})B_j| \cdots |A_{t,j} + \mathsf{rot}_f(\boldsymbol{id}_{t,j})B_j]$
    $\sigma = \sigma_t \sqrt{m(t+1)}\omega(\sqrt{\log(tm)})$
    $\boldsymbol{e}_j \leftarrow \mathsf{SamplePre}([A_j|C_j], S_j, l_j\boldsymbol{u}, \sigma)$, where $S_j \in SK_t$
    $z \leftarrow c' - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{c}_{0,\rho} \\ \vdots \\ \boldsymbol{c}_{t,\rho} \end{bmatrix} \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

7.2.2 Correctness

First recall that, based on Shamir's Secret Sharing Scheme (see Appendix B) and since $\boldsymbol{u}^\top \boldsymbol{s} = \boldsymbol{s}^\top \boldsymbol{u}$, we have

$$\sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{u}^\top \boldsymbol{s}_\rho = \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{s}_\rho^\top \boldsymbol{u} = \boldsymbol{s}^\top \boldsymbol{u} = \boldsymbol{u}^\top \boldsymbol{s}$$

Now the correctness is straightforward. We just substitute the values of $\boldsymbol{c}_{i,j}$ and c' in z and based on the Shamir's Secret Sharing Scheme we can recover the vector $\boldsymbol{s}$. Therefore, it is possible to cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get

the right value of M in z.

$$
\begin{aligned}
z &= c' - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{c}_{0,\rho} \\ \boldsymbol{c}_{1,\rho} \\ \vdots \\ \boldsymbol{c}_{t,\rho} \end{bmatrix} \mod q \\
&= c' - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} A_\rho^\top \boldsymbol{s}_\rho + \beta \boldsymbol{x}_\rho \\ (A_{1,j} + \mathsf{rot}_f(\boldsymbol{id}_{1,j})B_j)^\top \boldsymbol{s}_\rho + \beta R_{1,\rho}^\top \boldsymbol{x}_\rho \\ \vdots \\ (A_{t,j} + \mathsf{rot}_f(\boldsymbol{id}_{t,j})B_j)^\top \boldsymbol{s}_\rho + \beta R_{t,\rho}^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= c' - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top [A_\rho | C_\rho]^\top \boldsymbol{s}_\rho - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= c' - \sum_{\rho \in \mathbb{G}} l_\rho \boldsymbol{u}^\top \boldsymbol{s}_\rho - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + \beta x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \beta \boldsymbol{x}_\rho \\ \beta R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + \beta x - \beta \sum_{\rho \in \mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix} \boldsymbol{x}_\rho \\ R_\rho^\top \boldsymbol{x}_\rho \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}
$$

Note that for the correct decryption the error term must be less than $q/4$.

7.2.3 Security

In this section we prove the following theorem.

Theorem 7.2. *If the decision-*LWE *problem is infeasible, then the functional encryption scheme described in Section 7.2.1 is IND-wAH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.HFBE-SetUp$(1^n, 1^\mu, \boldsymbol{\omega}_1'^\star, \cdots, \boldsymbol{\omega}_d'^\star)$: The algorithm chooses randomly the l matrices A_j in $\mathbb{Z}_q^{n \times m}$ and a vector $\boldsymbol{u} \in \mathbb{Z}_q^n$, and it uses the algorithm $\mathsf{TrapGen}(q, n, m)$ to generate the l matrices $B_j^\star$ and $T_j^\star$. It defines the matrices $A_{i,j} \leftarrow A_j R_{i,j}^\star - \mathsf{rot}_f(\boldsymbol{id}_{i,j})B_j^\star$, for $j \in [1, l]$ and $i \in [1, d]$, where all matrices $R_{i,j}^\star$ are randomly chosen in $\{-1, 1\}^{m \times m}$. The algorithm also chooses $\boldsymbol{s}'^\star \in \mathbb{Z}_q^n$ at random and calculates $\boldsymbol{s}^\star$ such that all shares $\boldsymbol{s}_j^\star = \boldsymbol{s}'^\star$, i.e., it solves the equation $\boldsymbol{s}_j^\star = \boldsymbol{s} + \sum \boldsymbol{a}_i j^i$ for $\boldsymbol{s}_j^\star = \boldsymbol{s}'^\star$, $j \in [1, l]$ and $i \in [0, k-1]$. Finally, it outputs $PK = (\{A_j\}, \{A_{i,j}\}, \boldsymbol{u})$ and $MK = (\{R_{i,j}^\star\}, \{B_j^\star\}, \{T_j^\star\}, \boldsymbol{s}^\star, \boldsymbol{s}'^\star)$.

Sim.HFBE-KeyDerive$(PK, MK, \boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_t)$: Secret keys are now created using the trapdoors $T_j^\star$, sampled by the SampleBasisRight algorithm. It outputs $SK_t = \{S_j\}$, the bases of lattices $\Lambda_q^\perp\left(A_j|A_jR_{1,j}^\star - \mathsf{rot}_f(\boldsymbol{id}_{1,j}'^\star)B_j^\star|\cdots|A_jR_{t,j}^\star - \mathsf{rot}_f(\boldsymbol{id}_{t,j}'^\star)B_j^\star\right)$, using

$S_j \leftarrow \mathsf{SampleBasisRight}(A_j, B_j^\star, R_{i,j}^\star, T_j^\star, \sigma_t)$,

with $R_j^\star = [R_{1,j}^\star|\cdots|R_{t,j}^\star]$.

Note that we must have $\boldsymbol{id}_{i,j} \neq \boldsymbol{id}_{i,j}'^\star$, for $j \in [1, l]$ and $i \in [1, d]$, for the algorithm SampleBasisRight to work properly.

Sim.HFBE-Enc$(PK, M, \boldsymbol{\omega}_1'^\star, \cdots, \boldsymbol{\omega}_t'^\star)$: The algorithm differs from HFBE-Enc in the sense that it uses matrices $R_{i,j}^\star$ and $B_j^\star$ instead of matrices $R_{i,j}$ and B_j and vectors $\boldsymbol{s}^\star$ and $\boldsymbol{s}'^\star$ instead of vectors $\boldsymbol{s}$ and $\boldsymbol{s}_j$. It also uses constant β to calculate the ciphertext. Now we have

$\boldsymbol{c}_{0,j} = \beta(A_j^\top \boldsymbol{s}'^\star + \boldsymbol{x}_j)$,

$\boldsymbol{c}_{i,j} = \beta([A_{i,j} + \mathsf{rot}_f(\boldsymbol{id}_{i,j}'^\star)B_j^\star]^\top \boldsymbol{s}'^\star + R_{i,j}^{\star\top}\boldsymbol{x}_j)$ and

$c' = \beta(\boldsymbol{u}^\top \boldsymbol{s}^\star + x) + M\lfloor q/2 \rfloor$.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs HFBE-SetUp, answers $\mathcal{A}$'s secret key queries using algorithm HFBE-KeyDerive, and generates the challenge ciphertext using the HFBE-Enc with identities vectors $\boldsymbol{\omega}_1'^{\star 0}, \cdots, \boldsymbol{\omega}_t'^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.HFBE-SetUp with identities vectors $\boldsymbol{\omega}_1'^{\star 0}, \cdots, \boldsymbol{\omega}_d'^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HFBE-KeyDerive, and generates the challenge ciphertext using the Sim.HFBE-Enc algorithm with identities vectors $\boldsymbol{\omega}_1'^{\star 0}, \cdots, \boldsymbol{\omega}_t'^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.HFBE-SetUp with identities vectors $\boldsymbol{\omega}_1'^{\star 0}, \cdots, \boldsymbol{\omega}_d'^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.HFBE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.HFBE-SetUp with identities vectors $\boldsymbol{\omega}_1'^{\star 1}, \cdots, \boldsymbol{\omega}_d'^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HFBE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.HFBE-SetUp with identities vectors $\boldsymbol{\omega}_1'^{\star 1}, \cdots, \boldsymbol{\omega}_d'^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.HFBE-KeyDerive, and generates the challenge ciphertext using the Sim.HFBE-Enc algorithm with identities vectors $\boldsymbol{\omega}_1'^{\star 1}, \cdots, \boldsymbol{\omega}_t'^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs HFBE-SetUp, answers $\mathcal{A}$'s secret key queries using algorithm HFBE-KeyDerive, and generates the challenge ciphertext using the HFBE-Enc with identities vectors $\boldsymbol{\omega}_1'^{\star 1}, \cdots, \boldsymbol{\omega}_t'^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 7.4. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, matrices A_j are generated by $\mathsf{TrapGen}$ and matrices $A_{i,j}$ are uniformly random in $\mathbb{Z}_q^{n\times m}$. Instead, in Game 1, A_j are chosen uniformly at random and we have $A_{i,j} \leftarrow A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}_{i,j})B^\star_j$, where $B^\star_j$ is generated by $\mathsf{TrapGen}$ and the matrices $R^\star_{i,j}$ are uniformly and independently chosen at random in $\{-1,1\}^{m\times m}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathbb{Z}_q^n$.

Secret keys In Game 0, the secret key for identities vectors $\boldsymbol{\omega}_1, \cdots, \boldsymbol{\omega}_t$ is a set of basis of lattices $\Lambda_q^\perp(A_j|C_j)$, with $C_j = [A_{1,j} + \mathsf{rot}_f(\boldsymbol{id}_{1,j})B_j|\dots|A_{t,j} + \mathsf{rot}_f(\boldsymbol{id}_{t,j})B_j]$ sampled using the $\mathsf{SampleBasisLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleBasisRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext The challenge ciphertext components c' and $\boldsymbol{c}_{0,j}$ in both games are computed almost in the same way and are clearly indistinguishable by Lemma A.7. But, in Game 0, the challenge ciphertext components $\boldsymbol{c}_{i,j}$, for $i \in [1,t]$, are computed as follows:

$$\boldsymbol{c}_{i,j} = (A_{i,j} + \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j})B^\star_j)^\top \boldsymbol{s}'^\star + \beta R^{\star\top}_{i,j}\boldsymbol{x}_j \in \mathbb{Z}_q^m \ .$$

On the other hand, in Game 1, we have:

$$\begin{aligned}\boldsymbol{c}_{i,j} &= \beta((A_{i,j} + \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j})B^\star_j)^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j) \\ &= \beta((A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j})B^\star_j + \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j})B^\star_j)^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j) \ . \\ &= \beta((A_j R^\star_{i,j})^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j) \in \mathbb{Z}_q^m\end{aligned}$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(\{A_j\}, \{A_{i,j}\}, \{\boldsymbol{c}_{i,j}\})$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemmas A.4, A.5 and A.7 we have that the following two distributions are statistically indistinguishable for every fixed matrix $B^\star_j$ and every vector $\boldsymbol{x}_j \in \mathbb{Z}_q^m$:

$$\left(A_j, A_{i,j}, \beta R^{\star\top}_{i,j}\boldsymbol{x}_j\right) \approx_s \left(A_j, A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j})B^\star_j, R^{\star\top}_{i,j}\boldsymbol{x}_j\right) \ .$$

Since each $R^\star_{i,j}$ is chosen independently for every $i \in [1,d]$ and $j \in [1,l]$, then the joint distribution of them are statistically close:

$$\left(\{A_j\}, \{A_{i,j}\}, \{\beta R^{\star\top}_{i,j}\boldsymbol{x}_j\}\right) \approx_s \left(\{A_j\}, \{A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j})B^\star_j\}, \{R^{\star\top}_{i,j}\boldsymbol{x}_j\}\right) \ .$$

Since each $(A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j)^\top \boldsymbol{s}'^\star$ is statistically close to $A^\top_{i,j}\boldsymbol{s}'^\star$, it is possible to add each term to each side of the equation:

$$\begin{gathered}\big(\{A_j\},\{A_{i,j}\},\{A^\top_{i,j}\boldsymbol{s}'^\star + \beta R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big) \approx_s \\ \big(\{A_j\},\{A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j\},\{[A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j]^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big)\end{gathered}.$$

Then, we add $\mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^{\star\top}_j \boldsymbol{s}'^\star$ to each side of the equation:

$$\begin{gathered}\big(\{A_j\},\{A_{i,j}\},\{[A_{i,j} + \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j]^\top \boldsymbol{s}'^\star + \beta R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big) \approx_s \\ \big(\{A_j\},\{A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j\},\{[A_j R^\star_{i,j}]^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big)\end{gathered}.$$

Finally, we can multiply β in one side of the equation by Lemma A.7:

$$\begin{gathered}\big(\{A_j\},\{A_{i,j}\},\{[A_{i,j} + \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j]^\top \boldsymbol{s}'^\star + \beta R^{\star\top}_{i,j}\boldsymbol{x}_j\}\big) \approx_s \\ \big(\{A_j\},\{A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j\},\{\beta([A_j R^\star_{i,j}]^\top \boldsymbol{s}'^\star + R^{\star\top}_{i,j}\boldsymbol{x}_j)\}\big)\end{gathered}.$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 7.5. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*LWE*.*

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-LWE.

Init $\mathcal{B}$ is given $lm+1$ LWE challenge pairs $(\boldsymbol{a}_k, y_k) \in \mathbb{Z}^m_q \times \mathbb{Z}_q$, where either $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s}\rangle + x_k$ for a random $\boldsymbol{s} \in \mathbb{Z}^n_q$ and a noise term $x_k \leftarrow \Psi_\alpha$, or y_k is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_k, y_k)$. The i-th column of matrix A_j will be the vector $\boldsymbol{a}_{(j-1)m+i+1}$, and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrices $A_{i,j}$ are still calculated as in $\mathsf{Sim.HFBE\text{-}SetUp}$, i.e., $A_{i,j} \leftarrow A_j R^\star_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^\star_{i,j}) B^\star_j$.

Secret keys All private-key extraction queries are answered using $\mathsf{Sim.HFBE\text{-}KeyDerive}$.

Challenge Ciphertext The ciphertext $CT = (\boldsymbol{c}^\star_{i,j}, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_k, y_k)$, with $\boldsymbol{c}^\star_{0,j} = (y_{(j-1)n+1}, \cdots, y_{jn+1})$, $c'^\star = y_0 + M\lfloor q/2 \rceil$ and $\boldsymbol{c}^\star_{i,j} = R^{\star\top}_{i,j}\boldsymbol{c}_{0,j}$. If we have $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s}'^\star\rangle + x_k$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if y_k is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2. If $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s}'^\star\rangle + x_k$, then

$(y_{(j-1)m+1}, \ldots, y_{jm+1}) = (\langle \boldsymbol{a}_{(j-1)m+1}, \boldsymbol{s}'^{\star} \rangle + x_{(j-1)m+1}, \ldots, \langle \boldsymbol{a}_{jm+1}, \boldsymbol{s}'^{\star} \rangle + x_{jm+1})$. Therefore, for Game 1 we have

$$\begin{aligned} \boldsymbol{c}^{\star}_{0,j} &= \beta(A_j^{\top} \boldsymbol{s}'^{\star} + \boldsymbol{x}_j) \\ &= \beta(y_{(j-1)m+1}, \ldots, y_{jm+1}) \end{aligned}$$

and

$$\begin{aligned} \boldsymbol{c}^{\star}_{i,j} &= \beta((A_{i,j} + \mathsf{rot}_f(\boldsymbol{id}'^{\star}_{i,j})B^{\star}_j)^{\top} \boldsymbol{s}'^{\star} + R^{\star\top}_{i,j} \boldsymbol{x}_j) \\ &= \beta(A_j R^{\star}_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^{\star}_{i,j})B^{\star}_j + \mathsf{rot}_f(\boldsymbol{id}'^{\star}_{i,j})B^{\star}_j)^{\top} \boldsymbol{s}'^{\star} + R^{\star\top}_{i,j} \boldsymbol{x}_j) \\ &= \beta((A_j R^{\star}_{i,j})^{\top} \boldsymbol{s}'^{\star} + R^{\star\top}_{i,j} \boldsymbol{x}_j) \\ &= \beta R^{\star\top}_{i,j} (A_j^{\top} \boldsymbol{s}'^{\star} + \boldsymbol{x}_j) \\ &= R^{\star\top}_{i,j} \boldsymbol{c}_{0,j} \ . \end{aligned}$$

If all y_k is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the LWE challenge, because, as we showed, if y_k is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_k = \langle \boldsymbol{a}_k, \boldsymbol{s} \rangle + x_k$, then $\mathcal{A}$'s view is the same as in Game 1. □

Indistinguishability of Game 2 and Game 3

Lemma 7.6. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. A_j and $\boldsymbol{u}$ are random and $A_{i,j} \leftarrow A_j R^{\star}_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^{\star}_{i,j})B^{\star}_j$, with $\boldsymbol{\omega}'^{\star}_i = \boldsymbol{\omega}'^{\star 0}_i$ for Game 2 and $\boldsymbol{\omega}'^{\star}_i = \boldsymbol{\omega}'^{\star 1}_i$ for Game 3, for $i \in [1, d]$.

Secret keys All private-key extraction queries are answered using $\mathsf{Sim.HFBE\text{-}KeyDerive}$. The only difference is, again, that for Game 2, $\boldsymbol{\omega}'^{\star}_i = \boldsymbol{\omega}'^{\star 0}_i$ and $\boldsymbol{\omega}'^{\star}_i = \boldsymbol{\omega}'^{\star 1}_i$ for Game 3, for $i \in [1, d]$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrices $A_{i,j}$. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of $A_{i,j}$. From Lemmas A.4 and A.5 we can prove that $A_{i,j}$ for each game is statistically close to a uniformly random matrix, because

$A_{i,j} \leftarrow A_j R^{\star}_{i,j} - \mathsf{rot}_f(\boldsymbol{id}'^{\star}_{i,j})B^{\star}_j$.

□

7.2.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \le \sigma\sqrt{2m}$ from Lemma A.1 and $\|R_\rho \boldsymbol{e}\| \le 12\sqrt{tm+m}\|\boldsymbol{e}\|$ from Lemma A.2, for $R_\rho = [R_{1,rho}|\cdots|R_{t,\rho}]$, with $\max(t) = d$; therefore, for $\boldsymbol{e} = \begin{bmatrix}\boldsymbol{e}_1\\ \boldsymbol{e}_2\end{bmatrix}$:

$$\begin{aligned}
&\|\boldsymbol{e}_1 + R_\rho\boldsymbol{e}_2\| \le (\sigma_t\sqrt{m(d+1)} + 12\sqrt{m(d+1)}\sigma_t\sqrt{m(d+1)}) \qquad \text{so}\\
&\|\boldsymbol{e}_1 + R_\rho\boldsymbol{e}_2\| \le O(d^2\sigma_t m)\ .
\end{aligned}$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \le \|\boldsymbol{y}\| q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\begin{aligned}
\langle \boldsymbol{e}_1 + R_\rho\boldsymbol{e}_2, \boldsymbol{x}\rangle &\le O(d^2\sigma_t m) q\alpha_t w(\sqrt{\log 2m}) + O(d^2\sigma_t m)\sqrt{2m}/2\\
\langle \boldsymbol{e}_1 + R_\rho\boldsymbol{e}_2, \boldsymbol{x}\rangle &\le \widetilde{O}(\sigma_t d^2 m q\alpha_t) + O(\sigma_t d^2 m^{3/2})\ .
\end{aligned}$$

We also know that $|x| \le 2m$ from Lemma A.6 and $\beta = (l!)^2$ from construction. Therefore, to ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned}
&\left|\beta x - \sum_{\rho\in\mathbb{G}} \boldsymbol{e}_\rho^\top \begin{bmatrix}\beta\boldsymbol{x}_\rho\\ \beta R_\rho^\top\boldsymbol{x}_\rho\end{bmatrix}\right| < q/4\\
&\beta|x| - \sum_{\rho\in\mathbb{G}} \beta\boldsymbol{e}_\rho^\top \begin{bmatrix}\boldsymbol{x}_\rho\\ R_\rho^\top\boldsymbol{x}_\rho\end{bmatrix} < q/4\\
&\beta 2m - l\beta\boldsymbol{e}_\rho^\top \begin{bmatrix}\boldsymbol{x}_\rho\\ R_\rho^\top\boldsymbol{x}_\rho\end{bmatrix} < q/4\\
&\beta 2m - l\beta(\boldsymbol{e}_{1,\rho}^\top\boldsymbol{x}_\rho + \boldsymbol{e}_{2,\rho}^\top R_\rho^\top\boldsymbol{x}_\rho) < q/4\\
&\beta 2m - l\beta(\boldsymbol{e}_{1,\rho} + R\boldsymbol{e}_{2,\rho})^\top\boldsymbol{x}_\rho < q/4\\
&\beta 2m + l\beta\langle\boldsymbol{e}_{1,\rho} + R\boldsymbol{e}_{2,\rho}, \boldsymbol{x}_\rho\rangle < q/4\\
&\beta 2m + l\beta(\widetilde{O}(\sigma_t d^2 m q\alpha_t) + O(\sigma_t d^2 m^{3/2})) < q/4\\
&\widetilde{O}(l\beta\sigma_t d^2 m q\alpha_t) + O(l\beta\sigma_t d^2 m^{3/2})) < q/4\\
&\widetilde{O}(l(l!)^2\sigma_t d^2 m q\alpha_t) + O(l(l!)^2\sigma_t d^2 m^{3/2})) < q/4\\
&\widetilde{O}(2^{3l}\sigma_t d^2 m q\alpha_t) + O(2^{3l}\sigma_t d^2 m^{3/2})) < q/4
\end{aligned}$$

Note that $l(l!)^2 \le l^{2l+1} \le 2^{3l}$. To ensure that σ_t is sufficiently large for SampleBasisLeft and SampleBasisRight (Theorems 2.10 and 2.11), we have

$$\sigma_t > \|S\|\sqrt{m}\omega(\sqrt{\log m}).$$

To ensure that TrapGen (Theorem 2.1) can operate, we have

$$\begin{aligned} &m \geq 6n \log q \qquad \text{and} \\ &\|S\| \leq O(n \log q) \ . \end{aligned}$$

To ensure that the reduction applies (Theorem 2.12), we have

$$q > 2\sqrt{n}/\alpha \ .$$

Therefore, we need to set the parameters as

$$\begin{aligned} m &= 6n^{\delta+1}, \\ q &= 2^{3l} m^{2.5} \omega(\sqrt{\log n}), \\ \alpha_t &= (2^{3l} m^2 \omega(\sqrt{\log n}))^{-1}, \\ \sigma_t &= m\omega(\sqrt{\log n}), \end{aligned}$$

with δ such that $n^{\delta} = O(\log q)$.

7.2.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 7.2.1. Note that for security parameter n and modulus q, we have $m = O(n \log q)$ (see Section 7.2.4).

The master key MK comprised of l matrices of size $m \times m$ matrix, therefore its size is lm^2. The public key PK is comprised of a vector of length n and $2l + ld$ matrices of size $n \times m$; therefore its size is $n + 2lnm + ldmn$, which is $O(ldmn)$. The secret key is comprised of l matrices of size $n \times (t+1)m$, with $\max(t) = d$; therefore its size is $ln(d+1)m$, which is $O(ldnm)$. Finally, the ciphertext is comprised of an integer and $l(t+1)$ vectors of length m, with $\max(t) = d$; therefore its size is $1 + l(d+1)m$, which is $O(ldm)$.

The complexity of HFBE-SetUp is based on the complexity of the TrapGen algorithm. By Theorem 2.1 we have that the TrapGen algorithm is polynomial, and, since it is executed l times on HFBE-SetUp, the complexity will be $l \cdot \text{poly}(n)$. The complexity of HFBE-KeyDerive is based on the complexity of the SampleBasisLeft algorithm, that is executed l times, on the lt matrices additions of size $n \times m$ and on the lt matrices multiplications, with $\max(t) = d$. Therefore, we have that the complexity of HFBE-KeyDerive is $O(ldm^2 + l \cdot \text{poly}(n))$.

The HFBE-Enc algorithm calls the SpliVector algorithm which is linear, does lt matrix additions ($O(nm)$ each), lt matrix-matrix multiplications ($O(nm^2)$, each), $2l(t+1)$ matrix-vector multiplications ($O(nm)$ each), lt matrix-vector multiplications ($O(m^2)$ each), $l(t+1)$ constant-vector multiplications ($O(m)$), $l(t+1)$ vector additions ($O(m)$), one inner product ($O(n)$) and two simple additions $O(1)$, always with $\max(t) = d$. Therefore, the complexity of HFBE-Enc is based on the several multiplications. The HFBE-Dec algorithm does lt matrices additions ($O(nm)$, each), lt matrices multiplications ($O(nm^2)$, each), l inner products, a simple addiction, calls the SamplePre algorithm l times and calls the FindLagrangianCoef algorithm one time (which is $O(n)$). We have, by Theorem 2.6, that SamplePre is polynomial, therefore the complexity of the HFBE-Dec algorithm is based on SamplePre and the matrices multiplications.

Table 7.3 summarises the size of the main variables and Table 7.4 summarises the complexity of the four algorithms of the scheme described in Section 7.2.1.

Variable	**Size**
Public Key PK	$O(ldn^2 \log q)$
Master Key MK	$O(ln^2 \log^2 q)$
Secret Key SK	$O(ldn^2 \log q)$
Ciphertext CT	$O(ldn \log q)$

Table 7.3: Key Sizes of the general HFBE Scheme

Algorithm	**Complexity**
SetUp	$O(l \cdot \mathrm{poly}(n))$
KeyDerive	$O(l \cdot \mathrm{poly}(n) + ldn^2 \log^2 q)$
Enc	$O(ldn^3 \log^2 q)$
Dec	$O(l \cdot \mathrm{poly}(n) + ldn^3 \log^2 q)$

Table 7.4: Complexity of the general HFBE Scheme

Chapter 8

Ideal Lattice-Based Encryption

In this chapter we show a version of the IBE and HIBE schemes proposed by Agrawal, Boneh and Boyen [4] using ideal lattices. Section 8.1 describes the general scheme as proposed by Yang, Wu, Zhang and Chen [80] and Section 8.2 describes our contribution, a hierarchical version of the same scheme as proposed by Mochetti and Dahab [54].[1]

8.1 Ideal Lattice-Based Identity-Based Encryption

This Section reviews the IBE scheme proposed by Agrawal et al. [4] using ideal lattices as described by Yang et al. [80]. Section 8.1.1 describes the four algorithms that comprise the ideal IBE scheme, Sections 8.1.2, 8.1.3, 8.1.4 and 8.1.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

8.1.1 Description

As described in Section 3.2, an Identity-Based Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n)$, $\mathsf{KeyGen}(PK, MK, \boldsymbol{id})$, $\mathsf{Enc}(PK, M, \boldsymbol{id})$ and $\mathsf{Dec}(PK, SK, CT)$. In this section we describe each algorithm as presented by Yang et al. [80].

SetUp creates an ideal lattice and chooses at random the vectors that will form the public and master keys. KeyGen generates the secret key by encoding the identity $\boldsymbol{id}$ into the lattice basis. The secret key is a ring $\hat{\boldsymbol{e}}$ created by the $\mathsf{SampleLeft}$ algorithm (described in Section 2.2), using the matrix created with the identity as the lattice basis; therefore $\hat{\boldsymbol{e}} \in \Lambda_q^{\boldsymbol{u}}(\mathsf{Rot}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}}))$, with $\hat{\boldsymbol{a}}_{\boldsymbol{id}} = \left[\begin{smallmatrix} \hat{\boldsymbol{a}} \\ \hat{\boldsymbol{a}}_0 + \boldsymbol{id} \cdot \hat{\boldsymbol{b}} \end{smallmatrix}\right]$.

Enc uses the message M, the identity $\boldsymbol{id}$ and the vectors in the public key to create an integer c' and rings $\hat{\boldsymbol{c}}_0$ and $\hat{\boldsymbol{c}}_1$ that will compose the ciphertext for one bit. Finally,

[1] Although really similar, the schemes described in [54] were developed independently of the schemes described in [80].

Dec can recover the message from the ciphertext only if the identity used during the key generation is the same as the one used during the encryption.

Let $n = 2^\alpha$ be the security parameter, σ be the Gaussian parameter and $\mathcal{R} = \mathbb{Z}_q[x]/f(x)$, with $f(x) = x^n + 1$. Algorithms 8.1, 8.2, 8.3 and 8.4 describe the ideal IBE scheme.

Algorithm 8.1 ideal-IBE-SetUp(): Setup Algorithm for the ideal IBE Scheme

```
Input: security parameter $1^n$
Output: Public key $PK$ and master key $MK$
    $\hat{\boldsymbol{a}}, S \leftarrow \mathsf{IdealTrapGen}(n, k, q, \sigma)$
    $\hat{\boldsymbol{a}}_0, \hat{\boldsymbol{b}} \xleftarrow{\$} \mathcal{R}_q^k$
    $\boldsymbol{u} \xleftarrow{\$} \mathcal{R}_q$
    public key $PK = (\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}_0, \hat{\boldsymbol{b}}, \boldsymbol{u})$
    master key $MK = S$
```

Algorithm 8.2 ideal-IBE-KeyGen(): Key Generation Algorithm for the ideal IBE Scheme

```
Input: Public key $PK$, master key $MK$ and identity $\boldsymbol{id}$
Output: Secret key $SK$
    $\hat{\boldsymbol{c}} = \hat{\boldsymbol{a}}_0 + \boldsymbol{id} \cdot \hat{\boldsymbol{b}}$
    $\hat{\boldsymbol{e}} \leftarrow \mathsf{SampleLeft}(\mathsf{Rot}_f(\hat{\boldsymbol{a}}), \mathsf{Rot}_f(\hat{\boldsymbol{c}}), S, \boldsymbol{u}, \sigma)$
    secret key $SK = \hat{\boldsymbol{e}} \in \mathcal{R}_q^{2k}$
```

Algorithm 8.3 ideal-IBE-Enc(): Encryption Algorithm for the ideal IBE Scheme

```
Input: Public key $PK$, message $M$ and identity $\boldsymbol{id}$
Output: Ciphertext $CT$
    $\boldsymbol{s} \xleftarrow{\$} \mathcal{R}_q$
    $\hat{\boldsymbol{x}} \in \overline{\Psi}_\alpha^{kn}$ and $x \in \overline{\Psi}_\alpha$
    $\hat{\boldsymbol{r}}_\gamma \xleftarrow{\$} \mathcal{R}^k$ with coefficients in $\{-1, 1\}$, for $\gamma \in [1, k]$
    $R = [\mathsf{Rot}_f(\hat{\boldsymbol{r}}_1)^\top | \cdots | \mathsf{Rot}_f(\hat{\boldsymbol{r}}_k)^\top]$
    $\hat{\boldsymbol{c}}_0 = \hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}} \in \mathcal{R}_q^k$
    $\hat{\boldsymbol{c}}_1 = (\hat{\boldsymbol{a}}_0 + \boldsymbol{id} \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R^\top \hat{\boldsymbol{x}} \in \mathcal{R}_q^k$
    $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\hat{\boldsymbol{c}}_0, \hat{\boldsymbol{c}}_1, c')$
```

Algorithm 8.4 ideal-IBE-Dec(): Decryption Algorithm for the ideal IBE Scheme

```
Input: Public key $PK$, secret key $SK$ and ciphertext $CT$
Output: message $M$
    $z = c' - \hat{\boldsymbol{e}}^\top \left[\begin{smallmatrix}\hat{\boldsymbol{c}}_0 \\ \hat{\boldsymbol{c}}_1\end{smallmatrix}\right] \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

Note that $R^\top \hat{\boldsymbol{x}} = (\hat{\boldsymbol{r}}_1 \otimes \hat{\boldsymbol{x}}, \cdots, \hat{\boldsymbol{r}}_k \otimes \hat{\boldsymbol{x}})$.

8.1.2 Correctness

The correctness is straightforward. First we just substitute the values of $\hat{\boldsymbol{c}}_0$, $\hat{\boldsymbol{c}}_1$ and c' in z. If the identity used during the key generation is the same as the one used during the encryption, then $\hat{\boldsymbol{a}}_{\boldsymbol{id}}$ is $\left[\begin{smallmatrix}\hat{\boldsymbol{a}} \\ \hat{\boldsymbol{a}}_0 + \boldsymbol{id}\cdot\hat{\boldsymbol{b}}\end{smallmatrix}\right]$. We have for $f(x) = x^n + 1$ that $\hat{\boldsymbol{a}}_{\boldsymbol{id}} \cdot \boldsymbol{s} = \mathsf{Rot}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}})^\top \boldsymbol{s}$ (see Lemma C.1) and, therefore, $\hat{\boldsymbol{e}}^\top \hat{\boldsymbol{a}}_{\boldsymbol{id}} \cdot \boldsymbol{s} = \hat{\boldsymbol{e}}^\top \mathsf{Rot}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}})^\top \boldsymbol{s} = \boldsymbol{u}^\top \boldsymbol{s}$. So, we cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get the right value of M in z.

$$\begin{aligned}
z &= c' - \hat{\boldsymbol{e}}^\top \begin{bmatrix}\hat{\boldsymbol{c}}_0 \\ \hat{\boldsymbol{c}}_1\end{bmatrix} \mod q \\
&= c' - \hat{\boldsymbol{e}}^\top \begin{bmatrix}\hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}} \\ (\hat{\boldsymbol{a}}_0 + \boldsymbol{id} \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R^\top \hat{\boldsymbol{x}}\end{bmatrix} \mod q \\
&= c' - \hat{\boldsymbol{e}}^\top \begin{bmatrix}\hat{\boldsymbol{a}} \\ (\hat{\boldsymbol{a}}_0 + \boldsymbol{id} \cdot \hat{\boldsymbol{b}})\end{bmatrix} \cdot \boldsymbol{s} - \hat{\boldsymbol{e}}^\top \begin{bmatrix}\hat{\boldsymbol{x}} \\ R^\top \hat{\boldsymbol{x}}\end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \hat{\boldsymbol{e}}^\top \begin{bmatrix}\hat{\boldsymbol{x}} \\ R^\top \hat{\boldsymbol{x}}\end{bmatrix} \mod q \\
&= x + M\lfloor q/2 \rfloor - \hat{\boldsymbol{e}}^\top \begin{bmatrix}\hat{\boldsymbol{x}} \\ R^\top \hat{\boldsymbol{x}}\end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}$$

Note that for the correct decryption the error term must be less than $q/4$.

8.1.3 Security

In this section we prove the following theorem.

Theorem 8.1. *If the decision-*Ring-LWE *problem is infeasible, then the functional encryption scheme described in Section 8.1.1 is IND-AH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

$\textsf{\textbf{Sim.ideal-IBE-SetUp}}(1^n, \boldsymbol{id}^\star)$: The algorithm chooses random ring $\hat{\boldsymbol{a}} \in \mathcal{R}_q^k$ and $\hat{\boldsymbol{r}}_\gamma^\star \in \mathcal{R}^k$ with coefficients in $\{-1, 1\}$ and vector $\boldsymbol{u} \in \mathcal{R}_q$ and it uses $\mathsf{IdealTrapGen}(q, n, k, \sigma)$ to generate $\hat{\boldsymbol{b}}^\star \in \mathcal{R}_q^k$ and the basis $S^\star \in \mathbb{Z}^{kn \times kn}$ for $\Lambda_q^\perp(\mathsf{Rot}_f(\hat{\boldsymbol{b}}^\star))$. It then defines $\hat{\boldsymbol{a}}_0 \leftarrow \hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star$, with $R^\star = [\mathsf{Rot}_f(\hat{\boldsymbol{r}}_1^\star)^\top | \cdots | \mathsf{Rot}_f(\hat{\boldsymbol{r}}_k^\star)^\top]$ and outputs $PK = (\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}_0, \boldsymbol{u})$ and $MK = (\{\hat{\boldsymbol{r}}_\gamma^\star\}, \hat{\boldsymbol{b}}^\star, S^\star)$.

$\textsf{\textbf{Sim.ideal-IBE-KeyGen}}(PK, MK, \boldsymbol{id})$: Secret keys are now sampled by the $\mathsf{SampleRight}$ algorithm, using the trapdoor $S^\star$. It outputs

$SK = \hat{\boldsymbol{e}} \in \Lambda_q^{\boldsymbol{u}}(\mathsf{Rof}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}})$, with $\hat{\boldsymbol{a}}_{\boldsymbol{id}} = \left[\begin{smallmatrix} \hat{\boldsymbol{a}} \\ \hat{\boldsymbol{a}}R - (\boldsymbol{id} + \boldsymbol{id}^\star) \cdot \hat{\boldsymbol{b}}^\star \end{smallmatrix}\right]$, where

$\hat{\boldsymbol{e}} \leftarrow \mathsf{SampleRight}(\mathsf{Rof}_f(\hat{\boldsymbol{a}}), (\boldsymbol{id} - \boldsymbol{id}^\star)\mathsf{Rof}_f(\hat{\boldsymbol{b}}^\star), R^\star, S^\star, \boldsymbol{u}, \sigma)$,

with $R^\star = [\mathsf{Rot}_f(\hat{\boldsymbol{r}}_1^\star)^\top | \cdots | \mathsf{Rot}_f(\hat{\boldsymbol{r}}_k^\star)^\top]$.

Note that we must have $\boldsymbol{id} \neq \boldsymbol{id}^\star$ for the algorithm $\mathsf{SampleRight}$ to work properly.

$\textsf{\textbf{Sim.ideal-IBE-Enc}}(PK, M, \boldsymbol{id}^\star)$: The algorithm differs from $\mathsf{ideal\text{-}IBE\text{-}Enc}$ in the sense that it uses rings $\hat{\boldsymbol{r}}_\gamma^\star$ and $\hat{\boldsymbol{b}}^\star$ instead of rings $\hat{\boldsymbol{r}}_\gamma$ and $\hat{\boldsymbol{b}}$.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs $\mathsf{ideal\text{-}IBE\text{-}SetUp}$, answers $\mathcal{A}$'s secret key queries using algorithm $\mathsf{ideal\text{-}IBE\text{-}KeyGen}$, and generates the challenge ciphertext using the $\mathsf{ideal\text{-}IBE\text{-}Enc}$ with identity $\boldsymbol{id}^{\star 0}$ and M_0.

- **Game 1**: $\mathcal{C}$ runs $\mathsf{Sim.ideal\text{-}IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.ideal\text{-}IBE\text{-}KeyGen}$, and generates the challenge ciphertext using the $\mathsf{Sim.ideal\text{-}IBE\text{-}Enc}$ algorithm with $\boldsymbol{id}^{\star 0}$ and M_0.

- **Game 2**: $\mathcal{C}$ runs $\mathsf{Sim.ideal\text{-}IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 0}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.ideal\text{-}IBE\text{-}KeyGen}$, and generates the challenge ciphertext randomly.

- **Game 3**: $\mathcal{C}$ runs $\mathsf{Sim.ideal\text{-}IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.ideal\text{-}IBE\text{-}KeyGen}$, and generates the challenge ciphertext randomly.

- **Game 4**: $\mathcal{C}$ runs $\mathsf{Sim.ideal\text{-}IBE\text{-}SetUp}$ with identity $\boldsymbol{id}^{\star 1}$, answers $\mathcal{A}$'s secret key queries using $\mathsf{Sim.ideal\text{-}IBE\text{-}KeyGen}$, and generates the challenge ciphertext using the $\mathsf{Sim.ideal\text{-}IBE\text{-}Enc}$ algorithm with $\boldsymbol{id}^{\star 1}$ and M_1.

- **Game 5**: $\mathcal{C}$ runs $\mathsf{ideal\text{-}IBE\text{-}SetUp}$, answers $\mathcal{A}$'s secret key queries using algorithm $\mathsf{ideal\text{-}IBE\text{-}KeyGen}$, and generates the challenge ciphertext using the $\mathsf{ideal\text{-}IBE\text{-}Enc}$ with identity $\boldsymbol{id}^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 8.1. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, ring $\hat{\boldsymbol{a}}$ is generated by $\mathsf{IdealTrapGen}$ and ring $\hat{\boldsymbol{a}}_0$ is uniformly random in $\mathcal{R}_q^k$. Instead, in Game 1, $\hat{\boldsymbol{a}}$ is chosen uniformly at random and we have $\hat{\boldsymbol{a}}_0 \leftarrow \hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star$, where $\hat{\boldsymbol{b}}^\star$ is generated by $\mathsf{IdealTrapGen}$ and the rings $\hat{\boldsymbol{r}}_\gamma^\star$ are uniformly and independently chosen at random with coefficients in $\{-1, 1\}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathcal{R}_q$.

Secret keys In Game 0, the secret key for identity $\boldsymbol{id}$ is a ring $\hat{\boldsymbol{e}} \in \Lambda_q^{\boldsymbol{u}}(\mathsf{Rot}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}}))$, sampled using the $\mathsf{SampleLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext In both games the challenge ciphertext components c' and $\hat{\boldsymbol{c}}_0$ are computed the same way but, in Game 0, the challenge ciphertext component $\hat{\boldsymbol{c}}_1$ is computed as follows:

$$\hat{\boldsymbol{c}}_1 = (\hat{\boldsymbol{a}}_0 + \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}} \in \mathcal{R}_q^k \ .$$

On the other hand, in Game 1, we have:

$$\begin{aligned}\hat{\boldsymbol{c}}_1 &= (\hat{\boldsymbol{a}}_0 + \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}} \\ &= (\hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star + \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}} \ . \\ &= (\hat{\boldsymbol{a}}R^\star) \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}} \in \mathcal{R}_q^k\end{aligned}$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}_0, \hat{\boldsymbol{c}}_1)$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemma A.8 we have that the following two distributions are statistically indistinguishable for every fixed matrix $\hat{\boldsymbol{b}}^\star$, every $\boldsymbol{id}^\star$ and every vector $\hat{\boldsymbol{x}} \in \mathcal{R}_q^k$:

$$\left(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}_0, R^{\star\top}\hat{\boldsymbol{x}}\right) \approx_s \left(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star, R^{\star\top}\hat{\boldsymbol{x}}\right) \ .$$

Since $(\hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s}$ is statistically close to $\hat{\boldsymbol{a}}_0 \cdot \boldsymbol{s}$, it is possible to add each term to each side of the equation:

$$\begin{gathered}\left(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}_0, \hat{\boldsymbol{a}}_0 \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}}\right) \approx_s \\ \left(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star, (\hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}}\right)\end{gathered} \ .$$

Then, we add $(\boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s}$ to each side of the equation:

$$\begin{aligned}\Big(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}_0, (\hat{\boldsymbol{a}}_0 + \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}}\Big) &\approx_s \\ \Big(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star, (\hat{\boldsymbol{a}}R^\star) \cdot \boldsymbol{s} + R^{\star\top}\hat{\boldsymbol{x}}\Big) &.\end{aligned}$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 8.2. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*Ring-LWE *and decision-*LWE*.*

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-Ring-LWE.

Init $\mathcal{B}$ is given k Ring-LWE challenge pairs $(\boldsymbol{a}_j, \boldsymbol{y}_j) \in \mathcal{R}_q \times \mathcal{R}_q$, where either $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$ for a random $\boldsymbol{s} \in \mathcal{R}_q$ and a noise term $\boldsymbol{x}_j \leftarrow \Psi_\alpha^n$, or $\boldsymbol{y}_j$ is uniformly random in $\mathcal{R}_q$. And one LWE challenge pair $(\boldsymbol{a}_0, y_0) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where either $y_0 = \langle \boldsymbol{a}_0, \boldsymbol{s}\rangle + x_j$ for a noise term $x_j \leftarrow \Psi_\alpha$, or y_0 is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_j, \boldsymbol{y}_j)$. The i-th polynomial of ring $\hat{\boldsymbol{a}}$ will be the vector $\boldsymbol{a}_i$, for $1 \le i \le k$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The ring $\hat{\boldsymbol{a}}_0$ is still calculated as in Sim.ideal-IBE-SetUp, i.e., $\hat{\boldsymbol{a}}_0 \leftarrow \hat{\boldsymbol{a}}R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star$.

Secret keys All private-key extraction queries are answered using Sim.ideal-IBE-KeyGen.

Challenge Ciphertext The ciphertext $CT = (\hat{\boldsymbol{c}}_0^\star, \hat{\boldsymbol{c}}_1^\star, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_j, \boldsymbol{y}_j)$, with $\hat{\boldsymbol{c}}_0^\star = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m)$, $\boldsymbol{c}_1^\star = R^{\star\top}\hat{\boldsymbol{c}}_0^\star$ and $c'^\star = y_0 + M\lfloor q/2 \rceil$. If we have $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$ on the Ring-LWE challenge and $y_0 = \langle \boldsymbol{a}_0, \boldsymbol{s}\rangle + x$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if $\boldsymbol{y}_j$ is uniformly random in $\mathcal{R}_q$ and y_0 is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2. If $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$, then

$(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m) = (\boldsymbol{a}_1 \cdot \boldsymbol{s} + \boldsymbol{x}_1, \ldots, \boldsymbol{a}_k \cdot \boldsymbol{s} + \boldsymbol{x}_m) = \hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}}$.

Therefore, for Game 1 we have

$$\begin{aligned}\boldsymbol{c}_0^\star &= \hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}} \\ &= (y_1, \ldots, y_m)\ ,\end{aligned}$$

and

$$\begin{aligned}\boldsymbol{c}_1^\star &= (\hat{\boldsymbol{a}}_0 + \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R^{\star\top} \hat{\boldsymbol{x}} \\ &= (\hat{\boldsymbol{a}} R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}} + \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R^{\star\top} \hat{\boldsymbol{x}} \\ &= (\hat{\boldsymbol{a}} R^\star) \cdot \boldsymbol{s} + R^{\star\top} \hat{\boldsymbol{x}} \\ &= R^{\star\top} (\hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}}) \\ &= R^{\star\top} \hat{\boldsymbol{c}}_0^\star \,.\end{aligned}$$

If $\boldsymbol{y}_j$ is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the Ring-LWE and LWE challenges, because, as we showed, if $\boldsymbol{y}_j$ is uniformly random in $\mathcal{R}_q$ and y_0 is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_0 = \langle \boldsymbol{a}_0, \boldsymbol{s} \rangle + x_0$ and $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$, then $\mathcal{A}$'s view is the same as in Game 1.

□

Indistinguishability of Game 2 and Game 3

Lemma 8.3. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. $\hat{\boldsymbol{a}}$ and $\boldsymbol{u}$ are random and $\hat{\boldsymbol{a}}_0 \leftarrow \hat{\boldsymbol{a}} R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star$, with $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 0}$ for Game 2 and $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 1}$ for Game 3.

Secret keys All private-key extraction queries are answered using Sim.ideal-IBE-KeyGen. The only difference is, again, that for Game 2, $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 0}$ and, for Game 3, $\boldsymbol{id}^\star = \boldsymbol{id}^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for ring $\hat{\boldsymbol{a}}_0$. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of $\hat{\boldsymbol{a}}_0$. From Lemma A.8 we can prove that $\hat{\boldsymbol{a}}_0$ for each game is statistically close to a uniformly random matrix, because

$\hat{\boldsymbol{a}}_0 \leftarrow \hat{\boldsymbol{a}} R^\star - \boldsymbol{id}^\star \cdot \hat{\boldsymbol{b}}^\star$.

□

8.1.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma\sqrt{2kn}$ from Lemma A.1 and $\|R\boldsymbol{e}\| \leq 12\sqrt{2kn}\|\boldsymbol{e}\|$ from Lemma A.2; therefore, for $\boldsymbol{e} = \begin{bmatrix}\boldsymbol{e}_1\\ \boldsymbol{e}_2\end{bmatrix}$:

$$\begin{aligned}\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| &\leq (\sigma\sqrt{2kn} + 12\sqrt{2kn}\sigma\sqrt{2kn}) \qquad \text{so}\\ \|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| &\leq O(\sigma kn)\ .\end{aligned}$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \leq \|\boldsymbol{y}\|q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\begin{aligned}\langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq O(\sigma kn)q\alpha w(\sqrt{\log 2kn}) + O(\sigma kn)\sqrt{2kn}/2\\ \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq \widetilde{O}(\sigma knq\alpha) + O(\sigma(kn)^{3/2})\ .\end{aligned}$$

To ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned}&x - \boldsymbol{e}^\top\begin{bmatrix}\boldsymbol{x}\\ R^\top\boldsymbol{x}\end{bmatrix} < q/4\\ &x - \boldsymbol{e}_1^\top\boldsymbol{x} - \boldsymbol{e}_2^\top R^\top\boldsymbol{x} < q/4\\ &x - (\boldsymbol{e}_1 + R\boldsymbol{e}_2)^\top\boldsymbol{x} < q/4\\ &x - \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle < q/4\\ &\widetilde{O}(\sigma knq\alpha) + O(\sigma(kn)^{3/2}) < q/4\end{aligned}$$

To ensure that σ is sufficiently large for SampleLeft and SampleRight (Theorems 2.7 and 2.8), we have

$$\sigma > \|S\|\sqrt{2kn}\omega(\sqrt{\log 2kn}).$$

To ensure that IdealTrapGen (Theorem 2.2) can operate, we have

$$\begin{aligned}&k \geq \lceil \log q + 1\rceil \qquad \text{and}\\ &\|S\| = O(n\log q\sqrt{\omega(\log n)})\ .\end{aligned}$$

To ensure that the reduction applies (Theorem 2.14), we have

$$q > \omega(\sqrt{\log n})/\alpha\ .$$

Therefore, we need to set the parameters as

$$\begin{aligned}&k = O(\log q),\\ &q = (kn)^{2.5}\omega(\sqrt{\log n}),\\ &\alpha = (k^2n^2\omega(\sqrt{\log n}))^{-1},\\ &\sigma = kn\omega(\sqrt{\log n}).\end{aligned}$$

8.1.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 8.1.1. Note that for security parameter n and modulus q, we have $k = O(\log q)$ (see Section 8.1.4).

The master key MK is just an $kn \times kn$ matrix, therefore its size is $(kn)^2$. The public key PK is comprised of a vector of length n and three rings of lenght kn; therefore its size is $O(kn)$. The secret key is just a vector of length $2kn$; therefore its size is $O(km)$. Finally, the ciphertext is comprised of an integer and two vectors of length kn; therefore its size is also $O(kn)$.

The complexity of ideal-IBE-SetUp is based on the complexity of the IdealTrapGen algorithm. By Theorem 2.2, we have that the IdealTrapGen algorithm is polynomial, and, therefore, ideal-IBE-SetUp is also polynomial. The complexity of ideal-IBE-KeyGen is based on the complexity of the SampleLeft algorithm plus one ring-vector multiplication ($O(kn)$) and one ring addition ($O(kn)$). As before, we have that the SampleLeft algorithm is polynomial, by Theorem 2.7.

The ideal-IBE-Enc algorithm does three ring-vector multiplications ($O(kn)$), three ring addictions ($O(kn)$), k ring-ring multiplications ($O(k^2n)$), one inner product ($O(n)$) and two simple additions $O(1)$. Therefore, the complexity of ideal-IBE-Enc is based on the ring-ring multiplications. The ideal-IBE-Dec algorithm does only the inner product between two vectors and a simple addition. Since the vectors are of length $2kn$, we have that the complexity of ideal-IBE-Dec is $O(kn)$.

Table 8.1 summarises the size of the main variables and Table 8.2 summarises the complexity of the four algorithms of the scheme described in Section 8.1.1.

Variable	**Size**
Public Key PK	$O(n \log q)$
Master Key MK	$O(n^2 \log^2 q)$
Secret Key SK	$O(n \log q)$
Ciphertext CT	$O(n \log q)$

Table 8.1: Key Sizes of the general ideal IBE Scheme

Algorithm	**Complexity**
SetUp	$O(\text{poly}(n))$
KeyGen	$O(\text{poly}(n) + n \log q)$
Enc	$O(n \log^2 q)$
Dec	$O(n \log q)$

Table 8.2: Complexity of the general ideal IBE Scheme

8.2 Ideal Lattice-Based Hierarchical Identity-Based Encryption

This Section reviews the HIBE scheme proposed by Agrawal et al. [4] using ideal lattices as described by Mochetti et al. [54]. Section 8.2.1 describes the four algorithms that comprise the ideal HIBE scheme, Sections 8.2.2, 8.2.3, 8.2.4 and 8.2.5 give the correctness, security, parameters and complexity analysis of the underlined scheme, respectively.

8.2.1 Description

As described in Section 3.1, an Hierarchical Identity-Based Encryption Scheme consists of the following algorithms: $\mathsf{SetUp}(1^n, \mu)$, $\mathsf{KeyDerive}(PK, SK_{t-1}, \boldsymbol{id}_1, \cdots, \boldsymbol{id}_t)$, $\mathsf{Enc}(PK, M, \boldsymbol{id}_1, \cdots, \boldsymbol{id}_t)$ and $\mathsf{Dec}(PK, SK_t, CT)$. In this section we describe each algorithm as presented by Mochetti et al. [54]. The hierarchy is described by parameter μ and has maximum depth d.

SetUp creates an ideal lattice and chooses at random $d + 1$ rings and a vector that will form the public and master keys. KeyDerive generates the secret key by encoding each identity $\boldsymbol{id}_i$ into the lattice basis. Now, the secret key is a short basis for the lattice generate by this encoding, using the algorithm SampleBasisLeft, described in Section 2.2. Note that $SK_0 = MK$.

Enc uses the message M, the identities $\boldsymbol{id}_i$ and the rings in the public key to create an integer c', ring $\hat{\boldsymbol{c}}_0$ and t rings $\hat{\boldsymbol{c}}_i$, one for each level, that will compose the ciphertext for one bit. Finally, Dec uses the algorithm SamplePre with the basis that comprise SK_t to find a ring $\hat{\boldsymbol{e}} \in \Lambda_q^{\boldsymbol{u}}(A|C_1|\cdots|C_t)$ and then recover the message from the ciphertext only if all the identities $\boldsymbol{id}_i$ used are the same, for all $j \in [1, t]$.

Let $n = 2^\alpha$ be the security parameter, μ be the hierarchical parameter, σ_i (for $i \in [1, d]$) be the Gaussian parameters and $\mathcal{R} = \mathbb{Z}_q[x]/f(x)$, with $f(x) = x^n + 1$. Algorithms 8.5, 8.6, 8.7 and 8.8 describe the ideal HIBE scheme.

Algorithm 8.5 ideal-HIBE-SetUp(): Setup Algorithm for the ideal HIBE Scheme

```
Input: security parameter $1^n$ and hierarchical parameter $1^\mu$
Output: Public key $PK$ and master key $MK$
    $\hat{\boldsymbol{a}}, S \leftarrow \mathsf{IdealTrapGen}(n, k, q, \sigma)$
    $\hat{\boldsymbol{a}}_i \overset{\$}{\leftarrow} \mathcal{R}_q^k$, for $i \in [1, d]$
    $\hat{\boldsymbol{b}} \overset{\$}{\leftarrow} \mathcal{R}_q^k$
    $\boldsymbol{u} \overset{\$}{\leftarrow} \mathcal{R}_q$
    public key $PK = (\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}_i\}, \hat{\boldsymbol{b}}, \boldsymbol{u})$
    master key $MK = S$
```

Algorithm 8.6 ideal-HIBE-KeyDerive(): Key Generation Algorithm for the ideal HIBE Scheme

```
Input: Public key $PK$, secret key $SK_{t-1}$ and identities $\boldsymbol{id}_1, \cdots, \boldsymbol{id}_t$
Output: Secret key $SK_t$
    $\hat{\boldsymbol{c}}_i = \hat{\boldsymbol{a}}_i + \boldsymbol{id}_i \cdot \hat{\boldsymbol{b}}$, for $i \in [1, t]$
    $C = [\mathsf{Rot}_f(\hat{\boldsymbol{c}}_1)| \cdots |\mathsf{Rot}_f(\hat{\boldsymbol{c}}_{t-1})]$
    $A = \mathsf{Rot}_f(\hat{\boldsymbol{a}})$
    $C_t = \mathsf{Rot}_f(\hat{\boldsymbol{c}}_t)$
    $S_t \leftarrow \mathsf{SampleBasisLeft}([A|C], C_t, S_{t-1}, \sigma_t)$
    secret key $SK_t = S_t \in \mathbb{Z}_q^{n \times (t+1)kn}$
```

Algorithm 8.7 ideal-HIBE-Enc(): Encryption Algorithm for the ideal HIBE Scheme

```
Input: Public key $PK$, message $M$ and identities $\boldsymbol{id}_1, \cdots, \boldsymbol{id}_t$
Output: Ciphertext $CT$
    $\boldsymbol{s} \overset{\$}{\leftarrow} \mathcal{R}_q$
    $\hat{\boldsymbol{x}} \in \overline{\Psi}_\alpha^{kn}$ and $x \in \overline{\Psi}_\alpha$
    $\hat{\boldsymbol{r}}_{\gamma,i} \overset{\$}{\leftarrow} \mathcal{R}$ with coefficients in $\{-1, 1\}$, for $\gamma \in [1, k]$ and $i \in [1, t]$
    $R_i = [\mathsf{Rot}_f(\hat{\boldsymbol{r}}_{1,i})^\top | \cdots |\mathsf{Rot}_f(\hat{\boldsymbol{r}}_{k,i})^\top]$
    $\hat{\boldsymbol{c}}_0 = \hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}} \in \mathcal{R}_q^k$
    $\hat{\boldsymbol{c}}_i = (\hat{\boldsymbol{a}}_i + \boldsymbol{id}_i \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R_i^\top \hat{\boldsymbol{x}} \in \mathcal{R}_q^k$
    $c' = \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor$
    ciphertext $CT = (\hat{\boldsymbol{c}}_0, \{\hat{\boldsymbol{c}}_i\}, c')$
```

Algorithm 8.8 ideal-HIBE-Dec(): Decryption Algorithm for the ideal HIBE Scheme

```
Input: Public key $PK$, secret key $SK_t$ and ciphertext $CT$
Output: message $M$
    $\hat{\boldsymbol{c}}_i = \hat{\boldsymbol{a}}_i + \boldsymbol{id}_i \cdot \hat{\boldsymbol{b}}$, for $i \in [1, t]$
    $C_i = \mathsf{Rot}_f(\hat{\boldsymbol{c}}_i)$, for $i \in [1, t]$
    $C = [C_1 | \cdots | C_t]$
    $A = \mathsf{Rot}_f(\hat{\boldsymbol{a}})$
    $\sigma = \sigma_t \sqrt{kn(t+1)}\omega(\sqrt{\log(tkn)})$
    $\hat{\boldsymbol{e}} = \mathsf{SamplePre}([A|C], S_t, \boldsymbol{u}, \sigma)$
    $z = c' - \hat{\boldsymbol{e}}^\top \begin{bmatrix} \hat{\boldsymbol{c}}_0 \\ \hat{\boldsymbol{c}}_1 \\ \vdots \\ \hat{\boldsymbol{c}}_t \end{bmatrix} \mod q$
    if $|z| < q/4$, then $M = 0$; else $M = 1$
```

Note that $R_i^\top \hat{\boldsymbol{x}} = (\hat{\boldsymbol{r}}_{1,i} \otimes \hat{\boldsymbol{x}}, \cdots, \hat{\boldsymbol{r}}_{k,i} \otimes \hat{\boldsymbol{x}})$.

8.2.2 Correctness

The correctness is straightforward. First we just substitute the values of $\hat{\boldsymbol{c}}_0$, $\hat{\boldsymbol{c}}_i$ and c' in z. If the identity used during the key generation is the same as the one used during the encryption, then

$$\hat{\boldsymbol{a}}_{\boldsymbol{id}} = \begin{bmatrix} \hat{\boldsymbol{a}} \\ (\hat{\boldsymbol{a}}_1 + \boldsymbol{id}_1 \cdot \hat{\boldsymbol{b}}) \\ \vdots \\ (\hat{\boldsymbol{a}}_t + \boldsymbol{id}_t \cdot \hat{\boldsymbol{b}}) \end{bmatrix}.$$

We have for $f(x) = x^n + 1$ that $\hat{\boldsymbol{a}}_{\boldsymbol{id}} \cdot \boldsymbol{s} = \mathsf{Rot}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}})^\top \boldsymbol{s}$ (see Lemma C.1) and, therefore, $\hat{\boldsymbol{e}}^\top \hat{\boldsymbol{a}}_{\boldsymbol{id}} \cdot \boldsymbol{s} = \hat{\boldsymbol{e}}^\top \mathsf{Rot}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}})^\top \boldsymbol{s} = \boldsymbol{u}^\top \boldsymbol{s}$. So, we cancel the terms $\boldsymbol{u}^\top \boldsymbol{s}$, identify all terms that refer to the "noise" and get the right value of M in z.

$$\begin{aligned}
z &= c' - \hat{\boldsymbol{e}}^\top \begin{bmatrix} \hat{\boldsymbol{c}}_0 \\ \hat{\boldsymbol{c}}_1 \\ \vdots \\ \hat{\boldsymbol{c}}_t \end{bmatrix} \mod q \\
&= c' - \hat{\boldsymbol{e}}^\top \begin{bmatrix} \hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}} \\ (\hat{\boldsymbol{a}}_1 + \boldsymbol{id}_1 \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R_1^\top \hat{\boldsymbol{x}} \\ \vdots \\ (\hat{\boldsymbol{a}}_t + \boldsymbol{id}_t \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R_t^\top \hat{\boldsymbol{x}} \end{bmatrix} \mod q \\
&= c' - \hat{\boldsymbol{e}}^\top \begin{bmatrix} \hat{\boldsymbol{a}} \\ (\hat{\boldsymbol{a}}_1 + \boldsymbol{id}_1 \cdot \hat{\boldsymbol{b}}) \\ \vdots \\ (\hat{\boldsymbol{a}}_t + \boldsymbol{id}_t \cdot \hat{\boldsymbol{b}}) \end{bmatrix} \cdot \boldsymbol{s} - \hat{\boldsymbol{e}}^\top \begin{bmatrix} \hat{\boldsymbol{x}} \\ R_1^{\star\top} \hat{\boldsymbol{x}} \\ \vdots \\ R_t^{\star\top} \hat{\boldsymbol{x}} \end{bmatrix} \mod q \\
&= \boldsymbol{u}^\top \boldsymbol{s} + x + M\lfloor q/2 \rfloor - \boldsymbol{u}^\top \boldsymbol{s} - \hat{\boldsymbol{e}}^\top \begin{bmatrix} \hat{\boldsymbol{x}} \\ R^\top \hat{\boldsymbol{x}} \end{bmatrix} \mod q \\
&= x + M\lfloor q/2 \rfloor - \hat{\boldsymbol{e}}^\top \begin{bmatrix} \hat{\boldsymbol{x}} \\ R^\top \hat{\boldsymbol{x}} \end{bmatrix} \mod q \\
&= M\lfloor q/2 \rfloor + err \mod q
\end{aligned}$$

Note that for the correct decryption the error term must be less than $q/4$.

8.2.3 Security

In this section we prove the following theorem.

Theorem 8.2. *If the decision-*Ring-LWE *problem is infeasible, then the functional encryption scheme described in Section 8.2.1 is IND-AH-sAT-CPA.*

We need to define additional algorithms that will not be used in the actual scheme, but will be used in our security proof.

Sim.ideal-HIBE-SetUp$(1^n, 1^\mu, \boldsymbol{id}_1^\star, \cdots, \boldsymbol{id}_d^\star)$: The algorithm chooses random ring $\hat{\boldsymbol{a}} \in \mathcal{R}_q^k$ and $\hat{\boldsymbol{r}}_{\gamma,i}^\star \in \mathcal{R}$ with coefficients in $\{-1, 1\}$ and vector $\boldsymbol{u} \in \mathcal{R}_q$ and it uses algorithm IdealTrapGen(q, n, k, σ) to generate $\hat{\boldsymbol{b}}^\star \in \mathcal{R}_q^k$ and the basis $S^\star \in \mathbb{Z}^{kn \times kn}$ for $\Lambda_q^\perp(\mathsf{Rot}_f(\hat{\boldsymbol{b}}^\star))$. It then defines $\hat{\boldsymbol{a}}_i \leftarrow \hat{\boldsymbol{a}} R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star$, with $R_i^\star = [\mathsf{Rot}_f(\hat{\boldsymbol{r}}_{1,i}^\star)^\top | \cdots | \mathsf{Rot}_f(\hat{\boldsymbol{r}}_{k,i}^\star)^\top]$ and outputs $PK = (\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}_i\}, \boldsymbol{u})$ and $MK = (\{\hat{\boldsymbol{r}}_{\gamma,i}^\star\}, \hat{\boldsymbol{b}}^\star, S^\star)$.

Sim.ideal-HIBE-KeyDerive$(PK, MK, \boldsymbol{id}_1, \cdots, \boldsymbol{id}_t)$: Secret keys are now sampled by the SampleBasisRight algorithm, using the trapdoor $S^\star$. It outputs $SK_t = S_t$ which is a basis for lattice $\Lambda_q^{\perp}(\mathsf{Rof}_f(\hat{\boldsymbol{a}}_{(id)}))$, with $\hat{\boldsymbol{a}}_{(id)} = [\hat{\boldsymbol{a}}|\hat{\boldsymbol{a}}R_1^\star - (\boldsymbol{id}_1 + \boldsymbol{id}_1^\star)\cdot \hat{\boldsymbol{b}}^\star|\cdots|\hat{\boldsymbol{a}}R_t^\star - (\boldsymbol{id}_t + \boldsymbol{id}_t^\star)\cdot \hat{\boldsymbol{b}}^\star]$, where:

$S \leftarrow \mathsf{SampleBasisRight}(\mathsf{Rof}_f(\hat{\boldsymbol{a}}), B_{\boldsymbol{id}}^\star, R^\star, S^\star, \sigma_t)$,
with $R^\star = [R_1^\star|\cdots|R_t^\star]$ and $B_{\boldsymbol{id}}^\star = [(\boldsymbol{id}_1 - \boldsymbol{id}_1^\star)B^\star|\cdots|(\boldsymbol{id}_t - \boldsymbol{id}_t^\star)B^\star]$,
for $R_i^\star = [\mathsf{Rot}_f(\hat{\boldsymbol{r}}_{1,i}^\star)^\top|\cdots|\mathsf{Rot}_f(\hat{\boldsymbol{r}}_{k,i}^\star)^\top]$.

Note that we must have $\boldsymbol{id}_i \neq \boldsymbol{id}_i^\star$, for all $i \in [1,t]$, for the algorithm SampleRight to work properly.

Sim.ideal-HIBE-Enc$(PK, M, \boldsymbol{id}_1^\star, \cdots, \boldsymbol{id}_t^\star)$: The algorithm differs from ideal-HIBE-Enc in the sense that it uses rings $\hat{\boldsymbol{r}}_{i,\gamma}^\star$ and $\hat{\boldsymbol{b}}^\star$ instead of rings $\hat{\boldsymbol{r}}_{i,\gamma}$ and $\hat{\boldsymbol{b}}$.

For a probabilistic polynomial-time adversary $\mathcal{A}$, our proof of security will consist of the following sequence of six games between $\mathcal{A}$ and $\mathcal{C}$. The six games are defined as follows:

- **Game 0**: $\mathcal{C}$ runs ideal-HIBE-SetUp, answers $\mathcal{A}$'s secret key queries using the algorithm ideal-HIBE-KeyDerive, and generates the challenge ciphertext using the algorithm ideal-HIBE-Enc with identities $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_t^{\star 0}$ and M_0.
- **Game 1**: $\mathcal{C}$ runs Sim.ideal-HIBE-SetUp with identities $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_d^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.ideal-HIBE-KeyDerive, and generates the challenge ciphertext using the Sim.ideal-HIBE-Enc algorithm with $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_t^{\star 0}$ and M_0.
- **Game 2**: $\mathcal{C}$ runs Sim.ideal-HIBE-SetUp with identity identities $\boldsymbol{id}_1^{\star 0}, \cdots, \boldsymbol{id}_d^{\star 0}$, answers $\mathcal{A}$'s secret key queries using Sim.ideal-HIBE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 3**: $\mathcal{C}$ runs Sim.ideal-HIBE-SetUp with identity identities $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_d^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.ideal-HIBE-KeyDerive, and generates the challenge ciphertext randomly.
- **Game 4**: $\mathcal{C}$ runs Sim.ideal-HIBE-SetUp with identities $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_d^{\star 1}$, answers $\mathcal{A}$'s secret key queries using Sim.ideal-HIBE-KeyDerive, and generates the challenge ciphertext using the Sim.ideal-HIBE-Enc algorithm with $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_t^{\star 1}$ and M_1.
- **Game 5**: $\mathcal{C}$ runs ideal-HIBE-SetUp, answers $\mathcal{A}$'s secret key queries using the algorithm ideal-HIBE-KeyDerive, and generates the challenge ciphertext using the algorithm ideal-HIBE-Enc with identities $\boldsymbol{id}_1^{\star 1}, \cdots, \boldsymbol{id}_t^{\star 1}$ and M_1.

To prove the security of this scheme, we now show that each pair of consecutive games are indistinguishable, therefore proving that Game 0 and Game 5 are indistinguishable.

For simplicity reasons, assume that every time we refer to rings $\hat{\boldsymbol{a}}_i$ and $\hat{\boldsymbol{c}}_i$ and matrices $R_i^\star$ we are referring to all matrices or rings for $i \in [1,d]$.

Indistinguishability of Game 0 and Game 1 (or Game 4 and Game 5)

Lemma 8.4. *The view of the adversary $\mathcal{A}$ in Game 0 (resp. Game 4) is statistically close to the view of $\mathcal{A}$ in Game 1 (resp. Game 5).*

Proof.

SetUp In Game 0, ring $\hat{\boldsymbol{a}}$ is generated by $\mathsf{IdealTrapGen}$ and rings $\hat{\boldsymbol{a}}_i$ are uniformly random in $\mathcal{R}_q^k$. Instead, in Game 1, $\hat{\boldsymbol{a}}$ is chosen uniformly at random and we have $\hat{\boldsymbol{a}}_i \leftarrow \hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star$, where $\hat{\boldsymbol{b}}^\star$ is generated by $\mathsf{IdealTrapGen}$ and the rings $\hat{\boldsymbol{r}}_{\gamma,i}^\star$ are uniformly and independently chosen at random with coefficients in $\{-1, 1\}$. In both games the vector $\boldsymbol{u}$ is chosen at random in $\mathcal{R}_q$.

Secret keys In Game 0, the secret key for identity $\boldsymbol{id}$ is a basis of $\Lambda_q^\perp(\mathsf{Rot}_f(\hat{\boldsymbol{a}}_{\boldsymbol{id}}))$ sampled using the $\mathsf{SampleBasisLeft}$ algorithm. The same happens in Game 1 by using $\mathsf{SampleBasisRight}$. Thus, the secret keys have the same distribution in both games.

Challenge Ciphertext In both games the challenge ciphertext components c' and $\hat{\boldsymbol{c}}_0$ are computed the same way but, in Game 0, the challenge ciphertext components $\hat{\boldsymbol{c}}_i$ are computed as follows:

$$\hat{\boldsymbol{c}}_i = (\hat{\boldsymbol{a}}_i + \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R_i^{\star\top} \hat{\boldsymbol{x}} \in \mathcal{R}_q^k \ .$$

On the other hand, in Game 1, we have:

$$\begin{aligned} \hat{\boldsymbol{c}}_i &= (\hat{\boldsymbol{a}}_i + \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R_i^{\star\top} \hat{\boldsymbol{x}} \\ &= (\hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star + \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R_i^{\star\top} \hat{\boldsymbol{x}} \ . \\ &= (\hat{\boldsymbol{a}}R_i^\star) \cdot \boldsymbol{s} + R_i^{\star\top} \hat{\boldsymbol{x}} \in \mathcal{R}_q^k \end{aligned}$$

Let us now analyse the joint distribution of the public parameters and the challenge ciphertext in Game 0 and Game 1. We will show that the distributions of $(\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}_i\}, \{\hat{\boldsymbol{c}}_i\})$ in Game 0 and in Game 1 are statistically indistinguishable.

First notice that by Lemma A.8 we have that the following two distributions are statistically indistinguishable for every fixed matrix $\hat{\boldsymbol{b}}^\star$, every $\boldsymbol{id}_i^\star$ and every vector $\hat{\boldsymbol{x}} \in \mathcal{R}_q^k$:

$$\left(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}_i, R_i^{\star\top} \hat{\boldsymbol{x}}\right) \approx_s \left(\hat{\boldsymbol{a}}, \hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star, R_i^{\star\top} \hat{\boldsymbol{x}}\right) \ .$$

Since each $\hat{\boldsymbol{r}}_{\gamma,i}^\star$ is chosen independently for every i and γ, then the joint distribution of them are statistically close:

$$\left(\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}_i\}, \{R_i^{\star\top} \hat{\boldsymbol{x}}\}\right) \approx_s \left(\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star\}, \{R_i^{\star\top} \hat{\boldsymbol{x}}\}\right) \ .$$

Since $(\hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s}$ is statistically close to $\hat{\boldsymbol{a}}_i \cdot \boldsymbol{s}$, it is possible to add each term to each side of the equation:

$$\begin{aligned}\left(\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}_i\}, \{\hat{\boldsymbol{a}}_i \cdot \boldsymbol{s} + R_i^{\star\top}\hat{\boldsymbol{x}}\}\right) \approx_s \\ \left(\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star\}, \{(\hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R_i^{\star\top}\hat{\boldsymbol{x}}\}\right)\end{aligned} .$$

Then, we add $(\boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s}$ to each side of the equation:

$$\begin{aligned}\left(\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}_i\}, \{(\hat{\boldsymbol{a}}_i + \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star) \cdot \boldsymbol{s} + R_i^{\star\top}\hat{\boldsymbol{x}}\}\right) \approx_s \\ \left(\hat{\boldsymbol{a}}, \{\hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star\}, \{(\hat{\boldsymbol{a}}R_i^\star) \cdot \boldsymbol{s} + R_i^{\star\top}\hat{\boldsymbol{x}}\}\right)\end{aligned} .$$

To conclude, observe that the distribution on the left hand side is that of the public parameters and the challenge ciphertext in Game 0, while that on the right hand side is the distribution in Game 1.

□

Indistinguishability of Game 1 and Game 2 (or Game 3 and Game 4)

Lemma 8.5. *The view of the adversary $\mathcal{A}$ in Game 1 (resp. Game 3) is computationally indistinguishable from the view of $\mathcal{A}$ in Game 2 (resp. Game 4) under decision-*Ring-LWE*.*

Proof.

Suppose $\mathcal{A}$ can distinguish between Game 1 and Game 2 with non-negligible advantage. Then, it is possible to use $\mathcal{A}$ to build an algorithm $\mathcal{B}$ to solve *decision*-Ring-LWE.

Init $\mathcal{B}$ is given k Ring-LWE challenge pairs $(\boldsymbol{a}_j, \boldsymbol{y}_j) \in \mathcal{R}_q \times \mathcal{R}_q$, where either $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$ for a random $\boldsymbol{s} \in \mathcal{R}_q$ and a noise term $\boldsymbol{x}_j \leftarrow \Psi_\alpha^n$, or $\boldsymbol{y}_j$ is uniformly random in $\mathcal{R}_q$. And one LWE challenge pair $(\boldsymbol{a}_0, y_0) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$, where either $y_0 = \langle \boldsymbol{a}_0, \boldsymbol{s}\rangle + x_j$ for a noise term $x_j \leftarrow \Psi_\alpha$, or y_0 is uniformly random in $\mathbb{Z}_q$.

SetUp The public parameters are constructed using the vectors of the pairs $(\boldsymbol{a}_j, \boldsymbol{y}_j)$. The i-th polynomial of ring $\hat{\boldsymbol{a}}$ will be the vector $\boldsymbol{a}_i$, for $1 \le i \le k$ and vector $\boldsymbol{u}$ will be $\boldsymbol{a}_0$. The matrix $\hat{\boldsymbol{a}}_0$ is still calculated as in Sim.ideal-IBE-SetUp, i.e., $\hat{\boldsymbol{a}}_i \leftarrow \hat{\boldsymbol{a}}R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star$.

Secret keys All private-key extraction queries are answered using the algorithm Sim.ideal-HIBE-KeyDerive.

Challenge Ciphertext The ciphertext $CT = (\hat{\boldsymbol{c}}_0^\star, \hat{\boldsymbol{c}}_i^\star, c'^\star)$ is constructed based on the terms in the LWE challenge pairs $(\boldsymbol{a}_j, \boldsymbol{y}_j)$, with $\hat{\boldsymbol{c}}_0^\star = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_m)$, $\hat{\boldsymbol{c}}_i^\star = R_i^{\star\top}\hat{\boldsymbol{c}}_0^\star$ and $c'^\star = y_0 + M\lfloor q/2\rceil$. If we have $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$ on the Ring-LWE challenge and $y_0 = \langle \boldsymbol{a}_0, \boldsymbol{s}\rangle + x$ on the LWE challenge, then the ciphertext is distributed exactly as in Game 1, and if $\boldsymbol{y}_j$ is uniformly random in $\mathcal{R}_q$ and y_0 is uniformly random in $\mathbb{Z}_q$, then the ciphertext is distributed exactly as in Game 2. If $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$, then

$(y_1, \ldots, y_m) = (\boldsymbol{a}_1 \cdot \boldsymbol{s} + \boldsymbol{x}_1, \ldots, \boldsymbol{a}_k \cdot \boldsymbol{s} + \boldsymbol{x}_m) = \hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}}$.

Therefore, for Game 1 we have

$$\begin{aligned}\hat{\boldsymbol{c}}_0^\star &= \hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}} \\ &= (y_1, \ldots, y_m) \ ,\end{aligned}$$

and

$$\begin{aligned}\hat{\boldsymbol{c}}_i^\star &= (\hat{\boldsymbol{a}}_i + \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R_i^{\star\top} \hat{\boldsymbol{x}} \\ &= (\hat{\boldsymbol{a}} R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}} + \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}) \cdot \boldsymbol{s} + R_i^{\star\top} \hat{\boldsymbol{x}} \\ &= (\hat{\boldsymbol{a}} R_i^\star) \cdot \boldsymbol{s} + R_i^{\star\top} \hat{\boldsymbol{x}} \\ &= R_i^{\star\top} (\hat{\boldsymbol{a}} \cdot \boldsymbol{s} + \hat{\boldsymbol{x}}) \\ &= R_i^{\star\top} \hat{\boldsymbol{c}}_0^\star \ .\end{aligned}$$

If y_j is uniformly random in $\mathbb{Z}_q$ then the ciphertext is uniformly random, as the ciphertext generated by Game 2.

Guess $\mathcal{A}$ must guess whether it is interacting with Game 1 or Game 2 . The answer to this guess is also the answer to the Ring-LWE and LWE challenges, because, as we showed, if $\boldsymbol{y}_j$ is uniformly random in $\mathcal{R}_q$ and y_0 is uniformly random in $\mathbb{Z}_q$, then $\mathcal{A}$'s view is the same as in Game 2 and if $y_0 = \langle \boldsymbol{a}_0, \boldsymbol{s} \rangle + x_0$ and $\boldsymbol{y}_j = \boldsymbol{a}_j \cdot \boldsymbol{s} + \boldsymbol{x}_j$, then $\mathcal{A}$'s view is the same as in Game 1.

□

Indistinguishability of Game 2 and Game 3

Lemma 8.6. *The view of the adversary $\mathcal{A}$ in Game 2 is statistically indistinguishable from the view of $\mathcal{A}$ in Game 3.*

Proof.

SetUp The public parameters are generated in the same way in both games. $\hat{\boldsymbol{a}}$ and $\boldsymbol{u}$ are random and $\hat{\boldsymbol{a}}_i \leftarrow \hat{\boldsymbol{a}} R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star$, with $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 0}$ for Game 2 and $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 1}$ for Game 3.

Secret keys All private-key extraction queries are answered using Sim.ideal-IBE-KeyGen. The only difference is, again, that for Game 2, $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 0}$ and, for Game 3, $\boldsymbol{id}_i^\star = \boldsymbol{id}_i^{\star 1}$.

Challenge Ciphertext The challenge ciphertext in both games is randomly chosen.

All public parameters are randomly generated in both games, except for matrix $\hat{\boldsymbol{a}}_i$. Therefore, the indistinguishability of Game 2 and Game 3 only depends on the indistinguishability of $\hat{\boldsymbol{a}}_i$. From Lemma A.8 we can prove that $\hat{\boldsymbol{a}}_i$ for each game is statistically close to a uniformly random matrix, because

$\hat{\boldsymbol{a}}_i \leftarrow \hat{\boldsymbol{a}} R_i^\star - \boldsymbol{id}_i^\star \cdot \hat{\boldsymbol{b}}^\star$.

□

8.2.4 Parameters

In this section we analyse the several parameters of the scheme based on all the requirements used during construction, correction and security.

We know that $\|\boldsymbol{e}\| \leq \sigma_t\sqrt{(t+1)kn}$ from Lemma A.1 and $\|R\boldsymbol{e}\| \leq 12\sqrt{tkn+kn}\|\boldsymbol{e}\|$ from Lemma A.2, for $R = [R_1|\cdots|R_t]$, with $\max(t) = d$; therefore, for $\boldsymbol{e} = \begin{bmatrix}\boldsymbol{e}_1\\ \boldsymbol{e}_2\end{bmatrix}$:

$$\begin{aligned}
&\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq \sigma_t\sqrt{(d+1)kn} + d12\sqrt{(d+1)kn}\sigma_t\sqrt{(d+1)kn} \qquad \text{so}\\
&\|\boldsymbol{e}_1 + R\boldsymbol{e}_2\| \leq O(\sigma_t d^2 kn)\ .
\end{aligned}$$

From Lemma A.3 we have that $\langle \boldsymbol{y}, \boldsymbol{x}\rangle \leq \|\boldsymbol{y}\|q\alpha w(\sqrt{\log n}) + \|\boldsymbol{y}\|\sqrt{n}/2$; therefore:

$$\begin{aligned}
\langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq O(\sigma_t d^2 kn)q\alpha_t\omega(\sqrt{\log kn}) + O(\sigma_t d^2 kn)\sqrt{kn}/2\\
\langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle &\leq \widetilde{O}(\sigma_t d^2 knq\alpha_t) + O(\sigma_t d^2 (kn)^{3/2})\ .
\end{aligned}$$

To ensure that the error term is less than $q/4$, we need the following:

$$\begin{aligned}
&x - \boldsymbol{e}^\top\begin{bmatrix}\boldsymbol{x}\\ R^\top\boldsymbol{x}\end{bmatrix} < q/4\\
&x - \boldsymbol{e}_1^\top\boldsymbol{x} - \boldsymbol{e}_2^\top R^\top\boldsymbol{x} < q/4\\
&x - (\boldsymbol{e}_1 + R\boldsymbol{e}_2)^\top\boldsymbol{x} < q/4\\
&x - \langle \boldsymbol{e}_1 + R\boldsymbol{e}_2, \boldsymbol{x}\rangle < q/4\\
&\widetilde{O}(\sigma_t d^2 knq\alpha) + O(\sigma_t d^2 (kn)^{3/2}) < q/4
\end{aligned}$$

To ensure that σ_t is sufficiently large for SampleBasisLeft and SampleBasisRight (Theorems 2.10 and 2.11), we have

$$\sigma_t > \|S\|\sqrt{kn}\omega(\sqrt{\log kn}).$$

To ensure that IdealTrapGen (Theorem 2.2) can operate, we have

$$\begin{aligned}
&k \geq \lceil \log q + 1\rceil \qquad \text{and}\\
&\|S\| = O(n\log q\sqrt{\omega(\log n)})\ .
\end{aligned}$$

To ensure that the reduction applies (Theorem 2.14), we have

$$q > \omega(\sqrt{\log n})/\alpha\ .$$

Therefore, we need to set the parameters as

$$
\begin{aligned}
&k = O(\log q),\\
&q = (kn)^{2.5}\omega(\sqrt{\log n}),\\
&\alpha = (k^2 n^2 \omega(\sqrt{\log n}))^{-1},\\
&\sigma = kn\omega(\sqrt{\log n}).
\end{aligned}
$$

8.2.5 Complexity and Key Sizes

In this section we present an analysis of the size of the main variables and the complexity of the algorithms from the scheme described on Section 8.2.1. Note that for security parameter n, hierarchy's maximum depth d and modulus q, we have $k = O(\log q)$ (see Section 8.2.4).

The master key MK is just an $kn \times kn$ matrix, therefore its size is $(kn)^2$. The public key PK is comprised of a vector of length n and $d + 2$ rings of lenght kn; therefore its size is $O(dkn)$. The secret key is now a matrix, not a vector, of size $n \times (t + 1)kn$, with $\max(t) = d$; therefore its size is at most $n(d + 1)kn$, which is $O(dkn^2)$. Finally, the ciphertext is comprised of an integer and $t + 1$ vectors of length kn, with $\max(t) = d$; therefore its size is at most $1 + kn + dkn$, which is $O(dkn)$.

The complexity of ideal-HIBE-SetUp is based on the complexity of the IdealTrapGen algorithm. By Theorem 2.2 we have that the IdealTrapGen algorithm is polynomial, and, therefore, ideal-HIBE-SetUp is also polynomial. The complexity of ideal-HIBE-KeyDerive is based on the complexity of the SampleBasisLeft algorithm plus t ring-vector multiplications ($O(kn)$, each) and t ring additions ($O(kn)$, each). As before, we have that the SampleBasisLeft algorithm is polynomial, by Theorem 2.10.

The ideal-HIBE-Enc algorithm does $1 + 2t$ ring-vector multiplications ($O(kn)$, each), $1 + 2t$ ring addictions ($O(kn)$, each), tk ring-ring multiplications ($O(kn)$, each), one inner product ($O(n)$) and two simple additions $O(1)$. Therefore, the complexity of ideal-HIBE-Enc is based on the ring-ring multiplications with $\max(t) = d$.

The ideal-HIBE-Dec algorithm does t ring-vector multiplications ($O(kn)$, each), t ring addition operations ($O(kn)$, each), one inner product between two vectors of length tkn, a few simple additions and multiplications and calls the SamplePre algorithm. We have, by Theorem 2.6, that SamplePre is polynomial, therefore the complexity of the ideal-HIBE-Dec algorithm is based on SamplePre and the ring operations.

Table 8.3 summarises the size of the main variables and Table 8.4 summarises the complexity of the four algorithms of the scheme described in Section 8.2.1.

Variable	**Size**
Public Key PK	$O(dn \log q)$
Master Key MK	$O(n^2 \log^2 q)$
Secret Key SK	$O(dn^2 \log q)$
Ciphertext CT	$O(dn \log q)$

Table 8.3: Key Sizes of the ideal HIBE Scheme

Algorithm	**Complexity**
SetUp	$O(\text{poly}(n))$
KeyDerive	$O(\text{poly}(n) + dn \log q)$
Enc	$O(dn \log^2 q)$
Dec	$O(\text{poly}(n) + dn \log q)$

Table 8.4: Complexity of the ideal HIBE Scheme

Chapter 9

Results and Evaluation

This chapter summarizes and compares all results and schemes in this work, showing how their expansion to hierarchical versions and the use of ideal lattices affect the security proofs, the algorithms' efficiency and variable sizes and detailing the pitfalls and steps involving each expansion.

Table 9.1 gives a summary of keys and variable sizes for all schemes. Table 9.2 gives a summary of the complexity of the four algorithms for all schemes. Entries in the tables are given as functions of the security parameter n, maximum hierarchical depth d, vector length l and modulo q.

	Public Key PK	**Master Key** MK	**Secret Key** SK	**Ciphertext** CT
IBE	$O(n^2 \log q)$	$O(n^2 \log^2 q)$	$O(n \log q)$	$O(n \log q)$
HIBE	$O(dn^2 \log q)$	$O(n^2 \log^2 q)$	$O(dn^2 \log q)$	$O(dn \log q)$
IPE	$O(ln^2 \log^2 q)$	$O(n^2 \log^2 q)$	$O(n \log q)$	$O(ln \log^2 q)$
HIPE	$O(dln^2 \log^2 q)$	$O(n^2 \log^2 q)$	$O(dn^2 \log q)$	$O(dln \log^2 q)$
HVE	$O(ln^2 \log q)$	$O(ln^2 \log^2 q)$	$O(ln \log q)$	$O(ln \log q)$
HHVE	$O(dln^2 \log q)$	$O(ln^2 \log^2 q)$	$O(dln^2 \log q)$	$O(dln \log q)$
FBE	$O(ln^2 \log q)$	$O(ln^2 \log^2 q)$	$O(ln \log q)$	$O(ln \log q)$
HFBE	$O(ldn^2 \log q)$	$O(ln^2 \log^2 q)$	$O(ldn^2 \log q)$	$O(ldn \log q)$
ideal IBE	$O(n \log q)$	$O(n^2 \log^2 q)$	$O(n \log q)$	$O(n \log q)$
ideal HIBE	$O(dn \log q)$	$O(n^2 \log^2 q)$	$O(dn^2 \log q)$	$O(dn \log q)$

Table 9.1: Key sizes for all schemes

	SetUp	**KeyGen**	**Enc**	**Dec**
IBE	$O(\text{poly}(n))$	$O(\text{poly}(n) + n^3 \log q)$	$O(n^3 \log q)$	$O(n \log q)$
HIBE	$O(\text{poly}(n))$	$O(\text{poly}(n) + dn^3 \log q)$	$O(dn^3 \log q)$	$O(\text{poly}(n) + dn^3 \log q)$
IPE	$O(\text{poly}(n))$	$O(\text{poly}(n) + ln^2 \log^2 q)$	$O(ln^2 \log^3 q)$	$O(ln \log^2 q)$
HIPE	$O(\text{poly}(n))$	$O(\text{poly}(n) + dln^2 \log^2 q)$	$O(dln^2 \log^3 q)$	$O(\text{poly}(n) + dln^2 \log^2 q)$
HVE	$O(l \cdot \text{poly}(n))$	$O(l \cdot \text{poly}(n))$	$O(ln^2 \log q)$	$O(ln \log q)$
HHVE	$O(l \cdot \text{poly}(n))$	$O(ldn^2 \log q + l \cdot \text{poly}(n))$	$O(dln^2 \log q)$	$O(l \cdot \text{poly}(n) + dln^2 \log q)$
FBE	$O(l \cdot \text{poly}(n))$	$O(l \cdot \text{poly}(n) + ln^3 \log q)$	$O(ln^3 \log q)$	$O(ln \log q)$
HFBE	$O(l \cdot \text{poly}(n))$	$O(l \cdot \text{poly}(n) + ldn^2 \log^2 q)$	$O(ldn^3 \log^2 q)$	$O(l \cdot \text{poly}(n) + ldn^3 \log^2 q)$
ideal IBE	$O(\text{poly}(n))$	$O(\text{poly}(n) + n \log q)$	$O(n \log^2 q)$	$O(n \log q)$
ideal HIBE	$O(\text{poly}(n))$	$O(\text{poly}(n) + dn \log q)$	$O(dn \log^2 q)$	$O(\text{poly}(n) + dn \log q)$

Table 9.2: Algorithm complexities for all schemes

9.1 Hierarchical Expansion

In some cryptosystems, such as the ones we described in this work, a trusted third party, or Private Key Generator (PKG), is needed to generate all the keys (master, public and secret keys). Since the PKG has all the keys, it can encrypt and decrypt any message and, therefore, it can be more difficult to prove the integrity and origin of a message. Besides, if the PKG is compromised the whole system is compromised, so the PKG can be a good target for adversaries. Finally, a secure channel between each user and the PKG is needed for the transmission of secret keys. Therefore, it is convenient to have a hierarchy of certificate authorities, reducing the workload on the PKG as it does not need to generate all secret keys anymore.

9.1.1 Defining the Lattice

To create a hierarchical expansion of a lattice-based predicate encryption scheme, we need to create a new lattice for each level, always concatenating the previous lattice basis with the new matrix for the current level. Now the secret key will a be the short basis of this newly created lattice, and it is possible to get this short basis using the SampleBasisLeft algorithm that calculates a short basis for this new lattice using the short basis for the previous level lattice. For level 0, we have that the secret key is the master key.

Therefore, the master key is a short basis S for the lattice $\Lambda(A)$, and for level 1 we can use the SampleBasisLeft algorithm with S and A to find a short basis S_1 for the lattice $\Lambda(A|A_1)$, where A_1 is a new matrix that represents this level. Now, for each level we can continue to compute the short basis S_t using the matrix S_{t-1} and concatenating A with all A_i. That is how the secret keys are calculated in each KeyDerive algorithm.

Now, using the short basis S_t for level t, it is possible to calculate a vector $\boldsymbol{e}$ such that

$\boldsymbol{u} = [A|A_1|\cdots|A_t]\boldsymbol{e}$ by calling the SamplePre algorithm. This vector is the same vector used in the non-hierarchical version as the secret key, but now it is calculated in the Dec algorithm using the short lattice basis that is used now as the secret key. This change is the core of the hierarchical algorithm, since it allows the KeyDerive algorithm to calculate the secret key for level t using only the secret for level $t-1$ and not the master key.

9.1.2 Sampling the Basis

Changing the definition of the lattice basis also affects the security proof, that now will use SampleBasisRight to calculate the new short basis in the simulation algorithms of the proof. Using SampleBasisRight in the simulation algorithms of the hierarchical versions will have the same outcome as using SampleRight in the simulation algorithms of the regular schemes.

Another effect is seen in the matrices $R \in \{-1, 1\}$: we must assure now that for each level the R matrix is chosen independently so that we have that each matrix $A_i \leftarrow AR + CB$, for random A and B and a constant matrix C in the simulation, is indistinguishable from a random matrix.

9.1.3 Increasing the Key Size

The main complexity effect in the hierarchical versions is the increase in the secret key size. As we can see in Table 9.1, all keys except the master key, are at least d times larger for all the hierarchical schemes, where d is the maximum hierarchical level. The secret keys are even bigger since they are now a matrix, instead of a vector. This also reflects on the efficiency of Enc and Dec that have to perform more operations in a larger number of matrices, increasing their complexity also by a factor d as we can see in Table 9.2.

Tables 9.3 and 9.4 show the real size of each variable for two security levels, 128 bits and 256 bits respectively, and for $d = 3$. The key sizes are calculated based on the results by Lindner et al. [45] and describe the practical effect of using hierarchical versions of the schemes.

We notice a large increase on the secret key size from 54 KB to 27 MB (for 128 bits) and from 22 KB to 5 MB (for 256 bits), while the master key size remains the same. This happens because the master key is the secret key for level 0, i.e., the short basis generated by the SampleBasisRight algorithm without any concatenation and it does depend on the hierarchical level. We also have an increase in the public key and the ciphertext size.

Since the main problem with lattice-based schemes is the large size of the keys, the practical feasibility of hierarchical schemes proves to be a real challenge.

	IBE	**HIBE**
Public Key	20736	37632
Master Key	497664	497664
Secret Key	54	27648
Ciphertext	54	108

Table 9.3: Comparing hierarchical and non-hierarchical key sizes in KB for $n = 128$ bits, $q = 2053$ bits and $d = 3$

	IBE	**HIBE**
Public Key	4356	8668
Master Key	95832	95832
Secret Key	22	5808
Ciphertext	22	44

Table 9.4: Comparing hierarchical and non-hierarchical key sizes in KB for $n = 256$ bits, $q = 4093$ bits and $d = 3$

9.1.4 Splitting the Vector

We would also like to notice a specific effect that the hierarchical version has in the last two schemes, the Hidden Vector Encryption Scheme (HVE) and the Fuzzy Identity-Based Encryption Scheme (FBE). In the non-hierarchical version the vector split is vector $\boldsymbol{u}$, while in the hierarchical version vector $\boldsymbol{s}$ is split. This happens because if we split vector $\boldsymbol{u}$ in the hierarchical scheme we allow the user to decrypt the message only if $v_{t,j} = w_{t,j}$ (for the HHVE scheme) or $\boldsymbol{id}_j = \boldsymbol{id}'_j$ (for the HFBE scheme) for a single j. That is possible by calling algorithm $\mathsf{SamplePre}$ only once for $\boldsymbol{u}$ and this given j, instead of calling $\mathsf{SamplePre}$ for each share of $\boldsymbol{u}$ as would be expected. Then, making $z = c' - \boldsymbol{e}_j^\top \boldsymbol{c}_j$ will give the right message.

This happens because in the general version we split vector $\boldsymbol{u}$ during the KeyGen algorithm and this is not possible for the hierarchical version. Since we cannot guarantee that every time a share is created it is the same as before, the bases for each level will not always be the same, and the $\mathsf{KeyDerive}$ algorithm will not work properly.

Therefore, our main ideia consists of splitting vector $\boldsymbol{s}$ during encryption, guaranteeing that at least k Lagrangian coefficients must be correctly computed and that each group of $v_{t,j} = w_{t,j}$ (for the HHVE scheme) and $\boldsymbol{id}_j = \boldsymbol{id}'_j$ (for the HFBE scheme) must be equal for the right lattice basis to be calculated, and, therefore, for the correct vector $\boldsymbol{s}$ to be reconstructed.

9.2 Use of Ideal Lattices

Ideal lattices are a generalization of cyclic lattices, in which the lattice corresponds to ideals in a ring $\mathbb{Z}[x]/\langle f(x)\rangle$, for some irreducible polynomial function f. They can be used to decrease the parameters needed to describe a lattice, as shown in Section 2.1.1 and its basis pattern can be used to decrease the matrix multiplication complexity, as shown in Appendix C.

9.2.1 Using Rings Instead of Matrices

The change in the representation from matrices to rings (represented as vectors) has a direct effect on the efficiency of the operations done in the four algorithms of the scheme, as in the size of the main parameters used, but it also affects some steps of the security proof.

For the ideal lattice scheme, we can not use Lemmas A.4 and A.5 in the proof of the indistinguishability of Games 0 and 1, because both lemmas refer to matrices, not rings. Therefore, one important difference in this security proof is to use Lemma A.8 that refers to the indistinguishability of rings, and their multiplication, from random rings. This happens because in the simulation algorithm we replace matrix $A_0 \leftarrow AR + CB$ by ring $\hat{\boldsymbol{a}}_0 \leftarrow \hat{\boldsymbol{a}}R - \boldsymbol{id} \cdot \hat{\boldsymbol{b}}$.

9.2.2 Generating the Trapdoor

To make possible the use of ideal lattices in some cryptosystems, as the ones described in this work, we first need an algorithm to generate a short and a hard basis for an ideal lattice. Algorithm IdealTrapGen (Theorem 2.2) given by Stehlé et al. [73] generates these bases for an ideal lattice. The hard basis is described by a ring $\hat{\boldsymbol{g}}$ and the short basis, that will be used as the master key, is still a matrix S. Therefore, only the public key will be decreased, the master key size will remain the same.

9.2.3 Learning With Errors for Ideals Problem

The security proof of all the schemes are based on the reduction from the *decision*-LWE Problem, but this problem does not apply for ideal lattices. Two new problems were defined as suitable for ideal lattices: the Ring-LWE Problem and the Ideal-LWE Problem (see Section 2.3).

Lyubashevsky et al. [47] defined the Ring-LWE problem as finding vector $\boldsymbol{r}$ given vectors $\boldsymbol{a}$ and $\boldsymbol{b} = \boldsymbol{a} \cdot \boldsymbol{r} + \boldsymbol{e}$. While in the original LWE problem we had that b was a number, in the Ring-LWE we have that $\boldsymbol{b}$ is a vector. Now we can reduce an ideal lattice system, since the ring that defines the lattice basis is the concatenation of vectors and we have that $\hat{\boldsymbol{g}} \cdot \boldsymbol{s} = [\boldsymbol{r}_1 \cdot \boldsymbol{s}, \cdots, \boldsymbol{r}_k \cdot \boldsymbol{s}]$. The encryption will now be based on this multiplication. Lyubashevsky et al. proved that the decision Ring-LWE problem is as hard as γ-SIVP and γ-SVP in a quantum environment.

Stehlé et al. [73] defined another problem closer to the original LWE Problem. The Ideal-LWE Problem is defined as finding vector $\boldsymbol{r}$ given rings $\hat{\boldsymbol{a}}$ and $\hat{\boldsymbol{b}} = \mathsf{Rot}_f(\hat{\boldsymbol{a}})^\top \boldsymbol{r} + \hat{\boldsymbol{e}}$. The definition of $\hat{\boldsymbol{b}}$ is exactly the same as it will be in the original LWE Problem, because $\mathsf{Rot}_f(\hat{\boldsymbol{a}})$ gives the matrix that is the basis of the lattice. Although closer to the original one, this problem has one big issue: only the general, not the decision version, of this problem is known to be as hard as Ideal-SIS. There is no proof for the decision problem, making it impossible to use the reduction for this problem in all the schemes studied in this work. Note that for both problems the ideal lattice must be for the ring with $f(x) = x^n + 1$.

9.2.4 Decreasing the Key Size

One main effect of using ideal lattices is the decrease of the public key size. As we can see in Table 9.1, comparing schemes *ideal IBE* and *IBE*, for the general lattice scheme we have three $n \times m$ matrices as public key and for the ideal lattice scheme we have only three rings of length kn. Therefore the key size is reduced by a factor of n, for $m = kn$. Unfortunately, all the other parameters remain the same size asymptotically.

Tables 9.5 and 9.6 show the real size for the variables for two security levels, 128 bits and 256 bits respectively, based on the results by Lindner et al. [45]. We can see that, although most of parameters remain the same size asymptotically, for the real size we have a great improvement for all, specially the master key size. We also can highlight the importance of the use of ideal lattices to reduce the public key drastically from approximately 20 MB to only 13 KB (for 128 bits) and from approximately 4 MB to only 5 KB (for 256 bits). This happens because the parameter choice is also different for ideal lattices and for schemes with large key sizes, as lattice-based ones, the constants' choice can make a difference on the practical viability.

	IBE	**ideal-IBE**
Public Key	20736	13
Master Key	497664	13824
Secret Key	54	9
Ciphertext	54	9

Table 9.5: Comparing ideal and non-ideal key sizes in KB for $n = 128$ bits and $q = 2053$ bits

	IBE	**ideal-IBE**
Public Key	4356	5
Master Key	95832	2662
Secret Key	22	3
Ciphertext	22	3

Table 9.6: Comparing ideal and non-ideal key sizes in KB for $n = 256$ bits and $q = 4093$ bits

9.2.5 Improving the Complexity

Since the basis of an ideal lattice consists of the concatenation of k Toeplitz $n \times n$ matrices, the multiplication of the basis by a vector can be done in a more efficient way [59].

As shown in Table 9.2, comparing schemes *ideal IBE* and *IBE*, we have the largest improvement in the Enc algorithm, in which most of the multiplications are done. Since we are using ideal lattices and anti-circular matrices, and because $f(x) = x^n + 1$, we can enhance the multiplication operations by a factor of n^2. Note that this fact also makes the KeyGen algorithm more efficient for the ideal lattice version.

9.3 Hierarchical with Ideals

In Chapter 8 we combine the two features, describing a hierarchical IBE scheme based on ideal lattices. For a hierarchical scheme, ideal lattices only contribute with the decrease of the public key by a factor of n, as in the general lattice versions, as we can see in Table 9.1, comparing schemes *ideal HIBE* and *HIBE*. Unfortunately, the secret key continues to be larger than the non-hierarchical version. This happens because the secret keys are now a short basis of the lattice generated by the SampleBasisLeft algorithm and this short basis, although it is from an ideal lattice, does not have any distinguishable pattern. Therefore, the secret keys are still comprised of a large matrix.

Since multiplication with ideal lattices is more efficient, we have a decrease in the complexity of the main algorithms, especially encryption and decryption, as we can see in Table 9.2, comparing schemes *ideal HIBE* and *HIBE*. This is possible because the pattern found in the ideal lattice basis allows all operations to be performed with vectors and polynomials. Notice that we have an anti-circular matrix, since we use $f(x) = x^n + 1$, but we can have more efficient multiplications for any Toeplitz matrix (see Appendix C for more details).

Chapter 10

Conclusion

In this work we study several lattice-based predicate encryption schemes, a sub-class of functional encryption. Our main contribution is to show hierarchical versions of the main lattice-based functional encryption schemes used nowadays. Hierarchical schemes reduce the workload on a Thrusted Third Part as it does not need to generate all public and master keys. We also show detailed security proofs and analysis of some schemes using a special class of lattices called ideal lattices. Ideal lattices are usually applied to decrease the size of the parameters needed to describe a lattice and its basis pattern can be used to improve the scheme's complexity.

10.1 Future Work

Ideal lattices are very helpful for decreasing the size of the main variables and thus efficiency of the many matrix multiplications normally used in lattice-based schemes. Since their security is proved to be the same as regular lattices, creating a scheme exclusively for use with ideal lattices may provide even more improvements in the complexity of the algorithms and also decrease the ciphertext size.

A more detailed analysis with benchmarks for each scheme presented in this work can lead to more precise results on the practical effects of using ideal lattices and hierarchical versions. Implementing and addressing these practical issues is an important and underrated work. Not only the ideal and hierarchical versions should be studied along with their improvements, but the regular schemes should be compared with the functional schemes used nowadays and their feasibility should be analysed in detail.

Furthermore, there are no lattice-based hierarchical ABE schemes known, as well as very few studies on lattice-based WIBE schemes. Both schemes have several applications and, therefore, are promising research subjects.

Regarding their security, most of lattice-based functional schemes known have a weak security proofs; they are attribute hiding, in which the adversary must commit the challenge attributes at the beginning of the game proof. Therefore, it would be interest to build fully secure schemes that have a stronger security proof.

Finally, the main problem with the hierarchical schemes is the size of the secret key, even using ideal lattices; so, searching for other ways to improve these schemes is an important and interesting work.

Bibliography

[1] Michel Abdalla, Angelo De Caro, and Karina Mochetti. Lattice-based hierarchical inner product encryption. In *LATINCRYPT*, pages 121–138, 2012.

[2] Michel Abdalla, Dario Catalano, Alexander W. Dent, John Malone-Lee, Gregory Neven, and Nigel P. Smart. Identity-based encryption gone wild. In *ICALP (2)*, pages 300–311, 2006.

[3] Michel Abdalla, Dario Fiore, and Vadim Lyubashevsky. From selective to full security: Semi-generic transformations in the standard model. In *Public Key Cryptography*, pages 316–333, 2012.

[4] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, pages 553–572, 2010.

[5] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Fuzzy identity based encryption from lattices. In *IACR Cryptology ePrint Archive*, 2011.

[6] Shweta Agrawal, Xavier Boyen, Vinod Vaikuntanathan, Panagiotis Voulgaris, and Hoeteck Wee. Functional encryption for threshold functions (or fuzzy ibe) from lattices. In *Public Key Cryptography*, pages 280–297, 2012.

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

[8] Miklós Ajtai. Generating hard instances of lattice problems. *Electronic Colloquium on Computational Complexity (ECCC)*, 3(7), 1996.

[9] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, pages 75–86, 2009.

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

[11] Joonsang Baek, Willy Susilo, and Jianying Zhou. New constructions of fuzzy identity-based encryption. In *ASIACCS*, pages 368–370, 2007.

[12] Daniel J. Bernstein. Post-quantum cryptography. In *Encyclopedia of Cryptography and Security (2nd Ed.)*, pages 949–950. Springer, 2011.

[13] Johannes Blömer and Jean-Pierre Seifert. On the complexity of computing short linearly independent vectors and short bases in a lattice. In *STOC*, pages 711–720, 1999.

[14] Dan Boneh and Xavier Boyen. Efficient selective-ID secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[15] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[16] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

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

[18] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

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

[20] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[21] Angelo De Caro, Vincenzo Iovino, and Giuseppe Persiano. Hidden vector encryption fully secure against unrestricted queries. *IACR Cryptology ePrint Archive*, 2011:546, 2011.

[22] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, pages 523–552, 2010.

[23] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. *J. Cryptology*, 25(4):601–639, 2012.

[24] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[25] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[26] Irit Dinur, Guy Kindler, Ran Raz, and Shmuel Safra. Approximating CVP to within almost-polynomial factors is NP-hard. *Combinatorica*, 23(2):205–243, 2003.

[27] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. *IACR Cryptology ePrint Archive*, 2013:128, 2013.

[28] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

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

[30] Craig Gentry and Alice Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[31] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *CRYPTO*, pages 112–131, 1997.

[32] Oded Goldreich, Daniele Micciancio, Shmuel Safra, and Jean-Pierre Seifert. Approximating shortest lattice vectors is not harder than approximating closest lattice vectors. *Inf. Process. Lett.*, 71(2):55–61, 1999.

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

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

[35] Goichiro Hanaoka, Tsuyoshi Nishioka, Yuliang Zheng, and Hideki Imai. An efficient hierarchical identity-based key-sharing method resistant against collusion-attacks. In *ASIACRYPT 2009*, volume 5479 of *LNCS*, pages 348–362, Cologne, Germany, 2009. springer.

[36] J Hoffstein, N Howgrave-Graham, J Pipher, JH Silverman, and W Whyte. Hybrid lattice reduction and meet in the middle resistant parameter selection for NTRUencrypt. *NTRU Cryptosystems, Inc.*, 2007.

[37] Jeffrey Hoffstein, Jill Pipher, and Joseph H. Silverman. NTRU: A ring-based public key cryptosystem. In *ANTS*, pages 267–288, 1998.

[38] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, pages 466–481, 2002.

[39] Vincenzo Iovino and Giuseppe Persiano. Hidden-vector encryption with groups of prime order. In *Pairing*, pages 75–88, 2008.

[40] Jill Pipher Joseph H. Silverman, Jeffrey Hoffstein and Daniel Lieman. NTRU Cryptosystems, inc. In *https://www.securityinnovation.com/*, 2009.

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

[42] Subhash Khot. Hardness of approximating the shortest vector problem in lattices. *J. ACM*, 52(5):789–808, 2005.

[43] A.K. Lenstra, H.W.Jr. Lenstra, and Lászlo Lovász. Factoring polynomials with rational coefficients. *Math. Ann.*, 261:515–534, 1982.

[44] Jin Li, Qian Wang, Cong Wang, and Kui Ren. Enhancing attribute-based encryption with attribute hierarchy. *MONET*, 16(5):553–561, 2011.

[45] Richard Lindner and Chris Peikert. Better key sizes (and attacks) for LWE-based encryption. In *Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011*, CT-RSA'11, pages 319–339, Berlin, Heidelberg, 2011. Springer-Verlag.

[46] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact knapsacks are collision resistant. In *ICALP (2)*, pages 144–155, 2006.

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

[48] R McEliece. A public-key cryptosystem based on algebraic coding theory. *DSN progress report*, Jan 1978.

[49] Daniele Micciancio. Improving lattice based cryptosystems using the hermite normal form. In *CaLC*, pages 126–145, 2001.

[50] Daniele Micciancio. The shortest vector problem is NP-hard to approximate to within some constant. *SIAM Journal on Computing*, 30(6):2008–2035, 2001.

[51] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions from worst-case complexity assumptions. In *Proceedings of the 43rd Annual Symposium on Foundations of Computer Science - FOCS 2002.*, pages 356–365, Vancouver, Canada, 2002.

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

[53] Karina Mochetti and Ricardo Dahab. Expanding a lattice-based HVE scheme. In *SBSeg*, Belo Horizonte, Brazil, 2014.

[54] Karina Mochetti and Ricardo Dahab. Ideal lattice-based (H)IBE scheme. Technical report, Institute of Computing, UNICAMP, Campinas, Brazil, 2014.

[55] Phong Q. Nguyen. Cryptanalysis of the goldreich-goldwasser-halevi cryptosystem. In *CRYPTO*, pages 288–304, 1999.

[56] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.

[57] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *EUROCRYPT*, pages 591–608, 2012.

[58] Tatsuaki Okamoto and Katsuyuki Takashima. Efficient (hierarchical) inner-product encryption tightly reduced from the decisional linear assumption. *IEICE Transactions*, 96-A(1):42–52, 2013.

[59] Victor Y. Pan. *Structured matrices and polynomials: unified superfast algorithms.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.

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

[61] Chris Peikert. An efficient and parallel gaussian sampler for lattices. In *CRYPTO*, pages 80–97, 2010.

[62] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, pages 84–93, 2005.

[63] Yanli Ren and Dawu Gu. Efficient hierarchical identity based encryption scheme in the standard model. *Informatica (Slovenia)*, 32(2):207–211, 2008.

[64] Yanli Ren, Dawu Gu, Shuozhong Wang, and Xinpeng Zhang. New fuzzy identity-based encryption in the standard model. *Informatica, Lith. Acad. Sci.*, 21(3):393–407, 2010.

[65] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, 1978.

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

[67] Saeed Sedghi, Peter van Liesdonk, Svetla Nikova, Pieter H. Hartel, and Willem Jonker. Searching keywords with wildcards on encrypted data. In *SCN*, pages 138–153, 2010.

[68] Jae Hong Seo and Jung Hee Cheon. Fully secure anonymous hierarchical identity-based encryption with constant size ciphertexts. *IACR Cryptology ePrint Archive*, 2011:21, 2011.

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

[70] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[71] Peter W. Shor. Polynominal time algorithms for discrete logarithms and factoring on a quantum computer. In *ANTS*, page 289, 1994.

[72] Kunwar Singh, C. Pandurangan, and A. K. Banerjee. Adaptively secure efficient lattice (H)IBE in standard model with short public parameters. In *SPACE*, pages 153–172, 2012.

[73] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient public key encryption based on ideal lattices. In *ASIACRYPT*, pages 617–635, 2009.

[74] P. van Emde-Boas. Another NP-complete partition problem and the complexity of computing short vectors in a lattice. Technical Report 81-04, Math Inst., University of Amsterdam, Amsterdam, 1981.

[75] Guojun Wang, Qin Liu, and Jie Wu. Hierarchical attribute-based encryption for fine-grained access control in cloud storage services. In *ACM Conference on Computer and Communications Security*, pages 735–737, 2010.

[76] Guojun Wang, Qin Liu, Jie Wu, and Minyi Guo. Hierarchical attribute-based encryption and scalable user revocation for sharing data in cloud servers. *Computers & Security*, 30(5):320–331, 2011.

[77] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[78] Brent Waters. Functional encryption for regular languages. In *CRYPTO*, pages 218–235, 2012.

[79] Keita Xagawa. Improved (hierarchical) inner-product encryption from lattices. In *Public Key Cryptography*, pages 235–252, 2013.

[80] Xiao yuan Yang, Li qiang Wu, Min qing Zhang, and Xiao-Feng Chen. An efficient CCA-secure cryptosystem over ideal lattices from identity-based encryption. *Computers and Mathematics with Applications*, pages 1254–1263, 2013.

[81] Jiang Zhang, Zhenfeng Zhang, and Aijun Ge. Ciphertext policy attribute-based encryption from lattices. In *ASIACCS*, pages 16–17, 2012.

Appendix A

Lemmas on Matrices, Vectors and Rings

This chapter presents some important lemmas on matrices, vectors and rings length and randomness they are used during this work.

Lemma A.1 ([4])**.** *Let $q \geq 2$ and let A be a matrix in $\mathbb{Z}^{n \times m}$ with $m > n$. Let S be a basis for $\Lambda_q^{\perp}(A)$ and $\sigma \geq \|\widetilde{S}\|\omega(\sqrt{\log m})$. Then for $c \in \mathbb{R}^m$ and $\boldsymbol{u} \in \mathbb{Z}_q^n$:*

$$\Pr[\boldsymbol{e} \sim \mathcal{D}_{\Lambda_q^{\boldsymbol{u}}(A),\sigma} : \|\boldsymbol{e}\|\sqrt{m}\sigma] \leq \operatorname{negl}(n).$$

Lemma A.2 ([4])**.** *Let R be a $k \times m$ matrix chosen at random from $\{-1, 1\}^{k \times m}$. Then:*

$$\Pr[\|R\| > 12\sqrt{k+m}] < e^{-(k+m)}$$

Lemma A.3 ([4])**.** *Let $\boldsymbol{e}$ be some vector in $\mathbb{Z}^n$ and let $\boldsymbol{v} \xleftarrow{\$} \overline{\Psi}_{\alpha}^{n}$. Then the quantity $\langle \boldsymbol{e}, \boldsymbol{v} \rangle = \boldsymbol{e}^{\top}\boldsymbol{v}$ when treated as an integer in $[0, q-1]$ satisfies*

$$\langle \boldsymbol{e}, \boldsymbol{v} \rangle \leq \|\boldsymbol{e}\| \cdot (q\alpha \cdot \omega(\sqrt{\log n}) + \sqrt{n}/2)$$

with overwhelming probability (in n).

Lemma A.4 ([4])**.** *Let $m > (n+1)\log q = \omega(\log n)$ be an integer, let R be an $m \times k$ matrix chosen uniformly in $\{-1, 1\}^{m \times k}$ where $k = k(n)$ is polynomial in n. Let A and B be matrices chosen uniformly in $\mathbb{Z}^{n \times m}$ and $\mathbb{Z}^{n \times k}$ respectively. Then, for all vectors $\boldsymbol{x} \in \mathbb{Z}^m$, the distribution $(A, AR, R^{\top}, \boldsymbol{x})$ is statistically close to the distribution $(A, B, R^{\top}\boldsymbol{x})$.*

Lemma A.5 ([4])**.** *For any fixed $n \times m$ matrix X and uniformly random $n \times m$ matrix C, the matrix $C - X$ is uniformly random.*

Lemma A.6 ([6])**.** *Any vector $\boldsymbol{v} \xleftarrow{\$} \overline{\Psi}_{\alpha}^{n}$ has length $O(\alpha q\sqrt{n}) \leq 2n$ with all but exponentially small probability.*

Lemma A.7. *For any fixed constant x and uniformly random $n \times m$ matrix C, the matrix xC is uniformly random.*

Lemma A.8. *([51, Lemma 4.4]) Let $\hat{\boldsymbol{b}} \in \mathcal{R}^k$ be a sequence of arbitrary ring elements. If $\hat{\boldsymbol{a}} \in \mathcal{R}^k$ are independently and uniformly distributed ring elements, then $\hat{\boldsymbol{a}} \otimes \hat{\boldsymbol{b}} = \sum \boldsymbol{a}_i \boldsymbol{b}_i$ is uniformly distributed over the ideal generated by $\hat{\boldsymbol{b}}$. Note that for $k = 1$ we have that $\boldsymbol{a} \cdot \boldsymbol{b}$ is uniformly distributed for $\boldsymbol{a} \in \mathcal{R}$ and $\boldsymbol{b} \in \mathcal{R}$.*

Appendix B

Shamir's Secret Sharing

Shamir's Secret Sharing [69] is a *threshold scheme*, i.e., a scheme to divide a data into n parts in a way that it is only possible to recover the data with at least k parts, for $k \leq n$. Note that $k-1$ or fewer parts does not give enough information to determine the data. Threshold schemes are commonly used in the management of keys.

Shamir's threshold scheme is based on polynomial interpolation, i.e., given k points it is possible to define a polynomial of degree $k-1$. For a data d, the $\mathsf{Split}(d, n, k)$ algorithm chooses a random polynomial p of degree $k-1$, with $p(0) = d$, i.e., coefficient $a_0 = d$. Each share piece d_i, for $i \in [1, n]$ will be a point defined by the polynomial, so:

$$d_i = p(i).$$

To recover the data, the $\mathsf{Join}(\boldsymbol{x}, \boldsymbol{y})$ algorithm reconstructs the polynomial using k points. Several algorithms for polynomial evaluation and interpolation are known and can be used. One of the most efficient methods known is the *Lagrange Algorithm*.

The Lagrange Algorithm calculates k polynomials $l_j(x)$, called *Lagrangian coefficients*, based on the k given points $(x_j, y_j) = (i, d_i)$ and reconstruct the polynomial $p(x)$ as follows:

$$p(x) = \sum_{j=0}^{k} y_j l_j(x)$$

where,

$$l_j(x) = \prod_{m=0}^{k} \frac{x - x_m}{x_j - x_m}$$

Note that the data is, therefore:

$$d = p(0) = \sum_{j=0}^{k} y_j l_j(0)$$

Tables B.1 and B.2 describe the two algorithm for the Shamir's Secret Sharing Scheme.

Algorithm B.1 Split(): Split Algorithm for Shamir's Secret Sharing Scheme

```
Input: data $d$, number of parts $n$ and threshold $k$
Output: vector $\boldsymbol{d}$ of length $n$ with all share pieces $d_i$
    $a_0 \leftarrow d$
    for $i \leftarrow 1$ to $k$
        $a_i \overset{\$}{\leftarrow} \mathbb{Z}$
    for $i \leftarrow 1$ to $n$
        $d_i \leftarrow \sum_{j=0}^{k} a_j i^j$
    output $\boldsymbol{d}$
```

Algorithm B.2 Join(): Join Algorithm for Shamir's Secret Sharing Scheme

```
Input: vector $\boldsymbol{y}$ of length $k$ with share pieces $d_i$ and vector $\boldsymbol{x}$ with each position $i$
Output: data $d$
    for $j \leftarrow 0$ to $k$
        $l_j(x) = \prod_{m=0}^{k} \frac{x - x_m}{x_j - x_m}$
    $p(x) = \sum_{j=0}^{k} y_j l_j(x)$
    $d \leftarrow p(0)$
    output $d$
```

On some of the encryption schemes described in this work, we use the Shamir's Secret Sharing to split a vector of length n into m vectors that can be reconstructed only by finding the k lagrangian coefficients. Algorithms B.3 and B.4 define this idea.

Algorithm B.3 SplitVectors(): Algorithm to split a vector.

```
Input: vector $\boldsymbol{x}$ of length $n$, number of parts $m$ and threshold $k$
Output: $m$ vectors $\boldsymbol{x}_i$ of length $n$
    choose $n$ random polynomials $p_i(x)$ of degree $k - 1$, with $p_i(0) = x_i$
    $\boldsymbol{x}_i \leftarrow [p_1(i), p_2(i), \ldots, p_n(i)]$
```

Algorithm B.4 FindLagrangianCoef(): Algorithm to find the lagrangian coefficients.

```
Input: set $\mathbb{G}$ of size $k$ with each position $i$
Output: a vector $\boldsymbol{l}$ of length $n$ with each lagrangian coeficients (0 for all $i \notin \mathbb{G}$)
  for $i \leftarrow 0$ to $n$
    $l_i = 0$
  for $i \in \mathbb{G}$
    $l(x) = \prod_{j \in \mathbb{G}} \frac{x - j}{i - j}$
    $l_i \leftarrow l(0)$
```

Note that we now have

$$\begin{aligned}\sum_{i \in \mathbb{G}} l_i \boldsymbol{x}_i &= \sum_{i \in \mathbb{G}} l_i [p_1(i), p_2(i), \ldots, p_n(i)] \\ &= \sum_{i \in \mathbb{G}} [l_i p_1(i), l_i p_2(i), \ldots, l_i p_n(i)] \\ &= [\sum_{i \in \mathbb{G}} l_i p_1(i), \sum_{i \in \mathbb{G}} l_i p_2(i), \ldots, \sum_{i \in \mathbb{G}} l_i p_n(i)] \\ &= [x_1, x_2, \ldots, x_n] \\ &= \boldsymbol{x}.\end{aligned}$$

Lemma B.1. *([6, Lemma 3]) Let $\beta = (l!)^2$. Given $k \leq l$ numbers $x_1, \cdots, x_k \in [1, l]$ define the lagrangian coefficients*

$$l_j = \prod_{i \neq j} \frac{-x_i}{x_j - x_i}.$$

Then, for every $1 \leq j \leq k$, the value βl_j is an integer, and $|\beta l_j| \leq \beta^2 \leq (l!)^4$.

Appendix C

Multiplication of Toeplitz Matrices

A *Toeplitz matrix* is a matrix in which each descending diagonal from left to right is constant, i.e., $a_{i,j} = c_{i-j}$

$$A = \begin{pmatrix} a_0 & a_{-1} & a_{-2} & \cdots & a_{-n+1} \\ a_1 & a_0 & a_{-1} & \cdots & a_{-n+2} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ a_{n-2} & a_{n-3} & & \cdots & a_{-1} \\ a_{n-1} & a_{n-2} & & \cdots & a_0 \end{pmatrix}$$

Figure C.1: A Toeplitz matrix A.

A *circulant matrix* is a special kind of Toeplitz matrix in which each row is rotated one element to the right relative to the preceding row.

$$A = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \\ a_n & a_0 & \cdots & a_{n-2} & a_{n-1} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ a_2 & a_3 & \cdots & a_0 & a_1 \\ a_1 & a_2 & \cdots & a_n & a_0 \end{pmatrix}$$

Figure C.2: A circulant matrix A.

An *anti-circulant matrix* is a circulant matrix in which after the rotation, the first element of the row has its sign changed.

$$A = \begin{pmatrix} a_0 & a_1 & \cdots & a_{n-1} & a_n \\ -a_n & a_0 & \cdots & a_{n-2} & a_{n-1} \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ -a_2 & -a_3 & \cdots & a_0 & a_1 \\ -a_1 & -a_2 & \cdots & -a_n & a_0 \end{pmatrix}$$

Figure C.3: An anti-circulant matrix A.

We can use the discrete Fourier transform to obtain a faster multiplication between a circulant matrix and a vector [59]. While, for general matrices, this multiplications is $O(n^2)$, for a circulant matrix we have a $O(n \log n)$ time. Let $\boldsymbol{c}$ be the first row of the circulant matrix C, then we have that:

$$C\boldsymbol{x} = \mathcal{F}_n(\boldsymbol{c})\mathcal{F}_n(\boldsymbol{x}),$$

where

$$\mathcal{F}_n(\boldsymbol{x}) = \boldsymbol{y} \qquad \text{and} \qquad y_i = \frac{1}{n}\sum_{j=0}^{n-1} x_j \cdot e^{2\pi j i\sqrt{-1}/n}.$$

This algorithm can be expanded to any Toeplitz matrix T, by constructing a circular auxiliar matrix C as follows:

$$C = \begin{pmatrix} T & B \\ B & T \end{pmatrix},$$

with

$$B = \begin{pmatrix} 0 & a_{n-1} & a_{n-2} & \cdots & a_2 & a_1 \\ a_{1-n} & 0 & a_{n-1} & \cdots & a_3 & a_2 \\ \vdots & \vdots & \ldots & \vdots & \vdots & \\ a_{-2} & a_{-3} & a_{-4} & \cdots & 0 & a_{n-1} \\ a_{-1} & a_{-2} & a_{-3} & \cdots & a_{1-n} & 0 \end{pmatrix}.$$

Then to get the multiplication of the Toeplitz matrix T by a vector $\boldsymbol{x}$ we make:

$$C \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} T & B \\ B & T \end{pmatrix} \cdot \begin{pmatrix} \boldsymbol{x} \\ \boldsymbol{0} \end{pmatrix} = \begin{pmatrix} T\boldsymbol{x} \\ B\boldsymbol{x} \end{pmatrix}.$$

Lemma C.1. *For two polynomials $f(x) = a_0 + a_1x^1 + \ldots a_{n-1}x^{n-1}$ and $g(x) = b_0 + b_1x^1 + \ldots b_{n-1}x^{n-1}$, let $\boldsymbol{f}$ be the vectorial representation of $f(x)$ in which each position is* *a coefficient of $f(x)$, i.e., $\boldsymbol{v} = (a_0, a_1, \ldots, a_n)$ and let G be the anti-circulant matrix in which the first row is the vector representation of $g(x)$, then:*

$$G^\top \boldsymbol{f} = (f \cdot g) \bmod x^n + 1$$