2023

Contributions to Lattice–based Cryptography

Quoc Huy Le
University of Wollongong

UNIVERSITY
OF WOLLONGONG
AUSTRALIA

Contributions to Lattice–based Cryptography

Quoc Huy Le

This thesis is presented as part of the requirements for the conferral of the degree:

Doctor of Philosophy

Supervisor:
Distinguished Professor Willy Susilo

Co-supervisors:
Doctor Dung Hoang Duong & Professor Josef Pieprzyk

The University of Wollongong
School of Computing and Information Technology

February, 2023

This research has been conducted with the support of a CSIRO Data61 Scholarship and CSIRO Data61 Top up.

Declaration

I, **Quoc Huy Le**, declare that this thesis is submitted in partial fulfilment of the requirements for the conferral of the degree **Doctor of Philosophy**, from the University of Wollongong, is wholly my own work unless otherwise referenced or acknowledged. This document has not been submitted for qualifications at any other academic institution.

Quoc Huy Le

February 2, 2023

Abstract

Post–quantum cryptography (PQC) is a new and fast–growing part of Cryptography. It focuses on developing cryptographic algorithms and protocols that resist quantum adversaries (i.e., the adversaries who have access to quantum computers). To construct a new PQC primitive, a designer must use a mathematical problem intractable for the quantum adversary. Many intractability assumptions are being used in PQC. There seems to be a consensus in the research community that the most promising are intractable/hard problems in lattices. However, lattice–based cryptography still needs more research to make it more efficient and practical.

The thesis contributes toward achieving either the novelty or the practicality of lattice–based cryptographic systems. In particular, we claim the following advancements.

- We construct a forward–secure blind signature scheme over lattices for the first time.
- We develop a trapdoor delegation mechanism for polynomials. Then we use the mechanism to construct the first hierarchical identity–based encryption that applies the problem of degree–parameterised middle–product learning with errors.
- We introduce a cryptosystem called delegatable fully key–homomorphic encryption (DFKHE). The cryptosystem is the basis for a generic construction of puncturable encryption (PE). Moreover, we present a construction of DFKHE (hence, PE) from lattices.
- We propose a cryptographic primitive called delegatable multiple inner product encryption (DMIPE). We show a generic construction for spatial encryption (SE) from DMIPE. Furthermore, we build DMIPE (hence, SE) from a hard lattice problem. Our lattice SE construction has sizes that are smaller than those of SEs over lattices published. The SE design follows a generic construction using the hierarchical inner product encryption. In addition, we demonstrate a formal definition (i.e., syntax and security notions) for the so–called allow–/deny–list encryption (ADE). Further, we propose some transformations that map ADE versions to SE.

Acknowledgements

Doing PhD is challenging but unforgettable to everyone and, of course, to me. However, I was fortunate to be guided and supported by many people, including my supervisors, family, and close friends.

First and foremost, my deep gratitude is for the principal supervisor, Distinguished Professor Willy Susilo, for his kind guidance and support. Without his and the co–supervisors' strong support, I could not have had a great chance to study at the University of Wollongong (UOW). I am inspired by his excellent research outcome and his extraordinary career. Also, he helped me a lot in my personal life. I am honoured to be a Distinguished Professor Willy Susilo's PhD student.

Sincerely thank my respectable co–supervisor, Professor Josef Pieprzyk from CSIRO Data61. His wholehearted support and guidance brought me an excellent motivation to do my PhD. Unfortunately, due to the Covid–19 pandemic, I did not have many chances to see him physically. That is indeed a pity in my PhD time. Being a PhD student of Professor Josef is also a special honour.

Nothing can best express my appreciation to Dr Dung Hoang Duong, my other co-supervisor, for his supervision and support. In particular, Dr Dung organised some summer schools on Cryptography in 2015–2016, where and when I had a chance to participate. After that, with his and other supervisors' priceless help, I was fortunately awarded a reputable CSIRO Data61 Scholarship to join UOW as a PhD candidate. So I truly owe him a debt of gratitude.

Also, I gratefully acknowledge my co–authors for their helpful discussion and excellent collaboration. Additionally, I thank everyone at the Institute of Cybersecurity and Cryptology (IC^2) and Room 6.214 in Building 6 for everything they shared with me.

I want to thank CSIRO Data61 for the scholarships they awarded me, giving me the tremendous financial support in making my PhD dream come true. I am indebted to all staff of UOW and CSIRO Data61, especially of the School of Computing and Information Technology, for their paper works and their lovely assistance during my PhD study.

I will not forget the valuable time that I had with friends – Thanh Khuc, Anh Ta, Yen Tran, Khanh Nguyen, Tuong Nguyen, Hieu Phan, Phong Nguyen, Truc Pham, Trang Le, Linh Nguyen, Long Le, Huy Tran and Priyanka Dutta. They refreshed me with their

active youth.

I also send the honest acknowledgement to all of my friends and students, including ex–colleagues in the Ernst Thälmann High School, Van Huong Nguyen, Huong Le, Hong Thai Pham, and Ha Tran – my undergraduate friends – for their kind encouragement.

Last but not least, my parents and siblings have always been beside me on my PhD journey with their great love. I also love them so much! In especially, I want to give my greatest love to my wife – Tuyet Chu – who has been going with me on the long way of my PhD study with true love.

What I have done in my PhD would not be possible without their all support. Once again, thank you all very much!

List of Abbreviations

In this part, we include some abbreviations frequently used throughout the thesis.

Table 1: List of Abbreviations.

ABE	Attribute–based Encryption
ADE	Allow–/Deny–list Encryption
ATK	Attack
BFE	Bloom Filter Encryption
BS	Blind Signature
CCA	Chosen Ciphertext Attack
CMA	Chosen Message Attack
CPA	Chosen Plaintext Attack
DFPE	Dual–form Puncturable Encryption
DFKHE	Delegatable Fully Key–homomorphic Encryption
DLWE	Decisional Learning with Errors
DMIPE	Delegatable Multiple Inner Product Encryption
DMPLWE	Degree–parameterised Middle–product Learning with Errors
DPT	Deterministic Polynomial Time
DS	Digital Signature
FHE	Fully Homomorphic Encryption
FKHE	Fully Key–homomorphic Encryption
FSBS	Forward–secure Blind Signature
FSE	Forward–secure Encryption
FSUF	Forward–secure Unforgeability
FuPE	Fully Puncturable Encryption
GSO	Gram–Schmidt Orthogonalisation
HIBE	Hierarchical Identity–based Encryption
HIPE	Hierarchical Inner Product Encryption
IBBE	Identity–based Broadcast Encryption

to be continued...

... continued from previous page

IBE	Identity–based Encryption
IPE	Inner Product Encryption
IND	Indistinguishability
INDr	Indistinguishability from Randomness
KEM	Key Encapsulation Mechanism
KH–IRKEM	Key–homomorphic Identity–based Revocable KEM
LWE	Learning with Errors
MLWE	Module Learning with Errors
MPLWE	Middle–product Learning with Errors
MSIS	Module Short Integer Solution
OMUF	One–more Unforgeability
OTS	One–time Signature
PAY	Payload–hiding
PE	Puncturable Encryption
PIBE	Puncturable Identity–Based Encryption
PLWE	Polynomial Learning with Errors
PKE	Public–key Encryption
PQC	Post–quantum Cryptography
PrE	Predicate Encryption
PPT	Probabilistic Polynomial Time
QROM	Quantum Random Oracle Model
RLWE	Ring Learning with Errors
ROM	Random Oracle Model
RSIS	Ring Short Integer Solution
sATT	Selective Attribute
SDM	Standard Model
SE	Spatial Encryption
sID	Selective Identity
SIS	Short Integer Solution
SIVP	Shortest Independent Vectors Problem
sPUN	Selective Puncture
sVAR	Selective Variable
SVP	Shortest Vector Problem
W2A	Worst–case to Average–case
WI	Witness Indistinguishability

Contents

List of Figures

List of Tables

List of Publications

The thesis includes the following papers.

1. [LDSP22b] **Huy Quoc Le**, Dung Hoang Duong, Willy Susilo, Josef Pieprzyk. *Spatial Encryption Revisited: from Delegatable Multiple Inner Product Encryption and More*. In: Atluri, V., Di Pietro, R., Jensen, C.D., Meng, W. (eds) Computer Security – ESORICS 2022. ESORICS 2022. Lecture Notes in Computer Science, vol 13554. Springer, Cham. https://doi.org/10.1007/978-3-031-17140-6_14.

2. [SDLP20] Willy Susilo, Dung Hoang Duong, **Huy Quoc Le**, Josef Pieprzyk. *Puncturable Encryption: A Generic Construction from Delegatable Fully Key– Homomorphic Encryption.* In: Chen L., Li N., Liang K., Schneider S. (eds) Computer Security – ESORICS 2020. Lecture Notes in Computer Science, vol 12309. Springer, Cham. https://doi.org/10.1007/978-3-030-59013-0_6.

3. [LDS+20] **Huy Quoc Le**, Dung Hoang Duong, Willy Susilo, Ha Thanh Nguyen Tran, Cuong Viet Trinh, Thomas Plantard, Josef Pieprzyk. *Lattice Blind Signatures with Forward Security.* In: Liu J., Cui H. (eds) Information Security and Privacy– ACISP 2020. Lecture Notes in Computer Science, vol 12248. Springer, Cham. https://doi.org/10.1007/978-3-030-55304-3_1.

4. [LDSP20] **Huy Quoc Le**, Dung Hoang Duong, Willy Susilo, Josef Pieprzyk. *Trapdoor Delegation and HIBE from Middle-Product LWE in Standard Model.* In: Conti M., Zhou J., Casalicchio E., Spognardi A. (eds) Applied Cryptography and Network Security– ACNS 2020. Lecture Notes in Computer Science, vol 12146. Springer, Cham. https://doi.org/10.1007/978-3-030-57808-4_7.

Other publications not included in the thesis are:

1. [LVD+21] **Huy Quoc Le**, Bay Vo, Dung Hoang Duong, Willy Susilo, Ngoc Thuy Le, Kazuhide Fukushima, Shinsaku Kiyomoto (2021). *Identity-based Linkable Ring Signatures from Lattices*. IEEE Access, vol. 9, pp. 84739-84755, 2021. https://doi.org/10.1109/ACCESS.2021.3087808.

2. [LDR+21] **Huy Quoc Le**, Dung Hoang Duong, Partha S Roy, Willy Susilo, Kazuhide Fukushima, Shinsaku Kiyomoto (2021). *Lattice-based signcryption with equality test in standard model.* Computer Standards & Interfaces 76, 103515, ISSN 0920-5489. https://doi.org/10.1016/j.csi.2021.103515.

3. [SDL20] Willy Susilo, Dung Hoang Duong, **Huy Quoc Le**. *Efficient Post-quantum Identity-based Encryption with Equality Test.* 2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS), 2020, pp. 633-640. https://doi.org/10.1109/ICPADS51040.2020.00088.

4. [LSK+19] **Huy Quoc Le**, Willy Susilo, Thanh Xuan Khuc, Minh Kim Bui, Dung Hoang Duong (2019). *A Blind Signature From Module Lattices*. 2019 IEEE Conference on Dependable and Secure Computing (DSC), Hangzhou, China, 2019, pp. 1-8. https://doi.org/10.1109/DSC47296.2019.8937613.

Chapter 1

Introduction

The world today has seen an explosion of the Internet of Things (IoT) and proliferation of its applications. IoT can collect, transmit, process, and analyse real–time data. This situation naturally raises concerns regarding the users' privacy, integrity and confidentiality of data. Cryptographic algorithms (e.g., hashing, digital signatures, encryption) provide a rich collection of tools for data and information protection.

One of the most incredible technological discoveries in the 21st century is the invention of quantum computers. The technology behind quantum computing exploits laws of quantum mechanics that allow much faster calculations than what is possible for classical computers. Unfortunately, quantum computing is a double–sword tool. On the positive side, it has the potential to deliver an unprecedentedly high computing power. On the negative one, quantum computing in the hands of an adversary is a threat, which is difficult to overestimate. Remarkably, Shor, in his paper [Sho02], has shown that cryptographic algorithms based on the intractability of integer factorisation and discrete logarithm (hence the current standard cryptographic techniques) are completely insecure against quantum adversaries.

The rapid development of quantum technology [a] and the insecurity of commonly used cryptographic technologies against quantum threats (e.g., see the report [SCJ+16, Table 1] for a summary) motivate the research community to find new hardness assumptions intractable against quantum attackers. Besides, NIST [NIS16] has acknowledged a need to face the security challenge and announced in 2016 a standardisation for PQC algorithms. The standardisation has initiated a large–scale effort to select the best candidates for the post–quantum secure standards. It has attracted eighty two (82) submissions and has gone through three rounds with seven final candidates and eight alternatives. At the time of the

[a] The readers are referred to the link https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor for the breakthrough 127–qubit quantum processor built by IBM, accessed on 2 May 2022.

thesis writing, NIST announced the finalists. The finalists include CRYSTALS–Kyber [b], CRYSTALS–Dilithium [c], Falcon [d] and SPHINCS+ [e]. Note that 75% of them are lattice–based [f].

1.1 Lattice–based Cryptography

There is an overwhelming belief in the research community that intractable problems in lattices are the most promising tools for designing PQC algorithms. Intuitively, a lattice looks like a regular grid in space. Mathematically, a lattice comprises all linear combinations, with integral coefficients, of *basis vectors*. The dimension of a lattice is that of its basis. A lattice has invariant quantities called successive minima. Namely, the i–th successive minimum of an n–dimensional lattice is the smallest positive real number λ_i such that the n–dimensional ball centred at the origin (i.e., zero vector) having radius λ_i contains i linearly independent lattice vectors. Note that a lattice of dimension n has n successive minima.

In this section, we will sketch a brief history of lattices and the cryptography based on them. We refer readers to [Pei16] for an excellent survey on lattices and the related research directions. The history of lattices is dated back to the 18^{th} century when lattices were treated as nice abstract objects with no practical significance. At that time, lattices were intensively investigated by great mathematicians such as Gauss, Lagrange, and Minkowski, to name a few. In particular, Gauss and Lagrange used lattices to give proof of some well–known theorems, e.g., the quadratic reciprocity and the four–square in Number Theory. Later, in the early 1910s, Minkowski studied lattices as the theory of "geometry of numbers". Minkowski's first theorem and second theorem are famous results in the lattice theory [g].

Since the early 1980s, lattices have found new applications, especially with the discovery of the famous *LLL algorithm*, named after Lenstra, Lenstra, and Lovász [LLL82]. The LLL algorithm is used to transform a lattice basis into a new one that is more orthogonal and shorter. It, therefore, is a versatile tool in Mathematics. For instance, using LLL, one can factor integer polynomials [LLL82], solve integer programming problems [Len83], and find integer solutions to many problems [Sim10]. Also, one can use LLL in cryptanalysis, e.g., to break a variant of the well–known RSA algorithm [Cop97].

The groundbreaking result of Ajtai [Ajt96] has noticed the computational aspects of

[b] See https://pq-crystals.org/kyber/index.shtml.
[c] See https://pq-crystals.org/dilithium/index.shtml.
[d] See https://falcon-sign.info.
[e] See https://sphincs.org.
[f] See https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4.
[g] See, .e.g., https://web.eecs.umich.edu/ cpeikert/lic15/lec01.pdf

lattices. The author introduced a preliminary variant of the average–case problem, which was then called the *short integer solution* (SIS) later in [MR04]. Informally, the SIS problem requires finding a short integer (vector) solution to a system of random modular equations. Ajtai [Ajt96] also provided a reduction from worst–case problems in lattices to SIS. Two of worst–case problems are the approximate shortest independent vectors problem (γ–SIVP) and the approximate shortest vector problem (γ–SVP). Here γ–SIVP asks, given a n–dimensional lattice, to find a list of linearly independent lattice vectors whose length is at most $\gamma \cdot \lambda_n$. And γ–SVP requires to search for a lattice vector of length $\gamma \cdot \lambda_1$. Based on the worst–case lattice problems, Ajtai [Ajt96] presented a cryptographic one–way function. It is no exaggeration to say that the work [Ajt96] has laid the foundation of *lattice–based cryptography* – a branch of cryptography that focuses on building cryptosystems on worst–case lattice problems. Early results in lattice–based cryptography include: the cryptographic one–way function mentioned above; the first public–key encryption by Ajtai and Dwork [AD97]; the NTRU (now known as NTRUEncrypt) [HPS98]; the GGH Encryption and the GGH Signature [GGH97].

In the early 2000s, many works (e.g., [Mic01, Mic07, Reg03, MR04]) have developed theoretical aspects of the lattice–based cryptography and have greatly improved its efficiency. For example, inspired by the NTRU work on polynomial rings, Micciancio [Mic07] has introduced the polynomial ring variant of SIS (RSIS). The work [Mic07] has also proposed a ring version of the aforementioned one–way function. Regev [Reg03] has introduced Gaussian measures and harmonic analysis, which has resulted in several improvements to the results of [AD97]. Micciancio and Regev [MR04] have investigated the worst–case to average–case (W2A) reduction. At the heart of [MR04] are Gaussian measures.

In 2005, Regev [Reg05] significantly advanced the theory and applications of lattice–based cryptography. He has introduced the average–case *learning with errors* (LWE) problem. LWE generalises the *learning parities with noise* in Coding Theory. LWE can be considered as a dual variant of SIS. LWE has two main versions: search and decisional. Both versions enjoy the hardness equivalence through an elementary classical reduction. Informally, the decisional LWE (for 1 sample) is to decide whether a pair $(\mathbf{a}, c)$ is random over $\mathbb{Z}_q^n \times \mathbb{Z}_q$ or $(\mathbf{a}, c = \langle \mathbf{a}, \mathbf{s} \rangle + e \pmod q)$, where n is a positive integer, q is a prime, $\mathbf{a}$ is a public random vector in $\mathbb{Z}_q^n$, $\mathbf{s}$ is a secret vector in $\mathbb{Z}_q^n$, and e is a small error sampled from some distribution, e.g., a Gaussian distribution. Regev [Reg05] has provided a quantum reduction for LWE. The reduction says that one can transform any (classical, quantum) algorithm for LWE into a quantum algorithm for the (worst–case) lattice problems γ–GapSVP and γ–SIVP. Here, the γ–GapSVP problem is a decisional variant of γ–SVP mentioned above. The γ–GapSVP problem is to decide whether $\lambda_1(\Lambda) \leq d$ or $\lambda_1(\Lambda) > \gamma \cdot d$ for some $d > 0$. Also, it is worth noting that several follow–ups, e.g.,

[Pei09, LM09, BLP+13] have improved the quantum reduction to a classical one for γ–GapSVP only (but not for γ–SIVP). Regev [Reg05] has also given the first public–key encryption (PKE) scheme from lattices. The PKE scheme is provably semantic secure, relying on LWE.

To address the efficiency of lattice–based cryptosystems, some follow–up works regarding algebraic structured LWE/SIS variants have been introduced. In 2010, Lyubashevsky, Peikert and Regev formulated the notion of *ring–based LWE* (RLWE) together with worst–case hardness guarantees over *ideal lattices* [LPR10]. They also provided an essential toolkit for RLWE cryptography [LPR13]. In another work, Stehlé *et al.* [SSTX09] have considered the so–called polynomial LWE (PLWE), a special case of RLWE. Brakerski, Gentry, and Vaikuntanathan [BGV12] have generalised LWE and ring–LWE to a bigger class called *module LWE* (MLWE). Langlois and Stehlé [LS15] also have generalised SIS and RSIS to *module SIS* (MSIS). Both MLWE and MSIS enjoy a W2A reduction, as shown in [LS15]. More recently, in 2017, a new variant of PLWE, called *middle–product LWE problem* (MPLWE), was proposed by Roșca, *et al.* [RSSS17]. Roșca *et al.* [RSSS17] have shown that the hardness of MPLWE can be reduced to that of PLWE.

Upon the standard/ring/module versions of LWE/SIS, besides those mentioned above, numerous more advanced cryptographic constructions over lattices have been proposed. Well–known examples include a multi–bit amortised version of Regev's lattice–based cryptosystem by Peikert *et al.* [PVW08], digital signatures by Lyubashevsky [Lyu09, Lyu12], dual Regev encryption by Gentry *et al.* [GPV08], fully homomorphic encryption (FHE) by Gentry [Gen09], and so forth. Recall that FHE is a kind of encryption that allows performing computations directly on encrypted data. The result of the computations is still in encrypted form. The output of decrypting the result shall be the same as that of doing the computation directly on the unencrypted data.

With the advent of strong tools like *lattice trapdoors* [GPV08, MP12], one can realise many more powerful advanced cryptographic constructions. Some constructions enjoy security in the random oracle model (ROM), such as an identity–based encryption (IBE) and a hash–and–sign signature scheme in [GPV08]. Whilst others are provably secure in the standard model (SDM) such as (hierarchical) identity–based encryptions ((H)IBEs) [CHKP10, ABB10, MP12], digital signatures [Boy10, MP12, DM14, AS15]. We also note that [LVV19] has introduced a new trapdoor for MPLWE, which enables us to build IBE from MPLWE. Recall that in the random oracle model, one or more pseudorandom functions (i.e., those whose output's distribution looks random but not truly random, such as cryptographic hash functions) are considered truly random ones. Thus, any queries to the function(s) will be answered with a value randomly chosen from the function(s)' domain. However, in the standard model, any functions should be treated as their nature.

Therefore, the standard model is also called "the model without random oracles".

The development of lattice–based cryptography has also been accelerated by techniques developed from FHE such as key–switching (i.e., dimension reduction) [BV11b, BV11a], modulus switching [BV11a, BGV12] and a mechanism called *lattice homomorphic evaluations* [BGG+14]. In particular, the lattice homomorphic evaluations were inspired from the Gentry–Sahai–Walters FHE work [GSW13a]. The lattice homomorphic evaluations have realised many advanced cryptosystems, e.g., attribute–based encryption (ABE) [BGG+14, BV16], predicate encryption for circuits [GVW15], pseudo–random functions [BTVW17], fully secure ABE [Tsa19], adaptively secure inner product encryption [KNYY20] and many more.

We summarise some milestones in the development of lattice–based cryptography in Figure 1.1. Each milestone includes the author(s), the corresponding paper, the publishing year (in red colour texts), the main contribution(s)/the breakthrough(s) (in black or blue colour texts) and the concrete lattice–based constructions (in boldface texts).

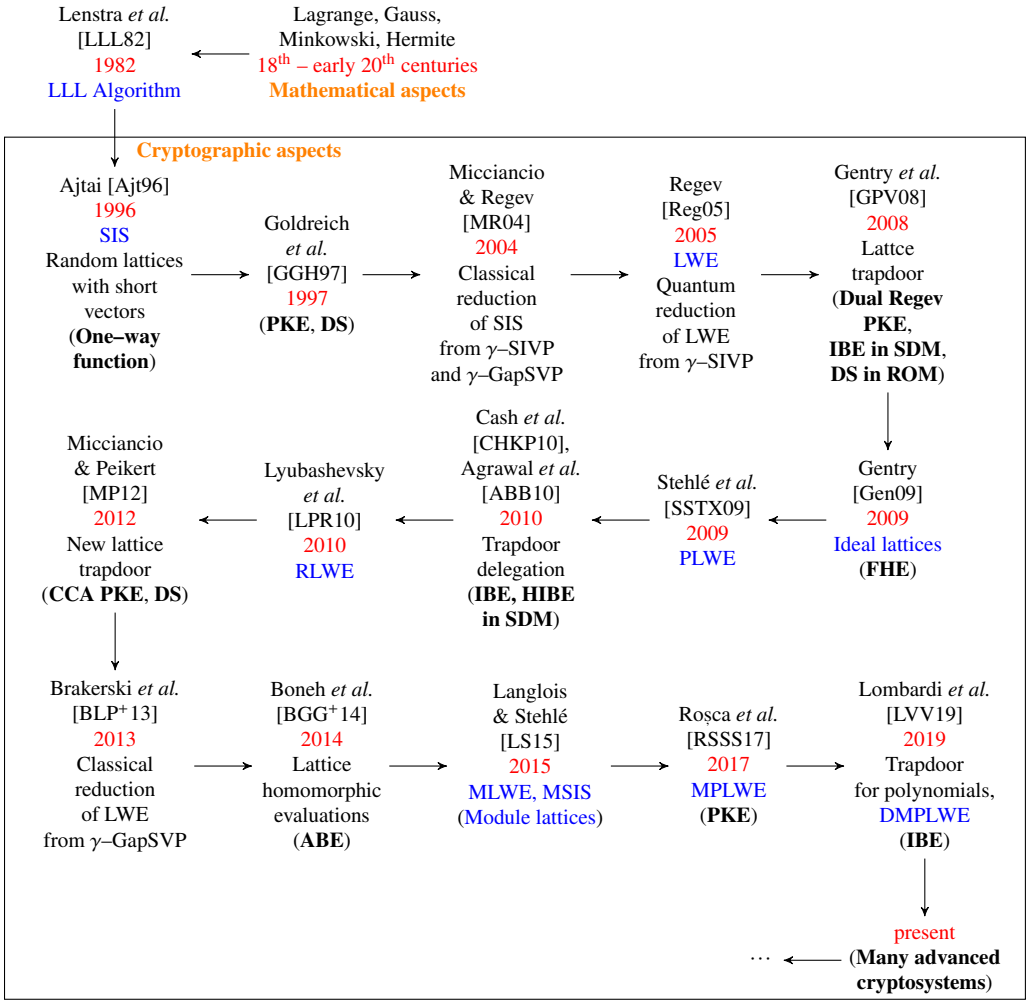

Figure 1.1: Development timeline of lattice–based cryptography.

Lattice–based cryptography is still developing and draws its inspiration from various

pure mathematics areas. The remarkable development of lattice–based cryptography is due to its advantages, as indicated in Peikert's survey [Pei16].

1.2 Contributions and Organisation

1.2.1 Contributions

This thesis is generally motivated by the following:

1. Many concepts of advanced cryptosystems applicable in real–life applications have not been constructed in the lattice setting.

2. Some lattice–based advanced cryptosystems need improvement for better efficiency.

Chapters	Advanced Cryptosystems	Contributions
Chapter 3	Forward-secure Blind Signatures	• Introducing the first forward-secure blind signature over lattices.
Chapter 4	HIBE from DMPLWE	• Proposing a delegation algorihtm for polynomials. • Putting forward the first HIBE based on DMPLWE.
Chapter 5	Puncturable Encryption (PE)	• Proposing a novel primitive called delegatable fully key-homomorphic encryption (DFKHE) • Introducing a generic PE construction from DFKHE. • Constructing a DFKHE and (hence, a new PE) system over lattices.
Chapter 6	Spatial Encryption (SE)	• Introducing a novel primitive called delegatable multiple inner product encryption (DMIPE) • Demonstrating security-preserving conversions between SE and DMIPE. • Giving a DMIPE construction over lattices. Since then, inducing an SE construction over lattices that is improved in sizes in comparison with the existing SEs. • Putting forward a formal framework for allow-/deny-list encryption (ADE) and demonstrating the transformations from some ADE variants to SE.

Figure 1.2: Overview of contributions.

This thesis builds theoretical constructions of advanced cryptosystems which are either unavailable in the literature or existing ones that need to be improved even in heuristic parameters. In addition, the thesis provides evidence that by using powerful tools/techniques, such as trapdoors and homomorphic evaluations based on lattices, one can actually construct many high–functional cryptographic systems. This contributes to the literature

of lattice–based cryptography and further demonstrates the potential of lattices in designing cryptographic protocols of different functionalities.

However, we stress that at the current stage, the contributions in this thesis are mainly on the theoretical aspects, and they should remain abstract constructions without proposing concrete parameters for practical usage or performance comparison with other works and without implementations. This is due to the following reasons:

(i) Some of the cryptosystems introduced in this thesis are the first ones of their types based on lattice problems, e.g., the lattice–based forward–secure blind signature given in Chapter 3 and the hierarchical identity–based encryption based on the degree–parameterised middle–product learning with errors problem presented in Chapter 4. Therefore, there are no other existing lattice–based constructions of the same functionalities for comparison. One may ask to compare our constructions with their counterparts based on other hardness assumptions. However, such comparisons are unfair because they rely on different design rationales.

(ii) Although some of the proposals in my thesis have lattice–based counterparts, they either (a) are implicit constructions induced from the corresponding generic frameworks (e.g., the puncturable encryption in Chapter 5) or (b) have no available implementations and concrete parameters (e.g., the spatial encryption in Chapter 6). In case (a), we compare the asymptotic performance of our puncturable encryption with the others, as in Table 1.3. In case (b), we compare the asymptotic performance of our spatial encryption with others, as in Table 1.4 and Table 6.1.

(iii) Proposing concrete parameters and implementing lattice–based cryptosystems are essential and complex research problems in their own right and are beyond the scope of my thesis. Indeed, these problems are interesting and vital but require intensive engineering investigations. Specifically, the former work needs systematic research on potential attacks (that can affect the underlying lattice problems) and parameter selection procedures for technical tools such as lattice trapdoors and homomorphic evaluations. The algorithms of lattice trapdoors and homomorphic evaluations involve many parameters in a heuristic manner, making heavy use of the Bachmann–Landau asymptotic notations $O(\cdot)$, $\tilde{O}(\cdot)$, $\omega(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$. These asymptotic complexity notations hide constants which affect the choices of specific parameters. Therefore, to specify conditions related to parameters, one may need intensive theoretical research and experiments. Consequently, the possible options for parameters in our constructions are theoretically guaranteed in an asymptotic sense, but their concrete values are elusive at the moment. The latter work (i.e., implementing cryptosystems) also needs considerable engineering skills for programming and optimising our proposed algorithms' running time and sizes. To the

best of our knowledge, none of the existing cryptosystems related to the topics of my thesis, in particular, ones that deploy lattice trapdoors and homomorphic evaluations in the literature, gives any practical implementation and concrete parameters (see, e.g., ones in [GPV08, ABB10, CHKP10, BGG+14, LVV19] etc.). Therefore, we leave the problem of investigating concrete parameters and practical implementations for future works. See Chapter 7 for further detail.

Specifically, the thesis focuses on selected advanced cryptosystems, namely, forward–secure blind signature, hierarchical identity–based encryption, puncturable encryption and spatial encryption. We categorise the contributions according to types of cryptosystem. Note that Figure 1.2 shows an overview of contributions. More details are as follows.

Contributions to Forward–secure Blind Signatures over Lattices. A *blind signature* (BS) [Cha83] allows signing without leaking information about the message content. Blind signatures have found many applications, such as electronic cash [PS96, Section 1] and electronic voting [Kuc10].

In general, a *key exposure attack* is one of the most severe threats against any cryptosystem. If the attack is successful, then secret keys are disclosed. This means that the cryptosystem is wholly broken. It is, therefore, necessary to protect cryptosystems from such attacks. Possible solutions for mitigating the key exposure attacks are discussed in [BM99]. An obvious solution can be to design systems free from such attacks. However, this could be difficult or perhaps impossible to achieve. An alternative approach is to minimise the damage caused by key exposure. This approach can be translated into the so–called *forward security*. Assuming that a system has multiple keys dependent on time, forward security guarantees that disclosing a current key does not make past keys insecure.

Günther [Gün90] has proposed the notion of forward security. Diffie *et al.* [DOW92] have used the notion to consider the security of authenticated key–exchange protocols. Both works [Gün90] and [DOW92] have referred to it as *forward secrecy*. Anderson [And02] has extended the forward security to cover the security of digital signature. Bellare and Milner [BM99] have investigated the secret–key exposure of digital signatures. They have formulated a security notion and defined forward–secure digital signatures. The authors also designed a forward–secure signature from the integer factorisation problem. Other works such as [AR00], and [IR01] have studied the efficiency of the Bellare–Milner signature.

Camenisch [CPS95] proposed blind signatures from the discrete logarithm problem without enjoying forward security. Duc *et al.* [DCK03] coined the first work that investigated forward security for BS. [DCK03] has adopted the syntax and the

security notion from [BM99] to forward–secure blind signatures (FSBSs). Their FSBS design is forward–secure unforgeable based on the strong RSA problem in the random oracle model applying the forking lemma of Pointcheval and Stern [PS96, PS00]. Several more works have followed up [DCK03] on FSBS. For instance, Chow *et al.* [CHYC05] and Jia *et al.* [JFC+10] have designed FSBSs using bilinear pairings. Lai and Chang [LC05] have designed FSBSs based on the factoring problem. In the lattice setting, the BS scheme designed by Rückert [Rüc10] has no forward security. Table 1.1 gives a summary of blind signatures with two options of "Forward–secure?" and "Over lattices?". To our knowledge, no lattice–based FSBS construction has been introduced prior to our work [LDS+20].

Table 1.1: Summary of BS constructions in the literature.

Works	Forward–secure?	Over lattices?
[Cha83]	No	No
[CPS95]	No	No
[DCK03]	Yes	No
[CHYC05]	Yes	No
[LC05]	Yes	No
[JFC+10]	Yes	No
[Rüc10]	No	Yes
Our work [LDS+20]	**Yes**	**Yes**

Our contribution in [LDS+20] is the first lattice–based construction of BS that achieves forward security, i.e., *the first FSBS over lattices*. The security argument of the proposed FSBS is based on the hardness of the SIS problem. The security is proven in the random oracle model. The main tools include: the Fiat–Shamir with aborts framework (Section 3.2.5), the GPV08 trapdoor (Section 2.5.3), the rejection sampling (Section 3.2.1), the oracle replay attack and the forking lemma (Section 3.2.3) and the witness indistinguishability (Section 3.2.4). We present this contribution in Chapter 3.

Contributions to Lattice Hierarchical Identity–based Encryption. *Identity– based encryption* (IBE) [Sha85] is a variant of public–key encryption (PKE), in which a user can use his identity (e.g., email address, user account) as public key. IBE has to include a mechanism that generates users' private keys from their identities. This setting helps mitigate the need to store a list of heavy certificates (generated in the typical PKE for users). Instead, the system must maintain the common parameters, lowering the memory cost.

Hierarchical identity–based encryption (HIBE) [HL02, GS02b] is a hierarchical variant of IBE. User identities in HIBE are arranged in a hierarchical structure that

can be pictured as a directed tree. Besides all IBE functionalities, HIBE also possesses a one–way delegation ability to derive a private key for a child identity from its parents' private key. By "one–way", we mean that it is impossible to get the parents' private key from any of their child's private keys. Some interesting applications of HIBE include broadcast encryption [DF03, YFDL04] and forward–secure encryption [CHK03].

Coming back to the lattice–based cryptography, the original idea by Ajtai [Ajt96, Ajt99] is to generate a hard random lattice instance together with its short vectors/bases. The idea is extended to the *preimage sampleable (trapdoor) functions* (what is known as the *GPV08 trapdoor*) by Gentry *et al.* [GPV08]. Basing on the GPV08 trapdoor, the work [GPV08] has built a hash–and–sign signature scheme and an IBE construction over lattices, which are secure in ROM. Also, the work [GPV08] introduced the so–called *dual Regev encryption* scheme and used it to design an IBE scheme secure in SDM. Agrawal *et al.* [AB09] have introduced IBE secure in SDM using the GPV08 trapdoor. Cash *et al.* [CHKP10] and Agrawal *et al.* [ABB10] independently have added delegation ability to lattice trapdoors. In particular, Cash *et al.* [CHKP10] have formulated *bonsai tree* principles that help generate and delegate short lattice bases. Agrawal *et al.* [ABB10] have also proposed two efficient and distinct trapdoor delegations. Trapdoor delegations by [CHKP10] and [ABB10] enable constructing some lattice–based advanced cryptosystems in SDM, including the first HIBE schemes without pairings [CHKP10, ABB10].

As an improvement of the GPV08 trapdoor, Micciancio and Peikert [MP12] have introduced a new notion of lattice trapdoor (called the *MP12 trapdoor* so that it is not mixed with the GPV08 trapdoor). A close relation between GPV08 and MP12 trapdoors is also shown in [MP12]. After the introduction of MPLWE, Lombardi *et al.* [LVV19] have proposed a variant of MPLWE called degree–parameterised MPLWE (DMPLWE) and a lattice trapdoor (henceforth called the *LVV19 trapdoor*). A dual Regev encryption [GPV08] for DMPLWE is also possible using the LVV19 trapdoor. The dual Regev encryption allows building IBE constructions in both ROM and SDM. Both designs are given in [LVV19]. The IBE construction in SDM of [LVV19] is adapted from the framework of [AB09]. We summarise the known lattice trapdoors as well as IBE and HIBE (if any) in Table 1.2. We notice that a DMPLWE–based HIBE construction in SDM cannot be directly obtained from the IBE construction in SDM of [LVV19] because a trapdoor delegation (for polynomials in DMPLWE) is missing.

We filled in the gap in the work [LDSP20]. Our contributions are as follows. We propose a delegation algorithm for the LVV19 trapdoor. Using the delegation, we

Table 1.2: Lattice trapdoors and IBE, HIBE constructions over lattices.

Trapdoor	Trapdoor delegation?	IBE over lattices?	HIBE over lattices?
GPV08 trapdoor [GPV08]	Yes [ABB10], [CHKP10]	Yes [ABB10], [CHKP10]	Yes [ABB10], [CHKP10]
MP12 trapdoor [MP12]	Yes [MP12]	Yes [MP12]	Yes [MP12]
LVV19 trapdoor [LVV19]	No	Yes [LVV19]	No
LVV19 trapdoor [LVV19]	**Ours [LDSP20]**		**Yes [LDSP20]**

construct *the first HIBE based on the hardness of DMPLWE*. The HIBE scheme is provably secure in SDM. Our work here uses both MP12 and LVV19 trapdoors ([MP12, LVV19] that will be summarised in Section 2.5.3) and a delegation algorithm for the LVV19 trapdoor (proposed in Section 4.2). This thesis includes the contributions in Chapter 4.

Contributions to Puncturable Encryption over Lattices. Puncturable encryption (PE) has been introduced by Green and Miers in [GM15]. It is a PKE variant that has a puncturing mechanism. The mechanism works as follows. Encryption requires *ciphertext tags* while decryption is controlled by *punctures*. In other words, a plaintext is encrypted with a list of ciphertext tags and a decryption key can be punctured on punctures. Puncturing the decryption key produces a new decryption key (*punctured key*). Whether or not the punctured key can decrypt a ciphertext depends on the resemblance of the ciphertext tags (embedded in the ciphertext) and the punctures (in the punctured key). The decryption fails if at least one ciphertext tag and one puncture are identical. Otherwise, the underlying plaintext is successfully recovered. Interesting applications of PE include asynchronous messaging system [GM15], forward–secret zero round–trip time protocol [GHJL17, DJSS18], public–key watermarking scheme [CHN$^+$16], and forward–secret proxy re–encryption [DKL$^+$18].

There have been several PE works in the literature, e.g., [GM15, GHJL17, DGJ18, DJSS18, SSS$^+$20] – see Table 1.3 (except the last row). Green & Miers [GM15] have proposed a generic framework of PE from attribute–based encryption (ABE). Günther *et al.* [GHJL17] have introduced a different approach based on HIBE and one–time signature (OTS). Both Derler *et al.* [DGJ18] and Derler *et al.* [DJSS18] have proposed their generic PE constructions from Bloom filter encryption (BFE). The BFE comes from identity–based broadcast encryption (IBBE) (in [DGJ18])

and IBE (in [DJSS18]). Recently, Sun *et al.* have constructed PE from key–homomorphic identity–based revocable key encapsulation mechanism (KH–IRKEM). The last five columns of Table 1.3 present information relating to the specific PE construction for each work. Here, "$< \infty$" means "bounded/predetermined", while "∞" stands for "unlimited/arbitrary". The column "Over lattices?" says whether or not the specific PE scheme is relied on lattices.

Note that some generic PE frameworks presented in Table 1.3 can be instantiated over lattices thanks to the corresponding lattice–based constructions of the source primitives. However, to our knowledge, no specific PE instantiation over lattices has existed prior to our work [SDLP20]. Even in the most recent work [SSS+20], the specific PE constructions are based on hard problems in pairings.

Table 1.3: The existing PE constructions in the literature in comparison with our PE.

Works	**Generic from**	**Over lattices?**	**Security proof model**	**# Ciphertext tags**	**# Punctures**	**Correctness error**
Green & Miers [GM15]	ABE	No	ROM	$< \infty$	∞	negligible
Günther *et al.* [GHJL17]	HIBE + OTS	No	SDM	$< \infty$	$< \infty$	negligible
Derler *et al.* [DGJ18]	BFE (IBBE)	No	SDM	1	$< \infty$	non–negligible
Derler *et al.* [DJSS18]	BFE (IBE)	No	ROM	1	$< \infty$	non–negligible
Sun *et al.* [SSS+20]	KH–IRKEM	No	SDM	$< \infty$ $< \infty$ ∞ $< \infty$	∞	negligible
Our work	DFKHE	Yes	SDM	$< \infty$	$< \infty$	negligible

Our contributions to PE are as follows. We introduce a primitive called *delegatable fully key– homomorphic* encryption (DFKHE). This primitive is a generalised variant of *fully key–homomorphic encryption* (FKHE) introduced in [BGG+14]. We describe a generic PE framework from DFKHE. We also give a concrete DFKHE construction from lattices. Since then, we get a lattice–based PE construction using the generic PE framework. The proposed lattice PE construction can handle a predetermined number of tags on a ciphertext. A ciphertext size linearly depends on

the fixed number of tags embedded in the ciphertext. However, the work of Brakerski and Vaikuntanathan [BV16] suggests that our lattice PE construction might be modified to handle an unbounded number of ciphertext tags. Furthermore, the lattice PE scheme supports a predetermined number of punctures. In addition, the secret key size increases quadratically with the number of punctures. Regarding security, the construction is *selective CPA secure* proven in SDM owing to that of the underlying DFKHE construction. Note that such a selective CPA can be converted into full CPA security with an exponential loss in parameters as discussed in [CHK04, Kil06, BB11, BGG$^+$14]. Finally, the construction also enjoys a negligible correctness error.

Please see the last row of Table 1.3 for our PE work in comparison with the existing PE work in the literature. Note that all PE constructions included in Table 1.3 enjoy the chosen plaintext attacks (CPA) security. The CPA security limits the adversary's ability to have only the information of ciphertexts of its chosen plaintexts. For further detail, please refer to Section 2.7.3.

Our main technical tools are the dual Regev encryption framework [GPV08] (given in Section 2.7.3), the leftover hash lemma [ABB10] (recapped in Section 2.5.1), the GPV08 trapdoor [GPV08] (reviewed in Section 2.5.3) and the lattice homomorphic evaluations [BGG$^+$14] (recalled in Section 2.5.4).

Details are presented in Chapter 5.

Contributions to Spatial Encryption over Lattices. Spatial encryption (SE) has been coined by Boneh and Hamburg [BH08] followed up by a systematic investigation in Hamburg's thesis [Ham11]. In SE, objects that are associated with ciphertexts (respectively, decryption keys) are affine points/vectors called *attribute vector* (respectively, affine/vector spaces called *predicate space*). We call an *affine SE* if SE involves affine objects. If SE involves vector objects – we call it a *linear SE*. However, one can easily transform an affine SE into a linear SE (see Section 6.3.1 of Chapter 6). Therefore, we only consider linear SEs. The main characteristic of SE is that a ciphertext can be decrypted if and only if the attribute vector associated with the ciphertext is an element of the predicate space, which is associated with the key. Let say the attribute vector x, the the predicate space V. Then informally

$$\mathsf{SE.Dec}(\mathsf{sk}_V, \mathsf{SE.Enc}(\mathsf{pk}, \mu, \mathbf{x})) = \mu \Leftrightarrow \mathbf{x} \in V,$$

where SE.Dec, SE.Enc, pk, sk and μ are decryption algorithm, encryption algorithm, public key, secret key for V and a plaintext in SE, respectively. Additionally, SE is also required to have a capacity that delegates a decryption key for a vector

space, say V, to a decryption key for any V's subspace.

Hamburg [Ham11] has also introduced doubly spatial encryption (DSE), a more expressive variant than SE. All objects involved in both decryption and encryption of DSE are spaces instead of vectors. Decryption is successful if the attribute and predicate spaces intersect (that is, their intersection is not empty). Both SE and DSE have many potential applications claimed in [BH08] and [Ham11]. In particular, one can employ SE to build (broadcast) (H)IBE, or even forward–secure encryption schemes.

As shown in Chapter 6, PE [GM15] and dual-form PE (DFPE) [DRSS21] can be built from SE. Generally, we can use SE to implement the so–called allow–/deny–list encryption (ADE) [DRSS21]. Then, one can consider IBE [Sha85], HIBE [GS02a], forward–secure puncturable IBE (fs–PIBE) [WCW+19], fully PE (FuPE) [DKL+18], PE and DFPE as sub–classes of ADE. Roughly speaking, ADE is a generalisation of PE, in which one can do puncturing on both *positive* and *negative tags* included in the *allow and deny* lists, respectively. Decryption is only enabled with positive tags. The main challenge for building ADE is to make sure that one can *positively puncture* and *negatively puncture* on multiple tags *in any order*. However, [DRSS21] did not give syntax nor security notions for ADE.

Table 1.4 (except the last row) shows various SE constructions in the literature. Most of them rely their security on the number-theoretic assumptions. Examples include bilinear decision Diffie–Hellman exponent (BDDHE) [BH08, Ham11], decisional bilinear Diffie–Hellman (DBDH) [ZC09, CW14, CZF12], and decisional linear (DLIN) [CZF12]. Some SE constructions are proven secure in the generic group model (GGM) [BH08, Ham11], while others – in SDM [ZC09, CZF12, CW14].

Notice that there is a generic SE framework from the *hierarchical inner product encryption* (HIPE) [OT09]. The framework was coined by Chen *et al.* [CLLW14] (called *Chen et al. framework* for short). Using the generic framework, one can realise lattice–based SE constructions, thanks to the lattice–based HIPE counterparts, such as HIPE of [ADCM12] and HIPE of [Xag15]. Both [ADCM12] and [Xag15] HIPE schemes are secure due to the LWE hardness. Unfortunately, such lattice–based SE constructions via Chen *et al.* framework are not free from a few weaknesses detailed below.

The notion of HIPE [OT09] is a generalisation of inner product encryption (IPE) [KSW08]. IPE ciphertext (and decryption key) involves a vector residing in a finite field. The decryption of IPE succeeds if and only if the inner product of two vectors (one in ciphertext and one in decryption key) returns zero. In other words, two vectors are *orthogonal* to each other. In HIPE, multiple attribute vectors are embedded

Table 1.4: (Doubly) SE constructions in the literature.

Works	Generic framework from	Instantiation based on	Security proof model	Security	Over lattices?
Boneh, Hamburg [BH08]		BDDHE	GGM	Selective & Adaptive	No
Hamburg [Ham11]		BDDHE	GGM	Selective	No
Chen, Wee [CW14]		DBDH	SDM	Selective	No
Chen *et al.* [CZF12]		DLIN, DBDH	SDM	Adaptive	No
Zhou, Cao [ZC09]		DBDH	SDM	Selective	No
Abdalla *et al.* [ADCM12]	HIPE via Chen [CLLW14]	LWE	SDM	Selective	Yes
Xagawa [Xag15]	HIPE via Chen [CLLW14]	LWE	SDM	Selective	Yes
Our work	DMIPE	LWE	SDM	Selective	Yes

into a ciphertext and multiple predicate vectors – in a decryption key. Therefore, the condition for successful decryption is now more complicated.

We now give more details on HIPE and on the Chen *et al.* framework to argue why the Chen *et al.* framework is not ideal for SE constructions. A HIPE is parameterised by a field $\mathbb{F}$ (e.g., $\mathbb{Z}_q$ for prime q), a *hierarchical format* $\Delta(\delta;\vec{\ell}_\delta) := (\delta;\ell_1,\ldots,\ell_\delta)$ of depth δ, where ℓ_i's are positive integers. For $k \le \delta$, define $\Gamma_i := \mathbb{F}^{\ell_i}$ and $\Gamma_{|k} := \Gamma_1 \times \cdots \times \Gamma_k$. Given $\vec{V} = (\mathbf{v}_1,\cdots,\mathbf{v}_k) \in \Gamma_{|k}$, the hierarchical predicate $f_{\vec{V}}(\cdot)$ indicated by $\vec{V}$ maps a vector $\vec{X} = (\mathbf{x}_1,\cdots,\mathbf{x}_t) \in \Gamma_{|t}$ to 1 iff the condition $k \le t \wedge \langle \mathbf{v}_i, \mathbf{x}_i\rangle = 0, \forall i \in [k]$ hold. Roughly, with $\Delta(\delta;\vec{\ell}_\delta)$–HIPE, one can do encryption with attribute vectors $\vec{X}$. Accordingly, producing decryption keys can be performed using predicate vectors contained in $\vec{V}$. Decryption is successful if $f_{\vec{V}}(\cdot)$ maps $\vec{X}$ to 1.

The Chen *et al.* [CLLW14] framework works as follows. It transforms a $\Delta(d)$–HIPE (here, we denote by $\Delta(d)$ the hierarchical format $\Delta(d;\vec{\ell}_d) := (d;\ell_1,\cdots,\ell_d)$ in which $\ell_1 = \cdots = \ell_d = d$) to a d–dimensional linear SE and vice versa for any positive integer d. The key idea is that the "belong to" relation in SE and the "orthogonal to" relation in HIPE can be swapped, i.e., for some vector space V, it holds that

$$\mathbf{x} \in V \text{ if and only if } \langle \mathbf{x}, \mathbf{v}\rangle = 0 \text{ for all } \mathbf{v} \in V^\perp. \tag{1.1}$$

Here $V^{\perp}$ is the orthogonal complement space of V. Now, denote $\mathcal{B}^{\perp}(V)$ to be a basis of $V^{\perp}$. Then "$\langle \mathbf{x}, \mathbf{v} \rangle = 0$ for all $\mathbf{v} \in V^{\perp}$" in Equation (1.1) is identical to "$\langle \mathbf{x}, \mathbf{v} \rangle = 0$ for all $\mathbf{v} \in \mathcal{B}^{\perp}(V)$". Deploying HIPE for SE, for each $\mathbf{x}$ and V in SE, one sets $\vec{X}$ and $\vec{V}$ to be $\vec{X} := (\mathbf{x}, \cdots, \mathbf{x})$ and $\vec{V} := \{\mathbf{v}_i : \mathbf{v}_i \in \mathcal{B}^{\perp}(V)\}$, respectively.

The above DMIPE construction incurs the following shortcomings:

- Because of the HIPE's complex structure, it may be hard to instantiate HIPE. There have been two HIPE constructions in the lattice setting of Abdalla *et al.* [ADCM12] and Xagawa [Xag15]. Hence, we have two corresponding SEs based on lattices. Note further that HIPE of [ADCM12], and [Xag15] are not efficient enough in terms of sizes for keys and ciphertext.
- SE encryption involves only one vector. SE decryption keys may take multiple vectors. Contrarily, HIPE encryption involves many vectors for its hierarchical format. Therefore, the Chen *et al.* framework duplicates the SE attribute vector to fit the HIPE hierarchical format. This may result in the size explosion of SE keys and ciphertext keys.

All discussions above raise the question:

"Can we construct SE from (a) primitive(s) other than HIPE, which results in better SE sizes?"

Our contributions to SE are as follows. First, we propose a primitive: delegatable multiple inner product encryption (DMIPE). This primitive notion stems from IPE endowed with the delegation ability to produce decryption keys. Specifically, one can produce a DMIPE ciphertext with *attribute vectors*. Moreover, one can generate a DMIPE decryption key from either other decryption keys or a master secret key. In the former case, one can do that by making the *predicate vectors* list's cardinality bigger. Successful decryption requires that the predicate vector and the attribute vector are "orthogonal" (i.e., their inner product is zero). Roughly, DMIPE generalises IPE differently and more naturally than HIPE. Table 1.5 quantitatively compares IPE, HIPE and DMIPE, whilst Figure 6.1 intuitively illustrate their relation with other primitives.

Second, we present a DMIPE design using LWE. The trapdoors and the homomorphic evaluations over lattices are the core tools for our DMIPE construction. In particular, the homomorphic evaluations focus on inner product functions. Our design is provably selective payload–hiding secure in SDM.

We show that there are *security–preserving* conversions between DMIPE and SE. This also implies that if a lattice DMIPE construction exists, then an SE con-

Table 1.5: Comparison of IPE variants

	IPE	**HIPE**	**DMIPE**
# Predicate vectors	1	$\leq d$	≥ 1
# Attribute vectors	1	d	1
Delegation?	No	Yes	Yes
Dimension of predicate and attribute vectors	same for all	may vary due to the hierarchical format	same for all

struction over lattices exists accordingly. In particular, in terms of sizes, our (d–dimensional) lattice SE construction built on DMIPE is more efficient than (d–dimensional) SE built on $\Delta(d)$–HIPEs from [ADCM12, Xag15].

The last row of Table 1.4 demonstrates an overview of our lattice-based SE work. Further, Table 6.1 in Chapter 6 compares our lattice–based SE with different lattice–based SEs, induced from the lattice–based HIPEs via the Chen *et al.* framework. Moreover, we formally present the syntax and security notions for the allow–/deny–list encryption (ADE). By definition, ADE includes PE [GM15], FuPE [DKL$^+$18], DFPE [DRSS21] and so on. We introduce three ADE versions: sADE, iADE and k–tADE which standing for standard ADE, inclusive ADE and k–threshold ADE, respectively.

Finally, we transform sADE and iADE (hence PE, FuPE, and DFPE) into SE. The main technical tools deployed for the SE work are the dual Regev encryption framework (given in Section 2.7.3), the leftover hash lemma (included in Section 2.5.1), the lattice trapdoor (Section 2.5.3) (that will be merged in a general framework), the lattice homomorphic evaluations (reviewed in Section 2.5.4) and some tools in Algebra (collected in Section 6.3.1). The readers are referred to Chapter 6 for more details.

1.2.2 Organisation of the Thesis

We structure the thesis as follows.

Chapter 1 sketches the history of lattice–based cryptography. It also presents the contributions as well as the organisation of the thesis.

Chapter 2 introduces notations and recaps some necessary knowledge on lattices and related topics, e.g., Gaussian distributions. This chapter also reviews the fundamental tools and frameworks of basic cryptographic primitives that are used in the thesis.

Chapter 3 focuses on constructing blind signatures over lattices supporting forward security. Part of the content of this chapter has been published in the proceedings of the 25th Australasian Conference on Information Security and Privacy (ACISP 2020) [LDS+20]. The paper [LDS+20] got the Best Paper Award in this conference. The author of this thesis is the first, and one of two corresponding authors of [LDS+20]. He contributed to finding the topic, the method to design the cryptosystem, the security analysis of the cryptosystem, and the writing of the manuscript.

Chapter 4 develops a trapdoor delegation for polynomials and proposes a hierarchical identity-based encryption from the degree–parameterised middle–product learning with errors problem. Part of the content of this chapter appeared in [LDSP20], which has been published in the proceedings of the 18th International Conference on Applied Cryptography and Network Security (ACNS 2020). The author of this thesis is the first, and one of the corresponding authors of [LDSP20]. Having received the topic from his supervisors, he contributed to developing the main method, the design of the cryptosystem, the security analysis of the cryptosystem and the writing of the manuscript.

Chapter 5 proposes a novel primitive called delegatable fully key homomorphic encryption (DFKHE) and a generic construction for puncturable encryption from (DFKHE). From an instantiation of the lattice–based DFKHE, one can get a lattice–based puncturable encryption construction. Part of the content of this chapter appeared in [SDLP20], which has been published in the proceedings of the 25th European Symposium on Research in Computer Security (ESORICS 2020). The author of this thesis is one of the corresponding authors of [SDLP20]. Having received the topic from his supervisors, he contributed to finding an appropriate theoretical framework, the design of the cryptosystems, the security analysis of the cryptosystems and the writing of the manuscript.

Chapter 6 revisits spatial encryption (SE) towards a more compact construction from lattices. In particular, we propose a novel primitive called delegatable multiple in-

ner product encryption (DMIPE). The chapter shows a security–preserving equivalence between DMIPE and SE. This chapter also considers allow–/deny–list encryption (ADE) and gives transformations that map ADE variants to SE. Part of the content of this chapter appeared in [LDSP22b], which has been published in the proceedings of the 27th European Symposium on Research in Computer Security (ESORICS 2022). The readers can find the full version of the paper at [LDSP22a]. The author of this thesis is the first and the corresponding author of [LDSP22b]. He contributed to finding the topic, the design of the cryptosystems, the security analysis of the results and the writing of the manuscript.

Chapter 7 presents some concluding remarks and discusses future works.

Chapter 2

Preliminaries

This chapter presents essential notions and definitions utilised throughout the thesis. Specifically, we start with notations (Section 2.1). We then discuss on the background of lattices (Section 2.2), the Gaussian distributions over lattices (Section 2.3). and hardness assumptions (Section 2.4). Next, we will recap some fundamental tools (Section 2.5), security proof models (Section 2.6) followed by basic cryptographic primitives (Section 2.7).

2.1 Notation

Sets. The notation $\mathbb{N}$ (respectively, $\mathbb{Z}$, $\mathbb{R}$) will stand for the set of natural numbers (respectively, integers and real numbers). We sometimes use $\mathbb{Z}^+$ (respectively, $\mathbb{R}^+$) to denote the set of positive integers (respectively, positive real numbers). We also use $\mathbb{Z}_q$ to denote the set of integers modulo q. For a discrete set S, we use $|S|$ standing for its cardinality. Given a positive integer n, $[n]$ stands for the set $\{1, 2, \cdots, n\}$.

Vectors, Matrices. A vector is written in small boldface and column form unless stated otherwise, e.g., $\mathbf{x}$. Zero vector is $\mathbf{0}$. A matrix is written in capital boldface, e.g., $\mathbf{B}$. The notation $\mathbf{b}^\top$ (respectively, $\mathbf{A}^\top$) denotes the transpose of vector $\mathbf{b}$ (respectively, matrix $\mathbf{A}$). The row concatenation of two vectors $\mathbf{a}$ and $\mathbf{b}$ will be written as $(\mathbf{a}^\top|\mathbf{b}^\top)$. The concatenation of two matrices $\mathbf{A}$ and $\mathbf{B}$ will be written as $[\mathbf{A}|\mathbf{B}]$. For vector $\mathbf{c}$ and matrix $\mathbf{A}$, $\mathbf{c}[i]$ and $\mathbf{A}[i]$ represent the i–th element of $\mathbf{c}$ and the i–th column of $\mathbf{A}$, respectively. While $\mathbf{A}[i, j]$ indicates the entry at row i and column j of matrix $\mathbf{A}$. The inner product of vectors $\mathbf{a} := (a_1, \cdots, a_n)$ and $\mathbf{b} := (b_1, \cdots, b_n)$ is denoted by $\langle \mathbf{a}, \mathbf{b} \rangle := \sum_{i=1}^{n} a_i b_i$. We use $\otimes$ to denote the tensor product of two matrices/vectors. For example, if $\mathbf{A} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix}$ and $\mathbf{B}$ is an arbitrary matrix then $\mathbf{A} \otimes \mathbf{B} = \begin{bmatrix} a_1 \cdot \mathbf{B} & a_2 \cdot \mathbf{B} \\ a_3 \cdot \mathbf{B} & a_4 \cdot \mathbf{B} \end{bmatrix}$.

Polynomials. We denote the set of polynomials having a degree less than n whose coefficients reside in the commutative ring R by $R^{<n}[x]$. We use small italic letters for polynomials over R, e.g., polynomial a. For $n \in \mathbb{N}$ and $i \in [n]$, let h_i's be polynomials. We call $\overline{\mathbf{h}} = (h_1, \cdots, h_n)$ an n–polynomial family (i.e., $\overline{\mathbf{h}}$ is a family of n polynomials). The notation $\overline{\mathbf{a}}|\overline{\mathbf{h}}$ denotes a concatenation of $\overline{\mathbf{a}}$ and $\overline{\mathbf{h}}$. The scalar product of two n–polynomial families $\overline{\mathbf{b}} = (b_1, \cdots, b_n)$ and $\overline{\mathbf{u}} = (u_1, \cdots, u_n)$ is defined as $\langle \overline{\mathbf{b}}, \overline{\mathbf{u}} \rangle := \sum_{i=1}^{n} b_i \cdot u_i$.

Let $f(x) \in R[x]$ be a polynomial over R. The notation $R[x]/\langle f(x) \rangle$ denotes the ring of polynomials over R modulo $f(x)$.

Security Games. We present a security notion through a game. The name of the game will be in the form

$$\mathsf{CS}_{\mathcal{B}}^{\mathsf{SEC}}(\lambda) \Rightarrow 0/1/\perp,$$

where CS is the considered cryptosystem, SEC stands for the mentioned security notion, $\mathcal{B}$ is the adversary playing the game with an implicit challenger (which is usually denoted by C), and λ is a parameter (e.g., security parameter). Here, "$\Rightarrow 0/1/\perp$" says that the game will return either 1 (if $\mathcal{B}$ wins) or 0 (if $\mathcal{B}$ loses) or even $\perp$ (if the game is aborted). We always assume that the challenger is honest in all security games mentioned in the thesis. Accordingly, the $\mathcal{B}$'s advantage in this game is denoted by

$$\mathsf{Adv}_{\mathcal{B},\,\mathsf{CS}}^{\mathsf{SEC}}(\lambda).$$

Figure 2.1 gives an example of the IND–CPA security game for PKE. Steps 1, 3 and 5 are done by the challenger. Steps 2 and 4 are performed by the adversary $\mathcal{A}$, where in Step 2, $\mathcal{A}$ can adaptively query to two oracles $\mathsf{EQ}(\cdot)$, and $\mathsf{DQ}(\cdot)$. After the queries, $\mathcal{A}$ releases two challenge plaintexts μ_0^*, μ_1^*. The oracles $\mathsf{EQ}(\cdot)$, and $\mathsf{DQ}(\cdot)$ are then described in enough detail right after in the box. In Step 4, $\mathcal{A}$ based on the challenge ciphertext ct^* to output a guess of the bit b (which is denoted as b'). The challenger returns 1 (i.e., $\mathcal{A}$ wins) if $b' = b$ and returns 0 (i.e., $\mathcal{A}$ loses) otherwise. The $\mathcal{A}$'s advantage in this game is denoted by $\mathsf{Adv}_{\mathcal{B},\mathsf{PKE}}^{\mathsf{IND\text{-}CPA}}(\lambda)$.

Complexity. We review some standard Bachmann–Landau notations like O, $\widetilde{O}$, ω, Ω, Θ and so on. We also define negligible as well as polynomial functions. All functions here are assumed to be in variable n.

- $f = O(g)$ if $\exists C > 0, \exists n_0$ such that $|f(n)| \leq C \cdot g(n)$ for all $n \geq n_0$.
- $f = \widetilde{O}(g)$ if $\exists k > 0$ such that $f(n) = O(g(n)\log^k(n))$.
- $f = o(g)$ if $\forall C > 0$, $\exists n_0$, $\forall n > n_0$ such that $|f(n)| < C \cdot g(n)$.

GAME $\mathsf{PKE}^{\mathsf{IND\text{-}CPA}}_{\mathcal{A}}(\lambda) \Rightarrow 1/0$:
1. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$;
2. $(\mu_0^*, \mu_1^*) \leftarrow \mathcal{A}^{\mathsf{EQ}(\cdot),\mathsf{DQ}(\cdot)}(\mathsf{pp}, \mathsf{pk})$;
3. $b \xleftarrow{\$} \{0,1\}$, $\mathsf{ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu_b^*)$;
4. $b' \leftarrow \mathcal{A}^{\mathsf{ct}^*,\mathsf{EQ}(\cdot)}(\mathsf{pp}, \mathsf{pk})$;
5. If $b' = b$, return 1. Otherwise, return 0.
Queried Oracles:
• Encryption Oracle $\mathsf{EQ}(\mu)$: Return the output of $\mathsf{PKE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu)$.
• Decryption Oracle $\mathsf{DQ}(\mathsf{ct})$: Return the output of $\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$.

Figure 2.1: An example of security game presentation.

- $f = \omega(g)$ if $g = o(f)$, i.e., $\forall C > 0$, $\exists n_0$, $\forall n > n_0$ such that $|f(n)| > C \cdot g(n)$.
- $f = \Theta(g)$ if $f = O(g)$ and $g = O(f)$. That is, $\exists C_1 > 0, \exists C_2 > 0$, $\exists n_0$, $\forall n > n_0$ such that $C_1 \cdot g(n) \leq |f(n)| \leq C_2 \cdot g(n)$.
- f is negligible in n (written as $f = \mathsf{negl}(n)$) if $f = o(1/n^k)$ for any $k > 0$. That is, $\forall k > 0$, $\forall C > 0$, $\exists n_0$, $\forall n > n_0$ such that $|f(n)| < C/n^k$.
- In particular, if $\exists k \in \mathbb{N}$ such that $f = O(n^k)$ then we write $f = \mathsf{poly}(n)$.

Algorithms, Distributions. For the uniform distribution over a set X, we write $\mathcal{U}(X)$. A random variable X follows distribution χ (respectively, density function f) is denoted as $X \sim \chi$ (respectively, $X \sim f$). Whereas, notation $a \leftarrow \chi$ says that a is sampled from the distribution χ. By $b \xleftarrow{\$} X$, we mentions sampling b uniformly at random from the set X. By "overwhelming probability", we mean the probability of "$1 - \mathsf{negl}(\lambda)$" for some implicit security parameter λ. An "efficient" algorithm is a polynomial–time one.

Suppose that A, B are two algorithms. We write $a \leftarrow \mathsf{A}(a_1, \cdots, a_n)$ to say that a is a randomised (probabilistic) output of A on input $(a_1, \cdots, a_n)$. This implicitly means A is probabilistic polynomial–time. On the other hand, we write $b := \mathsf{B}(b_1, \cdots, b_m)$ to say that b is a (deterministic) output of B on input $(b_1, \cdots, b_m)$. This implicitly means B is deterministic polynomial–time.

The statistical distance of X and X' (over a countable set S) is the quantity

$$\Delta(X, X') := \frac{1}{2} \sum_{x \in S} \left| \Pr[X = x] - \Pr[X' = x] \right|.$$

The following triangle inequality holds $\Delta(X_1, X_3) \leq \Delta(X_1, X_2) + \Delta(X_2, X_3)$, where X_1, X_2 and X_3 are three random variables over a countable set S.

Finally, through the text, the logarithm log is to base 2 unless otherwise stated.

Rounding Operations. The notation $\lfloor a \rceil$ means that a is mapped to the integer that is closest to a. If it is a tie then a is rounded up. For example, $\lfloor 5.3 \rceil = 5$, $\lfloor 8.5 \rceil = 8$. Whilst, $\lceil a \rceil$ always rounds a up to the smallest integer that is not less than a. If $a = 10.1$, for example, then $\lceil a \rceil = 11$. Of course, $\lceil 10 \rceil = 10$.

Norms. Throughout the thesis, we consider the following norms:

Euclidean norm. For a vector $\mathbf{v} = (v_1, ..., v_n) \in \mathbb{R}^n$, the Euclidean norm of $\mathbf{v}$ is denoted and defined by $\|\mathbf{v}\| := \sqrt{\sum_{i \in [n]} v_i^2}$. The (Euclidean) norm of a column matrix $\mathbf{A} = [\mathbf{a}_1 | \cdots | \mathbf{a}_n]$ is defined as $\|\mathbf{A}\| := \max_i \|\mathbf{a}_i\|$.

Gram–Schmidt (GS) norm. Let $\mathbf{A} = [\mathbf{a}_1 | \cdots | \mathbf{a}_k]$, be a matrix. The Gram–Schmidt Orthogonalisation (GSO) (cf. Section 2.2 for the definition) of $\mathbf{A}$ is denoted by $\widetilde{\mathbf{A}} := [\widetilde{\mathbf{a}}_1 | \cdots | \widetilde{\mathbf{a}}_k]$. The GS norm of $\mathbf{A}$ is defined as $\|\widetilde{\mathbf{A}}\|$.

Operator norm (a.k.a., sup norm). This norm, on input a matrix $\mathbf{R}$, returns the $\mathbf{R}$'s largest singular value, which is also computed by

$$s_1(\mathbf{R}) := \sup_{\mathbf{u}} \left(\frac{\|\mathbf{R}\mathbf{u}\|}{\|\mathbf{u}\|} \right) = \sup_{\|\mathbf{u}\|=1} (\|\mathbf{R}\mathbf{u}\|).$$

By definition, the following lemma holds.

Lemma 2.1.1. *For all vector* $\mathbf{u}$ *and matrix* $\mathbf{R}$ *such that the operation* $\mathbf{R}\mathbf{u}$ *makes sense, then*

$$\|\mathbf{R}\mathbf{u}\| \leq s_1(\mathbf{R}) \cdot \|\mathbf{u}\|.$$

Lemma 2.1.2 below gives an upper bound of $s_1(\mathbf{R})$ for any matrix $\mathbf{R}$.

Lemma 2.1.2 ([LVV19, Lemma 6])**.** *Let* $\mathbf{R} = (\mathbf{R}[i, j])_{i,j} \in \mathbb{R}^{m \times n}$ *be any matrix, it holds that*

$$s_1(\mathbf{R}) \leq \sqrt{mn} \cdot \max_{i,j} |\mathbf{R}[i, j]|. \tag{2.1}$$

Lemma 2.1.3 demonstrates an upper bound for the operator norm of a matrix (and its transpose) whose each entry is sampled from $\{-1, 1\}$.

Lemma 2.1.3 ([BGG+14, Lemma 2.5])**.** *For integer* $m > 0$*, if* $\mathbf{S}$ *is sampled uniformly from* $\{-1, 1\}^{m \times m}$ *then*

$$s_1(\mathbf{S}) \leq 20\sqrt{m}, \quad s_1(\mathbf{S}^\top) \leq 20\sqrt{m}.$$

Max–absolute–value norm. We denote this norm by $\|\cdot\|_{\max}$. It simply returns the maximum absolute value of the entries of an input vector/matrix. For example, for a vector $\mathbf{a}$ and a matrix $\mathbf{A}$, $\|\mathbf{a}\|_{\max} := \max_i |\mathbf{a}[i]|$ and $\|\mathbf{A}\|_{\max} :=$

$\max_{i,j}|\mathbf{A}[i,j]|$. The following lemma is the well–known result regarding the max–absolute–value norm.

Lemma 2.1.4. *Let* $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ *be vectors of dimensions* $m_1, m_2, m_3 \in \mathbb{N}$*, respectively. Let* $\mathbf{A}_1, \mathbf{A}_2$ *be matrices of appropriate dimensions. The following holds,*

1. $\|\mathbf{e}_1^\top \mathbf{A}_1\|_{\max} \leq m_1 \|\mathbf{e}_1^\top\|_{\max} \cdot \|\mathbf{A}_1\|_{\max}$.
2. $\|(\mathbf{e}_2^\top | \mathbf{e}_3^\top)\mathbf{A}_2\|_{\max} \leq (m_2\|\mathbf{e}_2^\top\|_{\max} + m_3\|\mathbf{e}_3^\top\|_{\max}) \cdot \|\mathbf{A}_2\|_{\max}$.

Infinity norm. We define the infinity norm as $\|\mathbf{v}\|_\infty := \max_{i\in[n]}|v_i|$. Correspondingly, $\|\mathbf{A}\|_\infty := \max_i \|\mathbf{A}_i\|_\infty$.

We have the following well–known property on the relation of infinity norm and Euclidean norm.

Lemma 2.1.5. *For any vector* $\mathbf{x} \in \mathbb{Z}^n$*,*

$$\|\mathbf{x}\|_\infty \leq \|\mathbf{x}\| \leq \sqrt{n} \cdot \|\mathbf{x}\|_\infty.$$

1–norm. The 1–norm is defined as $\|\mathbf{v}\|_1 := \sqrt{\sum_{i\in[n]}|v_i|}$. Correspondingly, $\|\mathbf{A}\|_1 := \max_i \|\mathbf{A}_i\|_1$.

Note that we can define the above norms for a set of vectors in the same way as for a matrix. For example, given a set of vectors $A = \{\mathbf{a}_1, \cdots, \mathbf{a}_k\}$, the Euclidean norm $\|A\| := \max_{i\in[k]}\|\mathbf{a}_i\|$. Also, remark that we can define the above norms for polynomials through its coefficient vector. If $f(x) = 2 + 5x + 4x^2 - x^3 \in \mathbb{Z}[x]$, for example, then $\|f\| = \|\mathbf{f}\| = \|(2,5,4,-1)\|$, where $\mathbf{f} = (2,5,4,-1)$.

Other Notations. The notation $\sqsubseteq$ will stand for the "subspace" relation (Chapter 6). For example, if V_1 is a subspace of V_2 then we write $V_1 \sqsubseteq V_2$. For a string x, notation $|x|$ means the length of x. The notation $\ll$ (respectively, $\gg$) means "much smaller than" (respectively, "much larger than"). We write $a \wedge b$ to mean "a and b".

2.2 Background of Lattices

Lattices and their hard problems are utilised throughout the thesis. Our cryptosystem proposals are based on the hardness of lattice assumptions. We will first present a formal definition of lattice and some related notions in Section 2.2.1. We then state some worst–case problems over lattices in Section 2.2.2.

2.2.1 Lattices

General Lattices. Roughly speaking, a lattice is a collection of points regularly arranged over space. We can define it mathematically as below.

Definition 2.2.1 (Lattice)**.** *Let* $\mathbf{a}_1, \cdots, \mathbf{a}_m$ *be* m *linearly independent vectors in* $\mathbb{R}^n$*. The lattice* $\Lambda = \Lambda(\mathbf{A})$ *with basis* $\mathbf{A} = [\mathbf{a}_1, \cdots, \mathbf{a}_m] \in \mathbb{R}^{n \times m}$ *comprises all linear integral combinations of* $\mathbf{a}_i$*'s, i.e.,*

$$\Lambda(\mathbf{A}) := \{\mathbf{A} \cdot \mathbf{x} : \mathbf{x} \in \mathbb{Z}^m\} = \left\{ \sum_{i=1}^{m} x_i \mathbf{a}_i : x_i \in \mathbb{Z} \right\}.$$

We say that each $\mathbf{a}_i$ *is a basis vector. We also say that* m *(respectively,* n*) is the lattice's rank (respectively, dimension). The lattice is full–rank if* $m = n$*.*

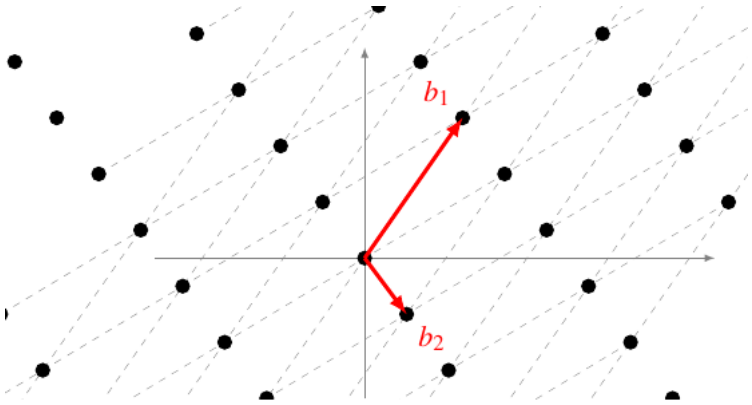

Figure 2.2: A lattice of dimension 2 with basis $\{\mathbf{b}_1, \mathbf{b}_2\}$**.**

One can prove that $\det(\mathbf{A}^\top \mathbf{A}) = \det(\mathbf{B}^\top \mathbf{B})$ given two different bases $\mathbf{A}$ and $\mathbf{B}$ (among infinitely many bases) of a lattice Λ. The invariant $\det(\Lambda(\mathbf{A})) := \sqrt{\det(\mathbf{A}^\top \mathbf{A})}$ is called the *determinant* (or *volume*) of lattice $\Lambda(\mathbf{A})$. Figure 2.2 illustrates a lattice of dimension 2 taking $\{\mathbf{b}_1, \mathbf{b}_2\}$ as a basis.

Denote $\mathcal{B}_n(0, \kappa)$ to be an n–dimensional sphere of radius $\kappa > 0$ (i.e., $\mathcal{B}_n(0, \kappa) = \{\mathbf{v} \in \mathbb{R}^n : \|\mathbf{v}\| \leq \kappa\}$). We are going to state the definition of *successive minima*, which are also invariant quantities for each lattice. Informally, the i–th successive minimum of a n–dimensional lattice is the smallest positive integer κ such that a closed ball of dimension n and radius κ contains i linearly independent lattice vectors. Formally,

Definition 2.2.2 (Successive Minima)**.** *Let* Λ *be an* n*–dimensional lattice. The* i*–th successive minimum* $\lambda_i(\Lambda)$ *of* Λ *is computed as*

$$\lambda_i(\Lambda) := \min\{\kappa : \dim(\mathsf{span}(\Lambda \cap \mathcal{B}_n(0, \kappa))) \geq i\}.$$

Here $\mathsf{span}(\mathbf{A}) := \{\mathbf{Ax} : \mathbf{x} \in \mathbb{R}^m\}$ for any $\mathbf{A} \in \mathbb{R}^{n\times m}$. Obviously, the shortest lattice vectors in Λ are ones having the length of the first successive minimum $\lambda_1(\Lambda)$.

The notion of dual lattices defined below will help determine the so–called smoothing parameters. Roughly speaking, the smoothing parameter is an essential measure of a lattice's quality as it has a close relation to the successive minima of a lattice (see, e.g., [MR04, Section 3] for this relation). We will formally define the smoothing parameter in Definition 2.3.2 and state its properties needed for our work later.

Definition 2.2.3 (Dual Lattices)**.** *Let Λ be n–dimensional lattice. We call the lattice defined by $\Lambda^* := \{\mathbf{x} \in \mathbb{R}^n : \forall \mathbf{v} \in \Lambda, \langle \mathbf{x}, \mathbf{v}\rangle \in \mathbb{Z}\}$ the dual of Λ.*

A lattice Λ is self–dual if the lattice is identical to its dual, that is, $\Lambda = \Lambda^*$. The following lemma is a well–known result about dual lattices.

Lemma 2.2.1. *For any positive integer m, $\mathbb{Z}^m$ is a self–dual lattice.*

Random q–ary Lattices. We will focus on a special sort of lattices which is called *random q–ary lattice* or *random modular lattice*. This is a lattice whose each coordinate is invariant under shifts by a fixed modulus q. Namely, given a matrix $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\mathbf{U} \in \mathbb{Z}_q^{n\times k}$ and a vector $\mathbf{u} \in \mathbb{Z}_q^n$, the following sets are q–ary lattices:

$$\begin{aligned}
\Lambda_q^{\perp}(\mathbf{B}) &:= \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Bx} = \mathbf{0} \,(\mathrm{mod}\ q)\},\\
\Lambda_q^{\mathbf{u}}(\mathbf{B}) &:= \{\mathbf{x} \in \mathbb{Z}^m : \mathbf{Bx} = \mathbf{u} \,(\mathrm{mod}\ q)\},\\
\Lambda_q^{\mathbf{U}}(\mathbf{B}) &:= \left\{\mathbf{X} \in \mathbb{Z}^{m\times k} : \mathbf{BX} = \mathbf{U} \,(\mathrm{mod}\ q)\right\}.
\end{aligned}$$

The shortness of a lattice basis can be measured through the norm of its Gram-Schmidt orthogonalisation (GSO). We, therefore, define the GSO below.

Gram–Schmidt Orthogonalisation. Given any set of vectors $A = \{\mathbf{a}_1, \cdots, \mathbf{a}_m\}$ of the same dimension. The GSO $\widetilde{A} = \{\widetilde{\mathbf{a}}_1, \cdots, \widetilde{\mathbf{a}}_m\}$ for A is computed as follows:

$$\begin{cases} \widetilde{\mathbf{a}}_1 & = \mathbf{a}_1 \\ \widetilde{\mathbf{a}}_i & = \mathbf{a}_i - \sum_{j=1}^{i-1} \frac{\langle \mathbf{a}_i, \widetilde{\mathbf{a}}_j\rangle}{\|\widetilde{\mathbf{a}}_j\|^2},\ i = 2, \cdots, m \end{cases}.$$

Similarly, we can define the GSO matrix for a matrix (hence for a basis of any lattice).

We will concern about lattice bases that are "short", which is determined in the following sense.

Definition 2.2.4. *A lattice basis is called short if its corresponding Gram–Schmidt norm is short.*

2.2.2 Lattice Worst–case Problems

One of the most fundamental lattice problems is the *shortest vector problem* (SVP). Roughly stating, SVP requires finding a lattice vector of the first minimum given a basis of the lattice.

Additionally, there are some worst–case variants of SVP which are proved to be NP–hard, such as the approximation SIVP and the decisional GapSVP. We recall them right below.

Definition 2.2.5 (γ–SIVP)**.** *On input an n–dimensional full-rank lattice* Λ*,* $\gamma := \gamma(n) \in \mathbb{R}^+$*, the* γ–SIVP *problem requires to find n linearly independent lattice vectors* $S := \{\boldsymbol{s}_1, \cdots, \boldsymbol{s}_n\} \subset \Lambda$ *such that* $\|S\| \leq \gamma \cdot \lambda_n(\Lambda)$*.*

Definition 2.2.6 (γ–GapSVP)**.** *On input an n–dimensional full–rank lattice* Λ*,* $\gamma := \gamma(n) \in \mathbb{R}^+$ *and* $d \in \mathbb{R}^+$*, the* γ–GapSVP *problem asks to decide whether* $\lambda_1(\Lambda) \leq d$ *or* $\lambda_1(\Lambda) > \gamma \cdot d$*.*

2.3 Gaussian Distributions over Lattices

The discrete Gaussian distributions are a very important measure for lattice–based cryptography. To review the distributions, we start with the definition of m–dimensional (continuous) Gaussian distribution centred at $\mathbf{c} \in \mathbb{R}^m$ with Gaussian parameter $\sigma > 0$ and characterised through the density function

$$\rho_{\sigma,\mathbf{c}}(\mathbf{x}) := \exp\left(-\frac{\pi\|\mathbf{x}-\mathbf{c}\|^2}{\sigma^2}\right).$$

Now, we give a formal definition for discrete Gaussian distributions.

Definition 2.3.1 (Discrete Gaussians)**.** *The discrete Gaussian distribution over* $\Lambda \subseteq \mathbb{Z}^m$*, parameterised by vector* $\mathbf{c} \in \mathbb{R}^m$ *and* $\sigma \in \mathbb{R}^+$*, is defined as*

$$D_{\Lambda,\sigma,\mathbf{c}}(\mathbf{v}) = \frac{\rho_{\sigma,\mathbf{c}}(\mathbf{v})}{\sum_{\mathbf{w}\in\Lambda}\rho_{\sigma,\mathbf{c}}(\mathbf{w})},\ \forall \mathbf{v} \in \Lambda.$$

We call $\mathbf{c}$ and σ the center and the Gaussian parameter, respectively. We will write ρ_σ (respectively, $D_{\Lambda,\sigma}$) for $\rho_{\sigma,\mathbf{0}}$ (respectively, $D_{\Lambda,\sigma,\mathbf{0}}$). Moreover, we will write ρ for $\rho_{\mathbf{0},1}$. Particularly, if the lattice is $\mathbb{Z}^m$, then we can write $D^m_{\sigma,\mathbf{c}}$ and D^m_σ respectively standing for $D_{\mathbb{Z}^m,\sigma,\mathbf{c}}$ and $D_{\mathbb{Z}^m,\sigma}$.

Before stating some actual results regarding discrete Gaussian distributions, we review the notion of *smoothing parameter* [MR04] for a lattice.Informally, the smoothing parameter is a quantity related to the Gaussian parameter such that a discrete Gaussian distribution over a lattice looks like a continuous one. Formally, we have the following definition.

Definition 2.3.2 ([MR04, Smoothing Parameters]). *For any n–dimensional lattice* Λ *and positive real* $\epsilon > 0$*, the smoothing parameter* $\eta_\epsilon(\Lambda)$ *is the smallest real number* $s > 0$ *such that* $\rho_{1/s}(\Lambda^* \setminus \{\mathbf{0}\}) \leq \epsilon$.

Let Λ be a lattice having an ordered basis $\mathbf{A} = (\mathbf{a}_i)_i$. Define Gram–Schmidt minimum to be

$$\widetilde{bl}(\Lambda) = \min_{\mathbf{A}} \|\widetilde{\mathbf{A}}\| = \min_{\mathbf{A}} \max_i \|\widetilde{\mathbf{a}_i}\|.$$

Lemma 2.3.1 ([GPV08, Lemma 3.1]). *For any n–dimensional lattice* Λ *and real* $\epsilon > 0$*, we have*

$$\eta_\epsilon(\Lambda) \leq \widetilde{bl}(\Lambda) \cdot \sqrt{\frac{\ln(2n(1+1/\epsilon))}{\pi}}.$$

Then, for any $\omega(\sqrt{\log n})$ *function, there is a negligible* $\epsilon = \epsilon(n)$ *for which*

$$\eta_\epsilon(\Lambda) \leq \widetilde{bl}(\Lambda) \cdot \omega(\sqrt{\log n}).$$

We sometimes deal with the smoothing parameter $\eta_\epsilon(\mathbb{Z})$ of $\mathbb{Z}$. We will give a lower and upper bound for it. By Lemma 2.2.1, $\mathbb{Z}$ is a self–dual lattice. Then,

$$\eta_\epsilon(\mathbb{Z}) = \min\{s : \rho_{1/s}(\mathbb{Z}) \leq 1 + \epsilon\}.$$

We know that for any $s > 0$, $\rho_{1/s}(\mathbb{Z}) = 1 + 2\sum_{i=1}^{\infty} \exp(-\pi s^2 i^2) > 1 + 2\exp(-\pi s^2)$. For any $0 < \epsilon < 1$, by choosing s fulfilling $\rho_{1/s}(\mathbb{Z}) \leq 1 + \epsilon$, then $2\exp(-\pi s^2) < \epsilon$, which implies that $s > \sqrt{\ln(2/\epsilon)/\pi}$. Therefore,

$$\eta_\epsilon(\mathbb{Z}) > \sqrt{\frac{\ln(2/\epsilon)}{\pi}}.$$

Moreover, by Lemma 2.3.1, we have

$$\eta_\epsilon(\mathbb{Z}) \leq \widetilde{bl}(\mathbb{Z}) \cdot \sqrt{\frac{\ln(2+2/\epsilon)}{\pi}} = \sqrt{\frac{\ln(2+2/\epsilon)}{\pi}}.$$

The equation yields the result presented in Lemma 2.3.2 below.

Lemma 2.3.2 (Bounds of $\eta_\epsilon(\mathbb{Z})$). *For any* $0 < \epsilon < 1$*, the following holds*

$$\sqrt{\frac{\ln(2/\epsilon)}{\pi}} < \eta_\epsilon(\mathbb{Z}) \leq \sqrt{\frac{\ln(2+2/\epsilon)}{\pi}}.$$

Lemma 2.3.3 ([GPV08, Corollary 5.4]). *Let* $m, n, q \in \mathbb{Z}^+$*, q is prime,* $m \geq 2n\log q$*. Then for all but* $2q^{-n}$ *fraction of all matrix* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ *and for any* $\sigma \geq \omega(\sqrt{\log m})$*, the distribution of* $\mathbf{u} := \mathbf{Ae} \pmod q$ *is statistically close to uniform over* $\mathbb{Z}_q^n$*, where* $\mathbf{e} \leftarrow D_{\mathbb{Z}^m,\sigma}$.

Below, we give some results related to Gaussian distributions.

Lemma 2.3.4 ([MR04, Lemma 4.4], [GPV08, Lemma 2.9]). *For any n–dimensional lattice* Λ, $\mathbf{c} \in \mathsf{span}(\Lambda)$ *and any* $0 < \epsilon < 1$, *any* $\sigma \geq \eta_\epsilon(\Lambda)$, *then*

$$\Pr_{\mathbf{x} \leftarrow D_{\Lambda,\sigma,\mathbf{c}}}[\|\mathbf{x} - \mathbf{c}\| \geq \sigma\sqrt{n}] \leq \frac{1+\epsilon}{1-\epsilon} \cdot 2^{-n}.$$

In particular, when $\Lambda = \mathbb{Z}$, we have Lemma 2.3.5 given below.

Lemma 2.3.5 ([GPV08, Lemma 4.2]). *For any* $\epsilon > 0$, *any* $\sigma \geq \eta_\epsilon(\mathbb{Z})$ *and any* $t > 0$,

$$\Pr_{x \sim D_{\mathbb{Z},\sigma,c}}[|x - c| \geq t \cdot \sigma] \leq 2e^{-\pi t^2} \cdot \frac{1+\epsilon}{1-\epsilon}.$$

Moreover, for $0 < \epsilon < 1/2$ *and* $t \geq \omega(\sqrt{\log n})$,

$$2e^{-\pi t^2} \cdot \frac{1+\epsilon}{1-\epsilon} \leq \mathsf{negl}(n).$$

Lemma 2.3.6 stated below is necessary for arguments related to the rejection sampling reviewed in Section 3.2.1.

Lemma 2.3.6 ([Lyu12, Lemma 4.5]). *For any* $\mathbf{c} \in \mathbb{Z}^m$, *if* $\sigma = \alpha \cdot \|\mathbf{c}\|$, *where* $\alpha > 0$, *we have* $\Pr\left[D^m_\sigma(\mathbf{x})/D^m_{\sigma,\mathbf{c}}(\mathbf{x}) \leq e^{12/\alpha + 1/(2\alpha^2)} : \mathbf{x} \leftarrow D^m_\sigma\right] \geq 1 - 2^{-100}$.

Remark 2.3.1. *In Lemma 2.3.6, if* $\alpha = 12$, *i.e.,* $\sigma = 12\|\mathbf{c}\|$ *then* $D^m_\sigma(\mathbf{x})/D^m_{\sigma,\mathbf{c}}(\mathbf{x}) \leq e^{1+1/288}$ *with probability not smaller than* $1 - 2^{-100}$.

Next, we present the definition of (B,ν)–*bounded distributions* (with $B, \nu > 0$) and then show that a discrete Gaussian distribution over $\mathbb{Z}$ is actually $(k\sigma, 2\exp(-k^2))$–bounded for any $k > 0$.

Definition 2.3.3 ([BV16, Definition 2.1]). *A distribution* χ *supported over* $\mathbb{Z}$ *is* (B,ν)–*bounded if*

$$\Pr_{x \leftarrow \chi}[|x| > B] < \nu.$$

Lemma 2.3.7 ([Lyu12, Lemma 4.4]). *For any positive* $k > 0$, *it holds that*

$$\Pr[|x| > k\sigma : \ x \leftarrow D_{\mathbb{Z},\sigma}] \leq 2\exp(-\frac{k^2}{2}).$$

Note that in Lemma 2.3.7, $1 - 2\exp(-72) \approx 1 - 2^{-100}$ if $k = 12$. Finally, this section gives a result according to Gaussian distributions and the sup norm.

Lemma 2.3.8 ([BGG+14, Lemma 2.5]). *For integers* $n, m, k, q > 0$ *and a positive real number* $\sigma > 0$, *matrices* $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{U} \in \mathbb{Z}_q^{n \times k}$, *if* $\mathbf{R}$ *is sampled from the distribution* $D_{\Lambda_q^{\mathbf{U}}(\mathbf{A}),\sigma}$, *then*

$$s_1(\mathbf{R}) \leq \sigma\sqrt{mk}, \quad s_1(\mathbf{R}^\top) \leq \sigma\sqrt{mk}.$$

2.4 Hardness Assumptions

Our specific constructions in the upcoming chapters base their security heavily on the average–case lattice hardness assumptions reviewed below.

2.4.1 Learning with Errors Problem

The learning with errors problem (LWE) is an average–case problem in lattices. For cryptographic purposes, two LWE versions usually taken into consideration are: (i) the search LWE and (ii) the decisional LWE. Roughly saying, the former is, given some LWE samples, to require recovering the LWE secret, while the latter requires distinguishing an LWE sample from a uniformly random one.

In the thesis, we only concentrate on the decisional variant of LWE (DLWE). A DLWE instance is parameterised by $n, m \in \mathbb{Z}^+$, a prime number q and a noise distribution χ over $\mathbb{Z}_q$. We denote such a DLWE instance by (n, m, q, χ)–DLWE.

Definition 2.4.1 (DLWE, [Reg09])**.** *A DLWE instance requires to distinguish the two joint distributions:*
(a) $(\mathbf{B}, \mathbf{r}^\top \mathbf{B} + \mathbf{x}^\top)$,
(b) $(\mathbf{B}, \mathbf{b}^\top)$, *in which* $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}, \mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^n, \mathbf{x} \leftarrow \chi^m, \mathbf{b} \xleftarrow{\$} \mathbb{Z}_q^m$.

Define by

$$\mathsf{Adv}_{\mathcal{S}}^{(n,m,q,\chi)-\mathsf{DLWE}} := \left|\Pr[\mathcal{S}(\mathbf{B}, \mathbf{r}^\top \mathbf{B} + \mathbf{x}^\top) = 1] - \Pr[\mathcal{S}(\mathbf{B}, \mathbf{b}^\top) = 1]\right|$$

the advantage of any probabilistic polynomial–time solver $\mathcal{S}$ *against* (n, m, q, χ)–DLWE.

Then, (n, m, q, χ)–DLWE *is hard if* $\mathsf{Adv}_{\mathcal{S}}^{(n,m,q,\chi)-\mathsf{DLWE}} \leq \mathsf{negl}(n)$ *for all* $\mathcal{S}$.

Regarding the DLWE hardness, there have been some related well–known works, such as [Reg09] with a quantum reduction from GapSVP. The reduction means that if there exists a solving algorithm for DLWE, one can (quantumly) transform it into a solving algorithm for GapSVP. After that, [Pei09, BLP+13] gave a classical reduction (i.e., the transformation now is not quantum but classical) from the worst–case GapSVP problem. We formally restate Lemma 2.4.1 for choosing parameters in our lattice designs.

Lemma 2.4.1 (DLWE Hardness, [BV16, Corollary 3])**.** *The* (n, m, q, χ)–DLWE *problem is at least as hard as the classical* γ–GapSVP *and the quantum* γ–SIVP*, where* $q = q(n) \leq 2^n$*,* $m = \Theta(n \log q) = \mathsf{poly}(n)$*,* $\chi = \chi(n)$ *such that* χ *is a* (B, ν)*–bounded for some* $B = B(n)$*,* $q/B \geq 2^{n^\nu}$ *and* $\gamma = 2^{\Omega(n^\nu)}$.

2.4.2 Degree–parameterised Middle–product Learning with Errors

The degree–parameterised middle–product LWE (DMPLWE) problem, introduced by Lombardi *et al.* [LVV19], is a variant of the middle–product LWE problem (MPLWE) [RSSS17]. MPLWE is, in turn, an LWE variant that exploits the so–called *middle–product*. Before presenting a formal definition for the DMPLWE problem, we first need to define the notion of middle–product.

Definition 2.4.2 (Middle–product, [RSSS17, Definition 3.1])**.** *Let d_a, $d_b, k, d \in \mathbb{Z}^+$ such that $d_a + d_b - 1 = 2k + d$. The middle–product of two polynomials $a \in \mathbb{Z}^{<d_a}[x]$ and $b \in \mathbb{Z}^{<d_b}[x]$ is defined as follows:*

$$\odot_d : \mathbb{Z}^{<d_a}[x] \times \mathbb{Z}^{<d_b}[x] \to \mathbb{Z}^{<d}[x], (a,b) \mapsto \left\lfloor \frac{ab \bmod x^{k+d}}{x^k} \right\rfloor. \tag{2.2}$$

The following lemma states a helpful property of middle–product.

Lemma 2.4.2 ([RSSS17, Lemma 3.3])**.** *Let $d, k, n \in \mathbb{Z}^+$. For all $r \in R^{<k+1}[x]$, $a \in R^{<n}[x]$, $s \in R^{<n+d+k-1}[x]$, it holds that $r \odot_d (a \odot_{d+k} s) = (r \cdot a) \odot_d s$.*

Let $n', t' \in \mathbb{Z}^+$, $q \geq 2$, $\mathbf{d} = (d_1, \cdots, d_{t'}) \in [\frac{n'}{2}]^{t'}$. Let χ be a distribution over $\mathbb{R}_q$. For $s \in \mathbb{Z}_q^{<n'-1}[x]$, define $\mathsf{DMP}_{q,n',\mathbf{d},\chi}(s)$ to be a distribution over $\prod_{i=1}^{t'}(\mathbb{Z}_q^{n'-d_i}[x] \times \mathbb{R}_q^{d_i}[x])$ that outputs $(f_i, \mathsf{ct}_i := f_i \odot_{d_i} s + e_i)_{i \in [t']}$, where $f_i \xleftarrow{\$} \mathbb{Z}_q^{<n'-d_i}[x]$ and $e_i \leftarrow \chi^{d_i}[x]$ for each $i \in [t']$.

Now, we define the DMPLWE problem.

Definition 2.4.3 (DMPLWE, [LVV19, Definition 9])**.** *A $(q, n', \mathbf{d}, \chi)$–$\mathsf{DMPLWE}$ instance requires distinguishing $\mathsf{DMP}_{q,n',\mathbf{d},\chi}(s)$ from $\prod_{i=1}^{t'} \mathcal{U}(\mathbb{Z}_q^{n'-d_i}[x] \times \mathbb{R}_q^{d_i}[x])$ through the same arbitrarily many samples of them.*

We define

$$\mathsf{ef}(f) := \max_{g \in \mathbb{Z}^{<2m-1}[x]} \frac{\|g \mod f\|_\infty}{\|g\|_\infty}$$

to be the *expansion factor* ([LM06]) of $f \in \mathbb{Z}[x]$ be degree–m polynomial.

Adapting the proof technique of [RSSS17], Lombardi *et al.* [LVV19] also showed that $(q, n', \mathbf{d}, \chi)$–$\mathsf{DMPLWE}$ is as hard as $(f, d, D_{\alpha \cdot q})$–$\mathsf{PLWE}$ with f such that $|\mathsf{ef}(f)| \leq \mathsf{poly}(n')$. We will define PLWE below.

Definition 2.4.4 (PLWE, [SSTX09])**.** *Let $n \in \mathbb{Z}^+$, $q \geq 2$. Let f be a degree–m integer polynomial. Let χ be a distribution over $\mathbb{R}[x]/\langle f \rangle$. Choose an $s \xleftarrow{\$} \mathbb{Z}_q[x]/\langle f \rangle$. A decisional problem polynomial learning with errors instance $(f, d, D_{\alpha \cdot q})$–$\mathsf{PLWE}(s)$ requires distinguishing the joint distribution of $\{(a_i, a_i \cdot s + e_i) : a_i \xleftarrow{\$} \mathbb{Z}_q[x]/\langle f \rangle, e_i \leftarrow \chi\}_{i \in I}$, from that of $\{(a_i, c_i) : a_i \xleftarrow{\$} \mathcal{U}(\mathbb{Z}_q[x]/\langle f \rangle), c_i \xleftarrow{\$} \mathcal{U}(\mathbb{R}_q[x]/\langle f \rangle)\}_{i \in I}$ for an arbitrarily index set I.*

The authors of [SSTX09] proved that $(f, d, D_{\alpha \cdot q})$–PLWE is as hard as SVP over ideal lattices in $\mathbb{Z}[x]/\langle f \rangle$.

Let $\zeta > 0$. We define $\mathcal{F}(\zeta, \mathbf{d}, n')$ to be the set of polynomials $f \in \mathbb{Z}[x]$ such that the following hold: (i) f is monic and has degree m, (ii) the constant coefficient of f is co–prime with q, (iii) $m \in \cap_{i=1}^{t'}[d_i, n' - d_i]$ and (iv) $\mathsf{ef}(f) < \zeta$. We now claim the hardness for DMPLWE via [LVV19].

Lemma 2.4.3 (Hardness of DMPLWE, [LVV19, Theorem 2])**.** *Let $n' \in \mathbb{Z}^+$, $q \geq 2$, $\mathbf{d} = (d_1, \cdots, d_{t'}) \in [\frac{n'}{2}]^{t'}$, and $\alpha \in (0, 1)$. Then, there is a probabilistic polynomial–time reduction from $(f, d, D_{\alpha \cdot q})$–PLWE for any $f \in \mathcal{F}(\zeta, \mathbf{d}, n')$ to $(q, n', \boldsymbol{d}, D_{\alpha' \cdot q})$–DMPLWE with $\alpha' = \alpha \cdot \zeta \cdot \sqrt{\frac{n'}{2}}$.*

2.4.3 Shortest Integer Solution Problem

Definition 2.4.5 (SIS, [Lyu12, Definition 3.1])**.** *Let $n, m, q \in \mathbb{Z}^+$ and $\eta \in \mathbb{R}^+$. Let $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. The (q, n, m, η)–SIS instance with respect to $\mathbf{B}$ asks to search for a short non–zero integer vector $\mathbf{w}$ in $\mathbb{Z}^m$ subject to $\mathbf{B}\mathbf{w} = \mathbf{0} \pmod q$ and $\|\mathbf{w}\| \leq \eta$.*

The following lemma demonstrates the hardness of (q, n, m, η)–SIS problem.

Lemma 2.4.4 (Hardness of SIS, [GPV08, Proposition 5.7])**.** *Let $m \in \mathbb{Z}^+$ be polynomial–bounded. Suppose that $\beta = \mathsf{poly}(n)$, q is prime and $q \geq \eta \cdot \omega(\sqrt{n \log n})$. Then, (q, n, m, η)–SIS is as hard as γ–SIVP problem for $\gamma = \eta \cdot \tilde{O}(\sqrt{n})$.*

For $d \in \mathbb{R}^+$, define (q, n, m, d)–SIS to be a distribution that outputs pairs $(\mathbf{A}, \mathbf{A} \cdot \mathbf{s} \pmod q)$, in which $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ and $\mathbf{s} \xleftarrow{\$} \{-d, \cdots, 0, \cdots, d\}$. The following lemma characterises the distribution.

Lemma 2.4.5 (Discussed in [Lyu12])**.** *For $d \gg q^{m/n}$, (q, n, m, d)–SIS is statistically close to $\mathcal{U}(\mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^n)$. Furthermore, given $(\mathbf{A}, \mathbf{u}) \sim (q, n, m, d)$–SIS, there exist many possible $\mathbf{s}$'s such that $\mathbf{A} \cdot \mathbf{s} = \mathbf{u} \pmod q$.*

2.5 Fundamental Tools

We employ some technical tools to design cryptosystems that base their security on lattice problems. In the following, we recap the common fundamental ones for completeness. Specifically, we recall the randomness extraction (leftover hash lemma) in Section 2.5.1, the gadget matrix in Section 2.5.2, the lattice trapdoors in Section 2.5.3. and the lattice homomorphic evaluations in Section 2.5.4.

2.5.1 Randomness Extraction

The following specific leftover hash lemma is helpful and standard to argue the security of many cryptographic schemes in the lattice setting.

Lemma 2.5.1 (Leftover Hash Lemma, [ABB10, Lemma 4], [BGG+14, Lemma 2.7])**.** *Let $m, n, q \in \mathbb{Z}^+$ subject to $q > 2$ prime, $k = \mathsf{poly}(n)$, and $m > (n+1)\log q + \omega(\log n)$. Let $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times k}$. Then, the joint distributions $(\mathbf{B}, \mathbf{BX}, \mathbf{X}^\top \mathbf{r})$ and $(\mathbf{B}, \mathbf{A}, \mathbf{X}^\top \mathbf{r})$ are statistically close. Here, $\mathbf{X} \xleftarrow{\$} \{-1,1\}^{m\times k}$ and $\mathbf{r} \in \mathbb{Z}_q^m$.*

Lemma 2.5.1 will be used in reasoning on indistinguishability of hybrid games in security proofs for specific constructions that based on the DLWE problem, namely in Chapters 5–6.

2.5.2 The Gadget Matrix

The gadget matrix was introduced in some works, e.g., [MP12, GSW13b], much involved in the MP12 trapdoor [MP12] (Section 2.5.3) and in the lattice homomorphic evaluation (Section 2.5.4). We formally define it below.

Let $n, q \in \mathbb{Z}^+$, and $k := \lceil \log q \rceil$. Define $\mathbf{G} = \mathbf{I}_n \otimes \mathbf{g}^\top \in \mathbb{Z}_q^{n\times nk}$, where $\mathbf{g}^\top = (1, 2, 4, ..., 2^{k-1}) \in \mathbf{Z}_q^k$, and $\mathbf{I}_n \in \mathbb{Z}^{n\times n}$ is an n–dimensional identity matrix. We call $\mathbf{G}$ the *gadget matrix*. Explicitly,

$$\mathbf{G} = \begin{bmatrix} \cdots \mathbf{g}^\top \cdots & & & \\ & \cdots \mathbf{g}^\top \cdots & & \\ & & \ddots & \\ & & & \cdots \mathbf{g}^\top \cdots \end{bmatrix} \in \mathbb{Z}^{n\times nk}.$$

One can find a short basis, say $\mathbf{S}_k \in \mathbb{Z}^{k\times k}$, for $\Lambda^\perp(\mathbf{g}^\top)$, i.e., $\mathbf{g}^\top \cdot \mathbf{S}_k = \mathbf{0} \in \mathbb{Z}_q^k$. Accordingly, the matrix $\mathbf{T}_\mathbf{G} := \mathbf{I}_n \otimes \mathbf{S}_k \in \mathbb{Z}^{nk\times nk}$ will be a short basis of $\Lambda^\perp(\mathbf{G})$. Precisely, the matrices $\mathbf{S}_k$ and $\mathbf{T}_\mathbf{G}$ are as follows:

$$\mathbf{S}_k = \begin{bmatrix} 2 & & & & q_0 \\ -1 & 2 & & & q_1 \\ & -1 & \ddots & & \vdots \\ & & \ddots & 2 & q_{k-2} \\ & & & -1 & q_{k-1} \end{bmatrix} \in \mathbb{Z}^{k\times k}, \quad \mathbf{T}_\mathbf{G} = \begin{bmatrix} \mathbf{S}_k & & & \\ & \mathbf{S}_k & & \\ & & \ddots & \\ & & & \mathbf{S}_k \end{bmatrix} \in \mathbb{Z}^{nk\times nk},$$

where $(q_0, q_1, \cdots, q_{k-1})$ is the binary expansion of q, i.e., $q = \sum_{i=0}^{k-1} 2^i q_i$. One can compute that $\|\widetilde{\mathbf{T}_\mathbf{G}}\| \le \sqrt{5}$ and $\|\mathbf{T}_\mathbf{G}\| \le \max\{\sqrt{5}, \sqrt{k}\}$.

Note that we can also define the gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ for any $m \geq nk$, with $k = \lceil \log q \rceil$ by first computing $\mathbf{K} := \mathbf{I}_n \otimes (1, 2, \cdots, 2^{k-1})$ and then adding more $(m - nk)$ zero columns to $\mathbf{K}$ (after $\mathbf{K}$'s last column).

Accompanying $\mathbf{G}$ is a polynomial–time algorithm, named $\mathbf{G}^{-1}$, that maps any matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times t}$ to a matrix $\mathbf{G}^{-1}(\mathbf{A}) \in \{0,1\}^{m\times t}$ subject to $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$. More formally,

Lemma 2.5.2. *Let $n, q, m \in \mathbb{Z}^+$ and $m \geq n\lceil \log q \rceil$. Let $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ be the gadget matrix defined as above. Then, for any $t \in \mathbb{N}$, there is a deterministic polynomial–time algorithm $\mathbf{G}^{-1} : \mathbb{Z}_q^{n\times t} \to \{0,1\}^{m\times t}$ that maps any matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times t}$ to $\mathbf{B} \in \{0,1\}^{m\times t}$ subject to $\mathbf{G} \cdot \mathbf{B} = \mathbf{A}$.*

Lemma 2.5.3 below gives the sup norm of an output of $\mathbf{G}^{-1}$.

Lemma 2.5.3 ([BGG+14, Claim 2.3])**.** *Given a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times t}$ and $\mathbf{R} = \mathbf{G}^{-1}(\mathbf{A})$. Then, $s_1(\mathbf{R}) \leq t$ and $s_1(\mathbf{R}^\top) \leq t$.*

Throughout the thesis, namely in Chapters 4–6, the notation $\mathbf{G}$ is exactly the gadget matrix mentioned here unless otherwise stated.

2.5.3 Lattice Trapdoors

Lattice trapdoors are one of powerful tools in designing many advanced cryptosystems (e.g., IBE, HIBE). The theory of lattice trapdoors has a long line of development, e.g., [Ajt96, GPV08, AP09, CHKP10, ABB10, MP12, LVV19] and so on. In this section, we classify lattice trapdoors into three classes: GPV08 trapdoor [GPV08], MP12 trapdoor [MP12] and LVV19 trapdoor [LVV19]. Regarding their usage, the GPV08 trapdoor will be employed in Chapter 3 and Chapter 5. We will take the MP12 trapdoor and the LVV19 trapdoor into account in Chapter 4. In Chapter 6, we will merge these trapdoors into a common framework suitable for the proposed construction.

GPV08 Trapdoor. The GPV08 trapdoor uses short bases of hard random lattices as trapdoors. The idea has initially been from the seminal work by [Ajt96] and systematically considered by Gentry *et al.* [GPV08] followed up by Cash *et al.* [CHKP10] and Agrawal *et al.* [ABB10].

Definition 2.5.1 (GPV08 Trapdoor)**.** *Let $\mathbf{A}$ be any random matrix. A matrix $\mathbf{T}_\mathbf{A}$ is called a σ–trapdoor for $\Lambda_q^\perp(\mathbf{A})$ (or for $\mathbf{A}$) if $\mathbf{T}_\mathbf{A}$ is a basis for the corresponding lattice $\Lambda_q^\perp(\mathbf{A})$ (that is, $\mathbf{A} \cdot \mathbf{T}_\mathbf{A} = \mathbf{0} \pmod q$) and $\|\widetilde{\mathbf{T}_\mathbf{A}}\| \leq \sigma$.*

The following lemma covers algorithms helpful to generate a random lattice together with a (short basis) trapdoor, extend and even randomise a trapdoor. We adapt it from [BGG+14, Lemmas 2.1–2.2].

Lemma 2.5.4 ([AP09, GPV08, ABB10, CHKP10, BGG+14])**.** *Let $n, m, q \in \mathbb{Z}^+$ and q be prime.*

1. $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$. TrapGen *is a trapdoor generation algorithm. It is probabilistic polynomial–time. Its inputs are $n > 0, q > 0$ and $m = \Theta(n \log q)$. Its output is a pair $(\mathbf{A}, \mathbf{T_A}) \in \mathbb{Z}_q^{n \times m} \times \mathbb{Z}^{m \times m}$, where $\mathbf{T_A}$ is a $O(\sqrt{n \log q})$–trapdoor for $\Lambda_q^\perp(\mathbf{A})$ such that $\mathbf{A}$ is negligibly far from uniform.*

2. $\mathbf{T_D} := \mathsf{ExtBasisRight}(\mathbf{D} := [\mathbf{A}|\mathbf{AS} + \mathbf{B}], \mathbf{T_B})$. ExtBasisRight *is a right trapdoor extension algorithm. It is deterministic polynomial–time. Its inputs are a matrix $(\mathbf{D} := [\mathbf{A}|\mathbf{AS} + \mathbf{B}]$, (where $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{S} \in \mathbb{Z}^{m \times m}$) and a trapdoor $\mathbf{T_B}$ for $\Lambda_q^\perp(\mathbf{B})$. Its outputs is a $\|\widetilde{\mathbf{T_B}}\| \cdot (1 + s_1(\mathbf{S}))$–trapdoor $\mathbf{T_D}$ for $\Lambda_q^\perp(\mathbf{D})$.*

3. $\mathbf{T_E} := \mathsf{ExtBasisLeft}(\mathbf{E} := [\mathbf{A}|\mathbf{B}], \mathbf{T_A})$. ExtBasisLeft *is a left trapdoor extension algorithm. It is deterministic polynomial–time. Its inputs are a matrix $\mathbf{E} := [\mathbf{A}|\mathbf{B}]$ (where $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$) and a trapdoor $\mathbf{T_A}$ for $\Lambda_q^\perp(\mathbf{A})$. Its output is a $\|\widetilde{\mathbf{T_A}}\|$–trapdoor $\mathbf{T_E}$ for $\Lambda_q^\perp(\mathbf{E})$.*

4. $\mathbf{R} \leftarrow \mathsf{SampleD}(\mathbf{A}, \mathbf{T_A}, \mathbf{U}, \sigma)$. SampleD *is a sampling algorithm. It is probabilistic polynomial–time. Its inputs are a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a trapdoor $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^\perp(\mathbf{A})$, a matrix $\mathbf{U} \in \mathbb{Z}_q^{n \times k}$ and a $\sigma \geq \|\widetilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log m}) \in \mathbb{R}^+$. Its output is a short matrix $\mathbf{R} \in \mathbb{Z}_q^{m \times k}$ whose distribution is statistically close to $D_{\Lambda_q^{\mathbf{U}}(\mathbf{A}), \sigma}$. Furthermore, $s_1(\mathbf{R}^\top) \leq \sigma\sqrt{mk}$, $s_1(\mathbf{R}) \leq \sigma\sqrt{mk}$.*

5. $\mathbf{T'_A} \leftarrow \mathsf{RandBasis}(\mathbf{A}, \mathbf{T_A}, \sigma)$. RandBasis *is a trapdoor randomising algorithm. It is probabilistic polynomial–time. Its input are a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, a trapdoor $\mathbf{T_A} \in \mathbb{Z}^{m \times m}$ for $\Lambda_q^\perp(\mathbf{A})$, and a $\sigma = \|\widetilde{\mathbf{T_A}}\| \cdot \omega(\sqrt{\log m}) \in \mathbb{R}^+$. Its output is a new basis $\mathbf{T'_A}$ for $\Lambda_q^\perp(\mathbf{A})$ whose distribution is statistically close to $(D_{\Lambda_q^\perp(\mathbf{A}), \sigma})^m$. Furthermore, $\|\widetilde{\mathbf{T'_A}}\| \leq \sigma\sqrt{m}$.*

MP12 Trapdoor (or G–Trapdoor). Micciancio and Peikert [MP12] have improved the GPV08 trapdoor by introducing a new notion of lattice trapdoors making use of primitive matrices.

Definition 2.5.2 (Primitive Matrices, [MP12, Section 4])**.** *A matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ is called primitive if $\mathbf{G} \cdot \mathbb{Z}^m = \mathbb{Z}_q^n$.*

From now on, we only focus on $\mathbf{G}$ that is primitive. More precisely, $\mathbf{G}$ can be seen to be the gadget matrix presented in Section 2.5.2 which is also a primitive matrix.

Definition 2.5.3 (**G**–Trapdoor, [MP12, Definition 5.2])**.** *Let* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ *and* $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$ *be matrices with* $m \geq w \geq n$*. A matrix* $\mathbf{R} \in \mathbb{Z}^{(m-w)\times w}$ *is called* **G***–trapdoor for* **A** *with tag* **H** *(which is an invertible matrix in* $\mathbb{Z}_q^{n\times n}$*) if*

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{R} \\ \mathbf{I}_w \end{bmatrix} = \mathbf{HG}.$$

The quality of **R** *is measured by* $s_1(\mathbf{R})$ *(i.e., sup norm of* **R***).*

Given **G** is primitive, then **A** is also primitive if **A** admits a **G**–trapdoor. We state a lemma that collects necessary algorithms for **G**–trapdoors.

Lemma 2.5.5 ([MP12])**.** *Let* $n,\ q \in \mathbb{Z}^+,\ q \geq 2,\ w = n\lceil \log q \rceil,\ m \geq w \geq n$*. Let* $\mathbf{G} \in \mathbb{Z}_q^{n\times w}$ *be a primitive matrix or even the gadget matrix. For* **G***–trapdoor, there exist the following algorithms:*

1. $(\mathbf{A}, \mathbf{R}) \leftarrow \mathsf{MPGenTrap}(1^n, 1^m, q)$. MPGenTrap *is a trapdoor generation algorithm. It is probabilistic polynomial–time. Its inputs are integers* n, q *and* m*. Its outputs is a pair* $(\mathbf{A}, \mathbf{R}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}^{(m-w)\times w}$*, where* **R** *is a* **G***–trapdoor for* **A** *such that* **A** *is negligibly far from uniform over* $\mathbb{Z}_q^{n\times m}$*.*

2. $\mathbf{x} \leftarrow \mathsf{MPSamTrap}(\mathbf{R}, \mathbf{A}, \mathbf{u}, \sigma)$[a]. MPSamTrap *is a sampling algorithm. It is probabilistic polynomial–time. Its inputs are a matrix* $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ *and its* **G***–trapdoor matrix* $\mathbf{R} \in \mathbb{Z}_q^{(m-w)\times w}$*, a Gaussian parameter* $\sigma \geq \omega(\sqrt{\log n}) \cdot \sqrt{7(s_1(\mathbf{R})^2 + 1)}$ [b]*, and any* $\mathbf{u} \in \mathbb{Z}_q^n$*. Its output is a vector* **x** *whose distribution is negligibly far from* $D_{\Lambda_q^{\mathbf{u}}(\mathbf{A}),\sigma}$*.*

3. $\mathbf{R}' \leftarrow \mathsf{MPDelTrap}([\mathbf{A}|\mathbf{A}_1], \mathbf{R}, \mathbf{H}', \sigma')$. MPDelTrap *is a trapdoor delegation algorithm. It is probabilistic polynomial–time. Its inputs are a matrix* $\mathbf{A}' := [\mathbf{A}|\mathbf{A}_1] \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{n\times w}$*, a* **G***–trapdoor* $\mathbf{R} \in \mathbb{Z}^{(m-w)\times w}$ *for* **A***, and an invertible* $\mathbf{H}' \in \mathbb{Z}_q^{n\times n}$*, a Gaussian parameter* $\sigma' \geq \eta_\epsilon(\Lambda_q^\perp(\mathbf{A}))$ [c]*. Its output is a* **G***–trapdoor* $\mathbf{R}' \in \mathbb{Z}^{m\times w}$ *for* $\mathbf{A}'$ *such that* $s_1(\mathbf{R}') \geq \sigma' \cdot O(\sqrt{m} + \sqrt{w})$ *except with negligible probability.*

LVV19 trapdoor. The LVV19 trapdoor is a trapdoor mechanism for polynomials involved in the DMPLWE problem. The mechanism was developed in [LVV19] and is a modification of the MP12 trapdoor. To formally introduce the LVV19 trapdoor, now we will define Toeplitz matrix and present its basic properties.

[a] In [MP12], the algorithm's name is SampleD. We change its name into MPSamTrap to distinguish it from SampleD in Lemma 2.5.4 which belongs to GPV08 trapdoor.

[b] See [MP12, Section 5.4, Quality analysis]

[c] By [MP12, Lemma 5.3], the basis $\mathbf{S}_\mathbf{A}$ of $\Lambda_q^\perp(\mathbf{A})$ satisfies $\|\widetilde{\mathbf{S}_\mathbf{A}}\| \leq s_1(\left[\begin{smallmatrix} \mathbf{I} & \mathbf{R} \\ \mathbf{0} & \mathbf{I} \end{smallmatrix}\right]) \cdot \|\widetilde{\mathbf{S}}\| = (s_1(\mathbf{R}) + 1) \cdot |\widetilde{\mathbf{S}}\|$ where $\mathbf{S}_\mathbf{A}$ is orthogonalised in suitable order, where $\|\widetilde{\mathbf{S}}\| = \sqrt{5}$. By Lemma 2.3.1, $\eta_\epsilon(\Lambda_q^\perp(\mathbf{A})) \leq \|\widetilde{\mathbf{S}_\mathbf{A}}\| \cdot \sqrt{\frac{\ln(2n(1+1/\epsilon))}{\pi}} \approx \|\widetilde{\mathbf{S}_\mathbf{A}}\| \cdot \omega(\sqrt{\log n})$.

Definition 2.5.4 (Toeplitz Matrix)**.** *Suppose that R is a ring. Let $d,n \in \mathbb{Z}^+$. For any $v \in R^{<n}[x]$, the Toeplitz matrix $\mathsf{Tp}^{n,d}(v) \in R^{(n+d-1)\times d}$ is a matrix whose column j comprises the coefficient vector of $x^{j-1} \cdot v$. These coefficients are put from the top going down with an increasing degree of x. The zeros 0's will be inserted, if any.*

For example, let $u(x) = 9 - 3x^2 + 7x^3 \in \mathbb{Z}[x]$ and $d = 3, n = 4$. Then,

$$\mathsf{Tp}^{4,3}(u) = \begin{bmatrix} 9 & 0 & 0 \\ 0 & 9 & 0 \\ -3 & 0 & 9 \\ 7 & -3 & 0 \\ 0 & 7 & -3 \\ 0 & 0 & 7 \end{bmatrix} \in R^{6\times 3}.$$

The Toeplitz matrix enjoys the properties given in Lemmas 2.5.6–2.5.8 below.

Lemma 2.5.6. *Let $v, w \in R^{<n}[x]$. Then,*

$$\mathsf{Tp}^{n,d}(v \pm w) = \mathsf{Tp}^{n,d}(v) \pm \mathsf{Tp}^{n,d}(w).$$

Lemma 2.5.7. *Let $v \in R^{<n}[x]$. It holds that*

$$\mathsf{Tp}^{n,d}(v) = [\mathsf{Tp}^{n+d-1,1}(v) | \mathsf{Tp}^{n+d-1,1}(x \cdot v) | \cdots | \mathsf{Tp}^{n+d-1,1}(x^{d-1} \cdot v)].$$

Lemma 2.5.8 ([LVV19, Lemma 7])**.** *For $k,n,d \in \mathbb{Z}^+$ and $v \in R^{<k}[x]$, if $w \in R^{<n}[x]$, then $\mathsf{Tp}^{k,n+d-1}(v) \cdot \mathsf{Tp}^{n,d}(w) = \mathsf{Tp}^{k+n-1,d}(v \cdot w)$.*

Definition 2.5.5 (Toeplitz Representation)**.** *Let $\overline{\mathbf{a}} = (a_1, \cdots, a_n)$ be an n–family of polynomials for any integer $n > 0$. A matrix $\mathbf{A}$ is called a Topelitz representation for $\overline{\mathbf{a}}$ if it is a concatenation of all Toeplitz matrices according to all polynomials a_1's in $\overline{\mathbf{a}}$..*

We are now going to give the definition for LVV19 trapdoor. The LVV19 trapdoor follows the MP12 trapdoor through Toeplitz representations. First, let $\tau = \lceil \log q \rceil$, $\gamma = (n + 2d - 2)/d$ and $\beta = \lceil \log n/2 \rceil$. Consider the family $\overline{\mathbf{g}} = (g_1, \cdots, g_{\gamma\tau}) \in (\mathbb{Z}_q^{n+d-1}[x])^{\gamma\tau}$, where

$$g_j = 2^\eta x^{d\zeta} \in \mathbb{Z}_q^{n+d-1}[x], \tag{2.3}$$

for $j = \zeta\tau + \eta + 1$ with $\eta \in \{0, \cdots, \tau - 1\}$, $\zeta \in \{0, \cdots, \gamma - 1\}$. Define the Toeplitz representation for $\overline{\mathbf{g}}$ as

$$\mathbf{G} = [\mathsf{Tp}^{n+d-1,d}(g_1) | \cdots | \mathsf{Tp}^{n+d-1,d}(g_{\gamma\tau})] \in \mathbb{Z}_q^{d\gamma \times d\gamma\tau}. \tag{2.4}$$

Definition 2.5.6 (LVV19 Trapdoor, formally stated via [LVV19])**.** *Let t_1, n_1, t_2, $n_2 \in \mathbb{Z}^+$. Let $\overline{\mathbf{a}} = (a_1, \cdots, a_{t_1}) \in (\mathbb{Z}_q^{<n_1}[x])^{t_1}$ and $\overline{\mathbf{g}} = (g_1, \cdots, g_{t_2}) \in (\mathbb{Z}_q^{n_2}[x])^{t_2}$. The family of*

polynomials td *is called a* $\overline{\mathbf{g}}$*–trapdoor for* $\overline{\mathbf{a}}$ *if there exist a Toeplitz representation* $\mathbf{A}$ *for* $\overline{\mathbf{a}}$*, a Toeplitz representation* $\mathbf{G}$ *for* $\overline{\mathbf{g}}$*, a Toeplitz representation* $\mathbf{T}$ *for* td *(with appropriate dimensions) such that*

$$\mathbf{A} \cdot \begin{bmatrix} \mathbf{T} \\ \mathbf{I} \end{bmatrix} = \mathbf{G}.$$

We state two basic algorithms with respect to the LVV19 trapdoor. The first algorithm $\mathsf{LVVGenTrap}(1^n)$ gives a way to generate a family of polynomials $\overline{\mathbf{a}}_\epsilon$ together with a $\overline{\mathbf{g}}$–trapdoor td_ϵ; whilst the second algorithm $\mathsf{LVVSamTrap}$ uses $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon)$ to sample a family of polynomials $\overline{\mathbf{r}}$ from a distribution close to a discrete Gaussian over $\mathbb{Z}$ under condition that $\sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i = u$, given a polynomial u of appropriate degree.

Lemma 2.5.9 ([LVV19])**.** *Let* $n, d, \gamma, t, \beta, \tau \in \mathbb{Z}^+$ *be such that* $q = \mathsf{poly}(n)$*,* $d \leq n$*,* $dt/n = \Omega(\log n)$*,* $d\gamma = n + 2d - 2$*,* $\tau := \lceil \log_2 q \rceil$*,* $\beta := \lceil \frac{\log_2(n)}{2} \rceil \ll q/2$*. For* $\overline{\mathbf{g}}$*–trapdoor, there exist the following algorithms:*

1. $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon) \leftarrow \mathsf{LVVGenTrap}(1^n)$*.* $\mathsf{LVVGenTrap}$ *is a trapdoor generation algorithm. It is probabilistic polynomial–time. Its input is a security parameter* n*. Its output is a pair* $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon)$*, where* td_ϵ *is a* $\overline{\mathbf{g}}$*–trapdoor for* $\overline{\mathbf{a}}_\epsilon$*. Furthermore, the distribution of* $\overline{\mathbf{a}}_\epsilon$ *is negligibly far from uniform over* $(\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$*.*

2. $\overline{\mathbf{r}} \leftarrow \mathsf{LVVSamTrap}(\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t+\gamma\tau}), \mathsf{td}_\epsilon = (\overline{\mathbf{w}}^{(1)}, \cdots, \overline{\mathbf{w}}^{(\gamma\tau)}), u, \sigma))$*.* $\mathsf{LVVSamTrap}$ *is a sampling algorithm. It is probabilistic polynomial–time. Its inputs are* $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t+\tau\gamma}) \in (\mathbb{Z}_q^{<n}[x])^{t+\tau\gamma}$ *and its* $\overline{\mathbf{g}}$*–trapdoor* td_ϵ*, a Gaussian parameter* $\sigma \geq \omega(\sqrt{\log(d\gamma)}) \cdot \sqrt{7(s_1^2(\mathbf{T}_\epsilon) + 1)}$*, and any* $u \in \mathbb{Z}_q^{<n+2d-2}[x]$*. Its output is a polynomial family* $\overline{\mathbf{r}} := (r_1, \cdots, r_{t+\gamma\tau})$ *whose distribution is negligibly far from* $(D_{\mathbb{Z}^{2d-1},\sigma}[x])^t \times (D_{\mathbb{Z}^d,\sigma}[x])^{\gamma\tau}$*. Moreover,*

$$\langle \overline{\mathbf{a}}_\epsilon, \overline{\mathbf{r}} \rangle := \sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i = u.$$

Here, matrix $\mathbf{T}_\epsilon$ *is defined as in Equation* (2.6)*.*

Below, we describe in detail $\mathsf{LVVGenTrap}$ and $\mathsf{LVVSamTrap}$ for reference later in Chapter 4.

- $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon) \leftarrow \mathsf{LVVGenTrap}(1^n)$. Do the following:
  1. Sample $\overline{\mathbf{a}}' = (a_1, \cdots, a_t) \xleftarrow{\$} (\mathbb{Z}_q^{<n}[x])^t$, and for all $j \in [\gamma\tau]$, sample $\overline{\mathbf{w}}^{(j)} = (w_1^{(j)}, \cdots, w_t^{(j)}) \leftarrow (\Gamma^d[x])^t$, where $\Gamma := \mathcal{U}(\{-\beta, \cdots, \beta\})$.
  2. For all $j \in [\gamma\tau]$, define $u_j = \langle \overline{\mathbf{a}}', \overline{\mathbf{w}}^{(j)} \rangle$ and $a_{t+j} = g_j - u_j$.
  3. Output $\overline{\mathbf{a}}_\epsilon := (a_1, \cdots, a_t, a_{t+1}, \cdots, a_{t+\gamma\tau})$ with its corresponding trapdoor $\mathsf{td}_\epsilon := (\overline{\mathbf{w}}^{(1)}, \cdots, \overline{\mathbf{w}}^{(\gamma\tau)})$.

- $\overline{\mathbf{r}} \leftarrow \mathsf{LVVSamTrap}(\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t+\gamma\tau}), \mathsf{td}_\epsilon = (\overline{\mathbf{w}}^{(1)}, \cdots, \overline{\mathbf{w}}^{(\gamma\tau)}), u, \sigma)$. Do the following:

  1. First, construct (implicitly) Toeplitz representations $\mathbf{A}'$, $\mathbf{A}_\epsilon, \mathbf{T}_\epsilon, \mathbf{G}$ for $\overline{\mathbf{a}}'$, $\overline{\mathbf{a}}_\epsilon$, td_ϵ, $\overline{\mathbf{g}}$, respectively as below:

$$\begin{aligned} \mathbf{A}' &= [\mathsf{Tp}^{n,2d-1}(a_1)|\cdots|\mathsf{Tp}^{n,2d-1}(a_t)], \\ \mathbf{A}_\epsilon :&= [\mathsf{Tp}^{n,2d-1}(a_1)|\cdots|\mathsf{Tp}^{n,2d-1}(a_t)|\mathsf{Tp}^{n+d-1,d}(a_{t+1})| \\ &\quad \cdots|\mathsf{Tp}^{n+d-1,d}(a_{t+\gamma\tau})], \end{aligned} \tag{2.5}$$

$$\mathbf{T}_\epsilon := \begin{bmatrix} \mathsf{Tp}^{d,d}(w_1^{(1)}) & \cdots & \mathsf{Tp}^{d,d}(w_1^{(\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{d,d}(w_t^{(1)}) & \cdots & \mathsf{Tp}^{d,d}(w_t^{(\gamma\tau)}) \end{bmatrix} \in \mathbb{Z}_q^{(2d-1)t \times d\gamma\tau}, \tag{2.6}$$

  and

$$\mathbf{I}_{d\gamma\tau} = \begin{bmatrix} \mathsf{Tp}^{1,d}(1) & \cdots & \\ \cdots & & \cdots \\ & \cdots & \mathsf{Tp}^{1,d}(1) \end{bmatrix} \in \mathbb{Z}_q^{d\gamma\tau \times d\gamma\tau}. \tag{2.7}$$

  Here, $\mathbf{I}_{d\gamma\tau}$ is the unit matrix of dimension $d\gamma\tau$. Then $\mathbf{A}_\epsilon = [\mathbf{A}'|\mathbf{G} - \mathbf{A}'\mathbf{T}_\epsilon]$ and hence $\mathbf{A}_\epsilon \cdot \left[\begin{smallmatrix} \mathbf{T}_\epsilon \\ \mathbf{I}_{d\gamma\tau} \end{smallmatrix}\right] = \mathbf{G}$.

  2. The Toeplitz matrix of u is $\mathbf{u} = \mathsf{Tp}^{n+2d-2,1}(u) \in \mathbb{Z}_q^{n+2d-2}$.

  3. Sample $\mathbf{r} \in \mathbb{Z}^{(2d-1)t+d\gamma\tau}$ from $D_{\Lambda_q^{\mathbf{u}}(\mathbf{A}),\sigma}$ by calling the algorithm $\mathsf{MPSamTrap}$ mentioned in Lemma 2.5.5 using the matrix $\mathbf{T}_\epsilon$, with noting that by Lemma 2.5.5

$$\sigma \geq \omega(\sqrt{\log(d\gamma)}) \cdot \sqrt{7(s_1(\mathbf{T}_\epsilon)^2 + 1)}, \tag{2.8}$$

  and by Lemma 2.1.2,

$$s_1(\mathbf{T}_\epsilon) \leq \sqrt{(2d-1) \cdot t \cdot (d\gamma\tau)} \cdot \beta. \tag{2.9}$$

  4. Parse $\mathbf{r}$ as $[\mathbf{r}_1^\top|\cdots|\mathbf{r}_{t+\gamma\tau}^\top]^\top$, and rewrite it as a Toeplitz representation for polynomials $r_1, \cdots, r_{t+\gamma\tau}$, where $\mathbf{r}_j = \mathsf{Tp}^{2d-1,1}(r_j)$, $\deg(r_j) < 2d-1$, $\forall j \in [t]$, $\mathbf{r}_j = \mathsf{Tp}^{d,1}(r_j)$, $\deg(r_{t+j}) < d$, $\forall j \in t+1, \cdots, t+\gamma\tau$.

  5. Output $\overline{\mathbf{r}} := (r_1, \cdots, r_{t+\gamma\tau})$. Note that $\langle \overline{\mathbf{a}}_\epsilon, \overline{\mathbf{r}} \rangle = \sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i = u$.

More details can be found in [LVV19, Section 5].

2.5.4 Lattice Homomorphic Evaluations

Lattice homomorphic evaluations [BGG+14] exploit some techniques developed for fully homomorphic encryption (FHE) regarding the gadget matrix $\mathbf{G}$ and the accompanied algorithm $\mathbf{G}^{-1}$. For what FHE is, please refer to Section 1.1. Here, we will make the term "homomorphic evaluations" clear. Informally speaking, the idea of lattice homomorphic evaluations is to homomorphically evaluate a function f through a list of "ciphertexts" of LWE form $\{\mathbf{c}_i := (x_i\mathbf{G} + \mathbf{B}_i)^\top \mathbf{s} + \mathbf{e}_i\}_{i\in[d]}$ to obtain an evaluated ciphertext of LWE form $\mathbf{c}_f := (f(x_1, \cdots, x_d) \cdot \mathbf{G} + \mathbf{B}_f)^\top \mathbf{s} + \mathbf{e}_f$. Here $\mathbf{B}_i$'s are uniformly random matrices. To do that, we need two algorithms called $\mathsf{Eval}_{\mathsf{pk}}$, $\mathsf{Eval}_{\mathsf{ct}}$ that work as follows:

$$\{\mathbf{B}_i\}_{i\in[d]} \xrightarrow{\mathsf{Eval}_{\mathsf{pk}},f} \mathbf{B}_f$$

$$\{\mathbf{c}_i\}_{i\in[d]} \xrightarrow{\mathsf{Eval}_{\mathsf{ct}},f} \mathbf{c}_f$$

Moreover, in security proof reductions we may replace uniformly random matrices $\mathbf{B}_i$'s with one computed as $\mathbf{AS}_i - \mathbf{x}^*\mathbf{G} = \mathbf{B}_i$ for some uniformly random matrix $\mathbf{A}$, short matrices $\mathbf{S}_i$'s and some vector $\mathbf{x}^*$. Then, we also need algorithm $\mathsf{Eval}_{\mathsf{sim}}$ that

$$\{\mathbf{AS}_i - \mathbf{x}^*\mathbf{G}\}_{i\in[d]} \xrightarrow{\mathsf{Eval}_{\mathsf{sim}},f} \mathbf{AS}_f - f(\mathbf{x}^*)\mathbf{G} = \mathbf{B}_f$$

The algorithms $\mathsf{Eval}_{\mathsf{pk}}$, $\mathsf{Eval}_{\mathsf{ct}}$ and $\mathsf{Eval}_{\mathsf{sim}}$ should work well together.

We will formally present the lattice homomorphic evaluations below. Let $n, q, d, m \in \mathbb{Z}^+$ such that $m = \Theta(n \log q)$. Let $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$ be the gadget matrix defined in Section 2.5.2. For $i \in [d]$, let $x_i \in \mathbb{Z}_q$, $x_i^* \in \mathbb{Z}_q$, $\mathbf{B}_i \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\mathbf{S}_i \in \mathbb{Z}_q^{m\times m}$ and $\mathbf{e}_i \in \mathbb{Z}_q^m$ with $\|\mathbf{e}_i\| \le \delta$. For $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, and for each $i \in [d]$, define

$$\mathbf{c}_i := (x_i\mathbf{G} + \mathbf{B}_i)^\top \mathbf{s} + \mathbf{e}_i \in \mathbb{Z}_q^m.$$

Let $\mathcal{F} = \{f : (\mathbb{Z}_q)^d \to \mathbb{Z}_q\}$ denote a family whose each function can be computed by depth τ, polynomial–size arithmetic circuits $(C_\lambda)_{\lambda\in\mathbb{N}}$.

Lemma 2.5.10 ([BGG+14, Section 4])**.** *There exist deterministic polynomial–time algorithms* $\mathsf{Eval}_{\mathsf{pk}}$, $\mathsf{Eval}_{\mathsf{ct}}$, $\mathsf{Eval}_{\mathsf{sim}}$ *and a constant* $\beta_{\mathcal{F}}$ *associated* $\mathcal{F}$ *such that the following properties hold.*

1. *If* $\mathbf{B}_f \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f \in \mathcal{F}, (\mathbf{B}_i)_{i=1}^d)$*, then* $\mathbf{B}_f \in \mathbb{Z}_q^{n\times m}$.
2. *If* $\mathbf{c}_f \leftarrow \mathsf{Eval}_{\mathsf{ct}}(f \in \mathcal{F}, ((x_i, \mathbf{B}_i, \mathbf{c}_i))_{i=1}^d)$*, and* $\mathbf{B}_f \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f, (\mathbf{B}_i)_{i=1}^d)$*, then*

$$\mathbf{c}_f := (f(x_1, \cdots, x_d) \cdot \mathbf{G} + \boldsymbol{B}_f)^\top \mathbf{s} + \mathbf{e}_f,$$

where $\|\mathbf{e}_f\| \leq \beta_{\mathcal{F}} \cdot \|\mathbf{e}_i\| < \beta_{\mathcal{F}} \cdot \delta$.

3. *If* $\mathbf{S}_f \leftarrow \mathsf{Eval}_{\mathsf{sim}}(f \in \mathcal{F}, ((x_i^*, \mathbf{S}_i))_{i=1}^d, \mathbf{A})$ *and* $\mathbf{B}_f \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f, (\mathbf{AS}_i - x_i^*\mathbf{G})_{i=1}^d)$ *then*

$$\mathbf{AS}_f - f(x_1^*, \cdots, x_d^*) \cdot \mathbf{G} = \mathbf{B}_f.$$

In particular, if $\mathbf{S}_1, \cdots, \mathbf{S}_d \xleftarrow{\$} \{-1, 1\}^{m \times m}$*, then* $s_1(\mathbf{S}_f) < \beta_{\mathcal{F}} \cdot \max_{i \in [d]} s_1(\mathbf{R}_i) \leq \beta_{\mathcal{F}} \cdot 20\sqrt{m}$ [d] *with all but negligible probability for all* $f \in \mathcal{F}$.

Lemma 2.5.11 gives an estimation of the constant $\beta_{\mathcal{F}}$ for the family $\mathcal{F}$ defined above.

Lemma 2.5.11 ([BGG+14, Lemma 4.7])**.** *For any* $f \in \mathcal{F}$*, Suppose that all of the input values to the multiplication gates (possibly except one) of circuits* C_λ *are bounded by some positive integer* $p \ll q$*. Then,*

$$\beta_{\mathcal{F}} = \left(\frac{p^d - 1}{p - 1} \cdot m\right)^\tau.$$

2.6 Security Proof Models

Throughout this thesis, we use the game–playing method to give security proof for cryptosystems. Following the approach, a security property is abstracted to a security game. This game is played by an adversary and a challenger. The game tries to mimic the natural behaviours of a realistic potential adversary against the cryptosystem. For example, the real–world adversary can try to "collect as much information about a system as possible" before "attacking". In the game, "collecting as much information about a system as possible" is modelled as being able to query some oracles to get back some desired information; while "attacking" is demonstrated via the action "challenging", when the adversary releases the target he wants to challenge. Winning or losing the game depends on what the adversary can reach after "challenging". Roughly speaking, if the oracles are working via their designed functionality, then we say that the game is in the standard model (SDM). For example, we consider a hash function as an oracle. Suppose further that the adversary can query to the oracle. Then, in SDM, the challenger replies to the query with the actual output produced by the hash function.

By contrast, due to either efficacy reasons or the difficulty in seeking an SDM proof, one can instead design a cryptosystem that is secure in the random oracle model (ROM) [BR93]. In ROM, one or many oracles (e.g., hash functions) can be replaced by random oracles, which return truly random values. Then, instead of using the oracle to produce the responses, the challenger instead can sample these responses uniformly at random over an appropriate set/space.

[d] This is due to Lemma 2.1.3

In the thesis, we will take both models into account.

2.7 Cryptographic Primitives

This section reviews some cryptographic primitives of interest in the upcoming chapters.

2.7.1 Digital Signature

Digital signatures can guarantee the authenticity of messages/documents. In a public–key digital signature scheme, a party (i.e., signer) uses a secret key to sign on a message/document to get a signature which can be verified together with the message/document just using the corresponding public key by any other party (i.e., verifier).

Syntax. A digital signature (DS) Π_{DS} consists of three polynomial–time algorithms $\mathsf{DS.Setup}$, $\mathsf{DS.Sign}$, and $\mathsf{DS.Verify}$. The algorithms work as follows:

- $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$. $\mathsf{DS.Setup}$ is the setup generation algorithm. It is probabilistic polynomial–time. Its input is a security parameter λ. Its outputs are a set of public parameters pp, a public key pk and a corresponding secret key sk.
- $\perp/\Sigma \leftarrow \mathsf{DS.Sign}(\mathsf{pp}, \mathsf{sk}, \mu)$. $\mathsf{DS.Sign}$ is the signing algorithm. It is probabilistic polynomial–time. Its inputs are a set of public parameters pp, a secret key sk and a message μ. Its output is either a signature Σ if it succeeds or a failure symbol $\perp$ otherwise.
- $1/0 := \mathsf{DS.Verify}(\mathsf{pp}, \mathsf{pk}, \mu, \Sigma)$. $\mathsf{DS.Verify}$ is the verification algorithm. It is deterministic polynomial–time. Its inputs are a set of public parameters pp, a public key pk, a message μ and a signature Σ. Its output is either 1 (if the signature Σ is non–$\perp$ and valid) or 0 (otherwise).

Correctness. The correctness of Π_{DS} is defined as follows. For any security parameter λ, any $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$ and any message μ, it holds that

$$\Pr[\mathsf{DS.Verify}(\mathsf{pp}, \mathsf{pk}, \mu, \mathsf{DS.Sign}(\mathsf{pp}, \mathsf{sk}, \mu)) = 1] = 1 - \mathsf{negl}(\lambda),$$

over the randomness of all algorithms involved.

Security Notions. For a DS, the basic security notion is EUF–aCMA (i.e., *existential unforgeability under adaptive chosen message attacks*). Assuming that any polynomial–time adversary can see signatures produced on some messages of its choice, the EUF–aCMA security guarantees that the adversary cannot produce any valid signature for a new message. A relaxation of EUF–aCMA is the static one (EUF–sCMA) that requires

the adversary specifies, in advance, a list of s queried messages $\mu_1, \cdots, \mu_s$ for some $s \in \mathbb{Z}^+$. Figure 2.3 presents the game for EUF–sCMA. The game for EUF–aCMA is precisely $\mathsf{DS}_{\mathcal{F}}^{\mathsf{EUF-sCMA}}(\lambda)$ but having no Step 1.

GAME $\mathsf{DS}_{\mathcal{F}}^{\mathsf{EUF-sCMA}}(\lambda) \Rightarrow 0/1$:
1. $\mu_1, \cdots, \mu_s \leftarrow \mathcal{F}(\lambda)$;
2. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DS.Setup}(1^\lambda)$;
3. $(\mu^*, \Sigma^*) \leftarrow \mathcal{F}^{\mathsf{SQ}(\mathsf{sk}, \mu_1, \cdots, \mu_s)}(\mathsf{pp}, \mathsf{pk})$;
4. If $\mathsf{DS.Verify}(\mathsf{pp}, \mathsf{pk}, \mu^*, \Sigma^*) = 1$ and $\mu^* \neq \mu_i$ for all $i \in [s]$, return 1. Otherwise, return 0.
Queried Oracles:
• Signing Oracle $\mathsf{SQ}(\mathsf{sk}, \mu_1, \cdots, \mu_s)$: Return $\Sigma_i \leftarrow \mathsf{DS.Sign}(\mathsf{pp}, \mathsf{sk}, \mu_i)$ for $i \in [s]$.

Figure 2.3: Existential unforgeability game for DS.

We formally define the EUF–sCMA security through the game $\mathsf{DS}_{\mathcal{F}}^{\mathsf{EUF-sCMA}}(\lambda)$ (Figure 2.3) and Definition 2.7.1.

Definition 2.7.1 (EUF–sCMA for Digital Signatures)**.** *Define the advantage of $\mathcal{F}$ in the game $\mathsf{DS}_{\mathcal{F}}^{\mathsf{EUF-sCMA}}(\lambda)$ (Figure 2.3) by*

$$\mathsf{Adv}_{\mathcal{F},\mathsf{DS}}^{\mathsf{EUF-sCMA}}(\lambda) := \left|\Pr[\mathsf{DS}_{\mathcal{F}}^{\mathsf{EUF-sCMA}}(\lambda) \Rightarrow 1] - 1/2\right|.$$

We say that Π_{DS} is EUF–sCMA if, for any efficient algorithm $\mathcal{F}$, it holds that

$$\mathsf{Adv}_{\mathcal{F},\mathsf{DS}}^{\mathsf{EUF-sCMA}}(\lambda) \leq \mathsf{negl}(\lambda).$$

A stronger security notion of DS is strong unforgeability against chosen message attacks (SUF–CMA) which additionally requires that the forger cannot produce a valid new signature for any previously queried messages. Specifically, Step 4 of $\mathsf{DS}_{\mathcal{F}}^{\mathsf{SUF-sCMA}}(\lambda)$ should be "If $\mathsf{DS.Verify}(\mathsf{pp}, \mathsf{pk}, \mu^*, \Sigma^*) = 1$ and $(\mu^*, \Sigma^*) \neq (\mu_i, \Sigma_i)$ for all $i \in [s]$, return 1. Otherwise, return 0.".

2.7.2 Blind Signature

Blind Signature. Blind signature (BS) [Cha83] is a kind of digital signature with the ability to make messages invisible to the signer while being signed. We remind its syntax and security notions below.

Syntax. A blind signature scheme is a system Π_{BS} that consists of the polynomial–time algorithms $\mathsf{BS.Setup}$, $\mathsf{BS.Sign}$, $\mathsf{BS.Verify}$ defined as follows:

- $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{BS.Setup}(1^\lambda)$. BS.Setup is the key setup algorithm. It is probabilistic polynomial–time. Here, λ denotes a security parameter. The algorithm outputs a set of system parameters pp, a public key pk and a secret key sk.

- $(\Sigma, \mathcal{V}) \leftarrow \mathsf{BS.Sign}(\mathsf{pp}, \mathsf{sk}, \mathsf{pk}, \mu)$. BS.Sign is the signing algorithm. It is a probabilistic polynomial–time interactive protocol between a user algorithm $\mathcal{U}(\mathsf{pp}, \mathsf{pk}, \mu)$ and a signer algorithm $\mathcal{S}(\mathsf{pp}, \mathsf{sk})$ in such a way that the user will get the signature Σ on message μ, while the signer gets a view $\mathcal{V}$. In the case that the interaction fails, $\Sigma := \perp$ and $\mathcal{V} := \perp$.

- $1/0 := \mathsf{BS.Verify}(\mathsf{pp}, \mathsf{pk}, \mu, \Sigma)$. BS.Verify is the verification algorithm. It is deterministic polynomial–time. Here, pp is a set of system parameters, pk – a public key, μ – a message and Σ – a signature. The algorithm returns either 1 (if the signature is non–$\perp$ and valid) or 0 (otherwise).

Correctness. The scheme Π_{BS} is correct if for any security parameter λ, any (pp, pk, sk) $\leftarrow \mathsf{BS.Setup}(1^\lambda)$, any message μ and any $(\Sigma, \mathcal{V}) \leftarrow \mathsf{BS.Sign}(\mathsf{pp}, \mathsf{sk}, \mathsf{pk}, \mu)$, it holds that

$$\Pr[\mathsf{BS.Verify}(\mathsf{pp}, \mathsf{pk}, \mu, \Sigma) = 1] = 1 - \mathsf{negl}(\lambda),$$

over the randomness of all algorithms.

Security Notions. The basic security requirements for a blind signature are *blindness* and *one–more unforgeability* (OMUF). The blindness guarantees that any message being signed is hidden from the signer. The OMUF security ensures that the number of valid signatures is always not larger than that of successful interaction between the user and the signer.

The blindness is defined in Figure 2.4 and Definition 2.7.2. Whereas the OMUF security is defined in Figure 2.5 and Definition 2.7.3.

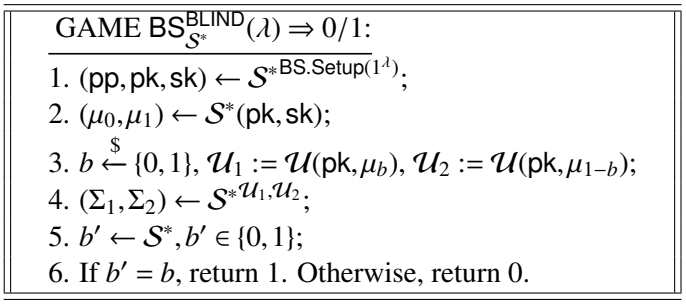

Figure 2.4: Blindness game for BS.

Definition 2.7.2 (Blindness for Blind Signatures)**.** *Define the advantage of $\mathcal{S}^*$ in the blindness game $\mathsf{BS}^{\mathsf{BLIND}}_{\mathcal{S}^*}(\lambda)$ (Figure 2.4) by*

$$\mathsf{Adv}^{\mathsf{BLIND}}_{\mathcal{S}^*,\mathsf{BS}}(\lambda) := \left|\Pr[\mathsf{BS}^{\mathsf{BLIND}}_{\mathcal{S}^*}(\lambda) \Rightarrow 1] - 1/2\right|.$$

We say that Π_{BS} is blind if, for any efficient algorithm $\mathcal{S}^*$, it holds that

$$\mathsf{Adv}^{\mathsf{BLIND}}_{\mathcal{S}^*,\mathsf{BS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

GAME $\mathsf{BS}^{\mathsf{OMUF}}_{\mathcal{U}^*}(\lambda) \Rightarrow 0/1$:

1. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{BS.Setup}(1^\lambda)$;
2. $(\mu_1, \Sigma_1), \cdots, (\mu_s, \Sigma_s) \leftarrow \mathcal{U}^{*\mathcal{S}(\mathsf{sk})}(\mathsf{pk})$ after at most $s-1$ successful signing interactions with $\mathcal{S}(\mathsf{sk})$;
3. If $\mathsf{BS.Verify}(\mathsf{pp}, \mu_i, \Sigma_i, \mathsf{pk}) = 1, \forall i \in [s]$ and $\mu_i \neq \mu_j$ for all $i, j \in [s]$, $i \neq j$, return 1. Otherwise, return 0.

Figure 2.5: One–more unforgeability game for blind signature.

Definition 2.7.3 (One–more Unforgeability for Blind Signatures)**.** *Define the advantage of $\mathcal{U}^*$ in the game $\mathsf{BS}^{\mathsf{OMUF}}_{\mathcal{U}^*}(\lambda)$ (Figure 2.5) by*

$$\mathsf{Adv}^{\mathsf{OMUF}}_{\mathcal{U}^*,\mathsf{BS}}(\lambda) := \left|\Pr[\mathsf{BS}^{\mathsf{OMUF}}_{\mathcal{U}^*}(\lambda) \Rightarrow 1]\right|.$$

We say that Π_{BS} is OMUF secure if, for any efficient algorithm $\mathcal{U}^*$, it holds that

$$\mathsf{Adv}^{\mathsf{OMUF}}_{\mathcal{U}^*,\mathsf{BS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

Forward–secure Blind Signatures. We will focus on a variant of blind signature that we call *forward–secure blind signatures* (FSBS). For the syntax and security notions of FSBS, we follow [DCK03], which is in turn adapted from [BM99].

Syntax. A FSBS scheme Π_{FSBS} composes of polynomial–time algorithms FSBS.Setup, FSBS.KeyUp, FSBS.Sign, and FSBS.Verify, which are described as follows:

- $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_\epsilon) \leftarrow \mathsf{FSBS.Setup}(1^\lambda)$. FSBS.Setup is the key setup algorithm. It is probabilistic polynomial–time. Its input is a security parameter λ. Its outputs are system parameters pp, a public key pk and an initial secret key sk_ϵ. Suppose that pp implicitly includes a time duration T.
- $\mathsf{sk}_{t+1} \leftarrow \mathsf{FSBS.KeyUp}(\mathsf{sk}_t, t)$. FSBS.KeyUp is the key update algorithm. It is probabilistic polynomial–time. Its input is a secret key sk_t at time point t. Its output is

a secret key sk_{t+1} for time point $t+1$. Note that after having sk_{t+1}, the algorithm deletes the key sk_t.

- $(\mathcal{V}, \Sigma) \leftarrow \mathsf{FSBS.Sign}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_t, t, \mu)$. FSBS.Sign is the signing algorithm. It is a probabilistic polynomial–time interactive protocol between a user $\mathcal{U}(\mathsf{pp}, \mathsf{pk}, t, \mu)$ and a signer $\mathcal{S}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_t, t)$. After the interaction, the user obtains a signature Σ on message μ corresponding to time point t, while the signer gets its own view $\mathcal{V}$. In the case that the interaction fails, $\Sigma := \perp$ and $\mathcal{V} := \perp$.

- $1/0 := \mathsf{FSBS.Verify}(\mathsf{pp}, \mathsf{pk}, t, \mu, \Sigma)$. FSBS.Verify is the verification algorithm. It is deterministic polynomial–time. Its inputs are system parameters pp, a public key pk, a time point t, a message μ and a signature Σ. Its output is either 1 (if the signature is non–$\perp$ and valid) or 0 (otherwise).

Correctness. The scheme Π_{FSBS} is correct if for any security parameter λ, any $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_\epsilon) \leftarrow \mathsf{FSBS.Setup}(1^\lambda)$, any time point t, any message μ and any $(\Sigma, \mathcal{V}) \leftarrow \mathsf{FSBS.Sign}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, t, \mu)$,

$$\Pr[\mathsf{FSBS.Verify}(\mathsf{pp}, \mathsf{pk}, t, \mu, \Sigma) = 1] = 1 - \mathsf{negl}(\lambda),$$

over the randomness of all algorithms.

GAME $\mathsf{FSBS}^{\mathsf{BLIND}}_{\mathcal{S}^*}(\lambda) \Rightarrow 1/0$:

1. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_\epsilon) \leftarrow \mathcal{S}^{*\mathsf{FSBS.Setup}(1^\lambda)}$;
2. $(\mu_0, \mu_1) \leftarrow \mathcal{S}^*(\mathsf{pp}, \mathsf{pk})$;
3. $b \xleftarrow{\$} \{0,1\}$, $\mathcal{U}_1 := \mathcal{U}(\mathsf{pp}, \mathsf{pk}, t_1, \mu_b)$, $\mathcal{U}_2 := \mathcal{U}(\mathsf{pp}, \mathsf{pk}, t_2, \mu_{1-b})$;
4. $(\Sigma_1, \Sigma_2) \leftarrow \mathcal{S}^{*\mathcal{U}_1, \mathcal{U}_2}$;
5. $b' \leftarrow \mathcal{S}^{*\Sigma_1, \Sigma_2}$;
6. if $b' = b$, return 1. Otherwise, return 0.

Figure 2.6: Blindness game for FSBS.

Security Notions. Two standard security notions required for FSBS are *blindness* and *forward–secure unforgeability* (FSUF) (following [DCK03]). The blindness of FSBS is the same as that of blind signature. On the other hand, the FSUF security is roughly similar to the unforgeability of digital signatures but stated in the forward security manner. The FSUF requires that the user cannot produce any valid signature for any time point prior to the corrupted time.

We formally define the blindness through the blindness game $\mathsf{FSBS}^{\mathsf{BLIND}}_{\mathcal{S}^*}(\lambda)$ (Figure 2.6) and Definition 2.7.4.

Definition 2.7.4 (Blindness)**.** *Define the advantage of $\mathcal{S}^*$ in the game $\mathsf{FSBS}^{\mathsf{BLIND}}_{\mathcal{S}^*}(\lambda)$ (Figure 2.6) by*

$$\mathsf{Adv}^{\mathsf{BLIND}}_{\mathcal{S}^*,\mathsf{FSBS}}(\lambda) := \left|\Pr[\mathsf{FSBS}^{\mathsf{BLIND}}_{\mathcal{S}^*}(\lambda) \Rightarrow 1] - 1/2\right|.$$

We say that Π_{FSBS} is blind if, for any efficient algorithm $\mathcal{S}^*$, it holds that

$$\mathsf{Adv}^{\mathsf{BLIND}}_{\mathcal{S}^*,\mathsf{FSBS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

We define the FSUF security through the game $\mathsf{FSBS}^{\mathsf{FSUF}}_{\mathcal{U}^*}(\lambda)$ (Figure 2.7), and then in Definition 2.7.5 in which the forger $\mathcal{U}^*$ is a malicious user (adversary). Moreover, the game $\mathsf{FSBS}^{\mathsf{FSUF}}_{\mathcal{U}^*}(\lambda)$ is put in the random oracle model (ROM) assuming that the construction would involve some random oracle. More specifically, we assume that the signing protocol will exploit some hash function considered as a random oracle. Whenever the adversary wants to make a signing query, it always makes a random oracle query in advance.

Definition 2.7.5 (Forward–secure Unforgeability for FSBS)**.** *Define the advantage of $\mathcal{U}^*$ in the game $\mathsf{FSBS}^{\mathsf{FSUF}}_{\mathcal{U}^*}(\lambda)$ (Figure 2.7) by*

$$\mathsf{Adv}^{\mathsf{FSUF}}_{\mathcal{U}^*,\mathsf{FSBS}}(\lambda) := \left|\Pr[\mathsf{FSBS}^{\mathsf{FSUF}}_{\mathcal{U}^*}(\lambda) \Rightarrow 1] - 1/2\right|.$$

We say that Π_{FSBS} is FSUF secure if, for any efficient algorithm $\mathcal{U}^*$, it holds that

$$\mathsf{Adv}^{\mathsf{FSUF}}_{\mathcal{U}^*,\mathsf{FSBS}}(\lambda) \leq \mathsf{negl}(\lambda).$$

2.7.3 Public–key Encryption

Roughly speaking, encryption is a process of transforming a *plaintext* into a *ciphertext*. Mainly, in the public–key encryption (PKE) scheme, one uses a public key to encrypt a plaintext so that only the corresponding secret key can decrypt the ciphertext. The thesis does not involve directly PKE but its advanced variants. However, we still give a description of it for completeness.

Syntax. A PKE system is a tuple Π_{PKE} of polynomial–time algorithms $\mathsf{PKE.Setup}$, $\mathsf{PKE.Enc}$ and $\mathsf{PKE.Dec}$. They work as follows:

- $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$. $\mathsf{PKE.Setup}$ is the key setup algorithm. It is probabilistic polynomial–time. Its input is a security parameter λ. It outputs are system parameters pp, a public key pk and a secret key sk.

GAME $\mathsf{FSBS}^{\mathsf{FSUF}}_{\mathcal{U}^*}(\lambda) \Rightarrow 1/0$:

1. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{FSBS.Setup}(1^\lambda)$;
2. $(\mu^*, t^*, \Sigma^*) \leftarrow \mathcal{U}^{*\mathsf{KQ}(\cdot),\mathsf{HQ}(\cdot,\cdot),\mathsf{SQ}(\cdot,\cdot),\mathsf{BQ}(\bar{t})}$;
   // **NOTE**: Only one break–in query to $\mathsf{BQ}(\bar{t})$ is allowed and $\mathsf{BQ}(\bar{t})$ is the last one being queried. After that, $\mathcal{U}^*$ has to output the forgery (μ^*, t^*, Σ^*). Further, $\mathsf{HQ}(\cdot)$ is optional depending on whether the security is considered in ROM or in SDM.
3. If $\mathsf{FSBS.Verify}(\mathsf{pp}, \mathsf{pk}, t^*, \mu^*, \Sigma^*) = 1$: return 1.
   Otherwise, return 0.

Queried Oracles:

- Key Update Oracle $\mathsf{KQ}(t)$: If $t < T-1$, update secret key sk_t to sk_{t+1} and update t to $t+1$. If $t = T-1$ then sk_T is given as an empty string.
  // **NOTE**: $\mathsf{HQ}(x, \cdot)$ and $\mathsf{SQ}(x, \cdot)$ for all $x < t$ are not allowed after $\mathsf{KQ}(t)$.
- Hashing Oracle $\mathsf{HQ}(t, \mu)$: Return a random value in an appropriate range.
- Signing Oracle $\mathsf{SQ}(t, \mu)$: Return a valid signature at time point t.
- Break-in Oracle $\mathsf{BQ}(\bar{t})$ $(\bar{t} < T-1)$: Return secret key $\mathsf{sk}_{\bar{t}}$ and close all oracles $\mathsf{KQ}(\cdot)$, $\mathsf{HQ}(\cdot,\cdot)$, $\mathsf{SQ}(\cdot,\cdot)$, $\mathsf{BQ}(\cdot)$. // **NOTE:** Only one query is allowed.

Figure 2.7: Forward–secure unforgeability game for FSBS.

- $\mathsf{ct} \leftarrow \mathsf{PKE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu)$. $\mathsf{PKE.Enc}$ is the encryption algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a public key pk and a plaintext μ. Its output is a ciphertext ct.

- $\mu / \perp := \mathsf{PKE.Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{ct})$. $\mathsf{PKE.Dec}$ is the decryption algorithm. It is deterministic polynomial–time. Its inputs are system parameters pp, a secret key sk and a ciphertext ct. Its output is either a plaintext μ (if it succeeds) or a failure symbol $\perp$ (otherwise).

Correctness. The correctness for Π_{PKE} requires that for any security parameter λ, and any $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$, and any plaintext μ, it holds that

$$\Pr[\mathsf{PKE.Dec}(\mathsf{pp}, \mathsf{sk}, \mathsf{PKE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu)) = \mu] = 1 - \mathsf{negl}(\lambda),$$

over the randomness of all algorithms.

Security Notions. For Π_{PKE}, any polynomial–time adversary should be unable to distinguish the ciphertexts of two distinct underlying plaintexts even of the adversary's choice. This requirement is called the *indistinguishability* (IND) security. Moreover, one typically considers the IND with one of the following three attack models:

- The chosen plaintext attack model (CPA): In this model, the adversary cannot query to the decryption oracle.

- The chosen ciphertext attack model (CCA1): In this model, the adversary can query to the decryption oracle, but the adversary performs this query before the adversary commits the target plaintexts.
- The stronger chosen ciphertext attack model (CCA2): In this model, the adversary can query to the decryption oracle at any time it wants; that is, the adversary can query to the decryption oracle even after it sees the target ciphertext.

More formally, the IND–ATK (where ATK $\in$ {CPA, CCA1, CCA2}) security of a PKE is defined through the following game $\mathsf{PKE}_{\mathcal{A}}^{\mathsf{IND\text{–}ATK}}(\lambda)$ (Figure 2.8) and Definition 2.7.6.

GAME $\mathsf{PKE}_{\mathcal{A}}^{\mathsf{IND\text{–}ATK}}(\lambda) \Rightarrow 1/0$:   (where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$)   1. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{PKE.Setup}(1^\lambda)$;   2. $(\mu_0^*, \mu_1^*) \leftarrow \mathcal{A}^{\mathsf{DQ}_1(\cdot)}(\mathsf{pp}, \mathsf{pk})$;   3. $b \xleftarrow{\$} \{0,1\}$, $\mathsf{ct}^* \leftarrow \mathsf{PKE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu_b^*)$;   4. $b' \leftarrow \mathcal{A}^{\mathsf{ct}^*, \mathsf{DQ}_2(\cdot)}(\mathsf{pp}, \mathsf{pk})$; // **NOTE**: $\mathsf{DQ}_2(\mathsf{ct}^*)$ is not allowed.   5. If $b' = b$, return 1. Otherwise, return 0.
Queried Oracles: • Decryption Oracle $\mathsf{DQ}_1(\mathsf{ct})$ (allowed only if $\mathsf{ATK} \in \{\mathsf{CCA1}, \mathsf{CCA2}\}$): Return the output of $\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$.   • Decryption Oracle $\mathsf{DQ}_2(\mathsf{ct})$ (allowed only if $\mathsf{ATK} = \mathsf{CCA2}$): Return the output of $\mathsf{PKE.Dec}(\mathsf{sk}, \mathsf{ct})$.

Figure 2.8: Security game for PKE.

Definition 2.7.6 (IND security for PKE)**.** *Define the advantage of the adversary $\mathcal{A}$ in the game $\mathsf{PKE}_{\mathcal{A}}^{\mathsf{IND\text{–}ATK}}(\lambda)$ as*

$$\mathsf{Adv}_{\mathcal{A},\,\mathsf{PKE}}^{\mathsf{IND\text{–}ATK}}(\lambda) := \left| \Pr[\mathsf{PKE}_{\mathcal{A}}^{\mathsf{IND\text{–}ATK}}(\lambda) \Rightarrow 1] - \frac{1}{2} \right|.$$

We say that Π_{PKE} is IND–ATK secure if for any polynomial–time adversary $\mathcal{A}$, it holds that

$$\mathsf{Adv}_{\mathcal{A},\,\mathsf{PKE}}^{\mathsf{IND\text{-}ATK}}(\lambda) \leq \mathsf{negl}(\lambda).$$

We involve much a specific lattice–based PKE, named *dual Regev*, in our lattice–based designs later. We recall it right below.

Dual Regev PKE. The encryption framework is introduced in [GPV08, Section 7] (for 1–bit plaintexts). It is parameterised by integers n, m, q, some Gaussian parameter $\sigma \geq \omega(\sqrt{\log m})$, a public matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, and some noise distribution χ. The dual Regev PKE encompasses polynomial–time algorithms $\mathsf{Dual.Setup}$, $\mathsf{Dual.Enc}$ and $\mathsf{Dual.Dec}$ described for t–bit plaintexts ($t \geq 1$) as follows.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Dual.Setup}(1^\lambda)$. On input a security parameter λ, do:
  1. Sample a matrix $\mathbf{E} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^{m\times t}$.
  2. Output $\mathbf{U} := \mathbf{AE} \in \mathbb{Z}_q^{n\times t}$ as public key, $\mathbf{E}$ as secret key.
- $\mathsf{ct} \leftarrow \mathsf{Dual.Enc}(\mathsf{pk}, \mu)$. On input a public key pk and a plaintext $\mu \in \{0,1\}^t$, do:
  1. Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\mathbf{x}_0 \leftarrow \chi^m$ and $\mathbf{x}_1 \leftarrow \chi^t$.
  2. Compute $\mathbf{c}_0 = \mathbf{A}^\top \mathbf{s} + \mathbf{x}_0 \in \mathbb{Z}_q^m$, and $\mathbf{c}_1 := \mathbf{U}^\top \mathbf{s} + \mathbf{x}_1 + \mu \cdot \lceil q/2 \rceil \in \mathbb{Z}_q^t$.
  3. Output ciphertext $\mathsf{ct} = (\mathbf{c}_0, \mathbf{c}_1)$.
- $\mu / \perp := \mathsf{Dual.Dec}(\mathsf{pk}, \mathsf{sk}, \mathsf{ct})$. On input a public key pk, a secret key sk and a ciphertext ct, do:
  1. Compute $\mu' = (\mu'_1, \cdots, \mu'_t) := \mathbf{c}_1 - \mathbf{E}^\top \mathbf{c}_0$.
  2. For $i \in [t]$, return 0 if $|\mu'_i| < q/4$; otherwise return 1.

One can modify the dual Regev PKE to have a variant working as follows. In the $\mathsf{Dual.Setup}$, generate $\mathbf{A}$ together with a trapdoor $\mathbf{T_A}$, then sample $\mathbf{U}$ uniformly at random. Then, use $\mathbf{T_A}$ to compute $\mathbf{E}$ such that $\mathbf{U} := \mathbf{AE} \in \mathbb{Z}_q^{n\times t}$ via the lattice trapdoor algorithms (reviewed in Section 2.5.3). This case, $\mathbf{T_A}$ is a secret key, while $(\mathbf{A}, \mathbf{U})$ is a public key.

2.7.4 Hierarchical Identity–based Encryption

Hierarchical identity–based encryption (HIBE) [GS02a] is a delegatable variant of identity–based encryption (IBE) [Sha85]. Recall that in an IBE system, a public key can be an identity, e.g., an email address or a user account. Further, there is a method that generates a corresponding private key from the master secret key on each identity. HIBE is quite the same as IBE, except that the identities in HIBE are arranged as a tree. Moreover, HIBE can derive a private key for a *child identity* from any private key for *parent identities*. We give the formal syntax and security notions for HIBE below.

Syntax. We denote by d the maximum depth of identities. Suppose that identities are presented as binary vectors. For example, $\mathsf{id} = (id_1, \cdots, id_\ell) \in \{0,1\}^\ell$ is a depth–ℓ identity. In this example, $\mathsf{id}|id_{\ell+1} = (id_1, \cdots, id_\ell, id_{\ell+1})$ is a child of id. A HIBE scheme Π_{HIBE} composes of polynomial–time algorithms $\mathsf{HIBE.Setup}$, $\mathsf{HIBE.Ext}$, $\mathsf{HIBE.Der}$, $\mathsf{HIBE.Enc}$ and $\mathsf{HIBE.Dec}$, defined as follows:

- $(\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{HIBE.Setup}(1^\lambda, 1^d)$. $\mathsf{HIBE.Setup}$ is the key setup algorithm. It is probabilistic polynomial–time. Its inputs are a security parameter λ and the maximum depth d. Its outputs are system parameters pp, a master public key mpk and a master secret key msk.

- $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{HIBE.Ext}(\mathsf{pp}, \mathsf{msk}, \mathsf{id})$. $\mathsf{HIBE.Ext}$ is the key extraction algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a master secret key msk and an identity id. Its output is a private key $\mathsf{sk}_{\mathsf{id}}$ for id.

- $\mathsf{sk}_{\mathsf{id}|id_{\ell+1}} \leftarrow \mathsf{HIBE.Der}(\mathsf{pp}, \mathsf{sk}_{\mathsf{id}}, \mathsf{id}|id_{\ell+1})$. $\mathsf{HIBE.Der}$ is the key derivation algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a secret key $\mathsf{sk}_{\mathsf{id}}$ for the identity id and a child $\mathsf{id}|id_{\ell+1}$ of id. Its output is a private key $\mathsf{sk}_{\mathsf{id}|id_{\ell+1}}$ for $\mathsf{id}|id_{\ell+1}$.

- $\mathsf{ct} \leftarrow \mathsf{HIBE.Enc}(\mathsf{pp}, \mathsf{mpk}, \mathsf{id}, \mu)$. $\mathsf{HIBE.Enc}$ is the encryption algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a master public key mpk, an identity id and a plaintext μ. Its output is a ciphertext ct.

- $\mu/\perp := \mathsf{HIBE.Dec}(\mathsf{pp}, \mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{ct})$. $\mathsf{HIBE.Dec}$ is the decryption algorithm. It is deterministic polynomial–time. Its inputs are system parameters pp, an identity id and its associated private key $\mathsf{sk}_{\mathsf{id}}$ and a ciphertext ct. Its output is either a plaintext μ (if it succeeds) or a failure symbol $\perp$ (otherwise).

Note that $\mathsf{HIBE.Der}$ works similarly to $\mathsf{HIBE.Ext}$ when its input secret key is the master secret key msk, which can be seen as the private key for any depth–0 identity.

Correctness. The HIBE scheme Π_{HIBE} is correct if for any security parameter λ, any d, any $(\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{HIBE.Setup}(1^\lambda, 1^d)$, any identity id, and any $\mathsf{sk}_{\mathsf{id}}$ generated either via $\mathsf{HIBE.Ext}(\mathsf{pp}, \mathsf{msk}, \mathsf{id})$ or from $\mathsf{HIBE.Der}(\mathsf{pp}, \mathsf{sk}_{\mathsf{id}'}, \mathsf{id})$ (where id' is any parent of id) and any plaintext μ, the following holds:

$$\Pr[\mathsf{HIBE.Dec}(\mathsf{pp}, \mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{HIBE.Enc}(\mathsf{pp}, \mathsf{mpk}, \mathsf{id}, \mu)) = \mu] = 1 - \mathsf{negl}(\lambda),$$

over the randomness of all involved algorithms.

Security Notions. The standard security of a HIBE is the *indistinguishability of ciphertexts under chosen identities against chosen plaintext/ciphertext attack* (for short, IND-ID-CPA/CCA). The IND-ID-CPA/CCA security is similar to the IND-CPA/CCA security of PKE, with an additional assumption that the adversary can see private keys concerning identities of its choice. A relaxation of IND–ID–CPA/CCA is IND–sID–CPA/CCA standing for "indistinguishability of ciphertexts under *selective* chosen identities against chosen–plaintext/ciphertext attack". We model IND–sID–CPA/CCA through the game $\mathsf{HIBE}_{\mathcal{A}}^{\mathsf{IND\text{–}sID\text{–}ATK}}(\lambda, d)$ given in Figure 2.9. In the game, the challenger can ask the adversary for its target identity id^* before generating keys. In Figure 2.9, if the target identity id^* is released together with μ_0^*, μ_1^* in Step 3 instead of in Step 1, then we have the game for IND–ID–ATK. Here ATK can be one of CPA, CCA1 and CCA2.

GAME $\mathsf{HIBE}_{\mathcal{A}}^{\mathsf{IND\text{-}sID\text{-}ATK}}(\lambda, d) \Rightarrow 1/0$:

(where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$)

1. $\mathsf{id}^* = (id_1^*, \cdots, id_k^*) \leftarrow \mathcal{A}(\lambda, d)$ with $k \leq d$;
2. $(\mathsf{pp}, \mathsf{mpk}, \mathsf{msk}) \leftarrow \mathsf{HIBE.Setup}(1^\lambda, 1^d)$;
3. $(\mu_0^*, \mu_1^*) \leftarrow \mathcal{A}^{\mathsf{PKQ}(\cdot),\ \mathsf{DQ}_1(\cdot,\cdot)}(\mathsf{pp}, \mathsf{mpk})$;
4. $b \xleftarrow{\$} \{0,1\}$, $\mathsf{ct}^* \leftarrow \mathsf{HIBE.Enc}(\mathsf{pp}, \mathsf{mpk}, \mathsf{id}^*, \mu_b^*)$;
5. $b' \leftarrow \mathcal{A}^{\mathsf{ct}^*,\ \mathsf{PKQ}(\cdot),\ \mathsf{DQ}_2(\cdot,\cdot)}(\mathsf{pp}, \mathsf{mpk})$;
   // **NOTE**: $\mathsf{DQ}_2(\mathsf{id}^*, \mathsf{ct}^*)$ is not allowed.
6. If $b' = b$, return 1. Otherwise, return 0.

Queried Oracles:

- Private Key Oracle $\mathsf{PKQ}(\mathsf{id})$: Return the output of $\mathsf{HIBE.Ext}(\mathsf{pp}, \mathsf{msk}, \mathsf{id})$.
  // **NOTE:** id is not a prefix of id^*.
- Decryption Oracle $\mathsf{DQ}_1(\mathsf{id}, \mathsf{ct})$ (allowed only if $\mathsf{ATK} \in \{\mathsf{CCA1}, \mathsf{CCA2}\}$): Return the output of $\mathsf{HIBE.Dec}(\mathsf{pp}, \mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{ct})$.
- Decryption Oracle $\mathsf{DQ}_2(\mathsf{id}, \mathsf{ct})$ (allowed only if $\mathsf{ATK} = \mathsf{CCA2}$): Return the output of $\mathsf{HIBE.Dec}(\mathsf{pp}, \mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{ct})$.

Figure 2.9: Security game for HIBE.

Moreover, there is one more version of IND–(s)ID–ATK in which the adversary just sends only one challenge plaintext, and the challenger C produces the challenge ciphertext ct^* as follows:

- Having received the challenge plaintext μ^*, C chooses $b \xleftarrow{\$} \{0,1\}$.
- If $b = 0$, then $\mathsf{ct}^* \leftarrow \mathsf{HIBE.Enc}(\mathsf{pp}, \mathsf{mpk}, \mathsf{id}^*, \mu_b^*)$.
- If $b = 1$, ct^* is randomly uniform over the ciphertext space.

We call this security version as *the indistinguishability from randomness under (selective) chosen identities against chosen plaintext/ciphertext attack* (INDr–(s)ID–CPA/CCA).

Now, we state the IND–sID–CPA/CCA security for HIBE in Definition 2.7.7 below.

Definition 2.7.7 (IND–sID–ATK security for HIBE)**.** *Define the advantage of the adversary $\mathcal{A}$ in the game $\mathsf{HIBE}_{\mathcal{A}}^{\mathsf{IND\text{–}sID\text{–}ATK}}(\lambda)$ as*

$$\mathsf{Adv}_{\mathcal{A},\mathsf{HIBE}}^{\mathsf{IND\text{–}sID\text{–}ATK}}(\lambda) := \left| \Pr[\mathsf{HIBE}_{\mathcal{A}}^{\mathsf{IND\text{–}sID\text{–}ATK}}(\lambda, d) \Rightarrow 1] - \frac{1}{2} \right|.$$

We say that Π_{HIBE} is IND–sID–ATK secure if for any polynomial–time adversary $\mathcal{A}$, it holds that

$$\mathsf{Adv}_{\mathcal{A},\mathsf{HIBE}}^{\mathsf{IND\text{-}sID\text{-}ATK}}(\lambda) \leq \mathsf{negl}(\lambda).$$

2.7.5 Puncturable Encryption

Puncturable encryption (PE) [GM15] is a subclass of public–key encryption with the ability to revoke decryption ability through *puncturing*. In PE, key generation, encryption and decryption will involve *tags*. Specifically, any plaintext is encrypted together with some *ciphertext tags*. On the other hand, decryption keys can be (possibly multiple times) punctured with other tags (also called *punctures*). Note that ciphertext tags and punctures belong to the same tag space. Suppose that one wants to decrypt a ciphertext using some decryption key. Then, "puncturing" means that whenever the key has been punctured with at least one puncture that also belongs to the list of ciphertext tags in the ciphertext, then decryption will fail. The "puncturing" also involves security requirements for PE.

Formally, the syntax and security notions of PE are given as follows.

Syntax. Besides a security parameter λ, we need some more parameters. We denote $d = d(\lambda)$ (respectively, $\mathcal{M} = \mathcal{M}(\lambda)$ and $\mathcal{T} = \mathcal{T}(\lambda)$) to be the maximum number of tags per ciphertext (respectively, the space of plaintexts and the set of valid tags). A PE system Π_{PE} consists of polynomial–time algorithms $\mathsf{PE.Gen}$, $\mathsf{PE.Enc}$, $\mathsf{PE.Pun}$ and $\mathsf{PE.Dec}$ stated as follows:

- $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{PE.Gen}(1^\lambda, 1^d)$. $\mathsf{PE.Gen}$ is the key generation algorithm. It is probabilistic polynomial–time. Its inputs are a security parameter λ and positive integer d indicating the maximum number of tags per ciphertext. Its outputs are system parameters pp, a public key pk and an initial secret key sk_0.

- $\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu, \mathsf{tg})$. $\mathsf{PE.Enc}$ is the encryption algorithm. It is probabilistic polynomial–time. Its inputs are a public key pk, a plaintext μ, and a list tg of ciphertext tags $t_1, \cdots, t_d$. Its output is a ciphertext ct.

- $\mathsf{sk}_i \leftarrow \mathsf{PE.Pun}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{i-1}, t_i^*)$. $\mathsf{PE.Pun}$ is the key puncturing algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a public key pk, a punctured key sk_{i-1} and a puncture t_i^*. Its outputs is a punctured key sk_i.

- $\mu/\perp := \mathsf{PE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_i, (\mathsf{ct}, \mathsf{tg}))$. $\mathsf{PE.Dec}$ is the decryption algorithm. It is deterministic polynomial–time. Its inputs are system parameters pp, a public key pk, a ciphertext ct, a punctured key sk_i and a list tg of tags $\{t_1, \cdots, t_d\}$. Its output is either a plaintext μ (if the decryption succeeds) or a failure symbol $\perp$ (otherwise).

Correctness. The system Π_{PE} is correct if for any $\lambda \in \mathbb{Z}^+$, any $d \in \mathbb{Z}^+$, any $\eta \in \mathbb{Z}^+$, any punctures $t_1^*, \cdots, t_\eta^* \in \mathcal{T}$, any ciphertext tags $\mathsf{tg} = (t_1, \cdots, t_d) \in \mathcal{T}$, any $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{PE.Gen}(1^\lambda, 1^d)$, any punctured key $\mathsf{sk}_i \leftarrow \mathsf{PE.Pun}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{i-1}, t_i^*)$, and any plaintext $\mu \in \mathcal{M}$ it holds sthat

- If $\{t_1^*, \cdots, t_\eta^*\} \cap \{t_1, \cdots, t_d\} = \emptyset$, then $\forall i \in \{0, \cdots, \eta\}$,

$$\Pr[\mathsf{PE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_i, (\mathsf{PE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu, \mathsf{tg}), \mathsf{tg})) = \mu] \geq 1 - \mathsf{negl}(\lambda).$$

- If there exist $j \in [d]$ and $k \in [\eta]$ such that $t_k^* = t_j$, then $\forall i \in \{k, \cdots, \eta\}$,

$$\Pr[\mathsf{PE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_i, (\mathsf{PE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu, \mathsf{tg}), \mathsf{tg})) = \mu] \leq \mathsf{negl}(\lambda),$$

over the randomness of all involved algorithms.

Security Notions. The standard security notion for PE is the indistinguishability under chosen punctures against chosen plaintext/ciphertext attacks (IND–PUN–CPA/CCA). It resembles the IND–ID–CPA/CCA for HIBE stated in Section 2.7.4 with the difference that IND–PUN–CPA/CCA relates punctures instead of identities. Precisely, the PE security game allows an adversary to query on punctures of its choice to get the corresponding punctured keys. Likewise, one can define the selective version (IND–sPUN–CPA/CCA) and the "indistinguishability from random" version INDr–(s)PUN–CPA/CCA for the PE's security notion. In Figure 2.10, we give the $\mathsf{PE}_{\mathcal{A}}^{\mathsf{IND\text{–}sPUN\text{–}ATK}}(\lambda, d) \Rightarrow 1/0/\perp$ game (where ATK belongs to {CPA, CCA1, CCA2}) for IND–sPUN–ATK which can be easily changed to one for IND–PUN–ATK.

Definition 2.7.8 (IND–sPUN–ATK security for PE)**.** *Define the advantage of the adversary $\mathcal{A}$ in the game $\mathsf{PE}_{\mathcal{A}}^{\mathsf{IND\text{–}sPUN\text{–}ATK}}(\lambda)$ as*

$$\mathsf{Adv}_{\mathcal{A},\,\mathsf{PE}}^{\mathsf{IND\text{–}sPUN\text{–}ATK}}(\lambda) := \left| \Pr[\mathsf{PE}_{\mathcal{A}}^{\mathsf{IND\text{–}sPUN\text{–}ATK}}(\lambda, d) \Rightarrow 1] - \frac{1}{2} \right|.$$

We say that Π_{PE} is IND–sPUN–ATK secure if for any polynomial–time adversary $\mathcal{A}$,

$$\mathsf{Adv}_{\mathcal{A},\,\mathsf{PE}}^{\mathsf{IND\text{–}sPUN\text{–}ATK}}(\lambda) \leq \mathsf{negl}(\lambda).$$

2.7.6 Spatial Encryption

Spatial encryption (SE), put forward by Boneh and Hamburg [BH08, Ham11], is a subclass of the so–called predicate encryption (PrE) [KSW08], in which the latter takes *attributes* and *predicates* into account. More precisely, encryption (respectively, key generation) of PrE will performed together with attributes (respectively, predicates). Typically, an attribute acts as a variable, say $\mathbf{x}$, while a predicate – a function, say f. Decryption is successful if, say $f(\mathbf{x}) = 1$.

SE has a *top* space T. A predicate is a vector (or an affine) subspace V of T. An attribute is a vector (or an affine point) $\mathbf{v}$ residing in T. Then $\mathbf{v} \in V$ is the condition for

GAME $\mathsf{PE}_{\mathcal{A}}^{\mathsf{IND\text{–}sPUN\text{–}ATK}}(\lambda, d) \Rightarrow 1/0/\perp$:

(where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$)

1. $\widehat{\mathsf{tg}} = (\widehat{t_1}, \cdots, \widehat{t_k}) \leftarrow \mathcal{A}(\lambda, d)$ with $k \leq d$;
2. $\mathsf{pT}^* \leftarrow \emptyset$, $\mathsf{cT}^* \leftarrow \emptyset$; // Set of punctures and set corrupted tags, respectively
3. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{PE.Setup}(1^\lambda, 1^d)$;
4. $(\mathsf{cT}^*, \widehat{\mu}_0, \widehat{\mu}_1) \leftarrow \mathcal{A}^{\mathsf{PKQ}(\cdot),\ \mathsf{CQ}_1(\cdot),\ \mathsf{DQ}_1(\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk})$;
   //**NOTE:** cT^* is updated and returned via the query to $\mathsf{CQ}_1(\cdot)$;
5. If $\{\widehat{t_1}, \cdots, \widehat{t_d}\} \cap \mathsf{cT}^* = \emptyset$, return $\perp$;
6. $b \xleftarrow{\$} \{0,1\}$, $\widehat{\mathsf{ct}} \leftarrow \mathsf{PE.Enc}(\mathsf{pp}, \mathsf{pk}, \widehat{\mathsf{tg}}, \widehat{\mu}_b)$;
7. $b' \leftarrow \mathcal{A}^{\widehat{\mathsf{ct}},\ \mathsf{PKQ}(\cdot),\ \mathsf{CQ}_2(\cdot),\ \mathsf{DQ}_2(\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk})$; // **NOTE**: $\mathsf{DQ}_2(\widehat{\mathsf{tg}}, \widehat{\mathsf{ct}})$ is not allowed
8. If $b' = b$, return 1. Otherwise, return 0.

Queried Oracles:

- Puncture Key Oracle $\mathsf{PKQ}(t_{i+1}^*)$: Compute $\mathsf{sk}_{i+1} \leftarrow \mathsf{PE.Pun}(\mathsf{pp}, \mathsf{sk}_i, t_{i+1}^*)$ and save sk_{i+1}. Update $\mathsf{pT}^* \leftarrow \mathsf{pT}^* \cup \{t_{i+1}^*\}$. Always return $\perp$.
- Corruption Oracle $\mathsf{CQ}_1(\mathsf{pT}^*)$: Assign $\mathsf{cT}^* \leftarrow \mathsf{pT}^*$. If $\{\widehat{t_1}, \cdots, \widehat{t_d}\} \cap \mathsf{cT}^* = \emptyset$, return $\perp$. Otherwise (i.e., $\{\widehat{t_1}, \cdots, \widehat{t_d}\} \cap \mathsf{cT}^* \neq \emptyset$), return $(\mathsf{cT}^*, \mathsf{sk}_{|\mathsf{cT}^*|})$.
  // **NOTE:** This oracle allows to be queried in only one time. Further, after execution of $\mathsf{CQ}(\cdot)$, all subsequent queries to $\mathsf{PKQ}(\cdot)$ are answered with $\perp$.
- Decryption Oracle $\mathsf{DQ}_1(\mathsf{tg}, \mathsf{ct})$ (allowed only if $\mathsf{ATK} \in \{\mathsf{CCA1}, \mathsf{CCA2}\}$): Return the output of $\mathsf{PE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{|\mathsf{cT}^*|}, (\mathsf{ct}, \mathsf{tg}))$.
- Corruption Oracle $\mathsf{CQ}_2(\mathsf{pT}^*)$: Always return $\perp$.
- Decryption Oracle $\mathsf{DQ}_2(\mathsf{tg}, \mathsf{ct})$ (allowed only if $\mathsf{ATK} = \mathsf{CCA2}$): Return the output of $\mathsf{PE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{\mathsf{id}}, (\mathsf{ct}, \mathsf{tg}))$.

Figure 2.10: Security game for PE.

successful decryption. However, SE requires that from a decryption key for any space $V_1 \sqsubseteq \mathsf{T}$, one can delegate a decryption key for $V_2 \sqsubseteq V_1$. If attributes and predicates are affine objects, we call such SE *affine*. If they are vector objects, then SE is called *linear*. We define the dimension of SE by that of the top space T.

Syntax. Formally, an SE system Π_{SE} encompasses polynomial–time algorithms $\mathsf{SE.Setup}$, $\mathsf{SE.Der}$, $\mathsf{SE.Del}$, $\mathsf{SE.Enc}$ and $\mathsf{SE.Dec}$ working as below:

- $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{SE.Setup}(1^\lambda, \mathsf{sp})$. $\mathsf{SE.Setup}$ is the key setup algorithm. It is probabilistic polynomial–time. Its inputs are a security parameter λ and setup parameters sp. Its outputs are public system parameters pp and a master secret key msk. Here pp implicitly defined a top space T. Then, msk is exactly the secret key sk_{T} with respect to T.

- $\mathsf{sk}_V \leftarrow \mathsf{SE.Der}(\mathsf{pp}, \mathsf{msk}, V)$. $\mathsf{SE.Der}$ is the key derivation algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a master secret key msk and a subspace V. Its output is a secret key sk_V for V.

- $\mathsf{sk}_{V_2} \leftarrow \mathsf{SE.Del}(\mathsf{pp}, \mathsf{sk}_{V_1}, V_2)$. SE.Del is the key delegation algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a secret key sk_{V_1} for the space V_1 and a space $V_2 \sqsubseteq V_1$. Its output is a secret key sk_{V_2} for V_2.

- $\mathsf{ct}_{\mathbf{x}} \leftarrow \mathsf{SE.Enc}(\mathsf{pp}, \mathbf{x}, \mu)$. SE.Enc is the encryption algorithm. It is probabilistic polynomial–time. Its inputs are system parameters pp, a plaintext μ and an affine point/vector $\mathbf{x}$. Its output is a ciphertext $\mathsf{ct}_{\mathbf{x}}$.

- $\mu / \perp := \mathsf{SE.Dec}(\mathsf{pp}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_V)$. SE.Dec is the decryption algorithm. It is deterministic polynomial–time. Its inputs are system parameters pp, a secret key sk_V and a ciphertext $\mathsf{ct}_{\mathbf{x}}$. Is output is either a plaintext μ underlying $\mathsf{ct}_{\mathbf{x}}$ (if $\mathbf{x} \in V$) or a failure symbol $\perp$ (otherwise).

GAME $\mathsf{SE}_{\mathcal{A}}^{\mathsf{PAY\text{-}sATT\text{-}ATK}}(\lambda, \mathsf{sp})$:
(where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$)
1. $\mathbf{x}^* \leftarrow \mathcal{A}(1^\lambda, \mathsf{sp})$;
2. $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{SE.Setup}(1^\lambda, \mathsf{sp})$;
3. $(\mu_0^*, \mu_1^*) \leftarrow \mathcal{A}^{\mathsf{KQ}(\cdot),\ \mathsf{DQ}_1(\cdot,\cdot)}(\mathsf{pp})$;
4. $b \xleftarrow{\$} \{0,1\}$, $\mathsf{ct}_{\mathbf{x}^*}^* \leftarrow \mathsf{SE.Enc}(\mathsf{pp}, \mathbf{x}^*, \mu_b^*)$;
5. $b' \leftarrow \mathcal{A}^{\mathsf{KQ}(\cdot),\ \mathsf{DQ}_2(\cdot,\cdot)}(\mathsf{pp}, \mathsf{ct}_{\mathbf{x}^*}^*)$. //**NOTE:** $\mathsf{DQ}_2(V, \mathsf{ct}_{\mathbf{x}^*}^*)$ is not allowed with $\mathbf{x}^* \in V$.
6. If $b' = b$, return 1. Otherwise, return 0.

Queried Oracles:
- Key Oracle KQ(V) (allowed only if $\mathbf{x}^* \notin V$): Return $\mathsf{sk}_V \leftarrow \mathsf{SE.Der}(\mathsf{pp}, \mathsf{msk}, V)$.
- Decryption Oracle $\mathsf{DQ}_1(V, \mathsf{ct}_{\mathbf{x}})$ (allowed only if $\mathsf{ATK} \in \{\mathsf{CCA1}, \mathsf{CCA2}\}$): Run $\mathsf{sk}_V \leftarrow \mathsf{SE.Der}(\mathsf{pp}, \mathsf{msk}, V)$, then return the output of $\mathsf{SE.Dec}(\mathsf{pp}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_V)$.
- Decryption Oracle $\mathsf{DQ}_2(V, \mathsf{ct}_{\mathbf{x}})$ (allowed only if $\mathsf{ATK} = \mathsf{CCA2}$): Run $\mathsf{sk}_V \leftarrow \mathsf{SE.Der}(\mathsf{pp}, \mathsf{msk}, V)$, then return the output of $\mathsf{SE.Dec}(\mathsf{pp}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_V)$.

Figure 2.11: Security game for SE.

Correctness. The SE system Π_{SE} is correct if for any $\lambda \in \mathbb{Z}^+$, any appropriate setup parameters sp, any $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{SE.Setup}(1^\lambda, \mathsf{sp}))$, any vector/affine point $\mathbf{x}$, any $\mathsf{sk}_V \leftarrow \mathsf{SE.Del}(\mathsf{pp}, \mathsf{sk}_{V'}, V)$ (for some V' such that $V \sqsubseteq V'$) or $\mathsf{sk}_V \leftarrow \mathsf{SE.Der}(\mathsf{pp}, \mathsf{msk})$, and any plaintext μ $\mathsf{ct}_{\mathbf{x}} \leftarrow \mathsf{SE.Enc}(\mathsf{pp}, \mathbf{x}, \mu)$, over the randomness of all involved algorithms, it holds that

- If $\mathbf{x} \in V$ then

$$\Pr[\mathsf{SE.Dec}(\mathsf{pp}, \mathsf{sk}_V, \mathsf{SE.Enc}(\mathsf{pp}, \mathbf{x}, \mu)) = \mu] \geq 1 - \mathsf{negl}(\lambda).$$

- Otherwise,

$$\Pr[\mathsf{SE.Dec}(\mathsf{pp}, \mathsf{sk}_V, \mathsf{SE.Enc}(\mathsf{pp}, \mathbf{x}, \mu)) = \mu] < \mathsf{negl}(\lambda).$$

It is also required that secret key sk_V for any subspace V follows the same distribution regardless of the way the key is produced. The way is either directly from SE.Der or indirectly generated using SE.Del.

Security Notions. There are several security notions for SE, following ones for PrE [KSW08]. They are combinations of payload–/attribute–hiding with selective/adaptive security.

- **Payload–hiding.** The security notion is the most basic for PrE, requiring that no information of the underlying plaintext is leaked through a ciphertext. However, the requirement is not required for attributes. When challenged, the adversary must send to the challenger an attribute, say $\mathbf{x}^*$ (and, of course, two plaintexts μ_0^*, μ_1^* of the same length). The restriction $f(\mathbf{x}^*) = 0$ must be fulfilled for any key queries according to the predicate f.

- **Attribute–hiding.** This is a stronger security notion than payload–hiding. Attribute–hiding requires that no information about either plaintext or attached attributes is leaked through a ciphertext. When challenged, the adversary must send to the challenger two attributes $\mathbf{x}_0^*$, $\mathbf{x}_1^*$ (along with two plaintexts μ_0^*, μ_1^* of the same length). The restriction $f(\mathbf{x}_0^*) = f(\mathbf{x}_1^*)$ must be satisfied for any key queries according to the predicate f. If there is a query for some f making $f(\mathbf{x}_0^*) = f(\mathbf{x}_1^*) = 1$, then it is a must that $\mu_0^* = \mu_1^*$. Note that the notion of *weak attribute-hiding* mentioned in [AFV11] is an intermediate one. It requires that $f(\mathbf{x}_0^*) = 0$ and $f(\mathbf{x}_1^*) = 0$ for any key query for predicates f's.

- **Selective security.** The adversary must release its target attribute(s) before the challenger outputs any keys.

- **Adaptive (full) security.** The adversary does not have to announce its target attribute(s) ahead of time. Instead, the adversary can do that when challenged.

However, we only formally define the selective payload–hiding (PAY–sATT–ATK) security for SE – see Definition 2.7.9 and the game $\mathsf{SE}_{\mathcal{A}}^{\mathsf{PAY\text{–}sATT\text{–}ATK}}(\lambda)$ in Figure 2.11. Note that $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$. The adaptive version can be obtained with a bit modification in Step 1 and Step 3 of the game $\mathsf{SE}_{\mathcal{A}}^{\mathsf{PAY\text{–}sATT\text{–}ATK}}(\lambda)$ in Figure 2.11. Namely, Step 1 will be deleted and in Step 3, the adversary will release the target attribute $\mathbf{x}^*$ and two plaintexts μ_0^*, μ_1^*. Accordingly, there is no any restrictions on the adversary's queries to Key Oracle $\mathsf{KQ}(\cdot)$ as well as Decryption Oracle $\mathsf{DQ}(\cdot, \cdot)$.

Definition 2.7.9 (PAY–sATT–ATK security for SE)**.** *Define the advantage of the adversary $\mathcal{A}$ in the game $\mathsf{SE}_{\mathcal{A}}^{\mathsf{PAY-sATT-ATK}}(\lambda, \mathsf{sp})$ as*

$$\mathsf{Adv}_{\mathcal{A},\,\mathsf{SE}}^{\mathsf{PAY-sATT-ATK}}(\lambda) := \left| \Pr[\mathsf{SE}_{\mathcal{A}}^{\mathsf{PAY-sATT-ATK}}(\lambda, \mathsf{sp}) \Rightarrow 1] - \frac{1}{2} \right|.$$

We say that an SE is PAY–sATT–ATK secure if for any polynomial–time adversary $\mathcal{A}$, it holds that

$$\mathsf{Adv}_{\mathcal{A},\,\mathsf{PE}}^{\mathsf{PAY-sATT-ATK}}(\lambda) \leq \mathsf{negl}(\lambda).$$

2.8 Summary

We have reviewed some preliminary materials for the upcoming chapters. Specifically, some knowledge regarding lattice–based cryptosystems, such as the background of lattices, the discrete Gaussian distributions, the lattice trapdoors, and the lattice homomorphic evaluations, was covered. We also gave a summary of fundamental tools that will be exploited. In addition, we reviewed related cryptographic primitives.

In Chapter 3, we will consider enhancing blind signatures with the forward security. The forward security ensures that exposing the current secret key does not affect the past keys. Our primary contribution presented in Chapter 3 is proposing the first lattice–based forward–secure blind signature.

Chapter 3

Forward–secure Blind Signatures over Lattices

Part of the content in this chapter appeared in Le et al. [LDS+20]. The author of this thesis is the first, and one of two corresponding authors of [LDS+20]. He contributed to finding the topic, the method to design the cryptosystem, the security analysis of the cryptosystem, and the writing of the manuscript.

3.1 Overview

The key exposure attacks aim to learn about the secret key of a cryptosystem. Therefore, they are one of the most dangerous attacks that can break cryptosystems completely. Among other solutions to preventing the attacks (e.g., see [BM99] for a summary), forward security is a promising method helping us minimize damages caused by secret–key disclosure. Given that a cryptosystem has a list of secret keys, each for a time session, forward security says that if an adversary compromises the current session key, the information of previous session keys is still hidden from the adversary. We illustrate the idea of forward security in Figure 3.1. Here, assume that at the time t_i, the key sk_i is exposed. Then, forward security ensures that all secret keys at times before t_i (i.e., $\mathsf{sk}_0, \cdots, \mathsf{sk}_{i-1}$) are still safe, while the ones after t_i (i.e., $\mathsf{sk}_{i+1}, \cdots$) would be compromised.

Blind signature (BS) is a kind of digital signature introduced by Chaum [Cha83]. In a BS, one blinds a message before sending it to be signed. This way, no information about message contents can be leaked to the signer. Cryptographic voting [Kuc10] and electronic cash schemes [PS96, Section 1] are remarkable applications of BS.

One may ask why and how the signer can sign the message if they cannot see its content. Fortunately, Chaum [Cha83] also gave an example to explain the idea of blind signature. Namely, if we write a signature on the outside of a carbon–lined envelope,

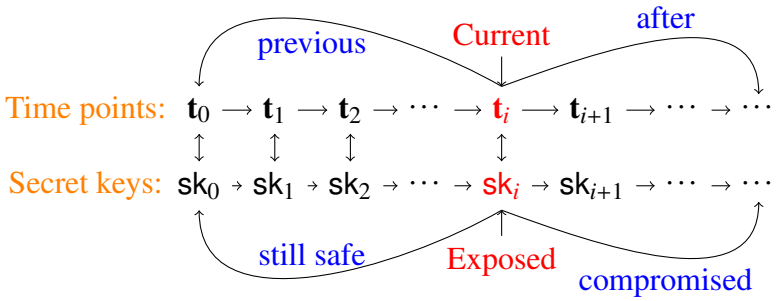

Figure 3.1: Illustration of forward security.

there is a carbon copy of the signature captured on a slip of paper inside the envelope. Obviously, in this case, we do not see the paper inside, but we can still sign on it.

In this chapter, we consider endowing a BS with forward security, i.e., forward–secure blind signature (FSBS). (For the formal definition of BS and FSBS, please refer to Section 2.7.2.) Certainly, FSBS has the same applications as BS and additionally enjoy the forward security. Furthermore, we put such a scheme into a quantum–safe scenario. In doing that, we propose the first construction of lattice–based blind signature enjoying forward security (in the random oracle model (ROM)). Specifically, we rely the construction's security on the short integer solution (SIS) problem. In the following, we give more details about the contributions and techniques/tools that we will present later in this chapter.

Forward Security. The approach to have forward security we apply in this work is to put time points into a structure of a directed binary tree and build up a key evolution mechanism working on the structure. In the structure, we represent each time point as a binary vector whose elements can be arranged via tree nodes going from the root to the leaf concerning the time point. This way, each leaf of the tree corresponds to a time point. Any smaller time points sit to the left of bigger ones. We use the key evolution mechanism to store and manage the secret keys so that the number of keys is minimal but enough to be used to update secret keys for larger time points. The binary structure and the key evolution mechanism ensure that using the stored secret keys cannot recover the keys for smaller time points (i.e., the past keys). We will give further detail below.

Lattice–based FSBS. To realise an FSBS construction over lattices, we deploy the lattice–based blind signature construction proposed by [Rüc10], inspired by Lyubashevsky's identification scheme [Lyu08a] in combination with a modified version of the Fiat–Shamir with aborts (see Section 3.2.5). Here "aborts" is implemented using the rejection sampling technique (see Section 3.2.1). This also allows the blindness for the lattice–based FSBS. Notice that the [Rüc10] construction's security is neither based on the learning with er-

SIGNER $\mathcal{S}(\mathsf{pp},\mathsf{pk},\mathsf{sk})$:

Phase 1:

01. $\mathsf{pk} := (\mathbf{F},\mathbf{K}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{n\times k}$
02. $\mathsf{sk} = \mathbf{S} \in \mathbb{Z}^{m\times k}$,
    $\mathbf{K} = \mathbf{F}\cdot\mathbf{S} \pmod q$)
03. $\mathbf{r} \in \mathbb{Z}^m \leftarrow D^m_{\sigma_2}$, $\mathbf{x} = \mathbf{F}\cdot\mathbf{r} \in \mathbb{Z}_q^n$
04. Send $\mathbf{x}$ to the user
    [**Go to Phase 2**]

Phase 3:

11. $\mathbf{z} = \mathbf{r} + \mathbf{S}\cdot\mathbf{e}$
12. Output $\mathbf{z}$ with probability
$$\min\left\{\frac{D^m_{\sigma_2}(\mathbf{z})}{M_2\cdot D^m_{\sigma_2,\ \mathbf{S}\cdot\mathbf{e}}(\mathbf{z})}, 1\right\}$$
13. Send $\mathbf{z}$ to the user
    [**Go to Phase 4**]

Phase 5:

18. **if** ($\mathsf{result} \neq \mathsf{accept}$):
19. Parse $\mathsf{result} := (\mathbf{a},\mathbf{b},\mathbf{e}',\mathbf{c})$
20. $\mathbf{u} := \mathbf{F}\cdot\mathbf{a} + \mathbf{x} + \mathbf{K}\cdot\mathbf{b} \pmod q$
    $\widehat{\mathbf{u}} := \mathbf{F}\cdot\mathbf{a} + \mathbf{F}\cdot\mathbf{z} - \mathbf{K}\cdot\mathbf{e}' \pmod q$
21. **if** $(\mathbf{e} - \mathbf{b} = \mathbf{e}' = H(\mathbf{u},\mathbf{c})$
    and $\mathbf{e}' = H(\widehat{\mathbf{u}},\mathbf{c})$
    and $\|\mathbf{z} + \mathbf{a}\| \geq \sigma_3\sqrt{m})$:
    restart from Phase 1
22. **Output:** the view $\mathcal{V} = (\mathbf{t},\mathbf{x},\mathbf{e},\mathbf{z})$

USER $\mathcal{U}(\mathsf{pp},\mathsf{pk},\mu)$:

Phase 2:

05. $\mathsf{pk} := (\mathbf{F},\mathbf{K}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{n\times k}$
06. $\mathbf{a} \leftarrow D^m_{\sigma_3}$, $\mathbf{b} \leftarrow D^k_{\sigma_1}$
07. $\mathbf{d}' \xleftarrow{\$} \{0,1\}^n$, $\mathbf{c} := \mathsf{COM}(\mu,\mathbf{d}')$,
    $\mathbf{u} = \mathbf{F}\cdot\mathbf{a} + \mathbf{x} + \mathbf{K}\cdot\mathbf{b} \pmod q$
08. $\mathbf{e}' = H(\mathbf{u},\mathbf{c}) \in \mathcal{R}_H$, $\mathbf{e} := \mathbf{e}' + \mathbf{b}$
09. Output $\mathbf{e}$ with probability
$$\min\left\{\frac{D^m_{\sigma_1}(\mathbf{e})}{M_1\cdot D^m_{\sigma_1,\ \mathbf{e}'}(\mathbf{e})}, 1\right\}$$
10. Send $\mathbf{e}$ back to the signer
    [**Go to Phase 3**]

Phase 4:

14. $\mathbf{z}' = \mathbf{z} + \mathbf{a}$
15. Output $\mathbf{z}'$ with probability
$$\min\left\{\frac{D^m_{\sigma_3}(\mathbf{z}')}{M_3\cdot D^m_{\sigma_3,\ \mathbf{z}}(\mathbf{z}')}, 1\right\}$$
i.e., **if** $(\|\mathbf{z}'\| < \sigma_3\sqrt{m})$:
    $\mathsf{result} := \mathsf{accept}$
else: $\mathsf{result} := (\mathbf{a},\mathbf{b},\mathbf{e}',\mathbf{c})$
16. **Output:** $(\mathbf{t},\mu,\Sigma = (\mathbf{d}',\mathbf{e}',\mathbf{z}'))$
    or $\perp$ when $\mathsf{result} \neq \mathsf{accept}$
17. Send result back to the signer
    [**Go to Phase 5**]

Figure 3.2: Signing algorithm in the SIS–friendly BS scheme.

rors (LWE) problem nor the SIS problem but the collision problem over ideal lattices. We emphasise that the Rückert construction based on ideal lattices seems to suffer from a flaw that has been indicated by Hauck *et al.* [HKLN20]. To avoid the flaw (as shown in a discussion given in Section 3.5), we first turn the [Rüc10]'s blind signature into an SIS–friendly form working with integer matrices and vectors. In the SIS–friendly blind signature, we generate the key pair as follows. We choose a matrix $\mathbf{S}$ via a short distribution in advance and set it as a secret key. We also sample a matrix $\mathbf{F}$ uniformly at random, then computes $\mathbf{K} = \mathbf{F}\cdot\mathbf{S} \pmod q$. Now, the public key is the pair $(\mathbf{F},\mathbf{K})$. We describe the signing algorithm of the SIS–friendly BS in Figure 3.2. There, we apply rejection sampling at Steps 09, 12, and 15. COM is a commitment function (See Section 3.2.6) and H is a hash function (See Section 3.2.2).

However, to equip the above SIS–friendly BS scheme with forward security, we must

embed the dependence on time points into the scheme and have a key evolution mechanism for updating keys. To this end, we do as follows. We use a binary tree structure whose each node corresponds to a random matrix. Namely, if the total number of time points is $T = 2^\ell$, then the tree has a depth of ℓ. The tree's leaves are labelled from left to right by consecutive time points $\mathbf{t} = 0$ up to $\mathbf{t} = T - 1$. The public key and initial secret key are the pair $(\mathbf{A}_0, \mathbf{T}_{\mathbf{A}_0})$ generated using the lattice trapdoor algorithms. We also sample random matrices $\mathbf{A}_j^{(0)}$, $\mathbf{A}_j^{(1)}$ for $j \in [\ell]$. Now, for any node $\mathbf{w}^{(i)} = (w_1, \cdots, w_i) \in \{0,1\}^i$, we build up a concatenated matrix of form $\mathbf{A}_{\mathbf{w}^{(i)}} = [\mathbf{A}_0 | \mathbf{A}_1^{(w_1)} | \cdots | \mathbf{A}_i^{(w_i)}]$. Then, we can compute a trapdoor for $\Lambda_q^\perp(\mathbf{A}_{\mathbf{w}^{(i)}})$ using $\mathbf{T}_{\mathbf{A}_0}$. If the node $\mathbf{w}^{(k)}$ is the ancestor of the node $\mathbf{w}^{(i)}$, then we can obtain a trapdoor for $\Lambda_q^\perp(\mathbf{A}_{\mathbf{w}^{(i)}})$ from a trapdoor for $\Lambda_q^\perp(\mathbf{A}_{\mathbf{w}^{(k)}})$. However, one cannot get a trapdoor for $\Lambda_q^\perp(\mathbf{A}_{\mathbf{w}^{(k)}})$ from a trapdoor of $\Lambda_q^\perp(\mathbf{A}_{\mathbf{w}^{(i)}})$. This is the main idea behind the key evolution (key update) mechanism.

Summary [Section 3.6]

Discussion on the Validity of the Proof of Theorem 3.4.3 [Section 3.5]
↑
The FSBS Construction [Section 3.4]
↑
Binary Tree Structure for Times [Section 3.3]
↑
Related Backkground [Section 3.2]
↑
Overview [Section 3.1]

Figure 3.3: The roadmap of this chapter.

Particularly, for each time point (i.e., each leaf of the binary tree), say $\mathbf{t}$, we form a matrix, say $\mathbf{F}_{(\mathbf{t})}$. This matrix plays the same role as $\mathbf{F}$ of the SIS–friendly BS scheme. Now, the matrix $\mathbf{K}$ is a random matrix sampled in advance. We then use the GPV08 trapdoor algorithms (Section 2.5.3) to generate an ephemeral key, say $\mathbf{S}_{(\mathbf{t})}$ using the secret key $\mathbf{T}_{\mathbf{F}_{(\mathbf{t})}}$ subject to the condition $\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{S}_{(\mathbf{t})} = \mathbf{K} \pmod q$. With respect to the time point $\mathbf{t}$, $\mathbf{S}_{(\mathbf{t})}$ plays the same role as $\mathbf{S}$. Note that we have the trapdoor $\mathbf{T}_{\mathbf{F}_{(\mathbf{t})}}$ for $\mathbf{F}_{(\mathbf{t})}$ by applying the delegation in the GPV08 trapdoor included in the key evolution mechanism. See Steps 01 – 03 of Phase 1 in Figure 3.9 for the mentioned modification. The modification to achieve forward security is the main technical contribution of our work in this chapter.

To summarise, the core techniques/tools used in this chapter are:

- a binary tree data structure used for representing the time points (see Section 3.4 below),
- a modification of Fiat–Shamir with aborts (see Section 3.2.5), in which aborting is implemented using the rejection sampling (see Section 3.2.1),

- a commitment function (see Section 3.2.6),
- a hash function (see Section 3.2.2), and
- the GPV08 trapdoor (see Section 2.5.3).

In the security analysis, we also need the following tools:

- the witness indistinguishability (Section 3.2.4),
- the oracle replay attack and the forking lemma (see Section 3.2.3).

Figure 3.3 shows the roadmap of this chapter.

3.2 Related Background

In this section, we will recall some functions, tools and techniques useful for our lattice–based FSBS construction.

3.2.1 Rejection Sampling

Suppose that we are working with two distributions. Suppose that f and g are their density functions, and that $f(x) \leq M \cdot g(x)$,$\forall x$ with $1 \leq M < \infty$. (See Figure 3.4 below.) At first, one wants to have samples following f. However, suppose further that sampling from f is much harder than sampling from g. If this is the case, the rejection sampling method can step in to help. The method allows using the distribution $g(x)$ to get samples that still follow $f(x)$. The rejection sampling works as follows:

- Repeat (up to at most M times)
  - sampling $x \leftarrow g$ (i.e., $x \sim g$), and
  - outputting x with probability $\frac{f(x)}{M \cdot g(x)}$
- Until x is accepted (i.e., $x \sim f$).

Particularly, we will need a rejection sampling version for discrete Gaussian distributions presented in Lemma 3.2.1. The lemma says that there is a universal constant $M = O(1)$ such that instead of sampling from $D_{\sigma}^{m}(\mathbf{z})$ and outputting with $1/M$, one can sample from $D_{\mathbf{w},\sigma}^{m}(\mathbf{z})$ and then output with probability $\min(\frac{D_{\sigma}^{m}(\mathbf{z})}{M \cdot D_{\mathbf{w},\sigma}^{m}(\mathbf{z})}, 1)$ for any $\mathbf{w}$ sampled from a suitable distribution h.

Lemma 3.2.1 (Rejection Sampling, [Lyu12, Theorem 3.4])**.** *Let $\delta \in \mathbb{R}^{+}$, and $W_{\delta} = \{\mathbf{w} \in \mathbb{Z}^{m} : \|\mathbf{w}\| \leq \delta\} \subset \mathbb{Z}^{m}$. Let $\sigma = \omega(\delta \log \sqrt{m}) \in \mathbb{R}^{+}$. Let $\chi : W_{\delta} \rightarrow \mathbb{R}$ be a probability distribution. For any $M \in \mathbb{R}^{+}$, define two following distributions $\mathcal{X}$ and $\mathcal{Y}$:*

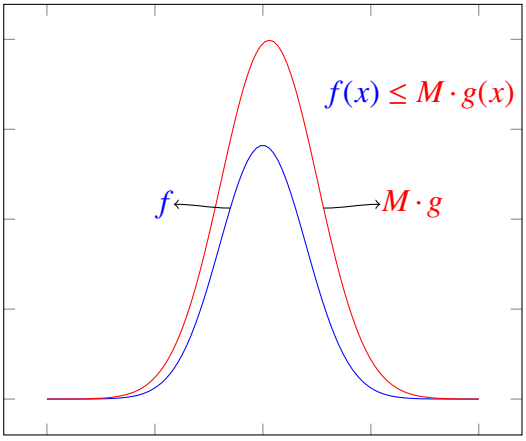

Figure 3.4: Rejection sampling.

- *($\mathcal{X}$):* $\mathbf{w} \sim \chi$, $\mathbf{x} \leftarrow D^m_{\mathbf{w},\sigma}$, *output* $(\mathbf{x}, \mathbf{w})$ *with probability* $\min(\frac{D^m_\sigma(\mathbf{x})}{M \cdot D^m_{\mathbf{w},\sigma}(\mathbf{x})}, 1)$, *and*
- *($\mathcal{Y}$):* $\mathbf{w} \sim \chi$, $\mathbf{x} \leftarrow D^m_\sigma$, *output* $(\mathbf{x}, \mathbf{w})$ *with probability* $1/M$.

Then there is a universal constant $M = O(1)$ *satisfying* $\Delta(\mathcal{X}, \mathcal{Y}) := 2^{-\omega(\log m)}/M$. *The probability for* $\mathcal{X}$ *to output something is not less than* $(1 - 2^{-\omega(\log m)})/M$. *Particularly, if* $\sigma = \alpha\delta$ *for any* $\alpha \in \mathbb{R}^+$ *then* $M = e^{12/\alpha + 1/(2\alpha^2)}$, $\Delta(\mathcal{X}, \mathcal{Y}) = 2^{-100}/M$.

We will exploit this tool in the lattice–based FSBS construction given in Section 3.4..

3.2.2 Hash Functions

Hash functions are an essential component of a digital signature (and its variants) constructions. The functions allow shortening the length of (large) messages to much smaller ones before, for example, signed. Computationally, a hash function takes a variable–length input string to return a fixed–length (generally shorter) string. It should run in polynomial–time. The input string is called *preimage*, while the output string is called *hash value*.

Definition 3.2.1 (Hash Functions)**.** *Take* λ *as security parameter. Let* $\mathcal{D} = \mathcal{D}(\lambda)$, $\mathcal{R} = \mathcal{R}(\lambda)$ *be the space of preimages and the space of hash values, respectively. Suppose additionally that* $\exists n \in \mathbb{Z}^+$ *such that* $|y| \leq n$, $\forall y \in \mathcal{R}$. *A hash function is a map* HF *that works as follows:*

$$\begin{array}{rccl} \mathsf{HF}: & \mathcal{D} & \to & \mathcal{R} \\ & x & \mapsto & y = \mathsf{HF}(x),\ |y| \leq n \ll |x| \end{array}$$

A *collision* occurs whenever there are at least two different preimages being mapped to the same hash value. Formally,

$$\exists x, x' \in \mathcal{D} : x \neq x' \text{ and } \mathsf{HF}(x) = \mathsf{HF}(x')$$

A hash function is called *one–way* if computing a hash value from a preimage is easy, but it is hard to recover the preimage from given a hash value. This is formally defined as in Definition 3.2.2.

Definition 3.2.2 (One–wayness)**.** *Let* HF *be a hash function defined as in Definition 3.2.1. We say* HF *is one–way if, for any hash value* $y \in \mathcal{R}$*, any efficient algorithm* $\mathcal{A}$*, it holds that*

$$\mathsf{Adv}^{\mathsf{OW}}_{\mathcal{A},\,\mathsf{HF}} := \Pr[x \leftarrow \mathcal{A}(y) : y = \mathsf{HF}(x)] \leq \mathsf{negl}(\lambda).$$

We also require a hash function to be *collision–resistant*. The security property guarantees that finding two distinct input strings for the same hash value is computationally infeasible. The requirement is stated in Definition 3.2.3.

Definition 3.2.3 (Collision–resistance)**.** *Let* HF *be a hash function defined as in Definition 3.2.1. We say* HF *is collision–resistant if, for any efficient algorithm* $\mathcal{A}$*, it holds that*

$$\mathsf{Adv}^{\mathsf{CR}}_{\mathcal{A},\,\mathsf{HF}} := \Pr[x, x' \leftarrow \mathcal{A}(\lambda) : x, x' \in \mathcal{D}, x \neq x', \mathsf{HF}(x) = \mathsf{HF}(x')] \leq \mathsf{negl}(\lambda).$$

Hash functions enjoying the one–way and collision–resistance will be involved in the lattice–based FSBS construction given in Section 3.4.

3.2.3 Rewinding, Oracle Replay Attack and Forking Lemma

One of the typical diagrams used to give provable security of a specific cryptosystem is proposing a reduction from a hard algorithmic problem to the desired security property. In the reduction, assuming an efficient attacker can disrupt the desired security of the cryptosystem, one constructs a solver algorithm for the hard algorithmic problem. In such a reduction, a popular approach (especially in proofs for digital signatures and their variants) is the *rewinding*. The core idea is that the solver rewinds the adversary's execution from a fixed point in the protocol multiple times until the solver achieves a favourable outcome.

In particular, we review, in detail, a version of the rewinding called the *oracle replay attack* by Pointcheval and Stern [PS00] implemented in the random oracle model. In the oracle replay attack, any algorithm (e.g., attackers, adversaries, solvers) is modelled as a probabilistic polynomial–time Turing machine in which one obtains the "probabilistic" via a random tape. The oracle replay attack works as follows:

- An attacker is supposed to make up to a predetermined number, say Q, of queries to a random oracle: each query q_i will be responded with a random value h_i.

- Replaying the attacker on n lists of random oracle queries that are different from some position, say $I+1$,

$$\begin{array}{ll} q_1,\cdots,q_{I-1},\ q_I, & q_{I+1}^{(1)},\cdots,q_Q^{(1)} \\ q_1,\cdots,q_{I-1},\ q_I, & q_{I+1}^{(2)},\cdots,q_Q^{(2)} \\ & \vdots \\ q_1,\cdots,q_{I-1},\ q_I, & q_{I+1}^{(n)},\cdots,q_Q^{(n)} \end{array} \tag{3.1}$$

  using n lists of corresponding responses which are

$$\begin{array}{ll} h_1,\cdots,h_{I-1},\ h_I^{(1)}, & h_{I+1}^{(1)},\cdots,h_Q^{(1)} \\ h_1,\cdots,h_{I-1},\ h_I^{(2)}, & h_{I+1}^{(2)},\cdots,h_Q^{(2)} \\ & \vdots \\ h_1,\cdots,h_{I-1},\ h_I^{(n)}, & h_{I+1}^{(n)},\cdots,h_Q^{(n)} \end{array} \tag{3.2}$$

  will result in n outputs, say $\Sigma^{(1)},\Sigma^{(2)},\cdots\Sigma^{(n)}$, respectively. Note here that it is required that $h_I^{(1)},h_I^{(2)},\cdots,h_I^{(n)}$ are mutually different.

- A solution to the intended hard problem can be extracted from the knowledge of $q_I,h_I^{(1)},\cdots,h_I^{(n)},\Sigma^{(1)},\cdots,\Sigma^{(n)}$.

The (general) following lemma by Bellare and Neven [BN06] gives a lower bound on the success probability of (a generalisation of) the oracle replay attack limited to 2 replays. Specifically, let A be an algorithm whose input is a tuple of $(x,h_1,...,h_Q;\mathsf{rt})$, where $x \xleftarrow{\$} \mathsf{IG}$ (IG is called the *input generator*), all the h_i's are chosen randomly from some set H and rt is a random tape. Here, IG is an input generator. The output of A is some index $I \in \{0,\cdots,Q\}$. Assume that A is run twice on the same random tape rt. Specifically, the algorithm outputs an index $I \in [Q]$ along with a side output Σ in the first run. Its input is $(x,h_1,\cdots,h_Q;\mathsf{rt}),$. In the second run on input $(x,h_1,\cdots,h_{I-1},h'_I,\cdots,h'_Q;\mathsf{rt})$ (noting that the input has been changed from the index I), the algorithm outputs an index $I' \in [Q]$ along with a side output Σ'. The forking lemma assures that the probability that two indexes are identical (i.e., $I = I'$) subject to $h_I \neq h'_I$ is not too small.

Lemma 3.2.2 (Forking Lemma, [BN06, Lemma 1])**.** *Let $Q \in \mathbb{Z}^+$ fixed. Let H be a set such that $|H| \geq 2$. Let $x \leftarrow \mathsf{IG}$ be a randomized algorithm whose output is denoted by x. Let $(I,\Sigma) \leftarrow \mathsf{A}(x,\ h_1,\cdots,h_Q;\mathsf{rt})$ be a randomized algorithm. Its input are $x \xleftarrow{\$} \mathsf{IG}$, $h_1,\cdots,h_Q \xleftarrow{\$} H$ and a random bit–string rt. Its output is a pair (I,Σ), where $I \in \{0,\cdots,Q\}$ and Σ is a side output. Let p be the accepting probability of A, which is defined as the probability that the A's output $I \geq 1$ over the randomness of $x,h_1,\cdots,h_Q$, and rt.*

The forking algorithm F_{A} *associated to* A *is the randomized algorithm that takes input* $x \xleftarrow{\$} \mathsf{IG}$ *and proceeds as in Figure 3.5.*

EXPERIMENT $\mathsf{F}_{\mathsf{A}}(x)$:
01. Pick a random bit–string rt.
02. $h_1, \cdots, h_Q \xleftarrow{\$} H$.
03. $(I, \Sigma) \xleftarrow{\$} \mathsf{A}(x, h_1, \cdots, h_Q; \mathsf{rt})$.
04. If $I = 0$, return $(0, \epsilon, \epsilon)$.
05. $h'_I, \cdots, h'_Q \xleftarrow{\$} H$.
06. $(I', \Sigma') \xleftarrow{\$} \mathsf{A}(x, h_1, \cdots, h_{I-1}, h'_I, \cdots, h'_Q; \mathsf{rt})$.
07. If $I = I'$ and $h_I \neq h'_I$, return $(1, \Sigma, \Sigma')$.
08. Otherwise, return $(0, \epsilon, \epsilon)$.

Figure 3.5: The forking algorithm F_{A}.

Define

$$frk := \Pr_{x \xleftarrow{\$} \mathsf{IG},\ (b,\Sigma,\Sigma') \leftarrow \mathsf{F}_{\mathsf{A}}(x)} [b = 1].$$

Then,

$$frk \geq p \cdot \left(\frac{p}{Q} - \frac{1}{|H|} \right).$$

We will deploy the oracle replay attack and the forking lemma to give security proof in ROM for the lattice–based forward–secure blind signatures given in Section 3.4. There, x is the system's public key; whilst h_i, h'_i's – random values replied to hash queries.

3.2.4 Witness Indistinguishability

Consider an interactive protocol between two parties where a party (the prover) wants to prove a public statement to the other (the verifier). In this situation, the witness indistinguishability [FS90] ensures that the verifier cannot tell which witness is being used by the prover (even if the verifier knows the set of all witnesses associating to the statement).

We will follow [Lyu08b] to formally review the notion of witness indistinguishability . For a string str and a relation $\mathsf{RL}(\cdot,\cdot)$, we define a witness set $\mathsf{WIT}_{\mathsf{RL}}$ to be one consisting of witnesses wit's such that the statement "$\mathsf{RL}(\mathsf{str}, \mathsf{wit}) = 1$" holds. In the lattice setting, for example, $\mathsf{str} := (\mathbf{A}, \mathbf{K})$, where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ and $\mathbf{K} \in \mathbb{Z}_q^{n \times m}$, $\mathsf{wit} := \mathbf{S} \leftarrow D_\sigma^{m \times m}$ and $\mathsf{RL}(\mathsf{str}, \mathsf{wit}) = 1 \Longleftrightarrow \mathbf{A} \cdot \mathbf{S} = \mathbf{K} \pmod{q}$.

Let $\mathcal{P}(\mathsf{str}, \mathsf{wit})$ and $\mathcal{V}(\mathsf{str}, \mathsf{aux})$ be randomised interactive algorithms. The former's input is a pair of public string and a witness $(\mathsf{str}, \mathsf{wit})$. The latter's input is a pair of public string and auxiliary input $(\mathsf{str}, \mathsf{aux})$. Let $(\mathcal{P}, \mathcal{V})$ denote an interactive protocol between the prover $\mathcal{P}(\mathsf{str}, \mathsf{wit})$ and the verifier $\mathcal{V}(\mathsf{str}, \mathsf{aux})$. $\mathcal{P}(\mathsf{str}, \mathsf{wit})$ wants to prove to $\mathcal{V}(\mathsf{str}, \mathsf{aux})$ a

public statement "$\mathsf{RL}(\mathsf{str},\mathsf{wit}) = 1$" associated with which there are several secret witnesses wit's. Define by $\mathcal{V}_{\mathcal{P}(\mathsf{str},\mathsf{wit})}(\mathsf{str},\mathsf{aux})$ the view of $\mathcal{V}(\mathsf{str},\mathsf{aux})$ received from the protocol.

Definition 3.2.4. *The protocol* $(\mathcal{P},\mathcal{V})$ *is witness indistinguishable if for all* $\mathcal{V}^*(\mathsf{str},\mathsf{wit}')$*, all large enough* str*, any auxiliary input* aux *and any two witnesses* $\mathsf{wit},\mathsf{wit}' \in \mathsf{WIT}_{\mathsf{RL}}$*, the statistical distance of two* $\mathcal{V}^*(\mathsf{str},\mathsf{aux})$*'s views is negligible in the length of* str*, i.e.,*

$$\Delta(\mathcal{V}^*_{\mathcal{P}(\mathsf{str},\mathsf{wit})}(\mathsf{str},\mathsf{aux}),\mathcal{V}^*_{\mathcal{P}(\mathsf{str},\mathsf{wit}')}(\mathsf{str},\mathsf{aux})) \leq \mathsf{negl}(|\mathsf{str}|).$$

We will employ the witness indistinguishability technique in the proof for the FSUF of the lattice forward–secure blind signature given in Section 3.4..

3.2.5 Fiat–Shamir with Aborts

Informally speaking, the Fiat–Shamir protocol [FS87] is an identification scheme of three moves between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$. $\mathcal{P}$ wants to convince $\mathcal{V}$ that $\mathcal{P}$ knows a witness wit for a public statement stm:

- *First move:* The prover delivers to the verifier a commitment com.
- *Second move:* The verifier provides the prover with a challenge cha.
- *Third move:* Having received the challenge cha, the prover replies with a response res. Basing on the knowledge of the statement stm and the *transcript* (com, cha, res), the verifier decides to accept or reject.

An augmented version called the *Fiat-Shamir with aborts* by Lyubashevsky [Lyu08b, Lyu09] allows the prover to abort when necessary before replying to the response res (Steps 3–4 in Figure 3.6). Aborting helps to protect some information about the witness while still maintaining the witness indistinguishability, as discussed in [Lyu09]. However, this may hurt the protocol's efficiency due to repeating the protocol multiple times.

We will design a lattice–based forward–secure blind signature, given in Section 3.4, exploiting a modified version of the Fiat–Shamir with aborts. In the construction, aborting occurs in several places, and we implement it using the rejection sampling (Section 3.2.1).

3.2.6 Commitment Functions

A commitment scheme is a cryptographic abstraction of a locked box, in which the sender can put, say, a message and then lock the box. After the box is locked, the message cannot be changed and is hidden from the receiver. Instead, the message will only get revealed if the sender gives the receiver the key. Likewise, a commitment scheme enables *hiding*

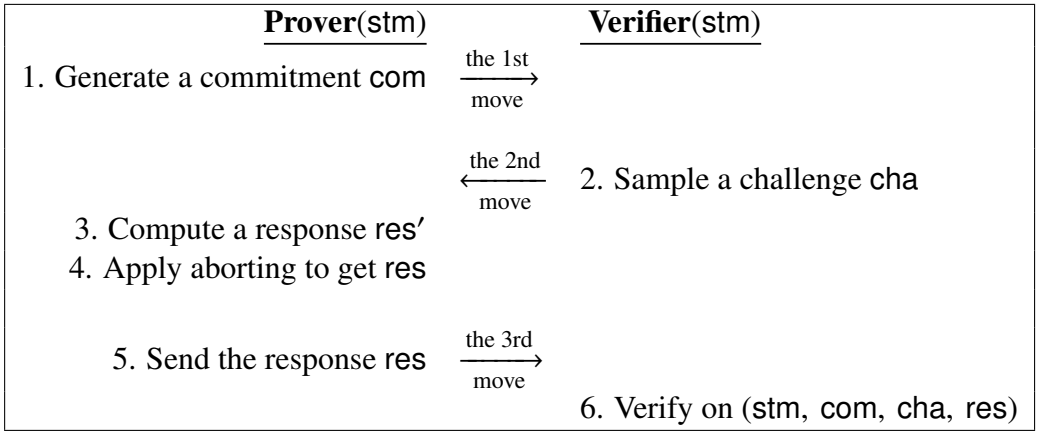

Figure 3.6: Fiat–Shamir with aborts.

a secret thing while keping it unchanged (i.e., *binding*) with the ability to open that thing later.

Let $n \in \mathbb{Z}^+$. More precisely, a commitment scheme acts as a function COM that maps two strings $(\nu,\mathbf{x}) \in \{0,1\}^* \times \{0,1\}^n$ to a string $\mathsf{com} \in \{0,1\}^n$. Furthermore, COM is required to have the two following properties in this thesis.

- **Statistically hiding**. Let $\mathcal{A}$ be any computationally unbounded algorithm. Let $\mathsf{com} = \mathsf{COM}(\mu,\mathbf{x})$, $\mathsf{com}' = (\nu',\mathbf{d}')$ and $(\nu,\mathbf{x}) \neq (\nu',\mathbf{x}')$. This property ensures that it is unable for $\mathcal{A}$ to statistically distinguish com and com'.

- **Computationally binding**. This property says that given a commitment string $\mathsf{com} := \mathsf{COM}(\mu,\mathbf{x})$, no polynomial–time algorithm can find $(\nu',\mathbf{x}')$ with $\nu' \neq \nu$ subject to $\mathsf{com} = \mathsf{COM}(\nu',\mathbf{x}')$.

See [HM96, KTX08, Rüc10] for more details. We utilise such a commitment function for our lattice–based forward–secure blind signature given in Section 3.4.

Next, we will present the binary tree structure for time points. The structure implemented in lattices using lattice trapdoors allows for achieving forward security in our lattice–based FSBS construction.

3.3 Binary Tree Structure for Times

In this section, we first present a detailed description of the binary tree structure for times. We then apply the structure to support forward security in the lattice–based forward–secure blind signature.

Binary Structure for Times. Suppose that we want to represent a time duration $T = 2^{\ell-1}$ for some integer $\ell \in \mathbb{Z}^+$ using the structure. To this end, we do the following:

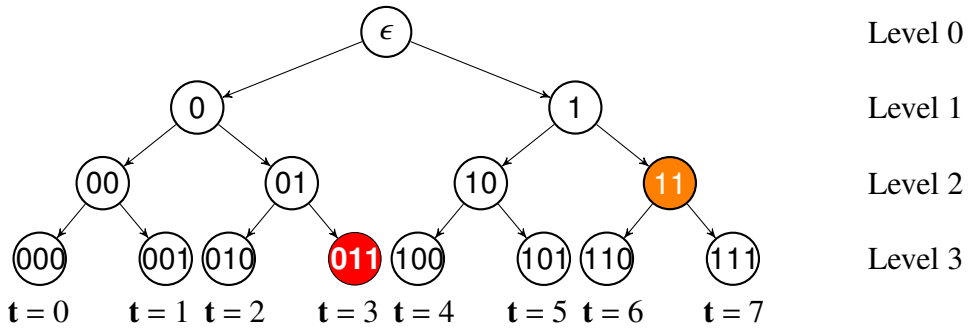

Figure 3.7: Binary tree for 8 time points (of depth $\ell = 3$).

- We partition the time duration $T = 2^{\ell-1}$ to 2^{ℓ} time points $\mathbf{t}$'s, each belonging to $\{0, \cdots, 2^{\ell} - 1\}$.

- We then build a binary tree of depth ℓ. The *root* sits at Level 0. The leaves reside at the highest level of the tree. At the smaller levels are the *nodes*. Remind that in a binary tree, each node (including the leaves) is assigned to a binary value with the following principle: A node at level $i+1$ $(0 \leq i \leq \ell - 1)$ will be assigned to either 0 if it is placed to the left of its level–i parent node or 1 – if it is placed to the right of the parent node.

- Each time point is attached to a tree leaf in such order that smaller time points sit to the left of the bigger ones. The readers can refer to Figure 3.7 for an illustration of a depth–3 binary tree for eight time points in which the root is denoted by ϵ. Each leaf (i.e., a time point) will correspond with a binary vector showing the unique path from the root ϵ to the leaf $\mathbf{t}$. We denote the binary vector by $\mathsf{bin}(\mathbf{t})$. We can see that for a leave $\mathbf{t}$, the binary vector is identical to the binary decomposition of $\mathbf{t}$. Taking the red–coloured leave $\mathbf{t} = 3$ in Figure 3.7 for example, its binary vector is $\mathsf{bin}(\mathbf{t}) = (0, 1, 1)$ as the path going from the root to the leave $\mathbf{t}$ is $\epsilon \rightarrow 0 \rightarrow 1 \rightarrow 1$.

- Similarly to leaves, each node will correspond with a binary vector showing the path going from the root to that node. Also, we denote by $\mathsf{bin}(\mathbf{w}^{(i)})$ the binary vector for a level–i node $\mathbf{w}^{(i)}$. We sometimes name nodes by their corresponding binary decomposition for convenience. For example, the orange–coloured node named $\textcircled{11}$ in Figure 3.7 corresponds with the path $\epsilon \rightarrow 1 \rightarrow 1$ and $\mathsf{bin}(\textcircled{11})$ =(1, 1).

Applying to the Lattice–based FSBS. Suppose that we have built a depth–ℓ binary tree as above for a time duration $T = 2^{\ell-1}$ (for $\ell \in \mathbb{Z}^{+}$). Now, we show how to apply the structure for forward security in our lattice–based forward–secure blind signature. We do as follows:

- We sample $(\mathbf{A}_0, \mathbf{T}_{\mathbf{A}_0})$ using the algorithm $\mathsf{TrapGen}$ (Lemma 2.5.4). Here $\mathbf{T}_{\mathbf{A}_0}$ is the initial secret key for the FSBS system.

- We also sample 2ℓ random matrices, say $\mathbf{A}_1^{(0)}$, $\mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_\ell^{(0)}$, $\mathbf{A}_\ell^{(1)}$. They have the same dimension as that of $\mathbf{A}_0$.

- We then place $\mathbf{A}_0$ at the binary tree's root ϵ. At level $i \in [\ell]$, we place $\mathbf{A}_i^{(0)}$ at the node assigned to binary value 0 and $\mathbf{A}_i^{(1)}$ at the node assigned to binary value 1. See Figure 3.8 for an illustration for $\ell = 3$.

- For any leave $\mathbf{t}$ having $\mathsf{bin}(\mathbf{t}) = (t_1, \cdots, t_\ell)$ we form the concatenated matrix $\mathbf{F}_{(\mathbf{t})} = [\mathbf{A}_0 | \mathbf{A}_1^{(t_1)} | \cdots | \mathbf{A}_\ell^{(t_\ell)}]$ according to the path $\epsilon \to t_1 \to \cdots \to t_\ell$. Similarly, for any node $\mathbf{w}^{(i)}$ that has $\mathsf{bin}(\mathbf{w}^{(i)}) = (w_1, \cdots, w_i)$, we assign it to the concatenated matrix $\mathbf{A}_{(\mathbf{w}^{(i)})} = [\mathbf{A}_0 | \mathbf{A}_1^{(w_1)} | \cdots | \mathbf{A}_i^{(w_i)}]$ according to the path $\epsilon \to w_1 \to \cdots \to w_i$. For example, for the red–coloured leave $\boxed{011}$ in Figure 3.7 (i.e., the leave where we placed the red–coloured matrix $\mathbf{A}_3^{(1)}$ according to $\mathbf{t} = 3$ in Figure 3.8), we form $\mathbf{F}_{(0,1,1)} := [\mathbf{A}_0 | \mathbf{A}_1^{(0)} | \mathbf{A}_2^{(1)} | \mathbf{A}_3^{(1)}]$ as $\mathsf{bin}(\mathbf{t}) = (0,1,1)$. With respect to the node $\boxed{11}$ in Figure 3.7 (i.e., the node where we placed the red–coloured matrix $\mathbf{A}_1^{(1)}$ in Figure 3.8), we form $\mathbf{A}_{(1)} = [\mathbf{A}_0 | \mathbf{A}_1^{(1)}]$.

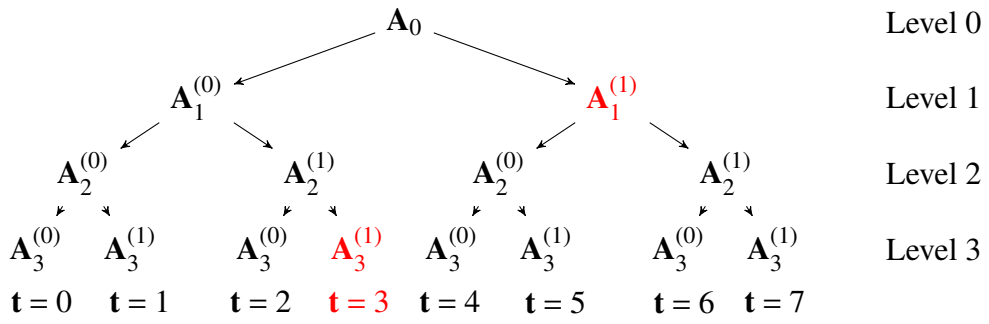

Figure 3.8: Binary tree of matrices.

For updating secret keys to achieve forward security, one can exploit the trapdoor delegation mechanism using $\mathsf{ExtBasisLeft}$ and $\mathsf{RandBasis}$ (Lemma 2.5.4).

- A secret key for some node can be easily computed using the initial secret key $\mathsf{sk}_\epsilon := \mathbf{T}_{\mathbf{A}_0}$. Specifically, the secret key $\mathbf{T}_{(\mathbf{w}^{(i)})}$ associated to the node $\mathbf{w}^{(i)} = (w_1, \cdots, w_i)$, is computed by generating

$$\mathbf{T}'_{(\mathbf{w}^{(i)})} \leftarrow \mathsf{ExtBasisLeft}(\mathbf{A}_{(\mathbf{w}^{(i)})}, \mathbf{T}_{\mathbf{A}_0}), \text{ where } \mathbf{A}_{(\mathbf{w}^{(i)})} = \left[\mathbf{A}_0 | \mathbf{A}_1^{(w_1)} | \cdots | \mathbf{A}_i^{(w_i)}\right],$$

and then randomising

$$\mathbf{T}_{(\mathbf{w}^{(i)})} \leftarrow \mathsf{RandBasis}(\mathbf{A}_{(\mathbf{w}^{(i)})}, \mathbf{T}'_{(\mathbf{w}^{(i)})}, \sigma_{0,i}) \text{ for some Gaussian parameter } \sigma_{0,i}.$$

By Lemma 2.5.4, $\sigma_{0,i} \geq \|\widetilde{\mathbf{T}'_{\mathbf{w}^{(i)}}}\| \cdot \omega(\sqrt{\log(\#\mathsf{col}(i))}) = \|\widetilde{\mathbf{T}_{\mathbf{A}_0}}\| \cdot \omega(\sqrt{\log(\#\mathsf{col}(i))})$, where

$\#\mathsf{col}(i)$ denotes the number of columns of $\mathbf{A}_{(\mathbf{w}^{(i)})}$. For example, if $\mathbf{A}_0, \mathbf{A}_j^{(w_j)} \in \mathbb{Z}_q^{n\times m}$ for all $j \in [i]$ then $\#\mathsf{col}(i) = (i+1)\cdot m$.

- Moreover, one can also update a secret key of a node at a higher level from the key of its ancestor nodes at a lower level. For example, suppose that $\mathbf{T}_{(\mathbf{w}^{(k)})}$ is a known secret key for the node $\mathbf{w}^{(k)} = (w_1, \cdots, w_k)$. Suppose that $\mathbf{w}^{(i)} = (w_1, \cdots, w_k, w_{k+1}, \cdots, w_i)$ for $k < i$. The key $\mathbf{T}'_{(\mathbf{w}^{(i)})}$ can be produced by first computing

$$\mathbf{T}'_{(\mathbf{w}^{(i)})} \leftarrow \mathsf{ExtBasisLeft}(\mathbf{A}_{(\mathbf{w}^{(i)})}, \mathbf{T}_{(\mathbf{w}^{(k)})}), \text{ where } \mathbf{A}_{(\mathbf{w}^{(i)})} = \left[\mathbf{A}_0|\mathbf{A}_1^{(w_1)}|\cdots|\mathbf{A}_2^{(w_k)}|\cdots|\mathbf{A}_i^{(w_i)}\right],$$

then randomising

$$\mathbf{T}_{(\mathbf{w}^{(i)})} \leftarrow \mathsf{RandBasis}(\mathbf{A}_{(\mathbf{w}^{(i)})}, \mathbf{T}'_{(\mathbf{w}^{(i)})}, \sigma_{0,i}) \text{ for some Gaussian parameter } \sigma_{0,i}.$$

- Similarly, a secret key for a leaf which corresponds to a time point can also be derived from any ancestor's secret key.

3.4 The FSBS Construction

Now, we present the proposed forward–secure blind signature. Before doing that, we summarise the system parameters and give their descriptions in Table 3.1.

Table 3.1: System parameters in our lattice–based FSBS construction.

Parameters	Definition
λ	Security parameter
ℓ	Depth of binary tree for time points
n	# row of matrices $\mathbf{A}_i^{(b)}$
q	System modulus
m	# column of matrices $\mathbf{A}_i^{(b)}$
k	# column of matrices $\mathbf{K}$
κ	Maximal Hamming weight of elements in $\mathcal{R}_H$
$T := 2^\ell$	Time duration
σ	Gaussian parameter used in SampleD
$\sigma_{0,i}$	Gaussian parameter used in RandBasis
σ_1	Gaussian parameter used in Phase 2 of FSBS.Sign
σ_2	Gaussian parameter used in Phase 1 of FSBS.Sign
σ_3	Gaussian parameter used in Phase 4 of FSBS.Sign
M_1	Rejection sampling constant used in Phase 2 of FSBS.Sign
M_2	Rejection sampling constant used in Phase 3 of FSBS.Sign
M_3	Rejection sampling constant used in Phase 4 of FSBS.Sign
β	the bound for the SIS instance

3.4.1 The Construction

The FSBS construction Π_{FSBS} includes efficient algorithms FSBS.Setup, FSBS.KeyUp, FSBS.Sign,and FSBS.Verify described as follows.

- $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_\epsilon) \leftarrow \mathsf{FSBS.Setup}(1^\lambda, 1^\ell)$. On input a security parameter λ and a binary tree depth ℓ, the algorithm outputs a set of public parameters pp, a public key pk and an initial secret key sk_ϵ. Do the following steps.

  1. Choose $n = n(\lambda)$, $q = \mathsf{poly}(n)$ prime, $m = \Theta(n \log q)$, k, κ, ℓ, time duration $T = 2^\ell$, Gaussian parameters σ, σ_1, σ_2, σ_3 and $\sigma_{0,i}$ for $i \in [\ell]$ as shown in Section 3.4.4.
  2. Take $\mathcal{M} = \{0,1\}^*$ as the message space.
  3. Select a matrix $\mathbf{K} \xleftarrow{\$} \mathbb{Z}_q^{n \times k}$. Similarly, sample matrices $\mathbf{A}_1^{(0)}, \mathbf{A}_1^{(1)}, \mathbf{A}_2^{(0)}, \mathbf{A}_2^{(1)}, \cdots, \mathbf{A}_\ell^{(0)}, \mathbf{A}_\ell^{(1)}$ from $\mathbb{Z}_q^{n \times m}$ uniformly at random.
  4. Run $\mathsf{TrapGen}(1^n, 1^m, q)$ (Lemma 2.5.4) to obtain a pair $(\mathbf{A}_0, \mathbf{T}_{\mathbf{A}_0})$, where $\mathbf{A}_0 \in \mathbb{Z}_q^{n \times m}$ and $\mathbf{T}_{\mathbf{A}_0} \in \mathbb{Z}^{m \times m}$ are a matrix and its associated trapdoor.
  5. Let $H : \{0,1\}^* \to \mathcal{R}_H$ be a collision–resistant and one–way hash function, where $\mathcal{R}_H := \{\mathbf{e}' \in \{-1,0,1\}^k : \|\mathbf{e}'\| \leq \kappa\}$.
  6. Let $\mathsf{COM} : \{0,1\}^* \times \{0,1\}^n \to \{0,1\}^n$ be a computationally binding and statistically hiding commitment function (see Section 3.2.6).
  7. Output $\mathsf{pp} := \{\lambda, n, q, m, \ell, T, k, \kappa, \sigma, \sigma_0, \sigma_1, \sigma_2, \sigma_3, \mathcal{M}, H, \mathsf{COM}\}$, $\mathsf{pk} := \{\mathbf{A}_0, \mathbf{A}_1^{(0)}, \mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_\ell^{(0)}, \mathbf{A}_\ell^{(1)}, \mathbf{K}\}$, and $\mathsf{sk}_\epsilon := \mathbf{T}_{\mathbf{A}_0}$ as public parameters, the public key and the initial secret key, respectively.

- $\mathsf{sk}_{\mathbf{t}+1} \leftarrow \mathsf{FSBS.KeyUp}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_\mathbf{t}, \mathbf{t})$. Taking as input a set of public parameters pp, a public key pk and a secret key $\mathsf{sk}_\mathbf{t}$ for a time point $\mathbf{t}$, the algorithm returns a secret key $\mathsf{sk}_{\mathbf{t}+1}$ for the time point $\mathbf{t}+1$. The algorithm needs a *key evolution mechanism* to delete all secret keys of internal nodes that can produce past keys. Additionally, it is required that the key evolution mechanism stores the smallest number of keys necessary for the signature to work properly. The key evolution mechanism works as follows.

  1. For any leaf $\mathbf{t}$, define the *minimal cover* $\mathsf{Node}(\mathbf{t})$ to be the smallest set of nodes that contains an ancestor of all leaves in $\{\mathbf{t}, \cdots, T-1\}$ but does not contain any ancestor of any leaf in $\{0, \cdots, \mathbf{t}-1\}$. For example, in Figure 3.7, $\mathsf{Node}(0) = \{\epsilon\}$, $\mathsf{Node}(1) = \{(0,0,1), (0,1), (1)\}$, $\mathsf{Node}(2) = \{(0,1), (1)\}$, $\mathsf{Node}(3) = \{(0,1,1), (1)\}$ (i.e., two black circles in the tree), $\mathsf{Node}(4) = \{(1)\}$, $\mathsf{Node}(5) = \{(1,0,1), (1,1)\}$, $\mathsf{Node}(6) = \{(1,1)\}$, $\mathsf{Node}(7) = \{(1,1,1)\}$.

2. The secret key $\mathsf{sk}_{\mathbf{t}}$ at time point $\mathbf{t}$ contains secret keys corresponding to all nodes (including leaves) in $\mathsf{Node}(\mathbf{t})$. For example, for the tree from Figure 3.7, we have $\mathsf{sk}_0 = \mathsf{sk}_\epsilon = \{\mathbf{T}_{\mathbf{A}_0}\}$, $\mathsf{sk}_1 = \{\mathbf{T}_{(1)}, \mathbf{T}_{(0,1)}, \mathbf{T}_{(0,0,1)}\}$, where $\mathbf{T}_1$, $\mathbf{T}_{(0,1)}$and $\mathbf{T}_{(0,0,1)}$ are associated trapdoors to $\mathbf{F}_{(1)} = [\mathbf{A}_0|\mathbf{A}_1^{(1)}]$, $\mathbf{F}_{(0,1)} = [\mathbf{A}_0|\mathbf{A}_1^{(0)}|\mathbf{A}_2^{(1)}]$ and $\mathbf{F}_{(0,0,1)} = [\mathbf{A}_0|\mathbf{A}_1^{(0)}|\mathbf{A}_2^{(0)}|\mathbf{A}_3^{(1)}]$, respectively. Notice that we compute $\mathsf{sk}_{\mathbf{t}}$ by using $\mathsf{ExtBasisLeft}$ and $\mathsf{RandBasis}$.

3. To update $\mathsf{sk}_{\mathbf{t}}$ to $\mathsf{sk}_{\mathbf{t}+1}$, determine the minimal cover $\mathsf{Node}(\mathbf{t}+1)$, then derive keys for all nodes in $\mathsf{Node}(\mathbf{t}+1) \setminus \mathsf{Node}(\mathbf{t})$ (if needed) using the keys in $\mathsf{sk}_{\mathbf{t}}$ as described above. Finally, delete all keys in $\mathsf{Node}(\mathbf{t}) \setminus \mathsf{Node}(\mathbf{t}+1)$. For example, for the tree in Figure 3.7 we have $\mathsf{sk}_1 = \{\mathbf{T}_{(1)}, \mathbf{T}_{(0,1)}, \mathbf{T}_{(0,0,1)}\}$, $\mathsf{sk}_2 = \{\mathbf{T}_{(0,1)}, \mathbf{T}_{(1)}\}$ since $\mathsf{Node}(2) \setminus \mathsf{Node}(1) = \{(0,1),(1)\}$ and $\mathsf{Node}(1) \setminus \mathsf{Node}(2) = \{(0,0,1)\}$.

- $(\Sigma, \mathcal{V}) \leftarrow \mathsf{FSBS.Sign}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{\mathbf{t}}, \mathbf{t}, \mu)$. On input a set of public parameters pp, a public key pk, a secret key $\mathsf{sk}_{\mathbf{t}}$ for a time point $\mathbf{t}$ (note that $\mathsf{sk}_{\mathbf{t}}$ contains $\mathbf{T}_{\mathbf{F}_{(\mathbf{t})}}$), and a message μ, the algorithm returns a signature Σ and a view $\mathcal{V}$. The algorithm is an interactive protocol between a user (i.e., the message author) and a signer. Here, we use a modification of Fiat–Shamir with aborts (Section 3.2.5), in which aborting is implemented using the rejection sampling (Section 3.2.1). The protocol consists of five phases: Phases 1, 3 and 5 are done by the signer, while Phases 2 and 4 – by the user. Figure 3.9 algorithmically details the signing algorithm. We describe $\mathsf{FSBS.Sign}$ in words as follows.

  – *Phase 1*: The signer constructs the matrix $\mathbf{F}_{(\mathbf{t})} = \left[\mathbf{A}_0|\mathbf{A}_1^{(t_1)}|\cdots|\mathbf{A}_\ell^{(t_\ell)}\right] \in \mathbb{Z}_q^{n\times(\ell+1)m}$ for the time point $\mathbf{t} = (t_1, \cdots, t_\ell) \in \{0,1\}^\ell$. Next, it computes an ephemeral secret key $\mathbf{S}_{(\mathbf{t})}$ using $\mathsf{SampleD}$ described in Lemma 2.5.4 satisfying that $\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{S}_{(\mathbf{t})} = \mathbf{K} \pmod q$. Note that $\mathbf{S}_{(\mathbf{t})}$ can also be computed at Phase 3 instead. The signer then samples $\mathbf{r} \in \mathbb{Z}^{(\ell+1)m}$ according to the discrete Gaussian distribution $D_{\sigma_2}^{(\ell+1)m}$. It finally computes and sends $\mathbf{x} = \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{r} \in \mathbb{Z}_q^n$ to the user.

  – *Phase 2*: Upon receiving $\mathbf{x}$, the user samples blind factors $\mathbf{a} \leftarrow D_{\sigma_3}^{(\ell+1)m}$, $\mathbf{b} \leftarrow D_{\sigma_1}^k$ and $\mathbf{d}' \xleftarrow{\$} \{0,1\}^n$. It computes $\mathbf{u} = \mathbf{x} + \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} + \mathbf{K} \cdot \mathbf{b}$ and hashes $\mathbf{u}$ with $\mathbf{c} := \mathsf{COM}(\mu, \mathbf{d}') \in \{0,1\}^n$ using H to obtain a *real challenge* $\mathbf{e}'$. The user runs the rejection sampling technique to get the *blinded challenge* $\mathbf{e}$, which is then sent back to the user.

  – *Phase 3*: The key $\mathbf{S}_{(\mathbf{t})}$ and $\mathbf{r}$ are used to compute $\mathbf{z} = \mathbf{r} + \mathbf{S}_{(\mathbf{t})} \cdot \mathbf{e}$. To make sure that no information of $\mathbf{S}_{(\mathbf{t})}$ is leaked, the signer applies the rejection sampling, which implies that the distribution of $\mathbf{z}$ and $\mathbf{r}$ are the same. Finally, the signer delivers the *blinded signature* $\mathbf{z}$ to the user.

– *Phase 4*: The user computes the unblinded signature $\mathbf{z}' = \mathbf{z} + \mathbf{a}$. Again, the user calls the rejection sampling to ensure that $\mathbf{z}'$ and $\mathbf{z}$ are independent of each other and $\mathbf{z}'$ is bounded in some desired domain. The user returns $(\mathbf{t}, \mu, \Sigma = (\mathbf{d}', \mathbf{e}', \mathbf{z}'))$ as the *final signature* if $\|\mathbf{z}'\| \leq \sigma_3 \sqrt{(1+\ell)m}$ holds. Otherwise, the user outputs "$\perp$". The user is required to confirm the validity of the final signature by sending result to the signer: result := accept means the final signature is good, while result := $(\mathbf{a}, \mathbf{b}, \mathbf{e}', \mathbf{c})$) is to ask the signer to restart the signing protocol.

– *Phase 5*: Having obtained result, the signer checks whether or not result ≠ accept. If not, it returns the *view* $\mathcal{V} = (\mathbf{t}, \mathbf{x}, \mathbf{e}, \mathbf{z})$. Otherwise, the signer checks some information before restarting the signing algorithm.

Note that because we use the rejection sampling in Phase 2 locally then it does not restart the signing algorithm. However, the rejection sampling in Phase 3 and Phase 4 can make the signing algorithm restart.

- $0/1 :=$ FSBS.Verify(pp, pk, $\mathbf{t}, \mu, \Sigma$). The algorithm accepts a signature Σ on the message μ for the time point $\mathbf{t} = (t_1, \cdots, t_\ell)$, a set of public parameters pp and public key pk as its input and performs the following steps:

  1. Parse $\Sigma = (\mathbf{d}', \mathbf{e}', \mathbf{z}')$.
  2. Form $\mathbf{F}_{(\mathbf{t})} := \left[\mathbf{A}_0 | \mathbf{A}_1^{(t_1)} | \cdots | \mathbf{A}_\ell^{(t_\ell)}\right] \in \mathbb{Z}^{n \times (1+\ell)m}$.
  3. Compute $\widehat{\mathbf{e}} := H(\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{z}' - \mathbf{K} \cdot \mathbf{e}' \pmod q,\ \mathsf{COM}(\mu, \mathbf{d}'))$.
  4. If $\|\mathbf{z}'\| \leq \sigma_3 \sqrt{(1+\ell)m}$ and $\widehat{\mathbf{e}} = \mathbf{e}'$, then output 1 (Valid), otherwise return 0 (Invalid).

3.4.2 Correctness

Theorem 3.4.1 (Correctness). *The correctness of* Π_{FSBS} *holds after at most* e^2 *restarts with probability not smaller than* $1 - 2^{-100}$.

Proof. First, we have an observation that: Remark 2.3.1 says that if $\sigma = 12\|\mathbf{c}\|$, then $D_\sigma^m(\mathbf{x})/(M \cdot D_{\sigma,\,\mathbf{c}}^m(\mathbf{x})) \leq e^{1+1/288}/M$ with probability greater than $1 - 2^{-100}$. Additionally, the rejection sampling requires that $D_\sigma^m(\mathbf{x})/(M \cdot D_{\sigma,\,\mathbf{c}}^m(\mathbf{x})) \leq 1$, which implies $M \geq e^{1+1/288}$. Therefore, $e^{1+1/288}$ is the best choice for M.

Assume that $(\mathbf{t}, \mu, \Sigma = (\mathbf{d}', \mathbf{e}', \mathbf{z}'))$ is a signature produced by FSBS.Sign(pp, pk, $\mathsf{sk}_{(\mathbf{t})}, \mu$) as shown in Figure 3.9. We first show that $H(\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{z}' - \mathbf{K} \cdot \mathbf{e}' \pmod q,\ \mathsf{COM}(\mu, \mathbf{d}')) = \mathbf{e}'$, which is equivalent to proving $\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{z}' - \mathbf{K} \cdot \mathbf{e}' = \mathbf{x} + \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} + \mathbf{K} \cdot \mathbf{b} \pmod q$. Indeed, using

SIGNER $\mathcal{S}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{(\mathbf{t})}, \mathbf{t})$:

Phase 1:

01. $\mathbf{F}_{(\mathbf{t})} := \left[\mathbf{A}_0 | \mathbf{A}_1^{(t_1)} | \cdots | \mathbf{A}_\ell^{(t_\ell)}\right] \in \mathbb{Z}_q^{n \times (\ell+1)m}$
02. $\mathbf{S}_{(\mathbf{t})} \in \mathbb{Z}^{(\ell+1)m \times k} \leftarrow \mathsf{SampleD}(\mathbf{F}_{(\mathbf{t})}, \mathbf{T}_{\mathbf{F}_{(\mathbf{t})}}, \mathbf{K}, \sigma)$
    (i.e., $\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{S}_{(\mathbf{t})} = \mathbf{K} \pmod q$)
03. $\mathbf{r} \in \mathbb{Z}^{(\ell+1)m} \leftarrow D_{\sigma_2}^{(\ell+1)m}$, $\mathbf{x} = \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{r} \in \mathbb{Z}_q^n$
04. Send $\mathbf{x}$ to the user
    [**Go to Phase 2**]

Phase 3:

11. $\mathbf{z} = \mathbf{r} + \mathbf{S}_{(\mathbf{t})} \cdot \mathbf{e}$
12. Output $\mathbf{z}$ with probability
$$\min\left\{\frac{D_{\sigma_2}^{(\ell+1)m}(\mathbf{z})}{M_2 \cdot D_{\sigma_2,\, \mathbf{S}_{(\mathbf{t})}\cdot\mathbf{e}}^{(\ell+1)m}(\mathbf{z})}, 1\right\}$$
13. Send $\mathbf{z}$ to the user
    [**Go to Phase 4**]

Phase 5:

18. **if** (result $\neq$ accept):
19. Parse result := $(\mathbf{a}, \mathbf{b}, \mathbf{e}', \mathbf{c})$
20. $\mathbf{u} := \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} + \mathbf{x} + \mathbf{K} \cdot \mathbf{b} \pmod q$
    $\widehat{\mathbf{u}} := \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} + \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{z} - \mathbf{K} \cdot \mathbf{e}' \pmod q$
21. **if** $(\mathbf{e} - \mathbf{b} = \mathbf{e}' = H(\mathbf{u}, \mathbf{c})$
    and $\mathbf{e}' = H(\widehat{\mathbf{u}}, \mathbf{c})$
    and $\|\mathbf{z} + \mathbf{a}\| \geq \sigma_3 \sqrt{(\ell+1)m})$:
    restart from Phase 1
22. **Output:** the view $\mathcal{V} = (\mathbf{t}, \mathbf{x}, \mathbf{e}, \mathbf{z})$

USER $\mathcal{U}(\mathsf{pp}, \mathsf{pk}, \mathbf{t}, \mu)$:

Phase 2:

05. $\mathbf{F}_{(\mathbf{t})} := \left[\mathbf{A}_0 | \mathbf{A}_1^{(t_1)} | \cdots | \mathbf{A}_\ell^{(t_\ell)}\right]$
06. $\mathbf{a} \leftarrow D_{\sigma_3}^{(\ell+1)m}$, $\mathbf{b} \leftarrow D_{\sigma_1}^k$
07. $\mathbf{d}' \xleftarrow{\$} \{0,1\}^n$, $\mathbf{c} := \mathsf{COM}(\mu, \mathbf{d}')$,
    $\mathbf{u} = \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} + \mathbf{x} + \mathbf{K} \cdot \mathbf{b} \pmod q$
08. $\mathbf{e}' = H(\mathbf{u}, \mathbf{c}) \in \mathcal{R}_H$, $\mathbf{e} := \mathbf{e}' + \mathbf{b}$
09. Output $\mathbf{e}$ with probability
$$\min\left\{\frac{D_{\sigma_1}^m(\mathbf{e})}{M_1 \cdot D_{\sigma_1,\, \mathbf{e}'}^m(\mathbf{e})}, 1\right\}$$
10. Send $\mathbf{e}$ back to the signer
    [**Go to Phase 3**]

Phase 4:

14. $\mathbf{z}' = \mathbf{z} + \mathbf{a}$
15. Output $\mathbf{z}'$ with probability
$$\min\left\{\frac{D_{\sigma_3}^{(\ell+1)m}(\mathbf{z}')}{M_3 \cdot D_{\sigma_3,\, \mathbf{z}'}^{(\ell+1)m}(\mathbf{z}')}, 1\right\}$$
i.e., **if** $(\|\mathbf{z}'\| < \sigma_3 \sqrt{(\ell+1)m})$:
    result := accept
else: result := $(\mathbf{a}, \mathbf{b}, \mathbf{e}', \mathbf{c})$
16. **Output:** $(\mathbf{t}, \mu, \Sigma = (\mathbf{d}', \mathbf{e}', \mathbf{z}'))$
    or $\perp$ when result $\neq$ accept
17. Send result back to the signer
    [**Go to Phase 5**]

Figure 3.9: Signing algorithm FSBS.Sign($\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{\mathbf{t}}, \mathbf{t}, \mu$).

the fact $\mathbf{K} = \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{S}_{(\mathbf{t})} \pmod q$, we easily have

$$\begin{aligned}
\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{z}' - \mathbf{K} \cdot \mathbf{e}' \pmod q &= \mathbf{F}_{(\mathbf{t})} \cdot (\mathbf{z} + \mathbf{a}) - \mathbf{K} \cdot (\mathbf{e} - \mathbf{b}) \pmod q \\
&= \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{z} + \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} - \mathbf{K} \cdot \mathbf{e} + \mathbf{K} \cdot \mathbf{b} \pmod q \\
&= \mathbf{F}_{(\mathbf{t})} \cdot (\mathbf{r} + \mathbf{S}_{(\mathbf{t})} \cdot \mathbf{e}) + \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} - \mathbf{K} \cdot \mathbf{e} + \mathbf{K} \cdot \mathbf{b} \pmod q \\
&= \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{r} + \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} + \mathbf{K} \cdot \mathbf{b} \pmod q \\
&= \mathbf{x} + \mathbf{F}_{(\mathbf{t})} \cdot \mathbf{a} + \mathbf{K} \cdot \mathbf{b} \pmod q.
\end{aligned}$$

Now by Lemma 2.3.4, $\|\mathbf{z}'\| \leq \sigma_3 \sqrt{(1+\ell)m}$ with overwhelming probability. Furthermore, in Phases 3 and 4, by applying the observation mentioned above to the rejection samplings, the signing algorithm can successfully produce a valid signature after at most

$M_2 \cdot M_3 = e^{1+1/288} \cdot e^{1+1/288} \approx e^2$ repetitions. The proof follows. □

3.4.3 Security Analysis

In this section, we will analyse the security requirements for our FSBS construction Π_{FSBS}. These are the blindness and the forward–secure unforgeability (FSUF). For the details of these requirements, please refer to Section 2.7.2. Roughly speaking, the blindness ensures that even an adversarial signer who knew two messages coming from a user cannot decide which message he/she has signed on. The forward–secure unforgeability guarantees that the (malicious) user cannot produce any valid signature for any time point prior to the corrupted time.

Blindness. The blindness of the proposed FSBS comes from the statistically hiding and computationally binding properties of the commitment functionCOM and the one–wayness and the collision–resistance of the hash function H, as shown in Theorem 3.4.2.

Theorem 3.4.2 (Blindness)**.** *Let* COM *be a statistically hiding and computationally binding commitment and H be a one–way and collision–resistant hash function. Then, the FSBS system* Π_{FSBS} *satisfies the blindness.*

Proof. We follow the game $\mathsf{FSBS}^{\mathsf{BLIND}}_{\mathcal{S}^*}(\lambda)$ (Figure 2.6) in proving the blindness of Π_{FSBS}. In the game, at Step 2, the adversarial signer $\mathcal{S}^*$ specifies two messages μ_0 and μ_1. The signer then sends these messages to the challenger C. The challenger C selects a random bit $b \xleftarrow{\$} \{0,1\}$. The challenger then plays the roles of two users $\mathcal{U}_b := \mathcal{U}(\mathsf{pp}, \mathsf{pk}, \mathbf{t}, \mu_b)$ and $\mathcal{U}_{1-b} := \mathcal{U}(\mathsf{pp}, \mathsf{pk}, \mathbf{t}, \mu_{1-b})$. After that, each of $\mathcal{U}_b$ and $\mathcal{U}_{1-b}$ interacts with $\mathcal{S}^*$ to produce signatures on μ_b and μ_{1-b}, respectively. From the interaction with $\mathcal{U}_b$ (respectively, $\mathcal{U}_{1-b}$), $\mathcal{S}^*$ gets $(\mathcal{V}_b, \Sigma_b)$ (respectively, $(\mathcal{V}_{1-b}, \Sigma_{1-b})$).

It is sufficient to demonstrate the independence of the information of $(\mathcal{V}_b, \Sigma_b)$ and $(\mathcal{V}_{1-b}, \Sigma_{1-b})$ from the underlying messages. We do that by showing that $\mathcal{V}_b$ and $\mathcal{V}_{1-b}$ are statistically indistinguishable. Also, we will show the statistical indistinguishability for Σ_b and Σ_{1-b}.

Let consider the distributions of $\mathcal{V}_b = (\mathbf{t}, \mathbf{x}_b, \mathbf{e}_b, \mathbf{z}_b)$ and $\mathcal{V}_{1-b} = (\mathbf{t}, \mathbf{x}_{1-b}, \mathbf{e}_{1-b}, \mathbf{z}_{1-b})$. Since $\mathbf{x}_b$, $\mathbf{z}_b$ and $\mathbf{x}_{1-b}$, $\mathbf{z}_{1-b}$ are produced by $\mathcal{S}^*$ itself. Thus, we only need to look at the distributions of $\mathbf{e}_b$ and $\mathbf{e}_{1-b}$, which are produced in Phase 2 (See Figure 3.9). We note that the distributions of both $\mathbf{e}_b$ and $\mathbf{e}_{1-b}$ are the same $D^k_{\sigma_1}$. This is due to the rejection sampling technique used at Step 09, Phase 2.

Now, we consider $\Sigma_b = (\mathbf{d}'_b, \mathbf{e}'_b, \mathbf{z}'_b)$ and $\Sigma_{1-b} = (\mathbf{d}'_{1-b}, \mathbf{e}'_{1-b}, \mathbf{z}'_{1-b})$. Again, due to the use of rejection sampling, the distributions of both $\mathbf{z}'_b$ and $\mathbf{z}'_{1-b}$ are identical, which is $D^{(1+\ell)m}_{\sigma_3}$ (Step 15, Phase 4). Furthermore, $\mathbf{d}'_b$, $\mathbf{d}'_{1-b}$ are uniformly random. Additionally, $\mathbf{e}'_b$ (respectively, $\mathbf{e}'_{1-b}$) is a hash value of the collision–resistant and one–way hash function

H on input $(\mathbf{u}_b,\ \mathbf{c}_b := \mathsf{COM}(\mu_b, \mathbf{d}'_b))$ (respectively, $(\mathbf{u}_{1-b},\ \mathbf{c}_{1-b} := \mathsf{COM}(\mu_{1-b}, \mathbf{d}'_{1-b}))$) in which COM is statistically hiding and computationally binding. This concludes that no information about the underlying messages is leaked through Σ_b and Σ_{1-b}.

Finally, note that restarts will occur in Phase 5 in case the user sends $\mathcal{S}^*$ the $\mathsf{result} := (\mathbf{a}, \mathbf{b}, \mathbf{e}', \mathbf{c})$. However, because the values $\mathbf{d}'$, $\mathbf{a}$ and $\mathbf{b}$ are fresh in every repetition, these restarts will not make the advantage of $\mathcal{S}^*$ better. The theorem follows. □

Forward–secure Unforgeability. We prove the forward secure unforgeability of Π_{FSBS} in Theorem 3.4.3. For the FSUF of FSBS, please refer to Figure 2.7. In the theorem, we assume that the forger $\mathcal{A}$ can make at most Q_H plain hash queries and at most Q_S signing queries. Note that each signing query also needs an additional hash query. Therefore, there are at most $Q := Q_H + Q_S$ hash queries in total.

We need the following lemma, which supports the witness indistinguishability argument mentioned in Section 3.2.4.

Lemma 3.4.1 (Adapted from [Lyu12, Lemma 4.2], [Lyu11, Lemma 5.2])**.** *Given a matrix* $\mathbf{F} \in \mathbb{Z}_q^{n\times(\ell+1)m}$*, where* $(\ell+1)m > 64 + n\log q/\log(2d+1)$ *and* $\mathbf{s} \xleftarrow{\$} \{-d, \cdots, 0, \cdots, d\}^{(\ell+1)m}$*. Then, there exists another* $\mathbf{s}' \xleftarrow{\$} \{-d, \cdots, 0, \cdots, d\}^{(\ell+1)m}$ *such that* $\mathbf{F}\cdot\mathbf{s} = \mathbf{F}\cdot\mathbf{s}' \pmod q$ *with probability at least* $1 - 2^{-100}$*.*

Notice that Lemma 2.4.5 also states the same result as Lemma 3.4.1 above. However, the condition in Lemma 2.4.5 (which is $d \gg q^{(\ell+1)m/n}$) is not so clear as the condition (which is $(\ell+1)m > 64 + n\log q/\log(2d+1)$) in Lemma 3.4.1.

Lemma 3.4.2 (Witness Indistinguishability)**.** *Consider the signing protocol in Figure 3.9. For any public parameters* pp*, any public key* pk*, any secret key* $\mathsf{sk}_\mathbf{t}$*, any time point* $\mathbf{t}$ *and any message* μ*, and for any* $\mathbf{S}_{(\mathbf{t})},\ \mathbf{S}^*_{(\mathbf{t})} \leftarrow D_\sigma^{(1+\ell)m\times k}$ *such that* $\mathbf{F}_{(\mathbf{t})}\cdot\mathbf{S}_{(\mathbf{t})} = \mathbf{F}_{(\mathbf{t})}\cdot\mathbf{S}^*_{(\mathbf{t})} = \mathbf{K} \pmod q$*, the resulting views* $\mathcal{V} = (\mathbf{t}, \mathbf{x}, \mathbf{e}, \mathbf{z})$ *(with respect to* $\mathbf{S}_{(\mathbf{t})}$*) and* $\mathcal{V}^* = (\mathbf{t}, \mathbf{x}^*, \mathbf{e}^*, \mathbf{z}^*)$ *(with respect to* $\mathbf{S}^*_{(\mathbf{t})}$*) are indistinguishable.*

Proof. To prove the lemma, we can do the same way as the proof for the blindness. Specifically, we consider the distribution of $\mathcal{V} = (\mathbf{t}, \mathbf{x}, \mathbf{e}, \mathbf{z})$ and $\mathcal{V}^* = (\mathbf{t}, \mathbf{x}^*, \mathbf{e}^*, \mathbf{z}^*)$. First, we remark that $\mathbf{x}$ and $\mathbf{x}^*$ are produced independently of $\mathbf{S}_{(\mathbf{t})},\ \mathbf{S}^*_{(\mathbf{t})}$. Next, we consider the distribution of $\mathbf{e}$, and $\mathbf{e}^*$, which are produced in Phase 2 (see Figure 3.9). The distribution is the same $D^k_{\sigma_1}$, due to the rejection sampling technique used at Step 09, Phase 2. Moreover, the distribution of $\mathbf{e}$ and $\mathbf{e}^*$ is independent of choosing $\mathbf{S}_{(\mathbf{t})},\ \mathbf{S}^*_{(\mathbf{t})}$.

Again, due to the rejection sampling technique used at Step 12, Phase 3, the distribution of $\mathbf{z}$ and $\mathbf{z}^*$ is the same, hiding the real ephemeral key of $\mathbf{S}_{(\mathbf{t})}$ and $\mathbf{S}^*_{(\mathbf{t})}$ that is being used. The lemma follows. □

Theorem 3.4.3 (Forward–secure Unforgeability)**.** *Suppose that the commitment function* COM *used in* Π_{FSBS} *is computationally binding and that there exists a forger* $\mathcal{A}$*, who can break the forward–secure unforgeability of* Π_{FSBS} *with success probability* succ_A*, using at most* Q_H *plain hash queries and at most* Q_S *signing queries. Let* γ *be the probability of a restart in* $\mathsf{FSBS.Sign}$*. Then, one can construct a polynomial–time algorithm* $\mathcal{B}$ *that solves a* $(q, n, (1+2\ell)m, \beta)$–$\mathsf{SIS}$ *instance with*

$$\beta = \max\{(2\sigma_3 + 2\sigma \cdot \sqrt{\kappa}) \cdot \sqrt{(1+\ell)\cdot m},\ (2\sigma_3 + \sigma_2) \cdot \sqrt{(1+\ell)\cdot m}\}$$

with the success probability succ_B *such that*

$$\mathsf{succ}_B \geq \left(1 - \frac{1}{T}\right) \cdot \min\left\{(1-\gamma)\cdot\left(\mathsf{succ}_A - \frac{1}{|\mathcal{R}_H|}\right)\left(\frac{\mathsf{succ}_A - \frac{1}{|\mathcal{R}_H|}}{Q} - \frac{1}{|\mathcal{R}_H|}\right), \mathsf{succ}_A \cdot \left(1 - \frac{1}{|\mathcal{R}_H|}\right)\right\},$$

where $Q := Q_S + Q_H$*.*

Proof. We prove the theorem through a reduction from an instance of the SIS problem exploiting the oracle replay attack and the forking lemma (Section 3.2.3) in ROM. Specifically, let $\mathcal{A}$ be a forger that can break the FSUF security of Π_{FSBS}. Now, the SIS solver $\mathcal{B}$, given an SIS instance, uses the instance to simulate the environment of the game $\mathsf{FSBS}^{\mathsf{FSUF}}_{\mathcal{A}}(\lambda)$ and plays as the challenger with the forger $\mathcal{A}$. The solver $\mathcal{B}$ runs $\mathcal{A}$ twice, each of which produces a forgery. Combining these two forgeries allows extracting a solution to the SIS instance. The details are below.

Instance. The solver $\mathcal{B}$ is given an instance $\mathbf{F} \xleftarrow{\$} \mathbb{Z}_q^{n\times(1+2\ell)m}$ of the $(q, n, (1+2\ell)m, \beta)$–$\mathsf{SIS}$ problem. Specifically, the instance requires to find $\mathbf{v} \in \mathbb{Z}^{(1+2\ell)m}$ such that $\|\mathbf{v}\| \leq \beta$ with $\beta = \max\{(2\sigma_3 + 2\sigma\sqrt{\kappa})\sqrt{(1+\ell)m},\ (2\sigma_3+\sigma_2)\sqrt{(1+\ell)m}\}$ and that

$$\mathbf{F}\cdot\mathbf{v} = 0 \ (\mathrm{mod}\ q). \tag{3.3}$$

Moreover, $\mathbf{F}$ can be split as $\mathbf{F} = \left[\mathbf{A}_0|\mathbf{U}_1^{(0)}|\mathbf{U}_1^{(1)}|\cdots|\mathbf{U}_\ell^{(0)}|\mathbf{U}_\ell^{(1)}\right]$ with $\mathbf{A}_0$, $\mathbf{U}_i^{(b)} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ for all $i \in [\ell]$ and for all $b \in \{0,1\}$.

Guessing the target. $\mathcal{B}$ guesses the target time point $\mathbf{t}^*$ that $\mathcal{A}$ wants to attack by choosing randomly $\mathbf{t}^* = (t_1^*, \cdots, t_\ell^*) \xleftarrow{\$} \{0, \cdots, T-1\}$. The probability of guessing a correct $\mathbf{t}^*$ is $1 - \frac{1}{T}$.

Initialise. $\mathcal{B}$ simulates the environment of the FSUF game by setting public parameters pp as in the $\mathsf{FSBS.Setup}$ algorithm. Nevertheless, the public key $\mathsf{pk} = \{\mathbf{A}_0, \mathbf{A}_1^{(0)}, \mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_\ell^{(0)}, \mathbf{A}_\ell^{(1)}, \mathbf{K}\}$ is set accordingly to the target $\mathbf{t}^* = (t_1^*, \cdots, t_\ell^*)$ via the following:

- Set $\mathbf{A}_i^{(t_i^*)} \leftarrow \mathbf{U}_i^{(t_i^*)}$ for all $i \in [\ell]$. For all $i \in [\ell]$ and for each bit $b \in \{0,1\}$ such that $b \neq t_i^*$, run $\mathsf{TrapGen}(1^n, 1^m, q)$ to generate $(\mathbf{A}_i^{(b)}, \mathbf{T}_{\mathbf{A}_i^{(b)}})$, where $\mathbf{T}_{\mathbf{A}_i^{(b)}}$ is a short basis of $\Lambda_q^\perp(\mathbf{A}_i^{(b)})$.

- To set matrix $\mathbf{K}$, $\mathcal{B}$ first sets $\mathbf{F}_{(\mathbf{t}^*)} = \left[\mathbf{A}_0|\mathbf{A}_1^{(t_1^*)}|\cdots|\mathbf{A}_\ell^{(t_\ell^*)}\right] \in \mathbb{Z}_q^{n\times(1+\ell)m}$ then samples $\mathbf{S}^* \leftarrow D_\sigma^{(1+\ell)m\times k}$ and finally assigns $\mathbf{K} := \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{S}^* \pmod q$. Let $d := \sigma\sqrt{(1+\ell)m}$. Here, we should choose σ to fulfil Lemma 2.3.3 (i.e., $\sigma \geq \omega(\sqrt{\log((1+\ell)m)})$), Lemma 2.4.5 (i.e., $d \gg q^{(1+\ell)m/n}$) and Lemma 3.4.1 (i.e., $(1+\ell)m > 64 + n\log q/\log(2d+1)$). Notice that Lemma 2.3.4 says $\|\mathbf{S}^*\| \leq d$ with overwhelming probability. According to Lemma 2.3.3, matrix $\mathbf{K}$ is statistically close to uniform over $\mathbb{Z}_q^{n\times k}$.

Suppose $\mathcal{A}$ makes up to Q_H plain hash queries and Q_S signing queries, each signing query in turn needs one hash query. Then the number of hash queries in total is $Q := Q_H + Q_S$. Then, $\mathcal{B}$ also prepares the list of replies for Q hash queries $\mathcal{L}_1 := \{\mathbf{r}_1, \cdots, \mathbf{r}_Q\}$, where each $\mathbf{r}_i \xleftarrow{\$} \mathcal{R}_H$. $\mathcal{B}$ then chooses a random tape ρ and runs $\mathcal{A}$ twice on $(\mathsf{pp}, \mathsf{pk}, \rho)$ in a black–box manner.

First execution of $\mathcal{A}$. $\mathcal{B}$ plays the role of signer and interacts with $\mathcal{A}$ as follows:

- **Setup.** The public parameters pp, and public key $\mathsf{pk} \leftarrow \{\mathbf{A}_0, \mathbf{A}_1^{(0)}, \mathbf{A}_1^{(1)}, \cdots, \mathbf{A}_\ell^{(0)}, \mathbf{A}_\ell^{(1)}, \mathbf{K}\}$ will be given to $\mathcal{A}$, while $\mathbf{T}_{\mathbf{A}_i^{(b)}}$'s and $\mathbf{S}^*$ are kept secret.

  In order to respond $\mathcal{A}$'s queries in ROM, $\mathcal{B}$ also creates and maintains a list $\mathcal{L}_H$ with

  $$\mathcal{L}_H = \{(\mathbf{u}, \mathbf{c}, \mathbf{e}') \in \mathbb{Z}_q^n \times \{0,1\}^n \times \mathcal{R}_H : \mathbf{e}' := H(\mathbf{u}, \mathbf{c})\}.$$

  That is, $\mathcal{L}_H$ comprises random oracle queries $(\mathbf{u}, \mathbf{c}) \xleftarrow{\$} \mathbb{Z}_q^n \times \{0,1\}^n$ and their corresponding hash values $\mathbf{e}' \in \mathcal{R}_H$.

- **Queries.** $\mathcal{B}$ responds to $\mathcal{A}$'s queries as follows:

  - *Queries to Key Update Oracle* $\mathsf{KQ}(\mathbf{t})$, $\mathbf{t} = (t_1, \cdots, t_\ell) \in \{0,1\}^\ell$: The query is aborted if $\mathbf{t} \leq \mathbf{t}^*$. Suppose that $\mathbf{t} > \mathbf{t}^*$ and that $k \leq \ell$ is the minimum index such that $t_k \neq t_k^*$. Then, use $\mathbf{T}_{\mathbf{A}_k^{(t_k)}}$ to compute the key $\mathbf{T}_{t_k}$ for the node corresponding to bit t_k. Precisely, compute

    $$\mathbf{T}'_{t_k} \leftarrow \mathsf{ExtBasisLeft}([\mathbf{E}|\mathbf{A}_k^{(t_k)}], \mathbf{T}_{\mathbf{A}_k^{(t_k)}}), \text{ where } \mathbf{E} = \left[\mathbf{A}_0|\mathbf{A}_1^{(t_1)}|\cdots|\mathbf{A}_{k-1}^{(t_{k-1})}\right],$$

    and

    $$\mathbf{T}_{t_k} \leftarrow \mathsf{RandBasis}([\mathbf{E}|\mathbf{A}_k^{(t_k)}], \mathbf{T}'_{t_k}, \sigma_0).$$

    Using $\mathbf{T}_{t_k}$, compute all keys in $\mathsf{sk}_\mathbf{t}$ as in $\mathsf{FSBS.KeyUp}$.

  - *Queries to Hashing Oracle* $\mathsf{HQ}(\mathbf{u}, \mathbf{c})$*:* Being queried with $(\mathbf{u}, \mathbf{c})$, $\mathcal{B}$ checks if the list $\mathcal{L}_H$ has the query already. If it does, $\mathcal{B}$ references to the corresponding hash value $\mathbf{e}'$ and forwards the value to the forger. If it does not, $\mathcal{B}$ takes a $\mathbf{r}_i \in \mathcal{L}_1$ that has not been used so far. $\mathcal{B}$ then assigns $\mathbf{e}' := \mathbf{r}_i$ and add the tuple $(\mathbf{u}, \mathbf{c}, \mathbf{e}')$ into the list $\mathcal{L}_H$. Finally, $\mathcal{B}$ responds the forger $\mathcal{A}$ with $\mathbf{e}'$.

– *Queries to Signing Oracle* $\mathsf{SQ}(\mathbf{t},\mu)$, $\mathbf{t} = (t_1,\cdots,t_\ell) \in \{0,1\}^\ell$: $\mathcal{B}$ constructs the matrix $\mathbf{F}_{(\mathbf{t})} := \left[\mathbf{A}_0|\mathbf{A}_1^{(t_1)}|\cdots|\mathbf{A}_\ell^{(t_\ell)}\right]$. In order to jointly sign on μ with $\mathcal{A}$, $\mathcal{B}$ needs to generate $\mathbf{S}_{(\mathbf{t})}$ as in Step 02, Phase 1. This is done depending on whether $\mathbf{t} = \mathbf{t}^*$.

  * If $\mathbf{t} \neq \mathbf{t}^*$, i.e., there exists a minimum index $k \leq \ell$ such that $t_k \neq t_k^*$. Then, $\mathcal{B}$ can compute $\mathbf{T}'_{\mathbf{F}_{(\mathbf{t})}} \leftarrow \mathsf{ExtBasisLeft}(\mathbf{F}_{(\mathbf{t})}, \mathbf{T}_{\mathbf{A}_k^{(t_k)}})$, $\mathbf{T}_{\mathbf{F}_{(\mathbf{t})}} \leftarrow \mathsf{RandBasis}(\mathbf{F}_{(\mathbf{t})}, \mathbf{T}'_{\mathbf{F}_{(\mathbf{t})}}, \sigma_0)$, and then $\mathbf{S}_{(\mathbf{t})} \leftarrow \mathsf{SampleD}(\mathbf{F}_{(\mathbf{t})}, \mathbf{T}_{\mathbf{F}_{(\mathbf{t})}}, \mathbf{K}, \sigma)$ (i.e., it holds that $\mathbf{F}_{(\mathbf{t})} \cdot \mathbf{S}_{(\mathbf{t})} = \mathbf{K} \pmod q$).
  * If $\mathbf{t} = \mathbf{t}^*$, $\mathcal{B}$ sets $\mathbf{S}^*$ to $\mathbf{S}_{(\mathbf{t}^*)}$ because $\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{S}^* = \mathbf{K} \pmod q$.

– *Query to Break–in Oracle* $\mathsf{BQ}(\mathbf{t}_0)$: If $\mathbf{t}_0 \leq \mathbf{t}^*$, then the query is aborted. If $\mathbf{t}_0 > \mathbf{t}^*$, then the break–in time, say $\bar{\mathbf{t}}$, can be set to $\mathbf{t}_0$ (i.e., $\bar{\mathbf{t}} := \mathbf{t}_0$). Note that because $\bar{\mathbf{t}} = \mathbf{t}_0 > \mathbf{t}^*$ then $\mathcal{B}$ can compute the secret key $\mathsf{sk}_{(\bar{\mathbf{t}})}$ in the same way as when replying to $\mathsf{KQ}(\mathbf{t})$ described earlier.

- **Forge.** Eventually, the forger $\mathcal{A}$ outputs a forgery $(\mathbf{t}'_1, \mu_1^*, \Sigma_1^*)$. $\mathcal{B}$ aborts the game if $\mathbf{t}'_1 \neq \mathbf{t}^*$. Otherwise, $\mathcal{B}$ accepts the forgery, which is valid in the following sense:

  (i) $\Sigma_1^* = (\mathbf{d}'_1, \mathbf{e}'_1, \mathbf{z}'_1)$.

  (ii) $\mathbf{e}'_1 := H(\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z}'_1 - \mathbf{K} \cdot \mathbf{e}'_1 \pmod q, \mathsf{COM}(\mu_1^*, \mathbf{d}'_1))$, where $\mathbf{F}_{(\mathbf{t}^*)} := \left[\mathbf{A}_0|\mathbf{A}_1^{(t_1^*)}|\cdots|\mathbf{A}_\ell^{(t_\ell^*)}\right] \in \mathbb{Z}^{n\times(1+\ell)m}$.

  (iii) $\|\mathbf{z}'_1\| \leq \sigma_3 \sqrt{(1+\ell)m}$.

  Let $I \in [Q]$ be the index such that $\mathbf{e}'_1 = \mathbf{r}_I$.

Second execution of $\mathcal{A}$. $\mathcal{B}$ runs $\mathcal{A}$ again on the same input pp, pk, ρ and the same hash queries as in the first execution. However, this time $\mathcal{B}$ samples new fresh $\{\mathbf{r}'_I, \cdots, \mathbf{r}'_Q\} \xleftarrow{\$} \mathcal{R}_H$ and uses the list $\mathcal{L}_2 := \{\mathbf{r}_1, \cdots, \mathbf{r}_{I-1}, \mathbf{r}'_I, \cdots, \mathbf{r}'_Q\}$ to reply to $\mathcal{A}$'s hash queries. Suppose that in this execution, $\mathcal{A}$ eventually outputs a new valid signature $(\mathbf{t}'_2, \mu_2^*, \Sigma_2^*)$, where $\Sigma_2^* = (\mathbf{d}'_2, \mathbf{e}'_2, \mathbf{z}'_2)$. Same as before, if $\mathbf{t}'_2 \neq \mathbf{t}^*$, then $\mathcal{B}$ aborts. Otherwise, $\mathcal{B}$ accepts the signature. That is, we have

(i) $\Sigma_2^* = (\mathbf{d}'_2, \mathbf{e}'_2, \mathbf{z}'_2)$.

(ii) $\mathbf{e}'_2 := H(\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z}'_2 - \mathbf{K} \cdot \mathbf{e}'_2 \pmod q, \mathsf{COM}(\mu_2^*, \mathbf{d}'_2))$, where $\mathbf{F}_{(\mathbf{t}^*)} := \left[\mathbf{A}_0|\mathbf{A}_1^{(t_1^*)}|\cdots|\mathbf{A}_\ell^{(t_\ell^*)}\right] \in \mathbb{Z}^{n\times(1+\ell)m}$.

(iii) $\|\mathbf{z}'_2\| \leq \sigma_3 \sqrt{(1+\ell)m}$.

It is easy to see that $\mathcal{B}$ is statistically perfect in playing the role of the real challenger in the FSUF game. In fact, this is because that $\mathcal{B}$ interacts with $\mathcal{A}$ almost following the real FSUF game with some exceptions as follows:

1. The public key generated by $\mathcal{B}$ has some matrices $\mathbf{A}_i^{(b)}$ that are not truly random but are generated by the algorithm $\mathsf{TrapGen}$. However, Lemma 2.5.4 guarantees that the distribution of such $\mathbf{A}_i^{(b)}$'s is negligibly far from uniform. This means that at this point, $\mathcal{A}$ is unable to distinguish the simulated game from the real one.

2. Again, in the public key of the real game, the matrix $\mathbf{K}$ is uniformly random. Instead, $\mathbf{K} := \mathbf{F} \cdot \mathbf{S}^* \pmod q$, where $\mathbf{S}^*$ is sampled from $D_\sigma^{(1+\ell)m \times m}$. However, Lemma 2.3.3 claims that the distribution of such a $\mathbf{K}$ is close to uniform with a careful choice of σ. Notice that the choice of σ does not affect (iii) above. This again means that at this point, $\mathcal{A}$ is unable to distinguish the simulated game from the real one.

3. When responding to query to Signing Oracle with $\mathbf{t} = \mathbf{t}^*$, the matrix $\mathbf{S}_{(\mathbf{t}^*)}$ is set to $\mathbf{S}^*$, which is not computed using $\mathsf{SampleD}$ but is sampled from $D_\sigma^{(1+\ell)m \times k}$. However, the point is that the forger $\mathcal{A}$ does not know any $\mathbf{S}_{(\mathbf{t})}$ (hence $\mathbf{S}^*$). Additionally, $\mathbf{z}$ generated in Step 12 at Phase 3 is outputted through the rejection sampling, which makes sure that $\mathbf{z} \sim D_{\sigma_2}^{(\ell+1)m}$ and that $\mathbf{z}$ is independent of any $\mathbf{S}_{(\mathbf{t})}$ and $\mathbf{S}^*$. Therefore, the view of $\mathcal{A}$ is independent of $\mathbf{S}^*$ in this case. As a result, $\mathcal{A}$ is unable to distinguish the simulated game from the real one.

Now, we are going to show the way for $\mathcal{B}$ to extract the solution to the SIS instance given in Equation (3.3). Notice that after two executions of $\mathcal{A}$ on the same random tape and the same hash queries, if $\mathbf{e}'_2 = \mathbf{e}'_1$, then $\mathcal{B}$ has to rerun $\mathcal{A}(\mathsf{pp}, \mathsf{pk}, \rho')$ using different random tapes ρ' and different hash queries. Fortunately, the forking lemma (Section 3.2.3) asserts that the probability that $\mathbf{e}'_1 = \mathbf{r}_I$, $\mathbf{e}'_2 = \mathbf{r}'_I$ and $\mathbf{e}'_2 \neq \mathbf{e}'_1$ using the same hash query as in the first execution (i.e., the I–th hash query) is at least

$$(1-\gamma) \cdot \left(\mathsf{succ}_A - \frac{1}{|\mathcal{R}_H|} \right) \left(\frac{\mathsf{succ}_A - \frac{1}{|\mathcal{R}_H|}}{Q} - \frac{1}{|\mathcal{R}_H|} \right).$$

Here the factor $1-\gamma$ is due to the probability of a restart.

Now, $\mathcal{B}$ returns the pairs

$$((\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z}'_1 - \mathbf{K} \cdot \mathbf{e}'_1, \mathsf{COM}(\mu_1^*, \mathbf{d}'_1)), (\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z}'_2 - \mathbf{K} \cdot \mathbf{e}'_2, \mathsf{COM}(\mu_2^*, \mathbf{d}'_2)). \tag{3.4}$$

Because the pair in Equation (3.4) are truly the same hash query and COM is computationally binding, we have $\mu_2^* = \mu_1^*$, $\mathbf{d}'_1 = \mathbf{d}'_2$ and

$$\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z}'_1 - \mathbf{K} \cdot \mathbf{e}'_1 = \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z}'_2 - \mathbf{K} \cdot \mathbf{e}'_2 \pmod q,$$

or equivalently,

$$\mathbf{F}_{(\mathbf{t}^*)} \cdot (\mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{S}^* \cdot (\mathbf{e}'_1 - \mathbf{e}'_2)) = \mathbf{0} \ (\mathrm{mod}\ q).$$

Set $\widehat{\mathbf{v}} := \mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{S}^* \cdot (\mathbf{e}'_1 - \mathbf{e}'_2)$.

At this point, we follow [Lyu11, Proof of Lemma 5.4] to show that $\widehat{\mathbf{v}} := \mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{S}^* \cdot (\mathbf{e}'_1 - \mathbf{e}'_2) \neq 0$. In fact, if $\mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{S}^* \cdot (\mathbf{e}'_1 - \mathbf{e}'_2) = 0$ then we can replace $\mathbf{S}^*$ by another one, say $\mathbf{S}'$, as follows. Let j be the index at which $\mathbf{e}'_1[j] \neq \mathbf{e}'_2[j]$. By Lemma 3.4.1, there exists another vector, say $\mathbf{s}'$, such that $\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{S}^*[j] = \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{s}' \pmod q$. Recall that $\mathbf{S}^*[j]$ is the j–th column of $\mathbf{S}^*$. Now, we form the matrix $\mathbf{S}'$ from $\mathbf{S}^*$ by replacing $\mathbf{S}^*[j]$ with $\mathbf{s}'$. This way guarantees that $\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{S}^* = \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{S}' \pmod q$, where $\mathbf{S}^*$ and $\mathbf{S}'$ have all the same columns except the j–th column. Further, this way also ensures that if $\mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{S}^* \cdot (\mathbf{e}'_1 - \mathbf{e}'_2) = \mathbf{0}$, then $\widehat{\mathbf{v}}' := \mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{S}' \cdot (\mathbf{e}'_1 - \mathbf{e}'_2) \neq \mathbf{0}$. This is because if $\mathbf{z}'_1 - \mathbf{z}'_2 - \mathbf{S}' \cdot (\mathbf{e}'_1 - \mathbf{e}'_2) = \mathbf{0}$, then it must be $\mathbf{S}' = \mathbf{S}^*$. This will be discussed in detail in Section 3.5 below. Stress that the view of $\mathcal{A}$ is independent of both $\mathbf{S}^*$ and $\mathbf{S}'$ by Lemma 3.4.2. We have shown that $\widehat{\mathbf{v}} \neq \mathbf{0}$ and $\mathbf{F}_{(\mathbf{t}^*)} \cdot \widehat{\mathbf{v}} = \mathbf{0} \pmod q$. We can easily check that $\|\widehat{\mathbf{v}}\| \leq 2(\sigma_3 + \sigma\sqrt{\kappa})\sqrt{(1+\ell)m}$, as $\|\mathbf{S}^*\| \leq \sigma\sqrt{(1+\ell)m}$, $\|\mathbf{z}'_i\| \leq \sigma_3\sqrt{(\ell+1)m}$, and $\|\mathbf{e}'_i\| \leq \sqrt{\kappa}$ for $i \in \{1,2\}$.

Particularly, we prove that if $\mathcal{A}$ can produce a forgery by restarting the signing interaction (with $\mathcal{B}$), then $\mathcal{B}$ is able to find a solution to the SIS problem given by Equation (3.3). Indeed, to restart the signing interaction, $\mathcal{A}$ delivers result:= $(\mathbf{a}, \mathbf{b}, \mathbf{e}', \mathbf{c})$ to $\mathcal{B}$. Now $\mathcal{B}$ with its view $\mathcal{V} = (\mathbf{t}, \mathbf{x}, \mathbf{e}, \mathbf{z})$, will check whether all the following equations

$$\mathbf{e} - \mathbf{b} = \mathbf{e}' = H(\mathbf{x} + \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{a} + \mathbf{K} \cdot \mathbf{b} \pmod q, \mathbf{c}), \tag{3.5}$$

$$\mathbf{e}' = H(\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{a} + \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z} - \mathbf{K} \cdot \mathbf{e}' \pmod q, \mathbf{c}), \tag{3.6}$$

$$\|\mathbf{z} + \mathbf{a}\| > \sigma_3\sqrt{(1+\ell)m} \tag{3.7}$$

hold or not. If yes, $\mathcal{B}$ restarts the interaction with $\mathcal{A}$. Let assume that afterwards $\mathcal{A}$ successfully produces a valid signature $\widehat{\Sigma} = (\widehat{\mathbf{d}'}, \widehat{\mathbf{e}'}, \widehat{\mathbf{z}'})$. Let $\widehat{\mathbf{b}} \in D^m_{\sigma_1}$ be such that $\mathbf{e} = \widehat{\mathbf{e}'} + \widehat{\mathbf{b}}$. Then, the following relations have to hold

$$\mathbf{e} - \widehat{\mathbf{b}} = \widehat{\mathbf{e}'} = H(\mathbf{x} + \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{a} + \mathbf{K} \cdot \widehat{\mathbf{b}} \pmod q, \mathbf{c}), \tag{3.8}$$

$$\widehat{\mathbf{e}'} = H(\mathbf{F}_{(\mathbf{t}^*)} \cdot \widehat{\mathbf{z}'} - \mathbf{K} \cdot \widehat{\mathbf{e}'} \pmod q, \mathsf{COM}(\mu^*, \widehat{\mathbf{d}'})), \tag{3.9}$$

$$\|\widehat{\mathbf{z}'}\| \leq \sigma_3\sqrt{(1+\ell)m}. \tag{3.10}$$

If $\widehat{\mathbf{e}'} = \mathbf{e}'$, then Equations (3.6)–(3.9) give $\mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{a} + \mathbf{F}_{(\mathbf{t}^*)} \cdot \mathbf{z} \pmod q = \mathbf{F}_{(\mathbf{t}^*)} \cdot \widehat{\mathbf{z}'} \pmod q$. Let $\widehat{\mathbf{v}} := \mathbf{a} + \mathbf{z} - \widehat{\mathbf{z}'}$, then $\widehat{\mathbf{v}} \neq 0$. This is true as otherwise $\mathbf{a} + \mathbf{z} = \widehat{\mathbf{z}'}$, which implies that $\|\mathbf{z} + \mathbf{a}\| \leq \sigma_3\sqrt{(1+\ell)m}$ (by Equation (3.10)). This contradicts Equation (3.7). Again, we have $\mathbf{F}_{(\mathbf{t}^*)} \cdot \widehat{\mathbf{v}} = \mathbf{0} \pmod q$, $\widehat{\mathbf{v}} \neq \mathbf{0}$ and $\|\widehat{\mathbf{v}}\| \leq \|\mathbf{a}\| + \|\mathbf{z}\| + \|\widehat{\mathbf{z}'}\| \leq (2\sigma_3 + \sigma_2)\sqrt{(1+\ell)m}$.

Now, if $\widehat{\mathbf{e}'} \neq \mathbf{e}'$, then $\mathcal{A}$ may hide $\widehat{\mathbf{e}'}$ in $\mathbf{e}$. Then $\mathbf{e} = \mathbf{e}' + \mathbf{b} = \widehat{\mathbf{e}'} + \widehat{\mathbf{b}'}$ for a $\widehat{\mathbf{b}'} \neq \mathbf{b}$. The

success probability for this case is at least

$$\mathsf{succ}_A \cdot \left(1 - \frac{1}{|\mathcal{R}_H|}\right),$$

because $\mathcal{A}$ had to predict the output of H to prepare $\mathbf{b}$.

Note that $\mathbf{F}_{(\mathbf{t}^*)} = \left[\mathbf{A}_0 | \mathbf{A}_1^{(t_1^*)} | \cdots | \mathbf{A}_\ell^{(t_\ell^*)}\right] = \left[\mathbf{A}_0 | \mathbf{U}_1^{(t_1^*)} | \cdots | \mathbf{U}_\ell^{(t_\ell^*)}\right]$. We can get $\mathbf{F}$ from $\mathbf{F}_{(\mathbf{t}^*)}$ by inserting into the gap between two sub–matrices in $\mathbf{F}_{(\mathbf{t}^*)}$ the remaining matrices $\{\mathbf{U}_i^{(1-t_i^*)}\}_i$ at relevant positions. We insert zeros into the corresponding position of $\widehat{\mathbf{v}}$ to get the desired solution $\mathbf{v}$ to the problem given by Equation (3.3). Obviously, $\mathbf{F} \cdot \mathbf{v} = \mathbf{0} \pmod q$, and $\|\mathbf{v}\| = |\widehat{\mathbf{v}}|$.

To summarise, we have shown that $\mathcal{B}$ can solve the $(q, n, (1+2\ell)m, \beta)$–SIS problem, with

$$\beta = \max\{(2\sigma_3 + 2\sigma\sqrt{\kappa})\sqrt{(1+\ell)m},\ (2\sigma_3 + \sigma_2)\sqrt{(1+\ell)m}\}.$$

□

3.4.4 Setting Parameters

Parameters for the FSBS system Π_{FSBS} are set heuristically as follows:

- First, we set λ as the security parameter, ℓ as the highest depth of the binary tree representing time points, $T = 2^\ell$ as the number of time points.
- Choose n, q, m and Gaussian parameters σ, σ_0, σ_1, σ_2, σ_3 such that TrapGen (Item 1, Lemma 2.5.4), ExtBasisLeft (Item 3, Lemma 2.5.4), SampleD (Item 4, Lemma 2.5.4), RandBasis (Item 5, Lemma 2.5.4) can work well. Specifically, we should choose $\sigma \geq \|\widetilde{\mathbf{T}_{\mathbf{A}_0}}\| \cdot \omega(\sqrt{\log((\ell+1)\cdot m)})$, $\sigma_1 = 12\|\mathbf{e}'\| = 12\sqrt{\kappa}$, $\sigma_2 = 12\|\mathbf{S}^* \cdot \mathbf{e}\| = 12\sigma\sigma_1\sqrt{(1+\ell)mk}$ and $\sigma_3 = 12\|\mathbf{z}\| = 12\sigma_2\sqrt{(1+\ell)m}$ (via Remark 2.3.1). Also, we choose $\sigma_{0,i} \geq \|\widetilde{\mathbf{T}_{\mathbf{A}_0}}\| \cdot \omega(\sqrt{\log((i+1)\cdot m)})$. Note that $m = \lceil 6n \log q \rceil$, $\|\widetilde{\mathbf{T}_{\mathbf{A}_0}}\| = O(\sqrt{n \log q}) = O(\sqrt{m})$.
- Due to Lemma 3.4.1, we should choose $(\ell+1)m > 64 + n\log q / \log(2d+1)$. Here $d = \sigma\sqrt{(1+\ell)m}$ by Lemma 2.3.4.
- To make sure the cardinality $|\mathcal{R}_H| \geq 2^\zeta$ for a desired positive integer ζ, we should choose positive integers k and κ such that

$$2^\kappa \cdot \binom{k}{\kappa} \geq 2^\zeta.$$

- Section 3.4.2 suggests setting $M_i := e^{1+1/288}$ for all $i \in [3]$.

- For $(q,n,(1+\ell)m,\beta)$–SIS to be hard by Lemma 2.4.4, we set m poly–bounded, $\beta = \mathsf{poly}(n)$ and $q \geq \beta \cdot \omega(\sqrt{n \log n})$, where $\beta = \max\{(2\sigma_3 + 2\sigma\sqrt{\kappa})\sqrt{(1+\ell)m}, (2\sigma_3 + \sigma_2)\sqrt{(1+\ell)m}\}$.

3.5 Discussion on the Validity of the Proof of Theorem 3.4.3

Hauck *et al.* in [HKLN20] have mentioned subtle flaws in existing lattice blind signature schemes including [Rüc10] and the paper [LSK+19]. The main problem that [HKLN20] showed is as follows. Rückert [Rüc10] reduces the one–more unforgeability security to the hardness of a collision problem, which is denoted as $Col(\mathcal{H}(R,m),D)$ in the Rückert's work. We omit its notational detail here. To prove the one–more unforgeability of the lattice–based blind signature, [Rüc10] applies the general forking lemma (see Section 3.2.3) to rewind the forger to get χ and χ' such that $h(\chi) = h(\chi')$ and applies the witness indistinguishability following [PS00, Lemma 8] to assure $\chi \neq \chi'$. Here h is an instance of $Col(\mathcal{H}(R,m),D)$. Obviously, $\chi \neq \chi'$ leads to a collision of h. In the lattice setting (e.g., in our work [LDS+20] and in this chapter), the collision implies a non–trivial solution to the SIS problem.

However, [HKLN20] reckons that using the witness indistinguishability and [PS00, Lemma 8] is insufficient to prove $\chi \neq \chi'$. Instead, [HKLN20] argued that one must apply [PS00, Lemma 9] and subsequent parts of Pointcheval and Stern's to prove $\chi \neq \chi'$. They also discussed that Lemma 8 and Lemma 9 of [PS00] only apply to the Okamoto–Schnorr scheme, not directly to other schemes. They concluded that the security proof for the OMUF of [Rüc10] (hence [LSK+19][a]) is not valid.

Fortunately, even though [LDS+20] (hence this chapter) follows the framework of [Rüc10], our argument for the forward–secure unforgeability security is quite different and much simpler. We deploy Lemma 3.4.1 and follow an argument by Lyubashevsky [Lyu11, Proof of Lemma 5.4]. We, therefore, do not rely on the argument of [PS00]. For ease of following, we summarise the idea below.

Suppose that the matrix $\mathbf{F}$ is an (q,n,m,β)–SIS instance. ($\mathbf{F}$ may be a part of a larger SIS instance.) Suppose further that after rewinding and from two forgeries, one gets $\chi_1 := \mathbf{z}_1 - \mathbf{S}\cdot\mathbf{c}_1$, $\chi_2 := \mathbf{z}_2 - \mathbf{S}\cdot\mathbf{c}_2$ with $\mathbf{c}_1 \neq \mathbf{c}_2$ satisfying that $\mathbf{F}\cdot\chi_1 = \mathbf{F}\cdot\chi_2 \pmod q$. The main goal is to prove $\chi_1 \neq \chi_2$ (with some probability), which implies a non-trivial of the SIS instance by choosing parameters such that $\|\chi_1 - \chi_2\| \leq \beta$. What we should do is as follows:

[a] However, we saw that [LSK+19] is free from the flaw because its proof does not apply the approach of [Rüc10]. We omit the detail because it is irrelevant to this thesis' content.

- Let $j \in [m]$ be the index at which $\mathbf{c}_1 \neq \mathbf{c}_2$, i.e., $\mathbf{c}_1[j] \neq \mathbf{c}_2[j]$.
- Of course, if $\chi_1 \neq \chi_2$ then everything is done.
- However, if $\chi_1 = \chi_2$, i.e.,

$$\mathbf{z}_1 - \mathbf{S} \cdot \mathbf{c}_1 = \mathbf{z}_2 - \mathbf{S} \cdot \mathbf{c}_2 \tag{3.11}$$

  then replace the matrix $\mathbf{S}$ by another matrix $\mathbf{S}'$. Here, $\mathbf{S}'$ is the same as $\mathbf{S}$ at all columns but the column j. We can do this by: (i) first choosing another vector $\mathbf{s}' \neq \mathbf{S}[j]$ subject to $\mathbf{F} \cdot \mathbf{s}' = \mathbf{F} \cdot \mathbf{S}[j] \pmod{q}$ (owing to Lemma 3.4.1); (ii) then setting $\mathbf{S}'$ to be $\mathbf{S}$ with the replacement the column $\mathbf{S}[j]$ of $\mathbf{S}$ by $\mathbf{s}'$. Now, let $\chi_1' := \mathbf{z}_1 - \mathbf{S}' \cdot \mathbf{c}_1$, $\chi_2' := \mathbf{z}_2 - \mathbf{S}' \cdot \mathbf{c}_2$. Notice that the forger is not aware of either $\mathbf{S}$ or $\mathbf{S}'$ being used due to the WI property (presented in Lemma 3.4.2).
- Prove $\chi_1' \neq \chi_2'$ by contradiction: If $\chi_1' = \chi_2'$ then we would get

$$\mathbf{z}_1 - \mathbf{S}' \cdot \mathbf{c}_1 = \mathbf{z}_2 - \mathbf{S}' \cdot \mathbf{c}_2. \tag{3.12}$$

  Subtracting Equations (3.11) and (3.12) side by side, we get $(\mathbf{S} - \mathbf{S}') \cdot (\mathbf{c}_1 - \mathbf{c}_2) = \mathbf{0}$, which is equivalent to

$$\begin{aligned}\mathbf{0} &= \begin{bmatrix} \vdots & \cdots & \vdots & & \vdots & & \vdots & \cdots & \vdots \\ 0 & \cdots & 0 & \mathbf{S}[j] - \mathbf{S}'[j] & 0 & \cdots & 0 \\ \vdots & \cdots & \vdots & & \vdots & & \vdots & \cdots & \vdots \end{bmatrix} \cdot \begin{pmatrix} \mathbf{c}_1[1] - \mathbf{c}_2[1] \\ \mathbf{c}_1[2] - \mathbf{c}_2[2] \\ \vdots \\ \mathbf{c}_1[j] - \mathbf{c}_2[j] \\ \vdots \\ \mathbf{c}_1[m-1] - \mathbf{c}_2[m-1] \\ \mathbf{c}_1[m] - \mathbf{c}_2[m] \end{pmatrix} \\ &= (\mathbf{S}[j] - \mathbf{S}'[j]) \cdot (\mathbf{c}_1[j] - \mathbf{c}_2[j]).\end{aligned}$$

  Since $\mathbf{c}_1[j] - \mathbf{c}_2[j] \neq 0$, then it must be $\mathbf{S}[j] = \mathbf{S}'[j]$, hence $\mathbf{S} = \mathbf{S}'$. This contradicts to the fact that $\mathbf{S} \neq \mathbf{S}'$.

To conclude, we have demonstrated that the proof for the forward–secure unforgeability in the paper [LDS+20] and Theorem 3.4.3 of this thesis does not suffer from the flaw shown in [HKLN20].

3.6 Summary

We have proposed the first forward–secure blind signature (FSBS) over lattices. The introduced FSBS's security relies on the short integer solution assumption. The main

tools used for the construction are the binary tree data structure used in representing the time periods, a modification of Fiat–Shamir with aborts, in which aborting is implemented using the rejection sampling, a commitment functions, a hash function, and the GPV08 trapdoor. We show that the proposed signature satisfies the blindness and the forward security unforgeability. The latter is proven in the random oracle model by exploiting the witness indistinguishability, the oracle replay attack and the forking lemma. We leave a lattice–based FSBS secure in the standard model for future research.

In Chapter 4, we will revisit the LVV19 trapdoor and develop a trapdoor delegation algorithm. Basing on that, we propose the first hierarchical identity–based encryption (Section 2.7.4) from the degree–parameterised middle–product learning with errors problem (Section 2.4).

Chapter 4

Hierarchical IBE from Degree–parameterised Middle–product LWE

Part of the content in this chapter appeared in Le et al. [LDSP20]. The author of this thesis is the first, and one of the corresponding authors of [LDSP20]. Having received the topic from his supervisors, he contributed to developing the main method, the design of the cryptosystem, the security analysis of the cryptosystem and the writing of the manuscript.

4.1 Overview

Identity– based encryption (IBE) [Sha85] is a kind of public–key encryption. An IBE has a mechanism that generates users' private keys from their identities (e.g., email addresses) which behave as public keys. There exists a hierarchical variant of IBE called hierarchical identity–based encryption (HIBE) [HL02, GS02b]. User identities in a HIBE are placed in a directed tree. In a HIBE, one can use a private key for an identity to produce a private key for its child identity. However, the reverse direction (i.e., computing a key for an identity from its child identity's private key) is impossible. One can deploy HIBE to construct broadcast encryption [DF03, YFDL04] and forward–secure encryption [CHK03].

In [LVV19], Lombardi *et al.* introduced the *LVV19 trapdoor*, which was reviewed in Section 2.5.3. The LVV19 trapdoor [LVV19] is essentially an adaptation of the MP12 trapdoor [MP12] to polynomials, which are involved in the degree–parameterised middle–product learning with errors (DMPLWE) problem. In more detail, the authors of [LVV19] used Toeplitz representation to represent a family of polynomials in such a way that they can apply the MP12 trapdoor to Toeplitz representations (that are concatenations of

Toeplitz matrices). [LVV19] introduced two algorithms called LVVGenTrap and LVVSamTrap in this thesis. The former helps generate a family of polynomials $\bar{\mathbf{a}}_\epsilon$ together with a trapdoor td_ϵ. The latter allows sampling a family of polynomials $\bar{\mathbf{r}}$ satisfying that $\langle \bar{\mathbf{a}}_\epsilon, \bar{\mathbf{r}} \rangle := \sum_{i=1}^{t+\gamma\tau} a_i \cdot r_i = u$. for a given polynomial u using the trapdoor td_ϵ. (Parameters will be described below.) Employing the trapdoor, Lombardi *et al.* [LVV19] also gave the first identity–based encryption (IBE) scheme relied on DMPLWE. However, the LVV19 trapdoor still lacks a delegation technique that helps derive a trapdoor for an expanded polynomial family. Therefore, a hierarchical identity–based encryption (HIBE) construction from DMPLWE has not been available in the literature.

In this chapter, we fill the above gap by introducing a delegation method for the LVV19 trapdoor, thanks to which we can construct the first HIBE scheme based on DMPLWE, which is provably secure in the standard model (SDM). See Figure 4.1 for an overview of our work in comparison with the [LVV19] work. For the formal definition of HIBE, please refer to Section 2.7.4.

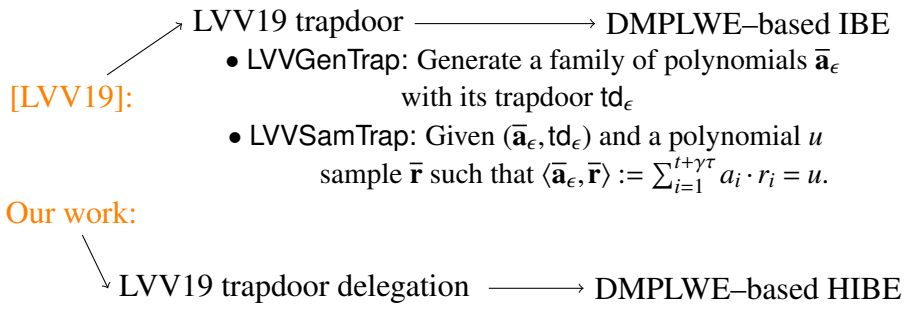

Figure 4.1: Overview of our work in comparison with [LVV19].

The primary technical point in delegating a trapdoor in the LVV19 trapdoor mechanism is to exploit the idea of MPDelTrap (Lemma 2.5.5) on Toeplitz representations. However, this should be done in a subtle (and not straightforward) way. We have to much care about the quantity and the form of Toeplitz matrices of polynomials in families that will be added to a given family. More intuitively, suppose that our goal is, given a trapdoor for $\bar{\mathbf{a}}$, to derive a trapdoor for $\bar{\mathbf{f}} := (\bar{\mathbf{a}}|\bar{\mathbf{h}})$ provided an added family $\bar{\mathbf{h}}$. Then, we will discuss, along the way, to see that in the *base case*, the number of polynomials in $\bar{\mathbf{h}}$ should be equal to that in the *gadget* family $\bar{\mathbf{g}}$ (given in Equation (4.2) below). Additionally, the Toeplitz representations for $\bar{\mathbf{h}}$ and for $\bar{\mathbf{g}}$ should be the same form. The cases that the number of polynomials in $\bar{\mathbf{h}}$ is a multiple of $\bar{\mathbf{g}}$ then can be handled recursively with some modifications to related algorithms.

Before jumping to the next sections, we define some common parameters for this chapter. We will work with parameters $n \in \mathbb{Z}^+$, a prime modulus $q = \mathsf{poly}(n)$, d, t, γ, τ, k, $m \in \mathbb{Z}^+$ satisfying that $d \leq n$, $m \geq 1$, $k \geq 1$, $dt/n = \Omega(\log n)$, $\tau := \lceil \log q \rceil$, $d\gamma = n + 2d - 2$, and $\beta := \lceil \frac{\log n}{2} \rceil$. Throughout this chapter, we consider the following *gadget* family consisting

of $\gamma\tau$ polynomials

$$\overline{\mathbf{g}} = (g_1, \cdots, g_{\gamma\tau}), \text{ with } g_j = 2^{\eta} x^{d\zeta}, \tag{4.1}$$

for $j = \zeta\tau + \eta + 1$ and $\eta \in \{0, \cdots, \tau - 1\}$, $\zeta \in \{0, \cdots, \gamma - 1\}$. Remember that

$$\mathbf{G} = [\mathsf{Tp}^{n+d-1,d}(g_1) | \cdots | \mathsf{Tp}^{n+d-1,d}(g_{\gamma\tau})] \in \mathbb{Z}_q^{d\gamma \times d\gamma\tau} \tag{4.2}$$

is a Toeplitz representation for $\overline{\mathbf{g}}$.

The primary contributions presented in this chapter are:

- proposing a delegation algorithm for the LVV19 trapdoor, given in Section 4.2, and
- introducing the first HIBE from the DMPLWE problem, presented in Section 4.3.

We exploit the main following technical tools at the core of our work:

- the MP12 trapdoor and the LVV19 trapdoor (Section 2.5.3),
- a delegation algorithm for the LVV19 trapdoor (will be proposed in Section 4.2).

In the following, we detail our contributions and techniques/tools.

LVV19 Trapdoor Delegation. Let $\overline{\mathbf{a}} = (a_1, \cdots, a_{t'})$ be a t'–family of polynomials. We can interpret any polynomial as a structured matrix, e.g. Toeplitz matrix [Pan01], and hence $\overline{\mathbf{a}}$ can be represented as a concatenated structured matrix, say $\mathbf{A}$. The trapdoor from [LVV19] is a modification for a family of polynomials of the trapdoor used in [MP12]. More specifically, in [LVV19], a trapdoor for the family $\overline{\mathbf{a}}$ is a collection td_a of *short* polynomials (here *short* means small coefficients) from which we form a matrix $\mathbf{R}$ such that $\mathbf{A} \cdot \left[\begin{smallmatrix}\mathbf{R}\\ \mathbf{I}\end{smallmatrix}\right] = \mathbf{G}$, where $\mathbf{G}$ is the concatenated structured matrix of $\overline{\mathbf{g}} = (g_1, \cdots, g_{\gamma\tau})$, namely $g_j = 2^{\eta} x^{d\zeta}$ for $j = \zeta\tau + \eta + 1$ with $\eta \in \{0, \cdots, \tau - 1\}$, $\zeta \in \{0, \cdots, \gamma - 1\}$. We call $\overline{\mathbf{g}}$ the *primitive family*. The trapdoor td_a is used to search for a t'–family of polynomials $\overline{\mathbf{r}} := (r_1, \cdots, r_{t'})$ (following some distribution that is close to uniform) such that $\langle \overline{\mathbf{a}}, \overline{\mathbf{r}} \rangle := \sum_{i=1}^{t'} a_i \cdot r_i = u$ for any given polynomial u of appropriate degree.

DMPLWE–based HIBE. For the construction of DMPLWE–based HIBE, we need to derive a trapdoor for an extended family of polynomials, say $\overline{\mathbf{f}} = (\overline{\mathbf{a}} | \overline{\mathbf{h}}) = (a_1, \cdots, a_{t'} | h_1, \cdots, h_{t''})$, from a trapdoor for $\overline{\mathbf{a}}$. To this end, we first proceed with the case $t'' = \gamma\tau$, i.e., the number of polynomials in $\overline{\mathbf{h}}$ has to be the same as the number in $\overline{\mathbf{g}}$. We transform $\overline{\mathbf{h}}$ into a matrix $\mathbf{H}$, and then apply the idea of trapdoor delegation from [MP12] to obtain the trapdoor td_f for $\overline{\mathbf{f}}$. We generalize the trapdoor delegation to the case $t'' = m\gamma\tau$, a multiple of $\gamma\tau$ for $m \geq 1$.

Using the proposed polynomial trapdoor delegation, we build the first HIBE based on DMPLWE, which is provably IND–sID–CPA secure in the standard model. To produce

a private key for an identity $\mathsf{id} = (id_1, \cdots, id_\ell)$ at depth ℓ, we form an *extended* family $\bar{\mathbf{f}}_{\mathsf{id}} = (\bar{\mathbf{a}}, \bar{\mathbf{h}}^{(1,id_1)}, \cdots, \bar{\mathbf{h}}^{(\ell,id_\ell)})$ in which each $\bar{\mathbf{h}}^{(i,\mathsf{bit})} = (h_1^{(i,\mathsf{bit})}, \cdots, h_{t'}^{(i,\mathsf{bit})})$ is a family of random polynomials. Then our trapdoor delegation helps to get a trapdoor for $\bar{\mathbf{f}}_{\mathsf{id}}$, which plays the role of the private key concerning the identity id. Deriving a private key for a child identity $\mathsf{id}|id_{\ell+1} = (id_1, \cdots, id_\ell, id_{\ell+1})$ from a parent identity $\mathsf{id} = (id_1, \cdots, id_\ell)$ is done in similar way by appending $\bar{\mathbf{h}}^{(\ell+1,id_{\ell+1})}$ to $\bar{\mathbf{f}}_{\mathsf{id}}$ so we get $\bar{\mathbf{f}}_{\mathsf{id}|id_{\ell+1}} = (\bar{\mathbf{a}}, \bar{\mathbf{h}}^{(1,id_1)}, \cdots, \bar{\mathbf{h}}^{(\ell,id_\ell)}, \bar{\mathbf{h}}^{(\ell+1,id_{\ell+1})})$. Then we use the trapdoor delegation to get its private key from the private key (trapdoor) of $\bar{\mathbf{f}}_{\mathsf{id}}$. For the security proof to work, we need to put a condition on t' such that t' is a multiple of $\gamma\tau$. Indeed, the condition ensures that the simulator can simulate an answer to a private key query of the adversary. The answer is generated using a trapdoor for some $\bar{\mathbf{h}}^{(i,id_i)}$, which is not chosen randomly but produced by a trapdoor.

Figure 4.2 shows the roadmap of this chapter.

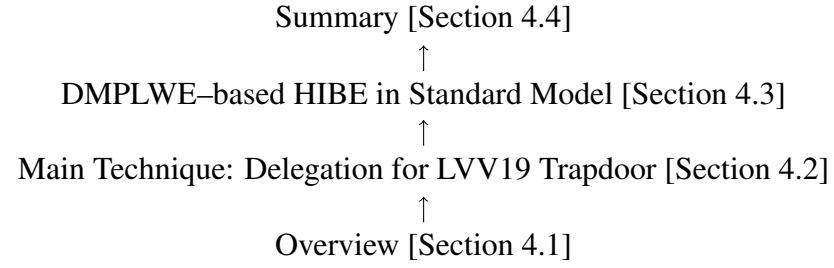

Figure 4.2: The roadmap of this chapter.

4.2 Main Technique: Delegation for LVV19 Trapdoor

In this section, we will be developing algorithms of delegating a $\bar{\mathbf{g}}$–trapdoor td (in the sense of Definition 2.5.6) for $(\bar{\mathbf{a}}|\bar{\mathbf{h}})$ given a $\bar{\mathbf{g}}$–trapdoor td_a for $\bar{\mathbf{a}} = (a_1, \cdots, a_{t_1})$ and given $\bar{\mathbf{h}} = (h_1, \cdots, h_{t_2})$, all polynomials a_i's and h_i's defined over $\mathbb{Z}_q$. Here t_1 and t_2 are two positive integers. Along the way, we will discuss to see what the values of t_1 and t_2 are and what the degrees of polynomials a_i's and h_j's should be.

Table 4.1: A description and comparison of Base Case, Middle Case and More General Case.

Case	Main Algorithm	Input	Output
Base Case	BasicLVVDelTrap	$(\bar{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t+\gamma\tau}),, \mathsf{td}_\epsilon,$ $\bar{\mathbf{h}} = (h_1, \cdots, h_{\gamma\tau}))$	trapdoor td_f for $\bar{\mathbf{f}} := (\bar{\mathbf{a}}_\epsilon\|\bar{\mathbf{h}})$
Middle Case	GenLVVDelTrap	$(\bar{\mathbf{a}} = (a_1, \cdots, a_{t+k\gamma\tau}), \mathsf{td}_a,$ $\bar{\mathbf{h}} = (h_1, \cdots, h_{\gamma\tau}))$	trapdoor td_f for $\bar{\mathbf{f}} := (\bar{\mathbf{a}}\|\bar{\mathbf{h}})$
More General Case	MoreGenLVVDelTrap	$(\bar{\mathbf{a}} = (a_1, \cdots, a_{t+k\gamma\tau}), \mathsf{td}_a,$ $\bar{\mathbf{h}} = (h_1, \cdots, h_{m\gamma\tau}))$	trapdoor td_f for $\bar{\mathbf{f}} := (\bar{\mathbf{a}}\|\bar{\mathbf{h}})$

For ease of presentation, we will consider three cases that we call *Base Case*, *Middle Case* and *More General Case*.

- In Base Case, we come up with an algorithm named BasicLVVDelTrap that takes as input $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon)$, $\overline{\mathbf{h}} = (h_1, \cdots, h_{\gamma\tau})$ to output a $\overline{\mathbf{g}}$–trapdoor for $\overline{\mathbf{f}} := (\overline{\mathbf{a}}_\epsilon | \overline{\mathbf{h}})$. Here the pair $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon)$, is directly generated by LVVGenTrap and remember that $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t+\gamma\tau})$. This algorithm uses LVVSamTrap as a procedure, which calls MPSamTrap in its course.

- We name the resulting algorithm in Middle Case by GenLVVDelTrap. Our goal is to use the algorithm to find a $\overline{\mathbf{g}}$–trapdoor td_f for $\overline{\mathbf{f}} := (\overline{\mathbf{a}} | \overline{\mathbf{h}})$ on the input of $(\overline{\mathbf{a}} = (a_1, \cdots, a_{t+k\gamma\tau}), \mathsf{td}_a, \overline{\mathbf{h}} = (h_1, \cdots, h_{\gamma\tau}))$. Remark that $(\overline{\mathbf{a}} = (a_1, \cdots, a_{t+k\gamma\tau}), \mathsf{td}_a)$ may be the direct output of either the algorithm BasicLVVDelTrap (if $k = 2$) or the algorithm GenVVDelTrap itself (if $k > 2$). This algorithm uses GenLVVSamTrap, a generalised version of LVVSamTrap, as a sub–procedure. GenLVVSamTrap in turn calls MPSamTrap in its course. In this case, we will also describe GenLVVSamTrap.

- In More General Case, we generalise GenLVVDelTrap to get MoreGenLVVDelTrap whose input is the tuple of $(\overline{\mathbf{a}} = (a_1, \cdots, a_{t+k\gamma\tau}), \mathsf{td}_a, \overline{\mathbf{h}} = (h_1, \cdots, h_{m\gamma\tau}))$. This algorithm calls GenLVVDelTrap as a procedure in its course.

We can see that BasicLVVDelTrap and GenVVDelTrap differ in the nature and the quantity of $(\overline{\mathbf{a}}, \mathsf{td}_a)$, while GenLVVDelTrap and MoreGenVVDelTrap differ in the the quantity of $\overline{\mathbf{h}}$. Please see Table 4.1 for the description and comparison of those cases. We present the flow of algorithms in Figure 4.3. The figure also shows the relation among Base Case (the blue rectangle), Middle Case (the orange rectangle) and More General Case (the red rectangle).

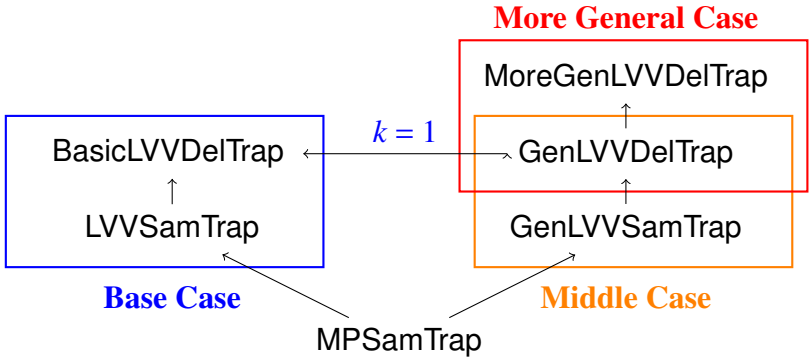

Figure 4.3: Flow of algorithms in our LVV19 trapdoor delegation.

We start with the base case in which $\overline{\mathbf{a}} := \overline{\mathbf{a}}^{(1)} = \overline{\mathbf{a}}_\epsilon$, and $\mathsf{td}_a := \mathsf{td}^{(1)} := \mathsf{td}_\epsilon$, with $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon) \leftarrow \mathsf{LVVGenTrap}(1^n)$.

Base Case. Let $(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon) \leftarrow \mathsf{LVVGenTrap}(1^n)$. Suppose that $\overline{\mathbf{h}} = (h_1, \cdots, h_{t_2})$. Let σ_1 be a Gaussian parameter to be determined. We will be discussing on a way to find a $\overline{\mathbf{g}}$–trapdoor td_a for $\overline{\mathbf{a}}^{(2)} := (\overline{\mathbf{a}}_\epsilon | \overline{\mathbf{h}}) = (a_1, \cdots, a_{t+\gamma\tau} | h_1, \cdots, h_{t_2})$.

Recall that by description of $\mathsf{LVVGenTrap}$ in Lemma 2.5.9, we have $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t+\gamma\tau})$ where $a_i \in \mathbb{Z}_q^{<n}[x]$ for $i \in [t]$, $a_i \in \mathbb{Z}_q^{<n+d-1}[x]$ for $t+1 \le i \le t+\gamma\tau$; and $\mathsf{td}_\epsilon := (\overline{\mathbf{w}}^{(1)}, \cdots, \overline{\mathbf{w}}^{(\gamma\tau)})$ in which $\overline{\mathbf{w}}^{(j)} = (w_1^{(j)}, \cdots, w_t^{(j)}) \leftarrow (\Gamma^d[x])^t$ with $\Gamma = \mathcal{U}(\{-\beta, \cdots, \beta\})$, satisfying that

$$\mathbf{A}_\epsilon \cdot \begin{bmatrix} \mathbf{T}_\epsilon \\ \mathbf{I}_{d\gamma\tau} \end{bmatrix} = \mathbf{G},$$

where

$$\mathbf{A}_\epsilon := [\mathsf{Tp}^{n,2d-1}(a_1) | \cdots | \mathsf{Tp}^{n,2d-1}(a_t) | \mathsf{Tp}^{n+d-1,d}(a_{t+1}) | \cdots | \mathsf{Tp}^{n+d-1,d}(a_{t+\gamma\tau})]$$

is a Toeplitz representation for $\overline{\mathbf{a}}_\epsilon$,

$$\mathbf{T}_\epsilon := \begin{bmatrix} \mathsf{Tp}^{d,d}(w_1^{(1)}) & \cdots & \mathsf{Tp}^{d,d}(w_1^{(\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{d,d}(w_t^{(1)}) & \cdots & \mathsf{Tp}^{d,d}(w_t^{(\gamma\tau)}) \end{bmatrix} \in \mathbb{Z}_q^{(2d-1)t \times d\gamma\tau} \tag{4.3}$$

is a Toeplitz representation for td_ϵ, and the $d\gamma\tau$–dimensional unit matrix

$$\mathbf{I}_{d\gamma\tau} = \begin{bmatrix} \mathsf{Tp}^{1,d}(1) & \cdots & \\ \cdots & & \cdots \\ & \cdots & \mathsf{Tp}^{1,d}(1) \end{bmatrix} \in \mathbb{Z}_q^{d\gamma\tau \times d\gamma\tau}.$$

Let call $\mathbf{R}^{(1)} := \mathbf{T}_\epsilon$. Suppose that matrix $\mathbf{H}$ is a Toeplitz representation for $\overline{\mathbf{h}}$. Define $\mathbf{A}^{(2)} := [\mathbf{A}_\epsilon | \mathbf{H}]$. Then, we have $\mathbf{A}^{(2)}$ as Toeplitz representation for $\overline{\mathbf{a}}^{(2)}$. Now, by definition, the trapdoor td_a will have a Toeplitz representation $\mathbf{R}^{(2)}$ such that $\mathbf{A}^{(2)} \cdot \begin{bmatrix} \mathbf{R}^{(2)} \\ \mathbf{I}_{t_3} \end{bmatrix} = \mathbf{G}$, for some integer $t_3 > 0$. We can easily determine that $t_3 := d\gamma\tau$, too. Thus

$$\mathbf{A}_\epsilon \cdot \mathbf{R}^{(2)} = \mathbf{G} - \mathbf{H}, \tag{4.4}$$

provided a trapdoor $\mathbf{R}^{(1)}$ (i.e., $\mathbf{T}_\epsilon$) for $\mathbf{A}_\epsilon$ and that $\mathbf{H}$ has the same dimension as $\mathbf{G}$.

The task of finding $\mathbf{R}^{(2)}$ satisfying Equation (4.4) can be done using $\mathsf{MPDelTrap}$ in Lemma 2.5.5. However, in our setting, the Toeplitz matrix $\mathbf{H}$ should be such that the matrix $\mathbf{U} := \mathbf{G} - \mathbf{H}$ is still a Toeplitz representation of some polynomials, from which we can correctly recover these polynomials. Moreover, the matrix $\mathbf{R}^{(2)}$ should also be structured so that we can easily convert it into appropriate polynomials r_i's. In comparison with the matrix $\mathbf{G}$, and by the basic properties of Toeplitz matrices (Lemmas 2.5.6–2.5.8), we see that $\mathbf{H}$ should be

$$\mathbf{H} = [\mathsf{Tp}^{n+d-1,d}(h_1)|\cdots|\mathsf{Tp}^{n+d-1,d}(h_{\gamma\tau})] \in \mathbb{Z}_q^{d\gamma\times d\gamma\tau}.$$

Thus, $t_2 := \gamma\tau$ and polynomials $h_1,\cdots,h_{\gamma\tau}$ should have $\deg(h_i) < n+d-1$ for all $i \in [\gamma\tau]$. Therefore,

$$\mathbf{G}-\mathbf{H} = [\mathsf{Tp}^{n+d-1,d}(g_1-h_1)|\cdots|\mathsf{Tp}^{n+d-1,d}(g_{\gamma\tau}-h_{\gamma\tau})].$$

Now, for $i = 1,\cdots,\gamma\tau$, letting $u_i = g_i - h_i$, we have

$$\begin{aligned}\mathbf{G}-\mathbf{H} =&[\mathsf{Tp}^{n+2d-2,1}(u_1)|\cdots|\mathsf{Tp}^{n+2d-2,1}(x^{d-1}\cdot(u_1))|\\ &\cdots|\mathsf{Tp}^{n+2d-2,1}(u_{\gamma\tau})|\cdots|\mathsf{Tp}^{n+2d-2,1}(x^{d-1}\cdot(u_{\gamma\tau}))]\\ =&[\mathsf{Tp}^{n+2d-2,1}(v_1)|\cdots|\mathsf{Tp}^{n+2d-2,1}(v_{d\gamma\tau})],\end{aligned} \tag{4.5}$$

where $v_i = x^{\alpha}u_{\psi}$ for $i = \alpha + d(\psi-1)+1$, with $\alpha \in \{0,\cdots,d-1\}$, $\psi \in \{1,\cdots,\gamma\tau\}$. Note that the last equality in Equation (4.5) is due to Lemma 2.5.7, helping to transform a concatenation of Toeplitz matrices of $\gamma\tau$ polynomials u_i's into that of $d\gamma\tau$ polynomials v_i's.

Recall that the Toeplitz representation for $\overline{\mathbf{a}}_\epsilon = (a_1,\cdots,a_t,a_{t+1},\cdots,a_{t+\gamma\tau}) \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ is

$$\mathbf{A}_\epsilon = [\mathsf{Tp}^{n,2d-1}(a_1)|\cdots|\mathsf{Tp}^{n,2d-1}(a_t)|\mathsf{Tp}^{n+d-1,d}(a_{t+1})|\cdots|\mathsf{Tp}^{n+d-1,d}(a_{t+\gamma\tau})].$$

By letting $\mathbf{v}^{(i)} := \mathsf{Tp}^{n+2d-2,1}(v_i)$, by Equation (4.4) and Equation (4.5), we have to find $\mathbf{R}^{(2)} := [\mathbf{r}^{(1)}|\cdots|\mathbf{r}^{(d\gamma\tau)}]$ such that $\mathbf{A}_\epsilon[\mathbf{r}^{(1)}|\cdots|\mathbf{r}^{(d\gamma\tau)}] = [\mathbf{v}^{(1)}|\cdots|\mathbf{v}^{(d\gamma\tau)}]$, which is equivalent to $\mathbf{A}_\epsilon\mathbf{r}^{(i)} = \mathbf{v}^{(i)}$ for $1 \le i \le d\gamma\tau$. By the description of $\mathsf{LVVSamTrap}$ in Lemma 2.5.9, we see that $\mathbf{r}^{(i)}$ is generated by $\mathsf{MPSamTrap}(\mathbf{T}_\epsilon,\mathbf{A}_\epsilon,\mathbf{v}^{(i)},\sigma_1)$, where $\mathbf{r}^{(i)} \in \mathbb{Z}^{(2d-1)t+d\gamma\tau}$ is a vector sampled from $\mathcal{D}_{\Lambda_q^{\mathbf{v}^{(i)}}(\mathbf{A}),\sigma_1}$, where

$$\sigma_1 \ge \omega(\sqrt{\log(d\gamma)})\cdot\sqrt{7((2d-1)t\cdot(d\gamma\tau)\cdot\beta^2+1)}$$

following Equations (2.8)–(2.9). Moreover, $\mathbf{r}^{(i)}$ is a Toeplitz representation (in column) for $\overline{\mathbf{r}}^{(i)} = (r_1^{(i)},\cdots,r_{t+\gamma\tau}^{(i)})$, with $\deg(r_j^{(i)}) < 2d-1$, $\forall\, j \in [t]$, $\deg(r_{t+j}^{(i)}) < d$ $\forall\, j \in [\gamma\tau]$ and $\forall i \in [d\gamma\tau]$. Equivalently speaking, we can say that $\overline{\mathbf{r}}^{(i)} \leftarrow \mathsf{LVVSamTrap}(\overline{\mathbf{a}}_\epsilon,\mathsf{td}_\epsilon,v^{(i)},\sigma_1)$.

Putting all $\overline{\mathbf{r}}^{(i)}$'s for $i \in [d\gamma\tau]$ in $\mathsf{td}^{(2)} = (\overline{\mathbf{r}}^{(1)},\cdots,\overline{\mathbf{r}}^{(d\gamma\tau)})$, we have a trapdoor for $\overline{\mathbf{a}}$, and

its corresponding matrix representation is

$$\mathbf{R}^{(2)} = (\mathbf{R}^{(2)}[i,j])_{i,j} = \begin{bmatrix} \mathsf{Tp}^{2d-1,1}(r_1^{(1)}) & \cdots & \mathsf{Tp}^{2d-1,1}(r_1^{(d\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{2d-1,1}(r_t^{(1)}) & \cdots & \mathsf{Tp}^{2d-1,1}(r_t^{(d\gamma\tau)}) \\ \mathsf{Tp}^{d,1}(r_{t+1}^{(1)}) & \cdots & \mathsf{Tp}^{d,1}(r_{t+1}^{(d\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{d,1}(r_{t+\gamma\tau}^{(1)}) & \cdots & \mathsf{Tp}^{d,1}(r_{t+\gamma\tau}^{(d\gamma\tau)}) \end{bmatrix} \in \mathbb{Z}^{((2d-1)t+d\gamma\tau)\times d\gamma\tau}. \quad (4.6)$$

By properties of Toeplitz matrices, we can check that $\mathbf{A}^{(2)} \cdot \left[\begin{smallmatrix} \mathbf{R}^{(2)} \\ \mathbf{I}_{d\gamma\tau} \end{smallmatrix}\right] = \mathbf{G}$. Remark that by Lemma 2.3.5,

$$|\mathbf{R}^{(2)}[i,j]| \leq \omega(\log n) \cdot \sigma_1 \text{ with probability } 1 - \mathsf{negl}(n). \quad (4.7)$$

Hence, from Lemma 2.1.2

$$s_1(\mathbf{R}^{(2)}) \leq \sqrt{((2d-1)t + d\gamma\tau) \cdot (d\gamma\tau)} \cdot \omega(\log n) \cdot \sigma_1, \quad (4.8)$$

where σ_1 satisfies Equation (2.8).

The procedure for **Base Case** that we have discussed above is formally called BasicLVVDelTrap. We algorithmically show it in Algorithm 1.

Algorithm 1 $\mathsf{BasicLVVDelTrap}(\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}, \mathsf{td}_\epsilon, \sigma_1)$

Input: A $(t+\gamma\tau)$–family of polynomials $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_t, a_{t+1}, \cdots, a_{t+\gamma\tau}) \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, and its trapdoor td_ϵ, and a $\gamma\tau$–family of polynomials $\overline{\mathbf{h}} = (h_1, \cdots, h_{\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, and (implicitly) $\overline{\mathbf{g}} = (g_1, \cdots, g_{\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ as in Equation (4.1).

Output: The trapdoor $\mathsf{td}^{(2)}$ for $\overline{\mathbf{a}}^{(2)} = (a_1, \cdots, a_{t+\gamma\tau}, h_1, \cdots, h_{\gamma\tau})$.

1: Compute $\overline{\mathbf{u}} = (u_1, \cdots, u_{\gamma\tau}) \leftarrow \overline{\mathbf{g}} - \overline{\mathbf{h}} = (g_1 - h_1, \cdots, g_{\gamma\tau} - h_{\gamma\tau})$.
2: Define $v_i = x^\alpha u_\psi$ for $i = \alpha + d(\psi - 1) + 1$, with $\alpha \in \{0, \cdots, d-1\}$, $\psi \in [\gamma\tau]$.
3: For $i \in [d\gamma\tau]$, call $\mathsf{LVVSamTrap}(\overline{\mathbf{a}}_\epsilon, \mathsf{td}_\epsilon, v_i, \sigma_1)$ to get $\overline{\mathbf{r}}^{(i)} = (r_1^{(i)}, \cdots, r_{t+\gamma\tau}^{(i)})$, where $\deg(r_j^{(i)}) < 2d-1$ for $j \in [t]$, $\deg(r_{t+j}^{(i)}) < d$ for $j \in [\gamma\tau]$.
4: Return $\mathsf{td}^{(2)} = (\overline{\mathbf{r}}^{(1)}, \cdots, \overline{\mathbf{r}}^{(d\gamma\tau)})$.

Middle Case. Now, we generalise **Base Case** to the case in which $\overline{\mathbf{a}} = (a_1, \cdots, a_{t_1})$ and $\overline{\mathbf{h}} = (h_1, \cdots, h_{t_2})$, with $t_1 := t + k\gamma\tau$ (for $k \geq 2$), and again $t_2 = \gamma\tau$. We choose such a t_2 following discussion **Base Case** via which the matrix $\mathbf{H}$ has the same dimension as $\mathbf{G}$. The reason that we consider such t_1 comes from the following observation. Suppose that we delegate up to $(k-1)$ times. We begin with $\overline{\mathbf{a}}^{(1)} := \overline{\mathbf{a}}_\epsilon = (a_1, \cdots a_t, a_{t+\gamma\tau})$, and $\mathsf{td}^{(1)} := \mathsf{td}_\epsilon$. For the first delegation, we add $\gamma\tau$–family of polynomials $\overline{\mathbf{h}}^{(1)} := (h_1, \cdots, h_{\gamma\tau})$ to $\overline{\mathbf{a}}^{(1)}$ then

get $\overline{\mathbf{a}}^{(2)} := (\overline{\mathbf{a}}^{(1)}|\overline{\mathbf{h}}^{(1)}) = (a_1, \cdots a_t, a_{t+1}, \cdots, a_{t+\gamma\tau}, h_1, \cdots, h_{\gamma\tau})$ which can be rewritten as

$$\overline{\mathbf{a}}^{(2)} := (a_1, \cdots a_t, a_{t+1}, \cdots, a_{t+2\gamma\tau}),$$

if we set $a_{t+\gamma\tau+1} := h_1, \cdots, a_{t+2\gamma\tau} := h_{\gamma\tau}$. This way, at the $(k-1)$–th delegation we will have

$$\overline{\mathbf{a}}^{(k)} := (\overline{\mathbf{a}}^{(k-1)}|\overline{\mathbf{h}}^{(k-1)}) = (a_1, \cdots a_t, a_{t+1}, \cdots, a_{t+k\gamma\tau}).$$

Accordingly to the expansion of trapdoors, we will have to do sampling using some trapdoor $\mathsf{td}^{(k)}$ for $\overline{\mathbf{a}}^{(k)}$ for any $k \geq 2$ (instead of $k = 1$ as in **Base Case**), then we slightly modify LVVSamTrap in Lemma 2.5.9 and call it GenLVVSamTrap. If we execute the algorithm GenLVVSamTrap on input $(\overline{\mathbf{a}}^{(k)} = (a_1, \cdots, a_{t+k\gamma\tau}), \mathsf{td}^{(k)} = (\overline{\mathbf{r}}^{(1)}, \cdots, \overline{\mathbf{r}}^{(d\gamma\tau)}), u, \sigma_k)$, where $\overline{\mathbf{r}}^{(i)} = (r_1^{(i)}, \cdots, r_{t+(k-1)\gamma\tau}^{(i)})$ (with $k \geq 2$), then $\mathsf{td}^{(k)}$ should be interpreted as matrix $\mathbf{R}^{(k)}$ given in Equation (4.9) of which the last row's index is $t + (k-1)\gamma\tau$. Remind that the Toeplitz representation $\mathbf{R}^{(1)} = \mathbf{T}_\epsilon$ for $\mathsf{td}^{(1)} := \mathsf{td}_\epsilon$ is of form in Equation (4.3).

$$\mathbf{R}^{(k)} = \begin{bmatrix} \mathsf{Tp}^{2d-1,1}(r_1^{(1)}) & \cdots & \mathsf{Tp}^{2d-1,1}(r_1^{(d\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{2d-1,1}(r_t^{(1)}) & \cdots & \mathsf{Tp}^{2d-1,1}(r_t^{(d\gamma\tau)}) \\ \mathsf{Tp}^{d,1}(r_{t+1}^{(1)}) & \cdots & \mathsf{Tp}^{d,1}(r_{t+1}^{(d\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{d,1}(r_{t+(k-1)\gamma\tau}^{(1)}) & \cdots & \mathsf{Tp}^{d,1}(r_{t+(k-1)\gamma\tau}^{(d\gamma\tau)}) \end{bmatrix} \in \mathbb{Z}^{((2d-1)t+(k-1)d\gamma\tau)\times d\gamma\tau}, \tag{4.9}$$

where each entry $|\mathbf{R}^{(k)}[i,j]$ of $\mathbf{R}^{(k)}$ satisfies that (by Lemma 2.3.5)

$$|\mathbf{R}^{(k)}[i,j]| \leq \omega(\log n) \cdot \sigma_{k-1} \text{ with probability } 1 - \mathsf{negl}(n). \tag{4.10}$$

We describe GenLVVSamTrap right below.

- $\overline{\mathbf{r}} \leftarrow \mathsf{GenLVVSamTrap}(\overline{\mathbf{a}}^{(k)} = (a_1, \cdots, a_{t+k\gamma\tau}), \mathsf{td}^{(k)} = (\overline{\mathbf{r}}^{(1)}, \cdots, \overline{\mathbf{r}}^{(d\gamma\tau)}), u, \sigma_k)$. Do the following:

  1. Construct (implicitly) Toeplitz representation

     $$\mathbf{A}^{(k)} = [\mathsf{Tp}^{n,2d-1}(a_1)|\cdots|\mathsf{Tp}^{n,2d-1}(a_t)|\mathsf{Tp}^{n+d-1,d}(a_{t+1})|\cdots|\mathsf{Tp}^{n+d-1,d}(a_{t+k\gamma\tau})]$$

     for $\overline{\mathbf{a}}^{(k)}$, Toeplitz representation $\mathbf{G}$ for $\overline{\mathbf{g}}$ as in Equation (4.2), and Toeplitz representation $\mathbf{R}^{(k)}$ for $\mathsf{td}^{(k)}$ as in Equation (4.9).

  2. Compute Toeplitz matrix for u: $\mathbf{u} = \mathsf{Tp}^{n+2d-2,1}(u) \in \mathbb{Z}_q^{n+2d-2}$.

3. Sample vector $\mathbf{r} \in \mathbb{Z}^{(2d-1)t+kd\gamma\tau}$ from $D_{\Lambda_q^{\mathbf{u}}(\mathbf{A}^{(k)}),\sigma_k}$ by calling the algorithm $\mathbf{r} \leftarrow \mathsf{MPSamTrap}(\mathbf{R}^{(k)}, \mathbf{A}^{(k)}, \mathbf{u}, \sigma_k)$ mentioned in Lemma 2.5.5, with choosing

$$\sigma_k \geq \omega(\sqrt{\log(d\gamma)}) \cdot \sqrt{7(s_1(\mathbf{R}^{(k)})^2 + 1)}, \tag{4.11}$$

where

$$s_1(\mathbf{R}^{(k)}) \leq \sqrt{((2d-1)t + (k-1)d\gamma\tau) \cdot (d\gamma\tau)} \cdot |\mathbf{R}^{(k)}[i,j]|. \tag{4.12}$$

4. Split $\mathbf{r}$ into $\mathbf{r}^{(k)} = [\mathbf{r}_1^\top | \cdots | \mathbf{r}_{t+k\gamma\tau}^\top]^\top$, and rewrite it (in column) as a Toeplitz representation of polynomials $r_1, \cdots, r_{t+k\gamma\tau}$. Namely, $\mathbf{r}_j = \mathsf{Tp}^{2d-1,1}(r_j)$, $\deg(r_j) < 2d-1$, $\forall j \in [t]$, $\mathbf{r}_j = \mathsf{Tp}^{d,1}(r_j)$, $\deg(r_{t+j}) < d, \forall j \in t+1, \cdots, t+k\gamma\tau$.
5. Output $\overline{\mathbf{r}}^{(k)} := (r_1, \cdots, r_{t+k\gamma\tau})$. Note that

$$\langle \overline{\mathbf{a}}^{(k)}, \overline{\mathbf{r}}^{(k)} \rangle = \sum_{i=1}^{t+k\gamma\tau} a_i \cdot r_i = u.$$

Algorithm 2 $\mathsf{GenLVVDelTrap}(\overline{\mathbf{a}}^{(k)}, \overline{\mathbf{h}}, \mathsf{td}^{(k)}, \sigma_k)$

Input: A $(t+k\gamma\tau)$–family of polynomials $\overline{\mathbf{a}}^{(k)} = (a_1, \cdots, a_t, a_{t+1}, \cdots, a_{t+k\gamma\tau}) \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, and its trapdoor $\mathsf{td}^{(k)}$, and a $\gamma\tau$–family of polynomials $\overline{\mathbf{h}} = (h_1, \cdots, h_{\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, (implicitly) $\overline{\mathbf{g}} = (g_1, \cdots, g_{\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ as in Equation (4.1), and a Gaussian parameter σ_k.
Output: The trapdoor $\mathsf{td}^{(k+1)}$ for $\overline{\mathbf{a}}^{(k+1)} = (a_1, \cdots, a_{t+k\gamma\tau}, h_1, \cdots, h_{\gamma\tau})$.
1: Compute $\overline{\mathbf{u}} = (u_1, \cdots, u_{\gamma\tau}) \leftarrow \overline{\mathbf{g}} - \overline{\mathbf{h}} = (g_1 - h_1, \cdots, g_{\gamma\tau} - h_{\gamma\tau})$.
2: Define $v_i = x^\alpha u_\psi$ for $i = \alpha + d(\psi - 1) + 1$, with $\alpha \in \{0, \cdots, d-1\}$, $\psi \in [\gamma\tau]$.
3: For $i \in [d\gamma\tau]$, call $\mathsf{GenLVVSamTrap}(\overline{\mathbf{a}}^{(k)}, \mathsf{td}^{(k)}, v_i, \sigma_k)$ to get $\overline{\mathbf{r}}^{(i)} = (r_1^{(i)}, \cdots, r_{t+k\gamma\tau}^{(i)})$, where $\deg(r_j^{(i)}) < 2d-1$ for $j \in [t]$, $\deg(r_{t+j}^{(i)}) < d$ for $j \in [k\gamma\tau]$.
4: Return $\mathsf{td}^{(k+1)} = (\overline{\mathbf{r}}^{(1)}, \cdots, \overline{\mathbf{r}}^{(d\gamma\tau)})$.

We are back to the delegation. We present $\mathsf{GenLVVDelTrap}$ in Algorithm 2 in which we delegate a trapdoor for $\overline{\mathbf{a}}^{(k+1)} = (\overline{\mathbf{a}}^{(k+1)} | \overline{\mathbf{h}})$, with $\overline{\mathbf{h}} = (h_1, \cdots, h_{\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$ from a trapdoor for $\overline{\mathbf{a}}^{(k)} = (a_1, \cdots, a_t, a_{t+1}, \cdots, a_{t+k\gamma\tau}) \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$. The algorithm $\mathsf{GenLVVDelTrap}$ is quite the same as $\mathsf{BasicLVVDelTrap}$ but calling $\mathsf{GenLVVSamTrap}$ instead of $\mathsf{LVVSamTrap}$.

More General Case. Now, we are ready to come up with the main algorithm for the LVV19 trapdoor delegation called $\mathsf{MoreGenLVVDelTrap}$ presented in Algorithm 3. This is a more general algorithm than $\mathsf{GenLVVDelTrap}$ involving a multiple of $\gamma\tau$ polynomials in $\overline{\mathbf{h}}$, i.e., $\overline{\mathbf{h}} = (h_1, \cdots, h_{m\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{m\gamma\tau}$ for some $m \in \mathbb{Z}^+$.

Let us make few observations for MoreGenLVVDelTrap. The output td_f of this algorithm is $(\overline{\mathbf{r}}^{(1)}, \cdots, \overline{\mathbf{r}}^{(d\gamma\tau)})$ in which for $i \in [d\gamma\tau]$, $\overline{\mathbf{r}}^{(i)} = (r_1^{(i)}, \cdots, r_{t+(k+m-1)\gamma\tau}^{(i)})$ and $r_j^{(i)} \in \mathbb{Z}_q^{<n+d-1}[x]$ for $j \in [t]$, and $r_{t+j}^{(i)} \in \mathbb{Z}_q^{<d}[x]$ for $j \in [(k+m-1)\gamma\tau]$. The Toeplitz representation $\mathbf{R}^{(k+m-1)}$ for the trapdoor td_f has the form (4.13) below.

$$\mathbf{R}^{(k+m)} = \begin{bmatrix} \mathsf{Tp}^{2d-1,1}(r_1^{(1)}) & \cdots & \mathsf{Tp}^{2d-1,1}(r_1^{(d\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{2d-1,1}(r_t^{(1)}) & \cdots & \mathsf{Tp}^{2d-1,1}(r_t^{(d\gamma\tau)}) \\ \mathsf{Tp}^{d,1}(r_{t+1}^{(1)}) & \cdots & \mathsf{Tp}^{d,1}(r_{t+1}^{(d\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{d,1}(r_{t+k\gamma\tau}^{(1)}) & \cdots & \mathsf{Tp}^{d,1}(r_{t+k\gamma\tau}^{(d\gamma\tau)}) \\ \vdots & & \vdots \\ \mathsf{Tp}^{d,1}(r_{t+(k+m-1)\gamma\tau}^{(1)}) & \cdots & \mathsf{Tp}^{d,1}(r_{t+(k+m-1)\gamma\tau}^{(d\gamma\tau)}) \end{bmatrix}. \tag{4.13}$$

Algorithm 3 $\mathsf{MoreGenLVVDelTrap}(\overline{\mathbf{a}}, \overline{\mathbf{h}}, \mathsf{td}_a, \overline{\sigma}^{(k,m)})$

Input: A $(t+k\gamma\tau)$–family of polynomials $\overline{\mathbf{a}} = (a_1, \cdots, a_t, a_{t+1}, \cdots, a_{t+k\gamma\tau}) \in (\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{k\gamma\tau}$, and its trapdoor td_a, and a $m\gamma\tau$–family of polynomials $\overline{\mathbf{h}} = (h_1, \cdots, h_{m\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{m\gamma\tau}$, (implicitly) $\overline{\mathbf{g}} = (g_1, \cdots, g_{\gamma\tau}) \in (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$, and a list of Gaussian parameters $\overline{\sigma}^{(k,m)} = (\sigma_{k+1}, \cdots, \sigma_{k+m})$).

Output: The trapdoor td_f for $\overline{\mathbf{f}} = (a_1, \cdots, a_{t+k\gamma\tau}, h_1, \cdots, h_{m\gamma\tau})$.

1: Split $\overline{\mathbf{h}} = (\overline{\mathbf{h}}^{(1)}, \cdots, \overline{\mathbf{h}}^{(m)})$ where each $\overline{\mathbf{h}}^{(i)}$ is a $\gamma\tau$–family of polynomials.
2: $\mathsf{td}^{(1)} \leftarrow \mathsf{td}_a$, $\overline{\mathbf{a}}^{(1)} \leftarrow \overline{\mathbf{a}}$.
3: **for** $i = 1$ up to m **do**
4: $\quad \mathsf{td}^{(i+1)} \leftarrow \mathsf{GenLVVDelTrap}(\overline{\mathbf{a}}^{(i)}, \overline{\mathbf{h}}^{(i)}, \mathsf{td}^{(i)}, \sigma_{k+i})$.
5: $\quad \overline{\mathbf{a}}^{(i+1)} \leftarrow (\overline{\mathbf{a}}^{(i)}, \overline{\mathbf{h}}^{(i)})$.
6: **end for**
7: Return $\mathsf{td}_f = \mathsf{td}^{(m+1)}$.

Formally, we state the following theorem.

Theorem 4.2.1 (Main Trapdoor Delegation)**.** *Let n be a positive integer, $q = \mathsf{poly}(n)$ be a prime, and d, t, γ, τ, k, m be positive integers such that $d \le n$, $dt/n = \Omega(\log n)$, $d\gamma = n + 2d - 2$, $k \ge 1$, $m \ge 1$. Let $\tau := \lceil \log q \rceil$ and $\beta := \lceil \frac{\log n}{2} \rceil$. Let $\overline{\mathbf{g}}$ be $\gamma\tau$–family of polynomials as in Equation* (4.1)*. Let $\overline{\mathbf{a}} = (a_1, \cdots, a_{t+k\gamma\tau})$ be a $(t+k\gamma\tau)$–family of polynomials and its associated trapdoor td_a, where $a_i \in \mathbb{Z}_q^{<n}[x]$ for $i \in [t]$ and $a_i \in \mathbb{Z}_q^{<n+d-1}[x]$ for $t+1 \le i \le t+k\gamma\tau$. Suppose that $\overline{\mathbf{h}} = (h_1, \cdots, h_{m\gamma\tau})$ is a $m\gamma\tau$–family of polynomials in $\mathbb{Z}_q^{<n+d-1}[x]$ and $\overline{\sigma}^{(k,m)} = (\sigma_{k+1}, \cdots, \sigma_{k+m})$ is Gaussian parameters to be determined. Then, there exists a probabilistic polynomial–time algorithm,* $\mathsf{MoreGenLVVDelTrap}(\overline{\mathbf{a}}, \overline{\mathbf{h}}, \mathsf{td}_a, \overline{\sigma}^{(k,m)})$ *that outputs a trapdoor td_f for $\overline{\mathbf{f}} = (a_1, \cdots, a_{t+k\gamma\tau} | h_1, \cdots, h_{m\gamma\tau})$.*

Here, we set Gaussian parameters $\overline{\sigma}^{(k,m)} = (\sigma_{k+1}, \cdots, \sigma_{k+m})$ for any integers $k, m > 0$. Note that the Gaussian parameter σ_k in algorithm $\mathsf{GenLVVSamTrap}(\overline{\mathbf{a}}, \mathsf{td}_a, u, \sigma_k)$ has to satisfy Equation(4.11). From Equation (4.13), we can see that σ_{k+i} should satisfy

$$\sigma_{k+i} \geq \omega(\sqrt{\log(d\gamma)}) \cdot \sqrt{7(s_1(\mathbf{R}^{(k+i-1)})^2 + 1)},$$

with

$$s_1(\mathbf{R}^{(k+i-1)}) \leq \sqrt{((2d-1)t + (k+i-1)d\gamma\tau) \cdot (d\gamma\tau)} \cdot \omega(\log n) \cdot \sigma_{k+i-1}, \text{ where } i \in [m].$$

Remind that σ_1 and $s_1(\mathbf{R}^{(1)})$ follows Equations (2.8)–(2.9).

4.3 DMPLWE–based HIBE in Standard Model

We will specify a concrete HIBE system from DMPLWE (Section 2.4.2). The proposed HIBE construction is IND–sID–CPA secure in SDM. We follow the IBE of [AB09] to get our HIBE. Even though, the authors of [LVV19] also exploited the same approach, the private key $\mathsf{sk}_{\mathsf{id}}$ (with respect to an identity id) in IBE of [LVV19] is actually not a $\overline{\mathbf{g}}$–trapdoor. One thus cannot perform delegation on identities. Therefore, it is unable to construct a HIBE using the IBE of [LVV19]. Instead, our HIBE construction uses a $\overline{\mathbf{g}}$–trapdoor as the private key (with respect to a identity).

Specifically, in our HIBE construction, we represent an identity as a binary vector, e.g., $\mathsf{id} := (id_1, \cdots, id_\ell) \in \{0,1\}^\ell$ is an depth–ℓ identity. We will then correspond each entry id_j to a $(t + \gamma\tau)$–family of polynomials $\overline{\mathbf{h}}^{(j,id_j)}$ sampled uniformly at random. For example, if $\mathsf{id} := (0,1,1) \in \{0,1\}^3$ then id corresponds to $\{\overline{\mathbf{h}}^{(1,0)}, \overline{\mathbf{h}}^{(2,1)}, \overline{\mathbf{h}}^{(3,1)}\}$. The proposed HIBE's master secret key is the *root* trapdoor td_ϵ generated together with the *root* family $\overline{\mathbf{a}}_\epsilon$ by $\mathsf{LVVGenTrap}$. For each identity, says $\mathsf{id} := (id_1, \cdots, id_\ell)$, we set the family $\overline{\mathbf{f}}_{\mathsf{id}} = (\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}^{(1,id_1)}, \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)})$. To compute the identity id's private key, we delegate a trapdoor for $\overline{\mathbf{f}}_{\mathsf{id}}$ from the root trapdoor td_ϵ. Similarly, from the private key for an identity $\mathsf{id} := (id_1, \cdots, id_\ell)$, i.e., trapdoor for $\overline{\mathbf{f}}_{\mathsf{id}} = (\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}^{(1,id_1)} \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)})$, we can derive a trapdoor for $\overline{\mathbf{f}}_{\mathsf{id}|id_{\ell+1}} = (\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}^{(1,id_1)}, \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)}, \overline{\mathbf{h}}^{(\ell+1,id_{\ell+1})})$ with respect to the expanded identity $\mathsf{id}|id_{\ell+1} := (id_1, \cdots, id_\ell, id_{\ell+1})$.

4.3.1 The Construction

First, we summarise the system parameters and give their descriptions in Table 4.2. The proposed construction Π_{HIBE} ={HIBE.Setup, HIBE.Ext, HIBE.Der, HIBE.Enc, HIBE.Dec} works as below.

Table 4.2: System parameters in our DMPLWE–based HIBE construction.

Parameters	Definition
n	security parameter
λ	maximum depth of binary tree for identities
q	system modulus
γ, τ	# polynomials in $\overline{\mathbf{g}}$
$t, t' := t + \gamma\tau$	indicate # polynomials generated by LVVGenTrap
m	$t' = m\gamma\tau$
d, k	used to define the middle–product in DMPLWE
$\overline{\alpha} = (\alpha_1, \cdots, \alpha_\lambda)$	Gaussian parameter used in HIBE,Enc
$\overline{\Sigma} = (\overline{\sigma}^{(1)}, \cdots, \overline{\sigma}^{(\lambda)})$,	Gaussian parameter used in HIBE.Ext and HIBE.Der
$\overline{\Psi} = (\Psi_1, \cdots, \Psi_\lambda) \in (\mathbb{R}^+)^\lambda$	Gaussian parameter used in Phase 2 of HIBE.Dec

- $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{HIBE.Setup}(1^\lambda, 1^n)$. HIBE.Setup is a PPT algorithm. It takes as input a security parameter n, a maximum depth λ to output public parameters pp, and a master secret msk. To do that, HIBE.Setup performs the following:

  1. Set common parameters as follows:
     - $q = q(n) \in \mathbb{Z}^+$ be a prime; $d, k \in \mathbb{Z}^+$ such that $2d + k \leq n$ and $\gamma := \frac{n+2d-2}{d} \in \mathbb{Z}^+$, i.e., $d\gamma = n + 2d - 2$; $\beta := \lceil \frac{\log n}{2} \rceil$, $\tau := \lceil \log q \rceil$, t is a positive integer. Let $t' = t + \gamma\tau$, and plaintext space $\mathcal{M} := \{0,1\}^{<k+2}[x]$. Later, we will see that t should also be a multiple of $\gamma\tau$. Suppose that $t' = t + \gamma\tau = m\gamma\tau$ for some $m \geq 2$.
     - For Gaussian parameters used in HIBE.Enc: choose $\overline{\alpha} = (\alpha_1, \cdots, \alpha_\lambda) \in (\mathbb{R}^+)^\lambda$; for Gaussian parameters used in HIBE.Ext and HIBE.Der: choose $\overline{\Sigma} = (\overline{\sigma}^{(1)}, \cdots, \overline{\sigma}^{(\lambda)})$, where $\overline{\sigma}^{(\ell)} = (\sigma_1^{(\ell)}, \cdots, \sigma_m^{(\ell)}) \in (\mathbb{R}^+)^m$. For $\ell \in [\lambda]$, let $\overline{\Sigma}^{(\ell)} = (\overline{\sigma}^{(1)}, \cdots, \overline{\sigma}^{(\ell)})$; for Gaussian parameters used in HIBE.Dec: choose $\overline{\Psi} = (\Psi_1, \cdots, \Psi_\lambda) \in (\mathbb{R}^+)^\lambda$.

     They all are set heuristically in Section 4.3.4.
  2. For $\ell \in [\lambda]$, let $\chi_\ell := \lfloor D_{\alpha_\ell \cdot q} \rceil$ be the rounded Gaussian distribution.
  3. Use $\mathsf{LVVGenTrap}(1^n)$ to get a root family $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t'})$ and its associated root trapdoor td_ϵ, where $a_i \in \mathbb{Z}_q^{<n}[x]$ for $i \in [t]$ and $a_i \in \mathbb{Z}_q^{<n+d-1}[x]$ for $t+1 \leq i \leq t + \gamma\tau$.
  4. Select uniformly a random polynomial $u_0 \in \mathbb{Z}_q^{<n+2d-2}[x]$.
  5. Take family $\overline{\mathbf{g}}$ given in Equation (4.1) into account.
  6. For each $i \in [\lambda]$, $\mathsf{bit} \in \{0,1\}$, sample randomly $\overline{\mathbf{h}}^{(i,\mathsf{bit})} = (h_1^{(i,\mathsf{bit})}, \cdots, h_{t'}^{(i,\mathsf{bit})})$, where each $h_j^{(i,\mathsf{bit})} \in \mathbb{Z}_q^{<n}[x]$ for $j \in [t]$, and each $h_j^{(i,\mathsf{bit})} \in \mathbb{Z}_q^{<n+d-1}[x]$ for $j \in \{t+1, \cdots, t+\gamma\tau\}$. Note that if $h_j^{(i,\mathsf{bit})} \in \mathbb{Z}_q^{<n}[x]$ then also $h_j^{(i,\mathsf{bit})} \in \mathbb{Z}_q^{<n+d-1}[x]$ for any $d \geq$

1. This is why we will be able to well perform the delegation (presented in Section 4.2) on such a family $\overline{\mathbf{h}}^{(i,\mathsf{bit})}$. Let $\mathsf{HList} = \{(i, \mathsf{bit}, \overline{\mathbf{h}}^{(i,\mathsf{bit})}) : i \in [\lambda], \mathsf{bit} \in \{0,1\}\}$ be the ordered set of all $\overline{\mathbf{h}}^{(i,\mathsf{bit})}$'s.

7. Set the master secret key $\mathsf{msk} := \mathsf{td}_\epsilon$.
8. The remaining are included in the public parameters pp.
9. Output $(\mathsf{pp}, \mathsf{msk})$.

We denote $\mathsf{id} = (id_1, \cdots, id_\ell) \in \{0,1\}^\ell$ as an identity of depth $\ell \leq \lambda$. All following algorithms will always work on $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t'})$ and HList.

- $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{HIBE.Ext}(\mathsf{pp}, \mathsf{id}, \mathsf{msk})$. $\mathsf{HIBE.Ext}$ is a PPT algorithm. It takes as input an identity $\mathsf{id} = (id_1, \cdots, id_\ell)$, master secret key $\mathsf{msk} = \mathsf{td}_\epsilon$. It extracts and returns a key $\mathsf{sk}_{\mathsf{id}}$ for identity id by executing:

  1. Build $\overline{\mathbf{h}}_{\mathsf{id}} = (\overline{\mathbf{h}}^{(1,id_1)}, \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)})$.
  2. Output $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{MoreGenLVVDelTrap}(\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}_{\mathsf{id}}, \mathsf{msk}, \overline{\Sigma}^{(\ell)})$.

- $\mathsf{sk}_{\mathsf{id}|id_{\ell+1}} \leftarrow \mathsf{HIBE.Der}(\mathsf{pp}, \mathsf{id}|id_{\ell+1}, \mathsf{sk}_{\mathsf{id}})$. $\mathsf{HIBE.Der}$ is a PPT algorithm. It takes as input identities $\mathsf{id} = (id_1, \cdots, id_\ell)$, $\mathsf{id}|id_{\ell+1} = (id_1, \cdots, id_\ell, id_{\ell+1})$, private key $\mathsf{sk}_{\mathsf{id}} := \mathsf{td}_{\mathsf{id}}$ which is a $\overline{\mathbf{g}}$–trapdoor for $\overline{\mathbf{f}}_{\mathsf{id}} = (\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}^{(1,id_1)}, \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)})$. It outputs a key $\mathsf{sk}_{\mathsf{id}|id_{\ell+1}}$ for $\mathsf{id}|id_{\ell+1}$ by executing:

  1. Output $\mathsf{sk}_{\mathsf{id}|id_{\ell+1}} \leftarrow \mathsf{MoreGenLVVDelTrap}(\overline{\mathbf{f}}_{\mathsf{id}}, \overline{\mathbf{h}}^{(\ell+1,id_{\ell+1})}, \mathsf{td}_{\mathsf{id}}, \overline{\Sigma}^{(\ell+1)})$.

- $\overline{\mathsf{CT}} \leftarrow \mathsf{HIBE.Enc}(\mathsf{pp}, \mathsf{id}, \mu, u_0, \alpha_\ell)$. $\mathsf{HIBE.Enc}$ is a PPT algorithm. It takes as input an identity $\mathsf{id} = (id_1, \cdots, id_\ell)$, a plaintext $\mu \in \mathcal{M}$, a polynomial u_0, a Gaussian parameter α_ℓ. It returns ciphertext $\overline{\mathsf{CT}}$ by executing:

  1. Parse $(f_1, \cdots, f_{t'(\ell+1)}) \leftarrow \overline{\mathbf{f}}_{\mathsf{id}} = (\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}^{(1,id_1)} \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)})$.
  2. Sample polynomial $s \xleftarrow{\$} \mathbb{Z}_q^{<n+2d+k-1}[x]$.
  3. Sample polynomial $e_0 \leftarrow \chi_\ell^{k+1}[x]$, then compute: $\mathsf{CT}_0 = u_0 \odot_{k+2} s + 2e_0 + \mu$.
  4. For $i = 0$ to ℓ do:
     - For $j \in [t]$, sample error polynomial $e_{i\cdot t'+j} \leftarrow \chi_\ell^{2d+k}[x]$, and compute:

$$\mathsf{ct}_i = f_{i\cdot t'+j} \odot_{2d+k} s + 2e_{i\cdot t'+j}.$$

     - For $t+1 \leq j \leq t+\gamma\tau$, sample error polynomial $e_{i\cdot t'+j} \leftarrow \chi_\ell^{d+k+1}[x]$, and compute:

$$\mathsf{ct}_i = f_{i\cdot t'+j} \odot_{d+k+1} s + 2e_{i\cdot t'+j}.$$

5. Set $\mathsf{CT}_1 = (\mathsf{ct}_1, \cdots, \mathsf{ct}_{t'(\ell+1)})$.
6. Output ciphertext $\overline{\mathsf{CT}} = (\mathsf{CT}_0, \mathsf{CT}_1)$.

- $\mu/\perp := \mathsf{HIBE.Dec}(\mathsf{pp}, \mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \overline{\mathsf{CT}}, u_0, \Psi_\ell)$. HIBE.Dec is a DPT algorithm. It takes as input an identity $\mathsf{id} = (id_1, \cdots, id_\ell)$, the trapdoor $\mathsf{sk}_{\mathsf{id}} := \mathsf{td}_{\mathsf{id}}$ for $\bar{\mathbf{f}}_{\mathsf{id}} = (\bar{\mathbf{a}}_\epsilon, \bar{\mathbf{h}}^{(1,id_1)}, \cdots, \bar{\mathbf{h}}^{(\ell,id_\ell)})$, a ciphertext $\overline{\mathsf{CT}} = (\mathsf{CT}_0, \mathsf{CT}_1 = (\mathsf{ct}_1, \cdots, \mathsf{ct}_{t'(\ell+1)}))$, polynomial u_0, and Gaussian parameter Ψ_ℓ. It returns either a plaintext μ or a failure symbol by performing:

  1. Parse $(f_1, \cdots, f_{t'(\ell+1)}) \leftarrow \bar{\mathbf{f}}_{\mathsf{id}} = (\bar{\mathbf{a}}_\epsilon, \bar{\mathbf{h}}^{(1,id_1)}, \cdots, \bar{\mathbf{h}}^{(\ell,id_\ell)})$.
  2. Sample family of polynomials

$$\bar{\mathbf{r}} = (r_1, \cdots, r_{t'(\ell+1)}) \leftarrow \mathsf{GenLVVSamTrap}(\bar{\mathbf{f}}_{\mathsf{id}}, \mathsf{td}_{\mathsf{id}}, u_0, \Psi_\ell),$$

     i.e., $\langle \bar{\mathbf{f}}_{\mathsf{id}}, \bar{\mathbf{r}} \rangle = \sum_1^{t'(\ell+1)} r_i \cdot f_i = u_0$.
  3. If succeeds, it outputs $\mu = (\mathsf{CT}_0 - \sum_{i=1}^{t'(\ell+1)} \mathsf{ct}_i \odot_{k+2} r_i \mod q) \mod 2$. Otherwise, it outputs $\perp$.

4.3.2 Correctness

Lemma 4.3.1 (Correctness)**.** *The* Π_{HIBE} *construction described in Section 4.3.1 is correct with probability* $1 - \mathsf{negl}(n)$ *if*

$$\alpha_\ell < \frac{1}{4}\Big[t'(\ell+1)\cdot(k+1)\cdot\omega(\log n)\cdot\Psi_\ell + \omega(\sqrt{\log n})\Big]^{-1}, \tag{4.14}$$

for all $\ell \in [\lambda]$.

Proof. Suppose that $\overline{\mathsf{CT}} := (\mathsf{CT}_0, \mathsf{CT}_1 = (\mathsf{ct}_1, \cdots, \mathsf{ct}_{t'(\ell+1)})) \leftarrow \mathsf{HIBE.Enc}(\mathsf{pp}, \mathsf{id}, \mu, u_0, \alpha_\ell)$. Over the randomness of HIBE.Setup, HIBE.Der, HIBE.Ext, HIBE.Enc, we will show that

$$\mathsf{HIBE.Dec}(\mathsf{pp}, \mathsf{id}, \mathsf{sk}_{\mathsf{id}}, \mathsf{HIBE.Enc}(\mathsf{pp}, \mathsf{id}, \mu, u_0, \alpha_\ell), u_0, \Psi_\ell) = \mu,$$

with probability $1 - \mathsf{negl}(n)$. By the property of the middle–product given in Lemma 2.4.2, we have

$$\begin{aligned} \mathsf{CT}_0 &= \mu + (\sum_1^t r_i \cdot f_i) \odot_{k+2} s + (\sum_{t+1}^{t'(\ell+1)} r_i \cdot f_i) \odot_{k+2} s + 2e_0 \\ &= \mu + \sum_1^t r_i \odot_{k+2} (\mathsf{ct}_i - 2e_i) + \sum_{t+1}^{t'(\ell+1)} r_i \odot_{k+2} (\mathsf{ct}_i - 2e_i) + 2e_0. \end{aligned} \tag{4.15}$$

Thus, $\mathsf{CT}_0 - \sum_{i=1}^{t'(\ell+1)} \mathsf{ct}_i \odot_{k+2} r_i = \mu + 2(e_0 - \sum_1^{t'(\ell+1)} r_i \odot_{k+2} e_i)$. Hence, if we choose parameters such that

$$\left\| e_0 - \sum_1^{t'(\ell+1)} r_i \odot_{k+2} e_i \right\|_\infty < q/4 \tag{4.16}$$

then the plaintext μ can be correctly recovered. To give a good choice of parameters, we evaluate the left–hand side of Equation (4.16) by bounding the size of coefficients of $e_0 - \sum_1^{t'(\ell+1)} r_i \odot_{k+2} e_i$. First, we see that

- for $i \in [t]$: $\deg(r_i) < d_r := 2d-1$, $\deg(e_i) < d_e := k+1$.
- for $i \in \{t+1, \cdots, t'(\ell+1)\}$: $\deg(r_i) < d_r := d$, $\deg(e_i) < d_e := d+k+1$.

Hence, $d_e + d_r - 1 = 2(d-1) + (k+2)$. Letting $\mathbf{r}_i = (\mathbf{r}_{i,0}, \cdots, \mathbf{r}_{i,d_r-1}), \mathbf{e}_i = (\mathbf{e}_{i,0}, \cdots, \mathbf{e}_{i,d_e-1})$ be the vectors of coefficients of r_i and e_i, respectively, by definition of the middle–product, it holds that

$$r_i \odot_{k+2} e_i = \sum_{j+w=d-1}^{d+k} \mathbf{r}_{i,j} \cdot \mathbf{e}_{i,w} \cdot x^{j+w}.$$

Lemma 2.3.5 guarantees that

$$\Pr[\|\mathbf{r}_i\|_\infty > \omega(\sqrt{\log n}) \cdot \Psi_\ell] = \mathsf{negl}(n),$$
$$\Pr[\|\mathbf{e}_i\|_\infty > \omega(\sqrt{\log n}) \cdot \alpha_\ell \cdot q] = \mathsf{negl}(n),$$

both of which imply that $\|r_i \odot_{k+2} e_i\|_\infty < (k+2) \cdot \omega(\log n) \cdot \Psi_\ell \cdot \alpha_\ell \cdot q$. As a result,

$$\left\| e_0 - \sum_1^{t'(\ell+1)} r_i \odot_{k+2} e_i \right\|_\infty \leq [t'(\ell+1) \cdot (k+2) \cdot \omega(\log n) \cdot \Psi_\ell + \omega(\sqrt{\log n})] \cdot \alpha_\ell \cdot q.$$

Following Equation (4.16), we should choose parameters to satisfy Equation (4.14). The proof follows. □

4.3.3 Security Analysis

Theorem 4.3.1. *Assuming the hardness of the* DMPLWE *assumption, the* Π_{HIBE} *system presented in Section 4.3.1 is INDr–sID–CPA secure in the standard model. Specifically, if there exists an adversary* $\mathcal{A}$ *that breaks the NDr–sID–CPA security of* Π_{HIBE}*, then one can build a solver* $\mathcal{B}$ *that solves the* $(q, n', \mathbf{d}, \chi)$ – DMPLWE *instance, where* $n' = n + 2d + k$

and $\mathbf{d} = (d_0, d_1, \cdots, d_{t'(\ell+1)})$ *with*

$$d_0 := k+2,$$
$$d_{i\cdot t'+j} := \begin{cases} 2d+k, & \text{if } j \in [t], \\ d+k+1, & \text{if } j \in \{t+1, \cdots, t+\gamma\tau\}, \end{cases}$$

for $i \in \{0, \cdots, \ell\}$.

Proof. We proceed the proof with a sequence of games: Game 0 through Game 4. We will show that two consecutive games, Game i and Game $(i+1)$, are indistinguishable by an INDr–sID–CPA adversary. Particularly, the indistinguishability of Game 3 and Game 4 are due to the hardness of the DMPLWE assumption. In other words, if the INDr–sID–CPA adversary can distinguish Game 3 and Game 4, then one can deploy the adversary to construct a DMPLWE solver. The games are described as follows.

Game 0**.** This is the original INDr–sID–CPA game between the adversary $\mathcal{A}$ and the challenger C as described in Section 2.7.4. Here, we focus only on the selective version via which at the beginning of the game, $\mathcal{A}$ must release his target identity $\mathsf{id}^* = (id_1^*, \cdots, id_\theta^*)$. Using the information, C then runs HIBE.Setup to generate system parameters. Specifically, C chooses randomly a family of polynomials $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t'})$ together with an associated trapdoor td_ϵ using LVVGenTrap. The challenger randomly samples a set of polynomial families $\overline{\mathbf{h}}^{(i,\mathsf{bit})} = (h_1^{(i,\mathsf{bit})}, \cdots, h_{t'}^{(i,\mathsf{bit})})$, and stores them in the list HList0, where each $h_j^{(i,\mathsf{bit})} \in \mathbb{Z}_q^{<n}[x]$ for $j \in [t]$, and each $h_j^{(i,\mathsf{bit})} \in \mathbb{Z}_q^{<n+d-1}[x]$ for $j \in \{t+1, \cdots, t+\gamma\tau\}$. The challenger C also takes randomly a polynomial u_0 from $\in \mathbb{Z}_q^{<n+2d-2}[x]$. Finally, the challenger sets $\mathsf{pp} := \{\overline{\mathbf{a}}_\epsilon, \mathsf{HList0}\}$ and sends it to the adversary $\mathcal{A}$, while it keeps $\mathsf{msk} := \mathsf{td}_\epsilon$ as the master secret key. Notice that at the **Challenge** phase, a challenge ciphertext $\overline{\mathsf{CT}}^*$ will be generated for the identity id^* by the challenger.

Game 1**.** This game is kept almost the same as Game 0. The modification is that instead of HList0, the challenger C uses LVVGenTrap to create two lists $\mathsf{HList1} := \{(i, \mathsf{bit}, \overline{\mathbf{h}}^{(i,\mathsf{bit})}) : i \in [\lambda], \mathsf{bit} \in \{0,1\}\}$ and $\mathsf{TList1} := \{(i, \mathsf{bit}, \mathsf{td}^{(i,\mathsf{bit})}) : i \in [\lambda], \mathsf{bit} \in \{0,1\}\}$ respectively containing $\overline{\mathbf{h}}^{(i,\mathsf{bit})}$'s and the corresponding trapdoor $\mathsf{td}^{(i,\mathsf{bit})}$'s for $0 \le i \le \lambda$, $\mathsf{bit} \in \{0,1\}$. The challenger C will then use the list HList1 instead of HList0. (The list TList1 is not used in this game but the next game.)

Game 2**.** This game is the same as Game 1, with a change in responding private key queries on $\mathsf{id} = (id_1, \cdots, id_\ell)$'s where $\ell \le \lambda$. (Note that each of id's is not prefix of the target id^*.) Namely, the challenger C chooses $\overline{\mathbf{a}}_\epsilon = (a_1, \cdots, a_{t+\gamma\tau})$ uniformly at random from $(\mathbb{Z}_q^{<n}[x])^t \times (\mathbb{Z}_q^{<n+d-1}[x])^{\gamma\tau}$. (The challenger thus does not have trapdoor td_ϵ.) Accordingly, C also builds a new algorithm PolyExtTrap which is shown

below to respond to each query from the adversary. Remark that PolyExtTrap does not require all of TList1 but only one $\mathsf{td}^{(j,id_j)} \in \mathsf{TList1}$ for any $j \in [\ell]$.

- $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{PolyExtTrap}(\overline{\mathbf{a}}_\epsilon,\ \mathsf{HList1},\ \mathsf{id},\ j,\ \mathsf{td}^{(j,id_j)})$. On input the root family $\overline{\mathbf{a}}_\epsilon$, list HList1, identity $\mathsf{id} = (id_1, \cdots, id_\ell)$, an index j and trapdoor $\mathsf{td}^{(j,id_j)} \in \mathsf{TList1}$, execute:
  1. Build $\overline{\mathbf{f}}_{\mathsf{id}} = (\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}^{(1,id_1)}, \cdots, \overline{\mathbf{h}}^{(j-1,id_{j-1})}, \overline{\mathbf{h}}^{(j+1,id_{j+1})}, \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)})$.
  2. $\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{MoreGenLVVDelTrap}(\overline{\mathbf{h}}^{(j,id_j)}, \overline{\mathbf{f}}_{\mathsf{id}}, \mathsf{td}^{(j,id_j)}, \overline{\Sigma}^{(\ell)})$.

Also note that the algorithm PolyExtTrap in this game is the reason why we require that the number of polynomials of each $\overline{\mathbf{h}}^{(j,id_j)}$ is the same as that of $\overline{\mathbf{a}}_\epsilon$ and that t is a multiple of $\gamma\tau$ as we claimed in HIBE.Setup in order for MoreGenLVVDelTrap to work well. In fact, this is because any $\overline{\mathbf{h}}^{(j,id_j)}$ and its corresponding $\mathsf{td}^{(j,id_j)}$ will play the role of $\overline{\mathbf{a}}_\epsilon$ and td_ϵ, respectively, for responding to the adversary's queries.

Game 3**.** This game is slightly modified from Game 2 as follows. Specifically, the challenger C respectively replace the lists HList1, TList1 with the lists $\mathsf{HList3} := \{(i, \mathsf{bit}, \overline{\mathbf{h}}^{(i,\mathsf{bit})}) : i \in [\lambda], \mathsf{bit} \in \{0,1\}\}$ and $\mathsf{TList3} := \{(i, \mathsf{bit}, \mathsf{td}^{(i,\mathsf{bit})}) : i \in [\lambda], \mathsf{bit} \in \{0,1\}\}$ created as follows:

- For each $j \in [\lambda]$ and $\mathsf{bit} \in \{0,1\}$ such that $\mathsf{bit} \neq id^*_j$, C calls $\mathsf{LVVGenTrap}(1^n)$ to generate $\overline{\mathbf{h}}^{(j,\mathsf{bit})}$ and its associated trapdoor $\mathsf{td}^{(j,\mathsf{bit})}$.
- For each $j \in [\lambda]$ and $\mathsf{bit} \in \{0,1\}$ such that $\mathsf{bit} = id^*_j$, C simply samples $\overline{\mathbf{h}}^{(j,\mathsf{bit})}$ uniformly at random and set $\mathsf{td}^{(j,\mathsf{bit})} = \perp$.

Now, getting a private key query on identity $\mathsf{id} = (id_1, \cdots, id_\ell)$ which is not a prefix of the target identity id^*, the challenger first finds an index $j^\dagger$ such that $id_{j^\dagger} \neq id^*_{j^\dagger}$. then answers the query with

$$\mathsf{sk}_{\mathsf{id}} \leftarrow \mathsf{PolyExtTrap}(\overline{\mathbf{a}}_\epsilon, \mathsf{HList3}, \mathsf{id}, j^\dagger, \mathsf{td}^{(j^\dagger, id_{j^\dagger})}),$$

where $\mathsf{td}^{(j^\dagger, id_{j^\dagger})} \in \mathsf{TList3}$. One more thing is that the challenge ciphertext is produced as $\overline{\mathsf{CT}}^* \leftarrow \mathsf{HIBE.Enc}(\mathsf{id}^*, \mu^*, u_0, \overline{\alpha})$ over HList3.

Game 4**.** This game is similar to Game 3, except that the challenger chooses the challenge ciphertext $\overline{\mathsf{CT}}^* = (\mathsf{CT}^*_0, \mathsf{CT}^*_1)$ uniformly at random.

One can easily show that the view of the INDr–sID–CPA adversary is identical in each pair of two consecutive games Game 0 and Game 1, Game 1 and Game 2, Game 2 and Game 3. However it needs more work to prove the indistinguishability of the two games Game 3 and Game 4. This can be done using the following reduction.

Reduction from DMPLWE. We proceed by contradiction. Suppose that the adversary $\mathcal{A}$ can distinguish between Game 3 and Game 4 with non–negligible probability. From $\mathcal{A}$, we construct an adversary $\mathcal{B}$ which can solve DMPLWE problem with the same probability. More specifically,

- $(q, n', \mathbf{d}, \chi)$–**DMPLWE Instance:** The adversary $\mathcal{B}$ are given $1 + t'(\ell+1)$ samples (f_z, ct_z) for $z \in \{0, \cdots, t'(\ell+1)\}$. $\mathcal{B}$ has to decide whether (f_z, ct_z) follows
  (i) $\prod_{z=0}^{t'(\ell+1)} \mathcal{U}(\mathbb{Z}_q^{n'-d_z}[x] \times \mathbb{R}_q^{d_z}[x])$, or
  (ii) $\mathsf{DMP}_{q,n',\mathbf{d},\chi}(s)$, where $n' = n + 2d + k$ and $\mathbf{d} = (d_0, d_1, \cdots, d_{t'(\ell+1)})$ is interpreted as follows:

$$d_0 := k + 2,$$
$$d_{i \cdot t' + j} := \begin{cases} 2d + k, & \text{if } j \in [t], \\ d + k + 1, & \text{if } j \in \{t+1, \cdots, t + \gamma\tau\}, \end{cases}$$

  for $i \in \{0, \cdots, \ell\}$, and f_z's are random in $\mathbb{Z}_q^{<n'-d_z}[x]$ for $z \in \{0, \cdots, t'(\ell+1)\}$.
  In other words, $\mathcal{B}$ is required to distinguish whether
  (i) all ct_z are random polynomials, or
  (ii) $\mathsf{ct}_z = f_z \odot_{d_z} s + 2e_z$ in $\mathbb{Z}_q^{<d_z}[x]$, for some $s \xleftarrow{\$} \mathbb{Z}_q^{<n'-1}[x]$ and $e_z \leftarrow \chi^{d_z}[x]$, for all $z \in \{0, \cdots, t'(\ell+1)\}$.
- **Targeting:** $\mathcal{B}$ gets the target identity id^* from the adversary $\mathcal{A}$.
- **Setup:** $\mathcal{B}$ creates the lists HListB and TListB in the same way as HList3 and TList3 in Game 3 and Game 4. Precisely,
  - For each $j \in [\lambda]$ and $\mathsf{bit} \in \{0,1\}$ such that $\mathsf{bit} \neq id_j^*$: $\mathcal{B}$ calls $\mathsf{LVVGenTrap}(1^n)$ to generate $\overline{\mathbf{h}}^{(j,\mathsf{bit})}$ and its associated trapdoor $\mathsf{td}^{(j,\mathsf{bit})}$.
  - For each $j \in [\lambda]$ and $\mathsf{bit} \in \{0,1\}$ such that $\mathsf{bit} = id_j^*$: $\mathcal{B}$ simply samples $\overline{\mathbf{h}}^{(j,\mathsf{bit})}$ uniformly at random and sets $\mathsf{td}^{(j,\mathsf{bit})} = \perp$.

  The list HListB contains all $\overline{\mathbf{h}}^{(j,\mathsf{bit})}$ and TListB contains all $\mathsf{td}^{(j,\mathsf{bit})}$.
- **Queries**: When answering the private key queries, $\mathcal{B}$ acts similarly to the way in Game 3 or in Game 4 and it can use one of trapdoors $\mathsf{td}^{(j,\mathsf{bit})} \neq \perp$.
- **Challenge:** In this phase, $\mathcal{B}$ produces the challenge ciphertext by choosing randomly $b \xleftarrow{\$} \{0,1\}$ and then setting

$$\overline{\mathsf{CT}}^* := (\mathsf{CT}_0^* := \mathsf{ct}_0 + \mu, \mathsf{CT}_1^* := (\mathsf{ct}_1, \cdots, \mathsf{ct}_{t'(\ell+1)})).$$

- **Guess:** $\mathcal{A}$ guesses the value of b. At this time, $\mathcal{B}$ outputs what $\mathcal{A}$ have guessed.

Analysis. Obviously, from what $\mathcal{A}$ can see, $\mathcal{B}$ behaves almost identically in both Game 3 and Game 4. However, the challenge ciphertexts in these two games are produced differently. In fact, if ct_z's follow DMPLWE then $\overline{\mathsf{CT}}^*$ are generated similarly to that in Game 3, whereas if ct_z's are randomly chosen then $\overline{\mathsf{CT}}^*$ are generated via the same way as in Game 4. This is the reason why $\mathcal{B}$ can solve the DMPLWE instance assuming that $\mathcal{A}$ can distinguish between Game 3 and Game 4. It is also easy to see that the success probability of $\mathcal{B}$ and that of $\mathcal{A}$ are the same. This argument completes the proof. □

4.3.4 Setting Parameters

We choose the parameters involved in the Π_{HIBE} construction as below:

- Set n to be a security parameter.
- Choose $q, d, t, \gamma, \beta, k, \tau$ to be such that: $q = \mathsf{poly}(n)$ prime, $\beta := \lceil \frac{\log n}{2} \rceil \ll q/2$, $\gamma := \frac{n+2d-2}{d} \in \mathbb{Z}^+$, $\tau := \lceil \log q \rceil$, $t' := t + \gamma\tau = m\gamma\tau$ (for some $m \geq 2$), $d \leq n$, $2d + k < n$, $dt/n = \Omega(\log n)$, and $d\gamma = n + 2d - 2 \leq 3n$.
- Set Gaussian parameters used in HIBE.Ext and HIBE.Der: Because the maximum depth of identities is λ, then it suffices to consider the longest series of Gaussian parameters $\overline{\Sigma} := \overline{\Sigma}^{(\lambda)} = (\overline{\sigma}^{(1)}, \cdots, \overline{\sigma}^{(\lambda)})$. Recall that each $\overline{\sigma}^{(i)} = (\sigma_1^{(i)}, \cdots, \sigma_m^{(i)}) \in (\mathbb{R}^+)^m$ consists of m elements. For convenience, we renumber $\overline{\Sigma}$ as $(\sigma_1, \cdots, \sigma_{m\lambda})$ keeping their original order. According to the maximal identity $\mathsf{id} = (id_1, \cdots, id_\lambda)$, we build $\overline{\mathbf{h}}_{\mathsf{id}} = (\overline{\mathbf{h}}^{(1,id_1)}, \cdots, \overline{\mathbf{h}}^{(\lambda,id_\lambda)})$ in which each $\overline{\mathbf{h}}^{(i,id_i)}$ consists of $m\gamma\tau$ polynomials. We now use a "regrouping technique" to split $\overline{\mathbf{h}}_{\mathsf{id}}$ into $(\overline{\mathbf{h}}^{(1)}, \cdots, \overline{\mathbf{h}}^{(m\lambda)})$, each $\overline{\mathbf{h}}^{(i)}$ consists of $\gamma\tau$ polynomials. Let $\overline{\mathbf{a}}^{(i+1)} = (\overline{\mathbf{a}}_\epsilon | \overline{\mathbf{h}}^{(1)} | \cdots | \overline{\mathbf{h}}^{(i)})$ with $\overline{\mathbf{a}}^{(1)} = \overline{\mathbf{a}}_\epsilon$. Then, MoreGenLVVDelTrap calls GenLVVDelTrap$(\overline{\mathbf{a}}^{(i)}, \overline{\mathbf{h}}^{(i)}, \mathsf{td}^{(i)}, \sigma_i)$ up to $m\lambda$ times, each time for one $i \in [m\lambda]$. Remember that $\mathsf{td}^{(1)} = \mathsf{td}_\epsilon$ and

$$\mathsf{td}^{(i+1)} \leftarrow \mathsf{GenLVVDelTrap}(\overline{\mathbf{a}}^{(i)}, \overline{\mathbf{h}}^{(i)}, \mathsf{td}^{(i)}, \sigma_i),$$

for $1 \leq i \leq m\lambda$. All σ_i's thus are set following Section 4.2, that is, for $1 \leq i \leq m\lambda - 1$,

$$\sigma_{i+1} \geq \omega(\sqrt{\log(d\gamma)}) \cdot \sqrt{7(s_1(\mathbf{R}^{(i)})^2 + 1)},$$

and

$$s_1(\mathbf{R}^{(i)}) \leq \sqrt{((2d-1)t + i \cdot d\gamma\tau) \cdot (d\gamma\tau)} \cdot \omega(\log n) \cdot \sigma_i.$$

Here, $\mathbf{R}^{(i)}$ is the Toeplitz representation of the private key (the trapdoor) for $\overline{\mathbf{a}}^{(i)} := (\overline{\mathbf{a}}_\epsilon | \overline{\mathbf{h}}^{(1)} | \cdots | \overline{\mathbf{h}}^{(i-1)})$. The formula for $\mathbf{R}^{(i)}$ was given in Equation (4.9) with noting that

the last row is indexed by $t+(i-1)\gamma\tau$. Also, remark that σ_1 and $\mathbf{R}^{(1)}$ are exactly σ and $\mathbf{T}_\epsilon$ in Equations (2.8)–(2.9).

- Now, it is the turn for Gaussian parameters $\overline{\Psi} = (\Psi_1, \cdots, \Psi_\lambda)$ used in $\mathsf{HIBE.Dec}$. Consider the case involving the identity $\mathsf{id} = (id_1, \cdots, id_\ell)$ (for $\ell \in [\lambda]$). Stress that for $\ell \in [\lambda]$, since Ψ_ℓ is involved in $\mathsf{GenLVVSamTrap}(\bar{\mathbf{f}}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}, u_0, \Psi_\ell)$ where $\mathsf{sk}_{\mathsf{id}} = \mathsf{td}_{\mathsf{id}}$ is a trapdoor for $\bar{\mathbf{f}}_{\mathsf{id}} = (\overline{\mathbf{a}}_\epsilon, \overline{\mathbf{h}}^{(1,id_1)}, \cdots, \overline{\mathbf{h}}^{(\ell,id_\ell)})$ which can be seen as $\overline{\mathbf{a}}^{(\ell m+1)} := (\overline{\mathbf{a}}_\epsilon | \overline{\mathbf{h}}^{(1)} | \cdots | \overline{\mathbf{h}}^{(\ell m)})$ (via the above "regrouping technique"). We should thus set $\Psi_\ell = \sigma_{\ell m}$, for $\ell \in [\lambda - 1]$. In particular, for the case involving the maximal identity $\mathsf{id} = (id_1, \cdots, id_\ell)$,

$$\Psi_\lambda \geq \omega(\sqrt{\log(d\gamma)}) \cdot \sqrt{7(s_1(\mathbf{R}^{(m\lambda)})^2 + 1)},$$

$$s_1(\mathbf{R}^{(m\lambda)}) \leq \sqrt{((2d-1)t + (m\lambda)d\gamma\tau) \cdot (d\gamma\tau)} \cdot \omega(\log n) \cdot \sigma_{m\lambda},$$

in which $\mathbf{R}^{(m\lambda)}$ is the Toeplitz representation for the private key (the trapdoor) for $\overline{\mathbf{a}}^{(m\lambda)} = (\overline{\mathbf{a}}_\epsilon | \overline{\mathbf{h}}^{(1)} | \cdots | \overline{\mathbf{h}}^{(m\lambda)})$.

- Finally, the Gaussian parameters $\overline{\alpha} = (\alpha_1, \cdots, \alpha_\lambda)$ used in $\mathsf{HIBE.Enc}$ should be chosen to fulfil Equation (4.14).

4.4 Summary

In this chapter, we have developed a delegation mechanism for the LVV19 trapdoor, which was presented in Section 2.5.3. Given a trapdoor for a polynomial family, this delegation method enables delegating a trapdoor for an expanded polynomial family. As an application, the delegation allows us to realize the first HIBE system from DMPLWE. Furthermore, we achieve the INDr–sID–CPA security of the proposed HIBE construction in SDM.

Regarding the trapdoor in the polynomial setting, we have some interesting open issues. First, the delegation technique requires that the number of polynomials in the added family (i.e., $\overline{\mathbf{h}} = (h_1, \cdots, h_{m\gamma\tau})$ in Theorem 4.2.1) should be a multiple of the number of polynomials in the gadget family $\overline{\mathbf{g}}$ (i.e., $\gamma\tau$ in the text). The question here is that if we could find a new trapdoor delegation method that would be applied well to $\overline{\mathbf{h}} = (h_1, \cdots, h_{t_2})$ for arbitrary $t_2 \geq 1$. Second, we follow the [AB09] for HIBE construction, which results in a quite large ciphertext in size. Thus, it is better if we could develop a mechanism allowing to delegate a trapdoor for $\overline{\mathbf{f}}'_{\mathsf{id}} = (\overline{\mathbf{a}}, \langle \overline{\mathbf{h}}^{(1,id_1)}, \overline{\mathbf{b}} \rangle, \cdots, \langle \overline{\mathbf{h}}^{(\ell,id_\ell)}, \overline{\mathbf{b}} \rangle)$, given random $\overline{\mathbf{b}}$ and its trapdoor td_b. This way, we might be able to apply the HIBE framework of [ABB10] for the DMPLWE–based HIBE construction. This framework might offer a better ciphertext size than that of our work here. Finally, the existing trapdoor method for polynomials exploits the Toeplitz matrices, which results in a complicated and heavy in

representation as well as in computation. It would therefore be fruitful if there could be a trapdoor (and delegation) method for polynomials involved in DMPLWE, which does not need Toeplitz representation.

In Chapter 5, an advanced cryptosystem called puncturable encryption (PE) will be taken into consideration. We will introduce a new primitive from which we can generically attain a PE. Also, we instantiate them based on a hard lattice problem.

Chapter 5

Puncturable Encryptions over Lattices

Part of the content in this chapter appeared in Susilo et al. [SDLP20]. The author of this thesis is one of the corresponding authors of [SDLP20]. Having received the topic from his supervisors, he contributed to finding an appropriate theoretical framework, the design of the cryptosystems, the security analysis of the cryptosystems and the writing of the manuscript.

5.1 Overview

This chapter dedicates to puncturable encryption (PE). The notion of PE has been introduced by Green and Miers [GM15] involving some tags embedded in ciphertexts (termed *ciphertext tags*) as well as in decryption keys (termed *punctures* or *key tags*). PE offers the ability to revoke individual plaintexts using the *puncturing mechanism*. Roughly, the puncturing mechanism says that if a key is embedded with a puncture already contained in a ciphertext, the key cannot decrypt the ciphertext; otherwise, one can use the key to regain the underlying plaintext. For the formal definition of PE, please refer to Section 2.7.5. Asynchronous messaging system, forward–secret zero round–trip time protocol, forward–secret proxy re–encryption and public–key watermarking scheme are several interesting applications of PE.

The puncturing mechanism is beneficial in various situations. One situation could be when the present decryption key of the cryptosystem is leaked. If this is the case, one can merely update the key using the puncturing mechanism on the current time, which plays as a puncture. Another situation where the puncturing mechanism could be applied is a system with multiple users. Assume that some users had breached the system's policy, and we want to revoke their decryption capacity. Here, tags are user identities. To keep

(respectively, to revoke) decryption capacity from some users, we do not (respectively, do) puncture their identities. Alternatively, if we want to hide some sensitive documents from some user, then a possible way is to puncture the user's decryption key on the document's identification.

The above–mentioned situations suggest that PE can provide forward security at a fine–grained level. By contrast, the forward–secure encryption (FSE) cryptosystem proposed by Canetti *et al.* [CHK03] guarantees forward security but not at a fine–grained level. For example, in FSE, if we revoke a user's decryption capability for a given time, we also remove the user's access to all plaintexts regarding prior periods.

In [SDLP20] and this chapter, we consider PE under an observation that it is possible to use the family of equality test functions to implement the puncturing property. Namely, such a function compares a ciphertext tag with a puncture to see if they are identical. More importantly, one can implement the functions as arithmetic circuits, which suggests we apply the idea of using the lattice homomorphic evaluations mentioned in Section 2.5.4. Specifically, we put forward the notion of *delegatable fully key–homomorphic encryption* (DFKHE). The notion is an augmented version of the fully key–homomorphic encryption (FKHE) proposed by Boneh *et al.* at Eurocrypt 2014 [BGG$^+$14]. We then present a generic construction framework for PE from DFKHE. Using the framework, we instantiate a PE construction over lattices. This is the *first* concrete PE construction in the lattice setting, The construction is selective secure under chosen plaintext attacks (CPA) in the standard model (SDM), and its security relies on the decision learning with errors (DLWE) problem.

In the following, we will give more detail about the contributions and techniques presented in this chapter.

DFKHE. We begin with a description of FKHE and DFKHE. At high-level description, an FKHE system possesses a mechanism that allows converting a ciphertext $ct_{\mathbf{x}}$ associated with a public variable $\mathbf{x}$ into the evaluated one ct_f, on the same plaintext, associated with the pair (y, f), where f is an efficiently computable function and $f(\mathbf{x}) = y$. To support this, FKHE offers a key–homomorphic evaluation algorithm named Eval. Roughly, we have $\mathsf{ct}_f \leftarrow \mathsf{Eval}(f, \mathsf{ct}_{\mathbf{x}})$. Accordingly, to successfully decrypt such an evaluated ciphertext, the decryptor needs to evaluate the initial secret sk to get sk_f. The FKHE system has an algorithm named Hom to do that, i.e., $\mathsf{sk}_f \leftarrow \mathsf{Hom}(\mathsf{sk}, (y, f))$. The drawback of FKHE is that it supports only one function f each evaluation time. In some cases, we would like to do the key-homomorphic evaluation for a list of functions $\{f_1, \cdots, f_k\}$ belonging to some function family $\mathcal{F}$. Therefore, we generalise FKHE by endowing it with two algorithms $\mathsf{ExtEval}$ and Del such that: (i) $\mathsf{ct}_{f_1,\cdots,f_k} \leftarrow \mathsf{ExtEval}(f_1, \cdots, f_k, \mathsf{ct}_{\mathbf{x}})$, transforming $(\mathsf{ct}_{\mathbf{x}}, \mathbf{x})$ into $(\mathsf{ct}_{f_1,\cdots,f_k}, (y, f_1, \cdots f_k))$ with $f_1(\mathbf{x}) = \cdots = f_k(\mathbf{x}) = y$; (ii) $\mathsf{sk}_{f_1,\cdots,f_k} \leftarrow \mathsf{Del}(\mathsf{sk}_{f_1,\cdots,f_{k-1}}, (y, f_k))$ that allows delegating the secret key for one more function each time from the evaluated

secret key $\mathsf{sk}_{f_1,\cdots,f_{k-1}}$. Doing that, we come up with the notion of DFKHE. At this point, the DFKHE can be viewed as the FKHE endowed with the two algorithms ExtEval and KDel.

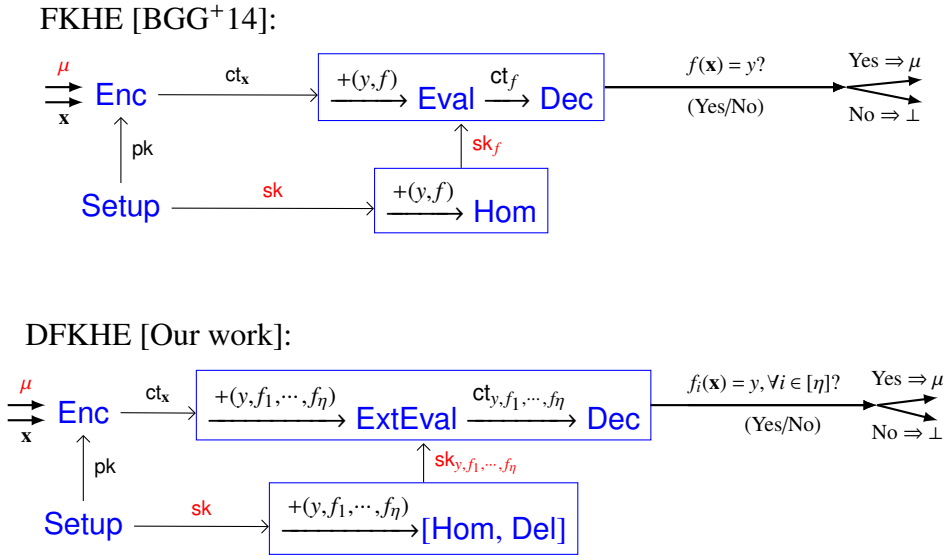

Figure 5.1: Pictorial description of DFKHE in comparison with FKHE.

Generic PE framework. Our generic PE framework is inspired by a simple but subtle observation that the requirements for the puncturing property (i.e., the equality of ciphertext tags and punctures) in a PE can be formalised through functions that arithmetic circuits can efficiently compute. We call such functions the *equality test functions*. We can see that in PE, the ciphertext tags play the role of variables $\mathbf{x}$'s, and the equality test functions act as the functions f's described in the FKHE system. Moreover, in FE, one more puncture added will define one more equality test function, which needs a delegation mechanism to take the function into account. This need can be easily met using the same idea as the key delegation Del mentioned above. In our work, in order to be able to employ the idea of DFKHE for $(y_0, \mathcal{F})$ to PE construction, we define an efficiently computable family $\mathcal{F}$ of equality test functions $f_{t^*}(\mathbf{t})$ allowing us to compare the puncture t^* with ciphertext tags $\mathbf{t} = (t_1, \cdots, t_d)$ under the definition that $f_{t^*}(\mathbf{t}) = y_0$ iff $t^* \neq t_j, \forall j \in [d]$, for some fixed value y_0.

Lattice–based DFKHE and Lattice–based PE. For a concrete DHKHE and PE construction, we exploit the LWE-based FKHE proposed in [BGG+14]. Specifically, we deploy the dual Regev encryption framework (Section 2.7.3). In this system, the ciphertext is $\mathsf{ct} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_1, \cdots, \mathbf{c}_d, \mathbf{c}_{\mathsf{out}})$, where $\mathbf{c}_i = (t_i\mathbf{G} + \mathbf{B}_i)^T\mathbf{s} + \mathbf{e}_i$ for $i \in [d]$. Here the gadget matrix $\mathbf{G}$ is a special one, whose associated trapdoor $\mathbf{T}_{\mathbf{G}}$ (i.e., a short basis for the q-ary lattice $\Lambda_q^{\perp}(\mathbf{G})$) is publicly known (see [MP12] for details). Also, there exist three evaluation algorithms named $\mathsf{Eval}_{\mathsf{pk}}$, $\mathsf{Eval}_{\mathsf{ct}}$, and $\mathsf{Eval}_{\mathsf{sim}}$ [BGG+14] (Section

2.5.4) which help us to homomorphically evaluate a circuit (function) on the ciphertext ct. More specifically, from $\mathbf{c}_i := [t_i\mathbf{G} + \mathbf{B}_i]^T\mathbf{s} + \mathbf{e}_i$, where $\|\mathbf{e}_i\| < \delta$ for all $i \in [d]$, and a function $f : (\mathbb{Z}_q)^d \to \mathbb{Z}_q$, we get $\mathbf{c}_f = [f(t_1, \cdots, t_d)\mathbf{G} + \mathbf{B}_f]^T\mathbf{s} + \mathbf{e}_f, \|\mathbf{e}_f\| < \Delta$, where $\mathbf{B}_f \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f, (\mathbf{B}_i)_{i=1}^d)$, $\mathbf{c}_f \leftarrow \mathsf{Eval}_{\mathsf{ct}}(f, ((t_i, \mathbf{B}_i, \mathbf{c}_i))_{i=1}^d)$, and $\Delta < \delta \cdot \beta$ for some β sufficiently small. Then, the algorithm $\mathsf{ExtEval}$ mentioned above can be implemented calling many times $\mathsf{Eval}_{\mathsf{pk}}$,$\mathsf{Eval}_{\mathsf{ct}}$, each time for each function. Meanwhile, $\mathsf{Eval}_{\mathsf{sim}}$ is only useful in the simulation for the security proof. In the LWE–based DFKHE construction, secret keys are the GPV08 trapdoors (Section 2.5.3) for q–ary lattices of form $\Lambda_q^{\perp}([\mathbf{A}|\mathbf{B}_{f_1}|\cdots|\mathbf{B}_{f_k}])$. For the key delegation Del, we can utilize the trapdoor delegation [GPV08, ABB10, CHKP10]. The leftover hash lemma (Section 2.5.1) is used to argue the indistinguishability of hybrids.

For the LWE–based PE instantiation, we employ the equality test function with $y_0 := 0 \pmod q$. Namely, for a puncture t^* and a list of ciphertext tags $t_1, \cdots, t_d$ we define $f_{t^*}(t_1, \cdots, t_d) := eq_{t^*}(t_1) + \cdots + eq_{t^*}(t_d)$, where $eq_{t^*} : \mathbb{Z}_q \to \mathbb{Z}_q$ satisfying that $\forall t \in \mathbb{Z}_q$, $eq_{t^*}(t) = 1 \pmod q$ iff $t = t^*$, otherwise $eq_{t^*}(t) = 0 \pmod q$. Such functions have also been employed in [BKM17] to construct a privately puncturable pseudorandom function. It follows from generic construction that our PE instantiation is selective CPA–secure.

Figure 5.2 shows the roadmap of this chapter.

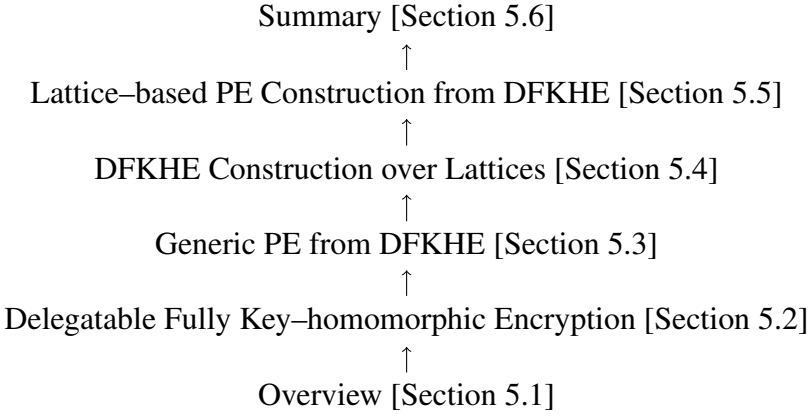

Figure 5.2: The roadmap of this chapter.

5.2 Delegatable Fully Key–homomorphic Encryption

We first present the *delegatable fully key–homomorphic encryption* (DFKHE) primitive. The primitive generalises the notion of fully key–homomorphic encryption (FKHE), which has been introduced in [BGG+14]. Informally, FKHE is a kind of public–key encryption that allows transforming a ciphertext $\mathsf{ct}_{\mathbf{x}}$ with respect to a public variable $\mathbf{x}$ into a new one ct_f with respect to a function f. A special value y is also involved in such a way that

$\mathbf{x}, (y, f)$ satisfy the following relation: the secret key which is produced associatively with (y, f) is able to decrypt the ciphertext ct_f as long as $f(\mathbf{x}) = y$.

The key idea of DFHKE is quite the same but allowing more functions to be involved, i.e., $f_1, \cdots, f_k$ instead of only one f, and decryption is successful only if $f_1(\mathbf{x}) = \cdots = f_k(\mathbf{x}) = y$.

5.2.1 Syntax

Let λ, $d = d(\lambda) \in \mathbb{Z}^+$. Let $\mathcal{T} = \mathcal{T}(\lambda)$ and $\mathcal{Y} = \mathcal{Y}(\lambda)$ be two finite sets. Let $\mathcal{F} = \mathcal{F}(\lambda)$ $:= \{f | f : \mathcal{T}^d \to \mathcal{Y}\}$ be a family of efficiently computable functions. Let sp be the set that contains system parameters including $d, \mathcal{T}, \mathcal{Y}$ and $\mathcal{F}$.

Then (λ, sp)–DFKHE is a collection Π_{DFKHE} of the main algorithms DFKHE.Setup, DFKHE.Hom, DFKHE.Eval, DFKHE.ExtEval, DFKHE.Del, DFKHE.Dec, DFKHE.Enc. We specify them right below.

- $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DFKHE.Setup}(1^\lambda, \mathsf{sp})$. DFKHE.Setup is the key setup algorithm. It is PPT. Its inputs are a security parameter λ and a set of system parameters sp. Its outputs are public parameters pp, a public key pk and a secret key sk.

- $\mathsf{sk}_{y,f} \leftarrow \mathsf{DFKHE.Hom}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, (y, f))$. DFKHE.Hom is the homomorphic key generation algorithm. It is PPT. Its inputs are public parameters pp, a public key pk, a secret key sk and a pair $(y, f) \in \mathcal{Y} \times \mathcal{F}$. Its output is a secret homomorphic key $\mathsf{sk}_{y,f}$.

- $\mathsf{sk}_{y,f_1,\cdots,f_{k+1}} \leftarrow \mathsf{DFKHE.Del}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_k}, (y, f_{k+1}))$. DFKHE.Del is the key delegation algorithm. It is PPT. Its inputs are public parameters pp, a public key pk, a function $f_{k+1} \in \mathcal{F}$ and a secret key $\mathsf{sk}_{y,f_1,\cdots,f_k}$. Its output is a delegated secret key $\mathsf{sk}_{y,f_1,\cdots,f_{k+1}}$. Furthermore, $\mathsf{sk}_{y,f_1,\cdots,f_k}$ can be produced either by DFKHE.Hom if $k = 1$, or iteratively by DFKHE.Del if $k > 1$.

- $(\mathsf{ct}, \mathbf{t}) \leftarrow \mathsf{DFKHE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu, \mathbf{t})$. DFKHE.Enc is the encryption algorithm. It is PPT. Its inputs are public parameters pp, a public key pk, a plaintext μ and a variable $\mathbf{t} \in \mathcal{T}^d$. Its output is a ciphertext ct. We call ct an encryption of μ under the variable $\mathbf{t}$.

- $\mathsf{ct}_{f_1,\cdots,f_k} \leftarrow \mathsf{DFKHE.ExtEval}(\mathsf{pp}, \mathsf{pk}, f_1, \cdots, f_k, (\mathsf{ct}, \mathbf{t}))$. DFKHE.ExtEval is the ciphertext evaluation algorithm. It is DPT. Its input are public parameters pp, public key pk, ciphertext ct and the associated variable $\mathbf{t} \in \mathcal{T}^d$. Its output is an evaluated ciphertext $\mathsf{ct}_{f_1,\cdots,f_k}$. Note that if $f_1(\mathbf{t}) = \cdots = f_k(\mathbf{t}) = y$, then we call $\mathsf{ct}_{f_1,\cdots,f_k}$ an encryption of μ under the public key $(y, f_1, \cdots, f_k)$.

- $\mu/\perp := \mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_k}, (\mathsf{ct}, \mathbf{t}))$. $\mathsf{DFKHE.Dec}$ is the decryption algorithm. It is DPT. Its inputs are a delegated secret key $\mathsf{sk}_{y,f_1,\cdots,f_k}$ and a ciphertext ct associated with $\mathbf{t} \in \mathcal{T}^d$. Its output is either a plaintext μ (if it succeeds) or a failure symbol $\perp$ (otherwise). Note that $\mathsf{DFKHE.Dec}$ succeeds in regaining a plaintext μ if $f_i(\mathbf{t}) = y$ for all $i \in [k]$.

  To regain μ, the algorithm first calls $\mathsf{DFKHE.ExtEval}(\mathsf{pp}, \mathsf{pk}, f_1, \cdots, f_k, (\mathsf{ct}, \mathbf{t}))$ and gets $\mathsf{ct}_{f_1,\cdots,f_k}$. It then uses $\mathsf{sk}_{y,f_1,\cdots,f_k}$ and opens $\mathsf{ct}_{f_1,\cdots,f_k}$.

5.2.2 Correctness

The system Π_{DFKHE} is correct if for any $\lambda \in \mathbb{Z}^+$, any $\mu \in \mathcal{M}$, any $k \in \mathbb{N}$, any $f_1, \cdots, f_k \in \mathcal{F}$, any $\mathbf{t} \in \mathcal{T}^d$, and any $y \in \mathcal{Y}$, over the randomness of $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DFKHE.Setup}(1^\lambda, \mathsf{sp})$, $(\mathsf{ct}, \mathbf{t}) \leftarrow \mathsf{DFKHE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu, \mathbf{t})$, $\mathsf{sk}_{y,f_1} \leftarrow \mathsf{DFKHE.Hom}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, (y, f_1))$ and $\mathsf{sk}_{y,f_1,\cdots,f_i} \leftarrow \mathsf{DFKHE.Del}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_{i-1}}, (y, f_i))$, $\mathsf{ct}_{f_1,\cdots,f_k} \leftarrow \mathsf{DFKHE.ExtEval}\ (\mathsf{pp}, \mathsf{pk}, f_1, \cdots, f_k, (\mathsf{ct}, \mathbf{t}))$ for all $i \in \{2, \cdots, k\}$, the following holds:

- $\Pr[\mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, \mathsf{DFKHE.Enc}(\mathsf{pp}, \mathsf{pk}, \mu, \mathbf{t})) = \mu] \geq 1 - \mathsf{negl}(\lambda)$,
- $\Pr[\mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, (\mathsf{ct}_{f_1,\cdots,f_k}, \mathbf{t})) = \mu] \geq 1 - \mathsf{negl}(\lambda)$,
- If $y = f_1(\mathbf{t}) = \cdots = f_k(\mathbf{t})$, then

$$\Pr[\mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_k}, (\mathsf{ct}, \mathbf{t})) = \mu] \geq 1 - \mathsf{negl}(\lambda).$$

- If there exists $i \in [k]$ such that $y \neq f_i(\mathbf{t})$, then

$$\Pr[\mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_k}, (\mathsf{ct}, \mathbf{t})) = \mu] \leq \mathsf{negl}(\lambda).$$

5.2.3 Security Notions

Similarly to PE, we can define the security notions of indistinguishability of ciphertexts under selective/adaptive variables against chosen plaintext/ciphertext attacks (IND–(s)VAR–ATK) for DFKHE, where ATK can be one of CPA, CCA1, CCA2. In the security notion IND–sVAR–ATK, the adversary has to release his target variable at the beginning of the game. Having received the public key from the challenger, the adversary accesses an oracle that outputs delegated secret keys. Using the information received from the oracle, the adversary challenges the challenger by submitting two plaintexts. The challenger, in turn, chooses randomly one of these two plaintexts to produce the challenge ciphertext then sends it to the adversary. Finally, the adversary has to guess which plaintext has been

encrypted. Depending on what ATK is, the adversary is allowed to query to the decryption oracle before challenging and/or after challenging.

The IND–VAR–ATK is quite the same as IND–sVAR–ATK. The difference is that in IND–VAR–ATK, the adversary does not need to announce his target variable in advance. Instead, the adversary attaches the target variable with the two challenge plaintexts.

We define the IND–sVAR–ATK security in the game $\mathsf{DFKHE}_{\mathcal{A}}^{\mathsf{IND\text{-}sVAR\text{-}ATK}}(\lambda, \mathsf{sp})$ (Figure 5.3) and then in Definition 5.2.1 below.

GAME $\mathsf{DFKHE}_{\mathcal{A}}^{\mathsf{IND\text{-}sVAR\text{-}ATK}}(\lambda, \mathsf{sp}) \Rightarrow 1/0$:

(where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$)

1. $\widehat{\mathsf{t}} = (\widehat{t_1}, \cdots, \widehat{t_d}) \leftarrow \mathcal{A}(\lambda, \mathsf{sp})$;
3. $(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DFKHE.Setup}(1^\lambda, \mathsf{sp})$;
4. $(\widehat{\mu_0}, \widehat{\mu_1}) \leftarrow \mathcal{A}^{\mathsf{DKQ}(\cdot,\cdot),\ \mathsf{DQ}_1(\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk})$;
6. $b \xleftarrow{\$} \{0,1\}$, $\widehat{\mathsf{ct}} \leftarrow \mathsf{DFKHE.Enc}(\mathsf{pp}, \mathsf{pk}, \widehat{\mathsf{t}}, \widehat{\mu_b})$;
7. $b' \leftarrow \mathcal{A}^{\widehat{\mathsf{ct}},\ \mathsf{DKQ}(\cdot,\cdot),\ \mathsf{DQ}_2(\cdot,\cdot)}(\mathsf{pp}, \mathsf{pk})$; // **NOTE**: $\mathsf{DQ}_2(\widehat{\mathsf{ct}}, \widehat{\mathsf{t}})$ is not allowed
8. If $b' = b$, return 1. Otherwise, return 0.

Queried Oracles:

- Delegated Key Oracle $\mathsf{DKQ}(y, (f_1, \cdots, f_k))$: Return $\perp$ if all $f_j(\widehat{\mathbf{t}}) = y$.
  Otherwise, return $\mathsf{sk}_{y,f_1,\cdots,f_k}$ which is computed as follows:
  Compute $\mathsf{sk}_{y,f_1} \leftarrow \mathsf{DFKHE.Hom}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}, (y, f_1))$
  and $\mathsf{sk}_{y,f_1,\cdots,f_i} \leftarrow \mathsf{DFKHE.Del}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_{i-1}}, (y, f_i))$, $\forall i \in \{2, \cdots, k\}$.
- Decryption Oracle $\mathsf{DQ}_1(\mathsf{ct}, \mathsf{t})$ (allowed only if $\mathsf{ATK} \in \{\mathsf{CCA1}, \mathsf{CCA2}\}$): Return the output of $\mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_k}, (\mathsf{ct}, \mathbf{t}))$ using the most recent key $\mathsf{sk}_{y,f_1,\cdots,f_k}$.
- Decryption Oracle $\mathsf{DQ}_2(\mathsf{ct}, \mathsf{t})$ (allowed only if $\mathsf{ATK} = \mathsf{CCA2}$): Return the output of $\mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{pk}, \mathsf{sk}_{y,f_1,\cdots,f_k}, (\mathsf{ct}, \mathbf{t}))$ using the most recent key $\mathsf{sk}_{y,f_1,\cdots,f_k}$.

Figure 5.3: Security game for DFKHE.

Definition 5.2.1 (IND–sVAR–ATK security for DFKHE)**.** *Define the advantage of the adversary $\mathcal{A}$ in the game $\mathsf{DFKHE}_{\mathcal{A}}^{\mathsf{IND\text{–}sVAR\text{–}ATK}}(\lambda)$ as*

$$\mathsf{Adv}_{\mathcal{A},\ \mathsf{DFKHE}}^{\mathsf{IND\text{–}sVAR\text{–}ATK}}(\lambda) := \left| \Pr[\mathsf{DFKHE}_{\mathcal{A}}^{\mathsf{IND\text{–}sVAR\text{–}ATK}}(\lambda, \mathsf{sp}) \Rightarrow 1] - \frac{1}{2} \right|.$$

We say that a DFKHE is IND–sVAR–ATK secure if, for any polynomial–time adversary $\mathcal{A}$, it holds that

$$\mathsf{Adv}_{\mathcal{A},\ \mathsf{DFKHE}}^{\mathsf{IND\text{–}sVAR\text{–}ATK}}(\lambda) \leq \mathsf{negl}(\lambda).$$

5.3 Generic PE Construction from DFKHE

Suppose that a ciphertext produced in the PE scheme is associated with tags $\mathbf{t} := (t_1, \cdots, t_d)$, and a secret key can be punctured on puncture t^*. Recall that the puncturing property of

PE says that if t^* is different from all elements of $\mathbf{t}$. The underlying plaintext of the ciphertext embedded with $\mathbf{t}$ can be recovered utilising the key punctured by t^*. Otherwise, the secret key cannot recover the underlying plaintext from the ciphertext.

The generic construction stems from the observation that the puncturing property of PE can be implemented through a family of functions that can check "equality". More specifically, we can implement ciphertext tags as an input vector of dimension d, say $\mathbf{t} := (t_1, \cdots, t_d)$. Also, for each puncture t^*, we can define a function indexed by the puncture, say f_{t^*}, in such a way that the returned value of $f_{t^*}(\mathbf{t})$ will depend on the equality of t^* and each element of $\mathbf{t}$. Formally, let $\lambda, d = d(\lambda) \in \mathbb{Z}^+$. Let $\mathcal{T} = \mathcal{T}(\lambda)$ be a finite set representing the *tag space* and $\mathcal{Y} = \mathcal{Y}(\lambda)$ be also a finite set. Additionally, we fix a special element y_0 in $\mathcal{Y}$. We consider equality test functions over the tag space $\mathcal{T}$ which are included in a family $\mathcal{F}$ defined as:

$$\mathcal{F} = \mathcal{F}(\lambda) := \left\{ f_{t^*} | t^* \in \mathcal{T}, \forall \mathbf{t} = (t_1, \cdots, t_d), f_{t^*} : \mathcal{T}^d \to \mathcal{Y} \right\}, \tag{5.1}$$

where

$$f_{t^*}(\mathbf{t}) = \begin{cases} y_0 & t^* \neq t_i, \forall i \in [d], \\ y_{t^*,\mathbf{t}} \in \mathcal{Y} \setminus \{y_0\} & otherwise. \end{cases} \tag{5.2}$$

Here, $y_{t^*,\mathbf{t}}$ is a value varied via t^* and $\mathbf{t}$. We will run DFKHE on the family to have PE.

5.3.1 The Generic Construction

Given a (λ, sp)–DFKHE system Π_{DFKHE} consisting of the algorithms DFKHE.Setup, DFKHE.Hom, DFKHE.Enc, DFKHE.ExtEval, DFKHE.Del and DFKHE.Dec, where sp contains system parameters including $d, \mathcal{T}, \mathcal{Y}$ and $\mathcal{F}$ which are defined earlier in Section 5.3.

From Π_{DFKHE}, we build a PE system Π_{PE} = (PE.Setup, PE.Enc, PE.Pun, PE.Dec) whose both tags and punctures belong to the space $\mathcal{T}$. The scheme Π_{PE} is described below:

- $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{PE.Setup}(1^\lambda, \mathsf{sp})$. For input a security parameter λ and the maximum number d of tags per ciphertext, run $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DFKHE.Setup}(1^\lambda, \mathsf{sp})$, and return $\mathsf{pk} := \mathsf{pk}$, and $\mathsf{sk}_0 := \mathsf{sk}$.

- $\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mathsf{pk}, \mu, \mathbf{t} = (t_1, \cdots, t_d))$. For a public key pk, a plaintext μ, and ciphertext tags $\mathbf{t} = (t_1, \cdots, t_d)$, return $\mathsf{ct} \leftarrow \mathsf{DFKHE.Enc}(\mathsf{pk}, \mu, \mathbf{t})$.

- $\mathsf{sk}_i \leftarrow \mathsf{PE.Pun}(\mathsf{pk}, \mathsf{sk}_{i-1}, t_i^*)$. For input pk, sk_{i-1} and a punctured tag t_i^*,

  - If $i = 1$: Output $\mathsf{sk}_1 \leftarrow \mathsf{DFKHE.Hom}(\mathsf{pk}, \mathsf{sk}_0, (y_0, f_{t_1^*}))$.

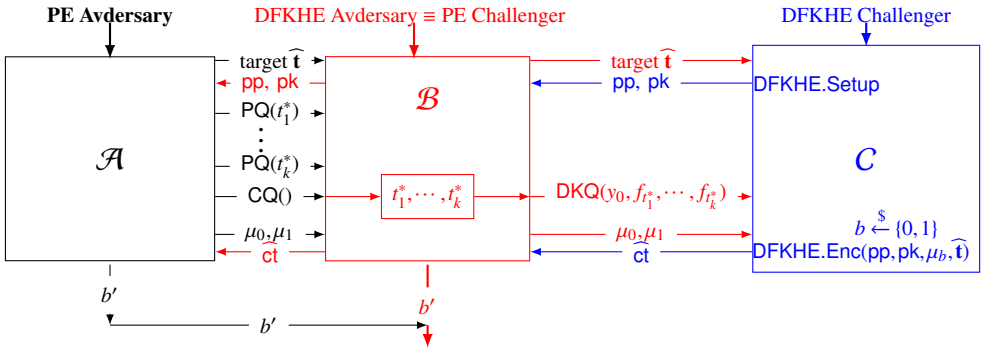

Figure 5.4: Reduction from DFKHE to PE.

– If $i \geq 2$: Output $\mathsf{sk}_i \leftarrow \mathsf{DFKHE.Del}(\mathsf{pk}, \mathsf{sk}_{i-1}, (y_0, f_{t_i^*}))$.

- $\mu/\perp := \mathsf{PE.Dec}(\mathsf{PE.pk}, (\mathsf{sk}_i, (t_1^*, \cdots, t_i^*)), (\mathsf{ct}, \mathbf{t}))$. For input the public key pk, a puncture key sk_i together with punctures $(t_1^*, \cdots, t_i^*)$, a ciphertext ct and its associated tags $\mathbf{t} = (t_1, \cdots, t_d)$, the algorithm first checks whether or not $f_{t_1^*}(\mathbf{t}) = \cdots = f_{t_i^*}(\mathbf{t}) = y_0$. If not, the algorithm returns $\perp$. Otherwise, it returns $\mu/\perp \leftarrow \mathsf{DFKHE.Dec}(\mathsf{sk}_i, \mathsf{ct})$.

5.3.2 Correctness

By the generic construction, it is easy to check that the correctness of Π_{PE} is guaranteed by that of Π_{DFKHE}.

5.3.3 Security

Theorem 5.3.1. *Assume that the underlying* Π_{DFKHE} *is selectively secure against chosen plaintext attacks. Then so is the* Π_{PE}. *described in Section 5.3.1.*

Proof. We prove by give a reduction from Π_{DFKHE} to Π_{PE}. Suppose that $\mathcal{A}$ is an adversary that can win the security game for the Π_{PE} with probability δ. We can construct an adversary $\mathcal{B}$ that wins the security game for the Π_{DFKHE} with probability at least the same.

The adversary $\mathcal{B}$ will take the role of the PE challenger. It gets the target tag from the PE adversary $\mathcal{A}$ and treats the target as its own target. It will also change the queries made by $\mathcal{A}$ into a form that is compatible with the DFKHE security game, and then forwards these reproduced queries to the DFKHE challenger $\mathcal{C}$. This way, $\mathcal{B}$ can simulate the responses for the queries coming from $\mathcal{A}$. Finally, $\mathcal{B}$ can make $\mathcal{A}$ output the guessed value of bit b. We visualise the reduction and the relation among $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ in Figure 5.4. In more detail, the reduction is as follows:

Initialise. The goal of $\mathcal{B}$ is to win the security game for (λ, sp)–DFKHE Π_{DFKHE} =

{DFKHE.Setup, DFKHE.Hom, DFKHE.Enc, DFKHE.Dec, DFKHE.ExtEval, and DFKHE.Del}.

Targeting. $\mathcal{B}$ simply uses the target $\widehat{\mathbf{t}} = (\widehat{t_1}, \cdots, \widehat{t_d})$ that $\mathcal{A}$ released in the game for PE Π_{PE} as $\mathcal{B}$'s own. $\mathcal{B}$ then sends the target to C.

Setup. $\mathcal{B}$ initializes $\mathsf{pT}^* \leftarrow \emptyset$ and $\mathsf{cT}^* \leftarrow \emptyset$. Here pT^* (respectively, cT^*) is a set containing punctured tags (respectively, all punctured tags at the time of the first corruption query). Once $\mathcal{B}$ gets the public key pk sent from C in the DFKHE security game, $\mathcal{B}$ gives $\mathcal{A}$ pk as the public key for the PE security game.

Query 1. Recall that for the PE security game, $\mathcal{A}$ can adaptively make puncture queries $\mathsf{PQ}(t_k^*)$, and it must make a corruption query $\mathsf{CQ}(\cdot)$ before challenging.

Every time, $\mathcal{A}$ makes a puncture query $\mathsf{PQ}(t_k^*)$, $\mathcal{B}$ does not have to reply the query. Instead, it just simply appends t_k^* to pT^* which will be updated as $\mathsf{pT}^* := \{t_1^*, t_2^*, \cdots, t_k^*\}$. Whenever $\mathcal{A}$ makes the corruption query $\mathsf{CQ}(\mathsf{pT}^*)$, $\mathcal{B}$ sets $\mathsf{cT}^* \leftarrow \mathsf{pT}^*$ and check whether or not

$$\{\widehat{t_1}, \cdots, \widehat{t_d}\} \cap \mathsf{cT}^* \neq \emptyset. \tag{5.3}$$

If not, $\mathcal{B}$ returns $\perp$. Otherwise, $\mathcal{B}$ sends the query $\mathsf{DKQ}(y_0, f_{t_1^*}, \cdots, f_{t_k^*})$ to C who will return the answer. Notice that the condition for a query $\mathsf{DKQ}(y_0, f_{t_1^*}, \cdots, f_{t_k^*})$ to be accepted is that there is at least one $j \in [k]$ satisfying $f_{t_j^*}(\widehat{\mathbf{t}}) \neq y_0$. This is guaranteed by the condition in Equation (5.3).

Challenge. $\mathcal{B}$ will forward to C two plaintexts μ_0, μ_1 that $\mathcal{A}$ has submitted to $\mathcal{B}$.

Query 2. Same as Query 1.

Guess. At this phase, $\mathcal{A}$ returns a bit b' for the PE security game. $\mathcal{B}$ just takes b' as its guess for the DFKHE security game.

We claim that the simulated PE environment for $\mathcal{A}$ is *perfect* in its view as $\mathcal{B}$ plays the role of PE challenger very well. Of course, $\mathcal{B}$ is also communicating well with the DFKHE challenger. What $\mathcal{B}$ gets from and what $\mathcal{B}$ sends back to $\mathcal{A}$ are what it forwards to and gets from C. That is, the information they are getting and sending is the same. Therefore, the successful probability of $\mathcal{A}$ in the PE game and that of $\mathcal{B}$ in the DFKHE game are the same. The proof follows. □

5.4 DFKHE Construction over Lattices

5.4.1 The Construction

First, we summarise the system parameters and give their descriptions in Table 5.1.

Table 5.1: System parameters in our lattice–based DFKHE construction.

Parameters	Definition
λ	Security parameter
η	maximum number functions to be delegated
n	# row of matrices $\mathbf{A}, \mathbf{G}, \mathbf{B}_i, \mathbf{U}$
q	System modulus
$\sigma_1, \cdots, \sigma_\eta$	Gaussian parameters used in $\mathsf{RandBasis}$
$\epsilon \in (0,1)$	control the trade–off of efficiency and security level
m	bit–length of plaintexts, # column of matrices $\mathbf{A}, \mathbf{G}, \mathbf{B}_i, \mathbf{U}$
d	maximum number of tags
(B, ν)	parameters for the bounded distribution χ used in DLWE

We construct the lattice–based DFKHE using the lattice trapdoor recalled in Section 2.5.3 and the lattice homomorphic evaluations in Section 2.5.4. We also involve the gadget matrix $\mathbf{G}$ defined in Section 2.5.2. In Section 5.4.3, we also involve the basis $\mathbf{T_G}$ for $\Lambda_q^\perp(\mathbf{G})$. We will briefly describe the lattice–based DFKHE right below.

In the construction of lattice–based DFKHE Π_{DFKHE}, the plaintext will be encrypted under the public key $\mathsf{pk} = \{\mathbf{A}, \mathbf{G}, \mathbf{B}_1, \cdots \mathbf{B}_d, \mathbf{U}\}$, where $(\mathbf{A}, \mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, $\mathbf{B}_1, \cdots, \mathbf{B}_d \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and under a list of at most d ciphertext tags $t_1, \cdots t_d \in \mathbb{Z}_q$ as well. Note that $\mathbf{T_A}$ will be kept as the initial secret key sk, which can decrypt any ciphertext and it can also be used to produce any delegated key. To encrypt a plaintext $\mu \in \{0,1\}^m$, we construct

$$\mathbf{H} := [\mathbf{A}|t_1\mathbf{G} + \mathbf{B}_1|\cdots|t_d\mathbf{G} + \mathbf{B}_d]$$

and then compute $\mathbf{c} = \mathbf{H}^\top\mathbf{s} + \mathbf{e}$ using a random vector $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$ as a ephemeral secret key, while $\mathbf{e}$ is obtained by choosing $\mathbf{e}_{\mathsf{in}} \leftarrow \mathcal{D}_{\mathbb{Z}_q^m,\sigma_1}$, sampling $\mathbf{S}_1, \cdots, \mathbf{S}_d \xleftarrow{\$} \{-1,1\}^{m\times m}$ and assigning $\mathbf{e}$ to $(\mathbf{e}_{\mathsf{in}}, \mathbf{e}_1, \cdots, \mathbf{e}_d) := (\mathbf{I}_m|\mathbf{S}_1|\cdots|\mathbf{S}_d)^\top\mathbf{e}_{\mathsf{in}}$. This way, we have a ciphertext of the form $(\mathsf{ct} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_1, \cdots, \mathbf{c}_d, \mathbf{c}_{\mathsf{out}}), (t_1, \cdots, t_d))$, where $\mathbf{c}_{\mathsf{in}} = \mathbf{A}^\top\mathbf{s} + \mathbf{e}_{\mathsf{in}}$, $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{U}^\top\mathbf{s} + \mathbf{e}_{\mathsf{out}} + \mu\lceil\frac{q}{2}\rceil$, $\mathbf{e}_{\mathsf{out}} \leftarrow \mathcal{D}_{\mathbb{Z}_q^m,\sigma_1}$, and $\mathbf{c}_i = (t_i\mathbf{G} + \mathbf{B}_i)^\top\mathbf{s} + \mathbf{e}_i$ for $i \in [d]$. Obviously, the size of the ciphertext is proportional to the number of ciphertext tags.

The key $\mathsf{sk}_{y,f_1,\cdots,f_i}$ delegated on the value y and functions $(f_1, \cdots, f_i)$ consists of a short basis (i.e., trapdoor) $\mathbf{T}_{y,f_1,\cdots,f_i}$ for the certain matrix $[\mathbf{A}|y\mathbf{G} + \mathbf{B}_{f_1}|\cdots|y\mathbf{G} + \mathbf{B}_{f_i}]$, where for all $j \in [i]$, $\mathbf{B}_{f_j}$ is computed by evaluating $\mathbf{B}_{f_j} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f_{t_j^*}, (\mathbf{B}_k)_{k=1}^d)$. This way, we can compute $\mathbf{T}_{y,f_1}$ from $\mathsf{sk} = \mathbf{T_A}$, compute $\mathbf{T}_{y,f_1,f_2}$ from $\mathbf{T}_{y,f_1}$ and so on, using $\mathsf{ExtBasisRight}$. For instance, to compute $\mathbf{T}_{y,f_1,\cdots,f_i}$ using $\mathbf{T}_{y,f_1,\cdots,f_{i-1}}$, we just compute and append $\mathbf{B}_{f_i}$ to $[\mathbf{A}|y\mathbf{G} + \mathbf{B}_{f_1}|\cdots|y\mathbf{G} + \mathbf{B}_{f_{i-1}}]$, and then run

$$\mathbf{T}_{y,f_1,\cdots,f_i} \leftarrow \mathsf{ExtBasisRight}([\mathbf{A}|y\mathbf{G} + \mathbf{B}_{f_1}|\cdots|y\mathbf{G} + \mathbf{B}_{f_{i-1}}|\mathbf{B}_{f_i}], \mathbf{T}_{y,f_1,\cdots,f_{i-1}}, \sigma_2).$$

The size of the delegated key is proportional to the number of functions f_i's.

For decryption, on input a ciphertext $\mathsf{ct}_{\mathbf{t}}$ associated with ciphertext tags $\mathbf{t} = (t_1, \cdots, t_d)$, we consider two cases:

- **Case 1:** Using the key $\mathsf{sk} = \mathbf{T}_{\mathbf{A}}$, directly compute a short basis $\mathbf{T}$ for $[\mathbf{A}|t_1\mathbf{G} + \mathbf{B}_1|\cdots|t_d\mathbf{G} + \mathbf{B}_d]$, then use $\mathbf{T}$ to sample $\mathbf{R}$ such that $\mathbf{R}^\top[\mathbf{A}|t_1\mathbf{G} + \mathbf{B}_1|\cdots|t_d\mathbf{G} + \mathbf{B}_d] = \mathbf{U} \pmod q$. The plaintext is recovered as

$$\bar{\mu} := (\bar{\mu}_1, \cdots, \bar{\mu}_m) \leftarrow \mathbf{c}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{c}_{\mathsf{in}}|\mathbf{c}_1|\cdots|\mathbf{c}_d).$$

- **Case 2:** Using the key $\mathsf{sk}_{y,f_1,\cdots,f_i}$ delegated on functions $\{f_1, \cdots, f_i\}$, compute $\mathbf{c}_{f_j} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(f_{t_j^*}, ((t_k, \mathbf{B}_k, \mathbf{c}_k))_{k=1}^d)$ for $j \in [i]$. At this point, we can set up the matrix $[\mathbf{A}|y\mathbf{G} + \mathbf{B}_{f_1}|\cdots|y\mathbf{G} + \mathbf{B}_{f_i}]$ and take $\mathbf{T}_{y,f_1,\cdots,f_i}$ to sample a short matrix $\mathbf{R}$ such that $\mathbf{R}^\top[\mathbf{A}|y\mathbf{G} + \mathbf{B}_{f_1}|\cdots|y\mathbf{G} + \mathbf{B}_{f_i}] = \mathbf{U} \pmod q$.

  In this case, the plaintext is calculated as $\bar{\mu} := \mathbf{c}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{c}_{\mathbf{in}}|\mathbf{c}_{f_1}|\cdots|\mathbf{c}_{f_i})$. Thanks to the properties of evaluation algorithms, decryption is successful if and only if $f_j(\mathbf{t}) = y$, meaning that $t_j^* \neq t_k$ for all $(j,k) \in [i] \times [d]$ as required.

Table 5.2 below shows the sizes of the public key, the initial secret key, delegated secret key and ciphertext in the LWE–based DFKHE construction.

Table 5.2: Sizes in our lattice–based DFKHE.

Public key size ($\mathbf{G}$ not included)	**Initial secret key size** ($\mathbf{T}_{\mathbf{A}}$)	**Delegated secret key size** ($\mathsf{sk}_{y,f_1,\cdots,f_k}, k \in [\eta]$)	**Ciphertext size** (ct)
$(d+1)\cdot\mathbb{Z}_q^{n\times m}$	$1\cdot\mathbb{Z}^{m\times m}$	$1\cdot D_{\mathbb{Z},\sigma_k}^{(k+1)m\times(k+1)m}$	$(d+2)\cdot\mathbb{Z}_q^m$

Formally, the DFKHE system Π_{DFKHE} consists of $\mathsf{DFKHE.Setup}$, $\mathsf{DFKHE.Hom}$, $\mathsf{DFKHE.Del}$, $\mathsf{DFKHE.Enc}$, $\mathsf{DFKHE.ExtEval}$ and $\mathsf{DFKHE.Dec}$ presented below.

- $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{DFKHE.Setup}(1^\lambda, \mathsf{sp})$. On a security parameter $\lambda \in \mathbb{N}$ and system parameters sp (which includes $d = d(\lambda)$), do the following:

  1. Choose $n = n(\lambda)$, $q = q(\lambda)$ to be positive integers such that $d < q$.
  2. Let $\eta \in \mathbb{N}$ be the maximum number of functions that can be delegated in the system.
  3. Let $\sigma_1, \cdots, \sigma_\eta$ be Gaussian parameters.
  4. Let $\epsilon \in (0,1)$ be a constant that helps determine the trade–off between the efficiency and the security level of the system, as mentioned in Lemma 2.4.1.
  5. The function family $\mathcal{F} := \{f | f : (\mathbb{Z}_q)^d \to \mathbb{Z}_q\}$ of functions over $\mathbb{Z}_q$ that can be evaluated by lattice evaluation algorithms ($\mathsf{Eval}_{\mathsf{pk}}$, $\mathsf{Eval}_{\mathsf{ct}}$, $\mathsf{Eval}_{\mathsf{sim}}$). Take the constant $\beta_{\mathcal{F}}$ defined for $\mathcal{F}$ via Lemma 2.5.11.

6. Choose m.
7. The plaintext space is $\mathcal{M} := \{0,1\}^m$, the tag space is $\mathcal{T} := \mathbb{Z}_q$.
8. Let χ be a (B,ν)–bounded noise distribution for which the $(n,2m,q,\chi)$–DLWE is hard.
9. Generate $(\mathbf{A},\mathbf{T_A}) \leftarrow \mathsf{TrapGen}(1^n,1^m,q)$, sample $\mathbf{U},\mathbf{B}_1,\cdots,\mathbf{B}_d \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$.
10. Take the gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$.
11. Output the public key $\mathsf{pk} = \{\mathbf{A},\mathbf{G},\mathbf{B}_1,\cdots\mathbf{B}_d,\mathbf{U}\}$ and the initial secret key $\mathsf{sk} := \mathbf{T_A}$.

- $\mathsf{sk}_{y,f_1} \leftarrow \mathsf{DFKHE.Hom}(\mathsf{sk},(y,f_1))$. On input the initial secret key sk and a pair $(y,f_1) \in \mathbb{Z}_q \times \mathcal{F}$, the algorithm peforms the following:

  1. $\mathbf{B}_{f_1} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f_1,(\mathbf{B}_k)_{k=1}^d)$, $\mathbf{E}_{y,f_1} \leftarrow \mathsf{ExtBasisLeft}([\mathbf{A}|y\mathbf{G}+\mathbf{B}_{f_1}],\mathbf{T_A})$.
  2. $\mathbf{T}_{y,f_1} \leftarrow \mathsf{RandBasis}([\mathbf{A}|y\mathbf{G}+\mathbf{B}_{f_1}],\mathbf{E}_{y,f_1},\sigma_1)$.
  3. Output the secret key $\mathsf{sk}_{y,f_1} := \mathbf{T}_{y,f_1}$.
     Here, we set $\sigma_1 = \omega(\beta_{\mathcal{F}} \cdot 20\sqrt{m} \cdot \sqrt{\log(2m)})$ for the security proof to work.

- $\mathsf{sk}_{y,f_1,\cdots,f_\eta} \leftarrow \mathsf{DFKHE.Del}(\mathsf{sk}_{y,f_1,\cdots,f_{\eta-1}},(y,f_\eta))$. Taking key $\mathsf{sk}_{y,f_1,\cdots,f_{\eta-1}} := \mathbf{T}_{y,f_1,\cdots,f_{\eta-1}}$ and a pair $(y,f_\eta) \in \mathbb{Z}_q \times \mathcal{F}$ as input, $\mathsf{DFKHE.Del}$ does the following:

  1. $\mathbf{B}_{f_\eta} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f_\eta,(\mathbf{B}_k)_{k=1}^d)$.
  2. $\mathbf{E}_{y,f_1,\cdots,f_\eta} \leftarrow \mathsf{ExtBasisLeft}([\mathbf{A}|y\mathbf{G}+\mathbf{B}_{f_1}|\cdots|y\mathbf{G}+\mathbf{B}_{f_{\eta-1}}|y\mathbf{G}+\mathbf{B}_{f_\eta}],\mathbf{T}_{y,f_1,\cdots,f_{\eta-1}})$.
  3. $\mathbf{T}_{y,f_1,\cdots,f_\eta} \leftarrow \mathsf{RandBasis}([\mathbf{A}|y\mathbf{G}+\mathbf{B}_{f_1}|\cdots|y\mathbf{G}+\mathbf{B}_{f_{\eta-1}}|y\mathbf{G}+\mathbf{B}_{f_\eta}],\mathbf{E}_{y,f_1,\cdots,f_\eta},\sigma_\eta)$.
  4. Output the secret key $\mathsf{sk}_{y,f_1,\cdots,f_\eta} := \mathbf{T}_{y,f_1,\cdots,f_\eta}$.

  We set $\sigma_\eta = \sigma_1 \cdot \omega(\sqrt{m\log m})^{\eta-1}$ and discuss on setting parameters in detail later.

- $\mathsf{ct_t} \leftarrow \mathsf{DFKHE.Enc}(\mu,\mathsf{pk},\mathbf{t})$. For the input consiting of (a plaintext $\mu = (\mu_1,\cdots,\mu_m) \in \mathcal{M}$, the public key pk and ciphertext tags $\mathbf{t} = (t_1,\cdots,t_d) \in \mathcal{T}^d$), perform the following steps:

  1. Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\mathbf{e}_{\mathsf{out}},\mathbf{e}_{\mathsf{in}} \leftarrow \chi^m$, and $\mathbf{S}_1,\cdots,\mathbf{S}_d \xleftarrow{\$} \{-1,1\}^{m\times m}$.
  2. Compute $\mathbf{e} \leftarrow (\mathbf{I}_m|\mathbf{S}_1|\cdots|\mathbf{S}_d)^\top \mathbf{e}_{\mathsf{in}} = (\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_1^\top|\cdots|\mathbf{e}_d^\top)^\top$.
  3. Form $\mathbf{H} \leftarrow [\mathbf{A}|t_1\mathbf{G}+\mathbf{B}_1|\cdots|t_d\mathbf{G}+\mathbf{B}_d]$
  4. Compute $\mathbf{c} = \mathbf{H}^\top\mathbf{s}+\mathbf{e} \in \mathbb{Z}_q^{(d+1)m}$, $\mathbf{c} = (\mathbf{c}_{\mathsf{in}}^\top|\mathbf{c}_1^\top|\cdots|\mathbf{c}_d^\top)^\top$, where $\mathbf{c}_{\mathsf{in}} = \mathbf{A}^\top\mathbf{s}+\mathbf{e}_{\mathsf{in}}$ and $\mathbf{c}_i = (t_i\mathbf{G}+\mathbf{B}_i)^\top\mathbf{s}+\mathbf{e}_i$ for $i \in [d]$.
  5. Compute $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{U}^\top\mathbf{s}+\mathbf{e}_{\mathsf{out}}+\mu\lceil\frac{q}{2}\rceil$.

6. Output the ciphertext $(\mathsf{ct}_{\mathbf{t}} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_1, \cdots, \mathbf{c}_d, \mathbf{c}_{\mathsf{out}}), \mathbf{t})$.

- $\mathbf{c}_{f_1,\cdots,f_\eta} \leftarrow \mathsf{DFKHE.ExtEval}(f_1, \cdots, f_\eta, \mathsf{ct}_{\mathbf{t}})$. For the input (a ciphertext $\mathsf{ct}_{\mathbf{t}} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_1, \cdots, \mathbf{c}_d, \mathbf{c}_{\mathsf{out}})$ and its associated tags $\mathbf{t} = (t_1, \cdots, t_d)$, and a list of functions $f_1, \cdots, f_\eta \in \mathcal{F}$), execute the following steps:

  1. Evaluate $\mathbf{c}_{f_j} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(f_j, ((t_k, \mathbf{B}_k, \mathbf{c}_k))_{k=1}^d)$ for $j \in [\eta]$.
  2. Output the evaluated ciphertext $\mathbf{c}_{f_1,\cdots,f_\eta} := (\mathbf{c}_{f_1}, \cdots, \mathbf{c}_{f_\eta})$.

- $\mu/\perp := \mathsf{DFKHE.Dec}(\mathsf{pp}, \mathsf{ct}_{\mathbf{t}}, \mathsf{sk}_{y,f_1,\cdots,f_\eta})$. For the input (a ciphertext $\mathsf{ct}_{\mathbf{t}} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_1, \cdots, \mathbf{c}_d, \mathbf{c}_{\mathsf{out}})$, the associated tags $\mathbf{t} = (t_1, \cdots, t_d)$, and a delegated key $\mathsf{sk}_{y,f_1,\cdots,f_\eta} := \mathbf{T}_{y,f_1,\cdots,f_\eta}$), execute the following steps:

  1. If $\eta = 0$ (i.e., $\mathsf{sk}_{y,f_1,\cdots,f_\eta} = \mathsf{sk} = \mathbf{T}_{\mathbf{A}}$):
     (a) Compute $\mathbf{E} \leftarrow \mathsf{ExtBasisLeft}([\mathbf{A}|t_1\mathbf{G} + \mathbf{B}_1|\cdots|t_d\mathbf{G} + \mathbf{B}_d], \mathbf{T}_{\mathbf{A}})$.
     (b) Compute $\mathbf{T} \leftarrow \mathsf{RandBasis}([\mathbf{A}|t_1\mathbf{G} + \mathbf{B}_1|\cdots|t_d\mathbf{G} + \mathbf{B}_d], \mathbf{E}, \sigma_d)$.
     (c) Sample $\mathbf{R} \leftarrow \mathsf{SampleD}([\mathbf{A}|t_1\mathbf{G} + \mathbf{B}_1|\cdots|t_d\mathbf{G} + \mathbf{B}_d], \mathbf{T}, \mathbf{U}, \sigma_d)$.
     (d) Compute $\bar{\mu} := (\bar{\mu}_1, \cdots, \bar{\mu}_m) \leftarrow \mathbf{c}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{c}_{\mathsf{in}}^\top|\mathbf{c}_1^\top|\cdots|\mathbf{c}_d^\top)$.
  2. If $\eta > 0$:
     (a) If $\exists j \in [\eta]$ s.t. $f_j(\mathbf{t}) \neq y$, then output $\perp$. Otherwise, go to Step (b).
     (b) For $i \in [\eta]$, compute $\mathbf{B}_{f_i} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f_i, (\mathbf{B}_k)_{k=1}^d)$.
     (c) Sample $\mathbf{R} \leftarrow \mathsf{SampleD}([\mathbf{A}|y\mathbf{G} + \mathbf{B}_{f_1}|\cdots|y\mathbf{G} + \mathbf{B}_{f_\eta}], \mathbf{T}_{y,f_1,\cdots,f_\eta}, \mathbf{U}, \sigma_\eta)$.
     (d) Evaluate $(\mathbf{c}_{f_1}, \cdots, \mathbf{c}_{f_\eta}) \leftarrow \mathsf{DFKHE.ExtEval}(f_1, \cdots, f_\eta, \mathsf{ct}_{\mathbf{t}})$.
     (e) Compute $\bar{\mu} := (\bar{\mu}_1, \cdots, \bar{\mu}_m) \leftarrow \mathbf{c}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{c}_{\mathsf{in}}^\top|\mathbf{c}_{f_1}^\top|\cdots|\mathbf{c}_{f_\eta}^\top)$.
  3. For $\ell \in [m]$, if $|\bar{\mu}_\ell| < q/4$ then output $\mu_\ell = 0$; otherwise, output $\mu_\ell = 1$.

5.4.2 Correctness

The following theorem states the correctness of the DFKHE system Π_{DFKHE}.

Theorem 5.4.1 (Correctness of Π_{DFKHE})**.** *Let*

$$M := 1 + \sigma_d \sqrt{m(d+1)} \cdot \sqrt{m + 400dm^2},$$
$$N := 1 + \sigma_\eta \sqrt{m(\eta+1)} \cdot \sqrt{m + 400\eta m^2 \beta_{\mathcal{F}}^2}.$$

The DFKHE system Π_{DFKHE} *is correct if the condition*

$$\max\{M, N\} < \frac{1}{4}(q/B) \tag{5.4}$$

holds, assumming that $f_j(\mathbf{t}) = y$ *for all* $j \in [\eta]$.

Proof. We consider the first case in DFKHE.Dec where $\eta = 0$ and $\mathsf{sk}_{y,f_1,\cdots,f_\eta} = \mathsf{sk} = \mathbf{T}_{\mathbf{A}}$. Then, we have:

$$\begin{aligned}
\bar{\mu} &= \mathbf{c}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{c}_{\mathsf{in}}^\top, \mathbf{c}_1^\top, \cdots, \mathbf{c}_d^\top)^\top \\
&= \mathbf{U}^\top\mathbf{s} + \mathbf{e}_{\mathsf{out}} + \mu\lceil\frac{q}{2}\rceil - \mathbf{R}^\top[\mathbf{A}^\top|(t_1\cdot\mathbf{G}+\mathbf{B}_1)^\top|\cdots \\
&\quad |(t_d\cdot\mathbf{G}+\mathbf{B}_d)^\top]\mathbf{s} - \mathbf{R}^\top(\mathbf{e}_{\mathsf{in}}|\mathbf{e}_1|\cdots|\mathbf{e}_d) \\
&= \mathbf{U}^\top\mathbf{s} - \mathbf{R}^\top[\mathbf{A}^\top|(t_1\mathbf{G}+\mathbf{B}_1)^\top|\cdots|(t_d\mathbf{G}+\mathbf{B}_d)^\top]\mathbf{s} \\
&\quad +\mu\lceil\frac{q}{2}\rceil + \mathbf{e}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_1^\top, \cdots, \mathbf{e}_d^\top)^\top \\
&= \mu\lceil\frac{q}{2}\rceil + \mathbf{e}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_1^\top, \cdots, \mathbf{e}_d^\top)^\top.
\end{aligned}$$

Recall that $\|\mathbf{e}_{\mathsf{out}}\|_\infty \le B, \|\mathbf{e}_{\mathsf{in}}\| \le \sqrt{m}\|\mathbf{e}_{\mathsf{in}}\|_\infty \le \sqrt{m}B$. Furthermore, for $i \in [d]$, by Lemma 2.1.3, we have

$$\|\mathbf{e}_i\| = \|\mathbf{S}_i^\top\mathbf{e}_{\mathsf{in}}\| \le s_1(\mathbf{S}_i^\top)\|\mathbf{e}_{\mathsf{in}}\| \le 20\sqrt{m}\cdot\sqrt{m}\|\mathbf{e}_{\mathsf{in}}\|_\infty \le 20mB.$$

This implies that

$$\|(\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_1^\top, \cdots, \mathbf{e}_d^\top)^\top\| \le \sqrt{m + 400dm^2}\cdot B.$$

And as $\mathbf{R} \in \mathbb{Z}^{m\times m(d+1)}$ then $s_1(\mathbf{R}^\top) \le \sigma_d m\sqrt{d+1}$ by Lemma 2.3.8. Therefore,

$$\begin{aligned}
\|\mathbf{e}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_1^\top, \cdots, \mathbf{e}_d^\top)^\top\|_\infty &\le \|\mathbf{e}_{\mathsf{out}}\|_\infty + s_1(\mathbf{R}^\top)\cdot\|(\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_1^\top, \cdots, \mathbf{e}_d^\top)^\top\| \\
&\le \left(1 + \sigma_d\sqrt{m(d+1)}\cdot\sqrt{m+400dm^2}\right)\cdot B := M.
\end{aligned} \tag{5.5}$$

Now, we consider the second case in DFKHE.Dec where $\eta > 0$. Then we have

$$\begin{aligned}
\bar{\mu} &= \mathbf{c}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{c}_{\mathsf{in}}^\top|\mathbf{c}_{f_1}^\top|\cdots|\mathbf{c}_{f_\eta}^\top)^\top \\
&= \mathbf{U}^\top\mathbf{s} + \mathbf{e}_{\mathsf{out}} + \mu\lceil\frac{q}{2}\rceil - \mathbf{R}^\top[\mathbf{A}^\top|(f_1(\mathbf{t})\cdot\mathbf{G}+\mathbf{B}_{f_1})^\top|\cdots \\
&\quad |(f_\eta(\mathbf{t})\cdot\mathbf{G}+\mathbf{B}_{f_\eta})^\top]\mathbf{s} - \mathbf{R}^\top(\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{f_1}^\top|\cdots|\mathbf{e}_{f_\eta}^\top)^\top \\
&= \mathbf{U}^\top\mathbf{s} - \mathbf{R}^\top[\mathbf{A}^\top|(y\mathbf{G}+\mathbf{B}_{f_1})^\top|\cdots|(y\mathbf{G}+\mathbf{B}_{f_\eta})^\top]\mathbf{s} \\
&\quad +\mu\lceil\frac{q}{2}\rceil + \mathbf{e}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{f_1}^\top|\cdots|\mathbf{e}_{f_\eta}^\top)^\top \\
&= \mu\lceil\frac{q}{2}\rceil + \mathbf{e}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{f_1}^\top|\cdots|\mathbf{e}_{f_\eta}^\top)^\top.
\end{aligned}$$

Remark that the third equality holds only if $f_j(\mathbf{t}) = y$ for all $j \in [\eta]$. Therefore, if there exists $j \in [\eta]$ such that $f_j(\mathbf{t}) \ne y$, then the equality (hence the decryption) fails. Here as $\mathbf{R} \in \mathbb{Z}^{m\times m(\eta+1)}$ then $s_1(\mathbf{R}^\top) \le \sigma_\eta m\cdot\sqrt{\eta+1}$ by Lemma 2.3.8. By Lemma 2.5.10, $\|\mathbf{e}_{f_j}\| < \|\mathbf{e}_i\|\cdot\beta_{\mathcal{F}} = 20mB\beta_{\mathcal{F}}$, for all $j \in [\eta]$. Then

$$\|(\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_{f_1}^\top, \cdots, \mathbf{e}_{f_\eta}^\top)^\top\| \leq \sqrt{m + 400\eta m^2 \beta_{\mathcal{F}}^2} \cdot B.$$

Therefore,

$$\begin{aligned} \|\mathbf{e}_{\mathsf{out}} - \mathbf{R}^\top (\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_{f_1}^\top, \cdots, \mathbf{e}_{f_\eta}^\top)^\top\|_\infty &\leq \|\mathbf{e}_{\mathsf{out}}\|_\infty + s_1(\mathbf{R}^\top) \cdot \|\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_{f_1}^\top, \cdots, \mathbf{e}_{f_\eta}^\top)^\top\| \\ &\leq \left(1 + \sigma_\eta \sqrt{m(\eta+1)} \cdot \sqrt{m + 400\eta m^2 \beta_{\mathcal{F}}^2}\right) \cdot B := N. \end{aligned} \tag{5.6}$$

From Equations (5.5)–(5.6), choosing parameters such that

$$\max\{M, N\} < q/4$$

will ensure the success of decryption. □

5.4.3 Security Analysis

We will show that the proposed DFKHE is indistinguishably secure under selective variables against chosen plaintext attacks (IND–sVAR–CPA). The security relies on the hardness of the DLWE assumption. Formally, we state the following theorem:

Theorem 5.4.2 (IND–sVAR–CPA of Π_{DFKHE})**.** *Assuming the hardness of* $(n, 2m, q, \chi)$*–*DLWE*, the proposed DFKHE* Π_{DFKHE} *is IND–sVAR–CPA. Specifically, if there exists an adversary* $\mathcal{A}$ *that breaks the NDr–sVAR–CPA security of* Π_{DFKHE}*, then one can build a solver* $\mathcal{B}$ *that solves the* $(n, 2m, q, \chi) - \mathsf{DLWE}$ *instance.*

Proof. The proof consists of sequences of four hybrid games, which we will prove to be indistinguishable from the view of an adversary. Specifically, the first in the sequence is the original IND–sVAR–CPA game. Our goal is to show that the adversary's advantage in the original one is negligible. Whereas in the last game, the adversary's advantage is restricted to zero. We base the indistinguishability of the third and last games on the DLWE assumption. We describe these games in detail as follows:

Game 0. This game is the original game $\mathsf{DFKHE}_{\mathcal{A}}^{\mathsf{IND\text{–}sVAR\text{–}ATK}}(\lambda)$ defined in Section 5.2.3. Recall that at the beginning of the game, the adversary $\mathcal{A}$ lets the challenger know a target variable $\widehat{\mathbf{t}} = (\widehat{t_1}, \cdots, \widehat{t_d})$ that it wants to challenge later. After that, the challenger produces the public key and the secret key $(\mathsf{pk}, \mathsf{sk})$, where $\mathsf{pk} = \{\mathbf{A}, \mathbf{G}, \mathbf{B}_1, \cdots \mathbf{B}_d, \mathbf{U}\}$, and $\mathsf{sk} = \mathbf{T}_{\mathbf{A}}$, with $(\mathbf{A}, \mathbf{T}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, $\mathbf{B}_1, \cdots \mathbf{B}_d \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$, $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. Only pk is sent to $\mathcal{A}$.

The adversary $\mathcal{A}$ can make delegate key queries $\mathsf{DKQ}(y, f_1, \cdots, f_k)$ to the challenger. If $(y, (f_1, \cdots, f_k)) \in \mathbb{Z}_q \times \mathcal{F}^k$ such that $f_1(\widehat{\mathbf{t}}) = \cdots = f_k(\widehat{\mathbf{t}}) = y$ then the query is aborted. Otherwise, the query will be responded with a key $\mathsf{sk}_{y, f_1, \cdots, f_k}$.

When challenged, the challenger produces the challenge ciphertext $\widehat{\mathsf{ct}}$. By doing that, $\widehat{\mathbf{S}}_1, \cdots, \widehat{\mathbf{S}}_d$ are sampled uniformly from $\{-1,1\}^{m\times m}$ at Step 1 of DFKHE.Enc.

Game 1. This game keeps almost things the same as Game 0, except some slight changes. Namely, $\widehat{\mathbf{S}}_1, \cdots, \widehat{\mathbf{S}}_d \in \{-1,1\}^{m\times m}$ are generated before challenging. Furthermore, for $i \in [d]$, matrix $\mathbf{B}_i$ can be computed as $\mathbf{B}_i := \mathbf{A}\widehat{\mathbf{S}}_i - \widehat{t}_i\mathbf{G}$.

Game 2. This game just modifies Game 1 in the point that matrix $\mathbf{A}$ is sampled uniformly at random from $\mathbb{Z}_q^{n\times m}$ instead of being generated via TrapGen.

Having no $\mathbf{T}_\mathbf{A}$, the challenger now uses the public basis $\mathbf{T}_\mathbf{G}$ for $\Lambda_q^\perp(\mathbf{G})$ as the key sk.

The challenger replies to a delegated key query $\mathsf{DKQ}(y, f_1, \cdots, f_k)$ as follows:

1. First, the challenger checks if $f_1(\widehat{\mathbf{t}}) = f_k(\widehat{\mathbf{t}}) = y$. If yes, it aborts the query. If not, i.e., there is some j with $f_j(\widehat{\mathbf{t}}) \neq y$, the challenger will go to next step. Suppose that $j = k$, that is, $f_k(\widehat{\mathbf{t}}) \neq y$.
2. Now the challenger uses evaluation algorithms to compute $\widehat{\mathbf{S}}_{f_i}$'s and $\mathbf{B}_{f_i}$'s. Specifically, for all $i \in [k]$, $\widehat{\mathbf{S}}_{fi} \leftarrow \mathsf{Eval}_{\mathsf{sim}}(f_i, ((\widehat{t}_j, \widehat{\mathbf{S}}_j))_{j=1}^d, \mathbf{A})$, $\mathbf{B}_{fi} = \mathbf{A}\widehat{\mathbf{S}}_{fi} - f_i(\widehat{\mathbf{t}})\mathbf{G}$. Notice that $\mathbf{B}_{f_i} = \mathsf{Eval}_{\mathsf{pk}}(f_i, (\mathbf{B}_j)_{j=1}^d)$. Here, by Item 3 of Lemma 2.5.10, we can bound $s_1(\widehat{\mathbf{S}}_{f_i}) \le \beta_\mathcal{F}$.
3. Compute $\mathbf{E}_{y,f_1,\cdots,f_k} \leftarrow \mathsf{ExtBasisRight}([\mathbf{A}|\mathbf{A}\widehat{\mathbf{S}}_{f_1} + (y - f_1(\widehat{\mathbf{t}}))\mathbf{G}|\cdots|\ \mathbf{A}\widehat{\mathbf{S}}_{f_k} + (y - f_k(\widehat{\mathbf{t}}))\mathbf{G}], \mathbf{T}_\mathbf{G})$. We can bound $\|\mathbf{E}_{y,f_1,\cdots,f_k}\| \le \|\widetilde{\mathbf{T}_\mathbf{G}}\|(1 + s_1(\widehat{\mathbf{S}}_{f_k})) = \sqrt{5}(1+\beta_\mathcal{F})$ for all $k \in [\eta]$ by Item 2 of Lemma 2.5.4.
4. Now, re–randomise $\mathbf{E}_{y,f_1,\cdots,f_k}$ to get $\mathbf{T}_{y,f_1,\cdots,f_k} \leftarrow \mathsf{RandBasis}([\mathbf{A}|\mathbf{A}\widehat{\mathbf{S}}_{f_1} + (y - f_1(\widehat{\mathbf{t}}))\mathbf{G}|\cdots|\mathbf{A}\widehat{\mathbf{S}}_{f_k} + (y - f_k(\widehat{\mathbf{t}}))\mathbf{G}], \mathbf{E}_{y,f_1,\cdots,f_k}, \sigma_k)$.
5. The challenger will return $\mathsf{sk}_{y,f_1,\cdots,f_k} := \mathbf{T}_{y,f_1,\cdots,f_k}$ back to the adversary.

Game 3. In this game the advantage of the adversary $\mathcal{A}$ is zero, because the two components $\widehat{\mathbf{c}}_{\mathsf{in}}$, $\widehat{\mathbf{c}}_{\mathsf{out}}$ in the challenge ciphertext $\widehat{\mathsf{ct}}$ is chosen randomly at random over the ciphertext space. Whereas, computing remaining components $\widehat{\mathbf{c}}_1, \cdots, \widehat{\mathbf{c}}_d$ of the challenge ciphertext and everything else in Game 2 are unchanged in this game. Hence, it is independent of the challenge plaintext.

Let $\mathbf{W}_i$ be the event that the adversary $\mathcal{A}$ wins Game i. Then

$$\left|\Pr[\mathbf{W}_3] - \frac{1}{2}\right| = 0.$$

Now, we claim and prove the indistinguishability of the above games through the following lemmas.

Lemma 5.4.1. Game 0 *and* Game 1 *are indistinguishable under the view of the adversary* $\mathcal{A}$*. That is,*

$$|\Pr[\mathbf{W}_1] - \Pr[\mathbf{W}_0]| \leq \mathsf{negl}(\lambda).$$

Proof. This is simply thanks to the leftover hash lemma (i.e., Lemma 2.5.1). In fact, Game 0 and Game 1 are just different in these two tuples $(\mathbf{A}, \mathbf{B}_i, \widehat{\mathbf{S}}_i^\top \mathbf{e}_{\mathsf{in}})$ and $(\mathbf{A}, \mathbf{A}\widehat{\mathbf{S}}_i - t_i\mathbf{G}, \widehat{\mathbf{S}}_i^\top \mathbf{e}_{\mathsf{in}})$. However, Lemma 2.5.1 ensures that their joint distributions are statistically close. Therefore, the adversary cannot distinguish them. □

Lemma 5.4.2. Game 1 *and* Game 2 *are indistinguishable under the view of the adversary* $\mathcal{A}$*. That is,*

$$|\Pr[\mathbf{W}_2] - \Pr[\mathbf{W}_1]| \leq \mathsf{negl}(\lambda).$$

Proof. These two games are different in the way the challenger replies $\mathsf{DKQ}(y, f_1, \cdots, f_k)$. However, we can see that, the returned key $\mathsf{sk}_{y,f_1,\cdots,f_k} := \mathbf{T}_{y,f_1,\cdots,f_k}$ produced using the trapdoor $\mathbf{T}_{\mathbf{A}}$ and the one produced using the trapdoor $\mathbf{T}_{\mathbf{G}}$ have the same distribution. This is because they both are also the output of the algorithm RandBasis. □

Lemma 5.4.3. Game 2 *and* Game 3 *are indistinguishable under the view of the adversary* $\mathcal{A}$*. That is,*

$$|\Pr[\mathbf{W}_3] - \Pr[\mathbf{W}_2]| \leq \mathsf{negl}(\lambda).$$

Proof. We prove this lemma using a reduction from the DLWE problem. By contradiction, suppose that $\mathcal{A}$ is able to distinguish Game 2 from Game 3 and its advantage is non–negligible. The reduction constructs a DLWE solver $\mathcal{B}$ employing the distinguisher $\mathcal{A}$. More specifically, the reduction works as follows:

$(n, 2m, q, \chi)$**–DLWE instance.** The solver $\mathcal{B}$ is given a matrix $\mathbf{F} := [\mathbf{A}|\mathbf{U}] \xleftarrow{\$} \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{n\times m}$, and a vector $\mathbf{c} := (\mathbf{c}_{\mathsf{in}}^\top | \mathbf{c}_{\mathsf{out}}^\top)^\top \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m$. Here, there are only two possibilities for $\mathbf{c}$. Namely,

- **Case 1:** $\mathbf{c}$ is random in $\mathbb{Z}_q^{2m}$; or
- **Case 2:** $\mathbf{c}$ is LWE samples, $\mathbf{c} = \mathbf{F}^\top \mathbf{s} + \mathbf{e}$, for some random vector $\mathbf{s} \in \mathbb{Z}_q^n$ and $\mathbf{e} := (\mathbf{e}_{\mathsf{in}}^\top | \mathbf{e}_{\mathsf{out}}^\top)^\top \leftarrow \chi^{2m}$.

The solver $\mathcal{B}$ wants to know which case $\mathbf{c}$ belongs to. Note that if $\mathbf{c}$ belongs to **Case 2**, then one can parse it as

$$\mathbf{c}_{\mathsf{in}} = \mathbf{A}^\top \mathbf{s} + \mathbf{e}_{\mathsf{in}}, \quad \mathbf{c}_{\mathsf{out}} = \mathbf{U}^\top \mathbf{s} + \mathbf{e}_{\mathsf{out}}. \tag{5.7}$$

Initialise. $\mathcal{B}$ receives from $\mathcal{A}$ the target variable $\widehat{\mathbf{t}} = (\widehat{t_1}, \cdots, \widehat{t_d})$.

Setup. $\mathcal{B}$ follows Game 2 to generate the public key and secret key. Namely, it samples $\widehat{\mathbf{S}}_1, \cdots, \widehat{\mathbf{S}}_d \xleftarrow{\$} \{-1,1\}^{m\times m}$ and sets $\mathbf{B}_i := \mathbf{A}\widehat{\mathbf{S}}_i - \widehat{t_i}\mathbf{G}$ for $i \in [d]$. The public key is $\mathsf{pk} = (\mathbf{A}, \mathbf{G}, \mathbf{B}_1, \cdots, \mathbf{B}_d, \mathbf{U})$ will be sent to $\mathcal{A}$. The secret key is $\mathsf{sk} = \mathbf{T}_{\mathbf{G}}$.

Query. For delegated key queries of $\mathcal{A}$, $\mathcal{B}$ responds following Game 2.

Challenge. Having received two plaintexts μ_0 and μ_1, $\mathcal{B}$ chooses a bit $b \xleftarrow{\$} \{0,1\}$ uniformly at random , then computes $\widehat{\mathbf{c}}_{\mathsf{out}} \leftarrow \mathbf{c}_{\mathsf{out}} + \mu_b\lceil\frac{q}{2}\rceil \in \mathbb{Z}_q$ and $\widehat{\mathbf{c}} \leftarrow [\mathbf{I}_m|\widehat{\mathbf{S}}_1|\cdots|\widehat{\mathbf{S}}_d]^\top \mathbf{c}_{\mathsf{in}} \in \mathbb{Z}_q^{(d+1)m}$.

- If $\mathbf{c}$ belongs to **Case 2**, i.e., $\mathbf{c}_{\mathsf{in}}$, $\mathbf{c}_{\mathsf{out}}$ satisfy Equation (5.7), then

$$\widehat{\mathbf{c}} = [\mathbf{I}_m|\widehat{\mathbf{S}}_1|\cdots|\widehat{\mathbf{S}}_d]^\top(\mathbf{A}^\top\mathbf{s} + \mathbf{e}_{\mathsf{in}}),$$
$$\widehat{\mathbf{c}}_{\mathsf{out}} = \mathbf{U}^\top\mathbf{s} + \widehat{\mathbf{e}}_{\mathsf{out}} + \mu_b\lceil\frac{q}{2}\rceil \in \mathbb{Z}_q,$$

  which are computed as in Game 2.

- If $\mathbf{c}$ belongs to **Case 1**; i.e., $\mathbf{c}_{\mathsf{in}}$, $\mathbf{c}_{\mathsf{out}}$ are random, then $\widehat{\mathbf{c}}$ is random (by Lemma 2.5.1 again). Because, in this case $\widehat{\mathbf{c}}_{\mathsf{out}}$ is also random, then $\widehat{\mathsf{ct}} := (\widehat{\mathbf{c}}, \widehat{\mathbf{c}}_{\mathsf{out}})$ is random in $\mathbb{Z}_q^{(d+2)m}$ which are computed as in Game 3.

Guess. $\mathcal{A}$ guesses which game between Game 2 and Game 3 that he has interacted with. At this point, $\mathcal{B}$ decide the DLWE instance.

Of course, the difference of Game 2 and Game 3 depends only on which case $\mathbf{c}$ belongs to. Thus, the advantage of $\mathcal{A}$ and $\mathcal{B}$ are the same. □

From Lemmas 5.4.1–5.4.3, we have

$$\begin{aligned}\left|\Pr[\mathbf{W}_0] - \frac{1}{2}\right| &= |\Pr[\mathbf{W}_0] - \Pr[\mathbf{W}_3]| + \left|\Pr[\mathbf{W}_3] - \frac{1}{2}\right| \\ &\le |\Pr[\mathbf{W}_1] - \Pr[\mathbf{W}_0]| + |\Pr[\mathbf{W}_2] - \Pr[\mathbf{W}_1]| \\ &+ |\Pr[\mathbf{W}_3] - \Pr[\mathbf{W}_2]| + \left|\Pr[\mathbf{W}_3] - \frac{1}{2}\right| \\ &\le \mathsf{negl}(\lambda).\end{aligned}$$

This completes the proof. □

Remark 5.4.1. *We can prove the proposed DFKHE to be fully secure through the following reduction: Suppose that there is an adversary $\mathcal{A}$ that can break the fully security of the proposed DFKHE system with some advantage ϵ. Using $\mathcal{A}$, we can build an adversary $\mathcal{B}$ that breaks the selective security.of the system . To do that, $\mathcal{B}$ has to guess the target variable $\widehat{\mathbf{t}} = (\widehat{t_1}, \cdots, \widehat{t_d}) \in \mathbb{Z}_q^d$ to be challenged by $\mathcal{A}$. Obviously, guessing $\widehat{\mathbf{t}} = (\widehat{t_1}, \cdots, \widehat{t_d})$*

incurs a loss of q^d in advantage following the proof for [BB11, Theorem 7.1]. That is, the advantage of $\mathcal{B}$ is only $\epsilon \cdot q^{-d}$.

5.4.4 Setting Parameters

We will heuristically present parameters for our construction:

- Let λ be a security parameter.
- By Lemma 2.4.1, the hardness of $(n, 2m, q, \chi)$–DLWE (in Lemma 5.4.3) is ensured by choosing ϵ, n, q, χ, such that $n = n(\lambda)$, $q = q(n) \leq 2^n$, $m = \Theta(n \log q) = \mathsf{poly}(n)$, and the distribution $\chi = \chi(n)$ is a (B, ν)–bounded for some $B = B(n)$ satisfying that $q/B \geq 2^{n^\epsilon}$. Note that to choose practical parameters, the "core–SVP hardness" methodology (e.g., see in [ABDo20, Section 5.2.1]) is usually used.
- Set Gaussian parameters $\sigma_1, \cdots, \sigma_\eta$ such that TrapGen (Item 1, Lemma 2.5.4, ExtBasisRight (Item 2, Lemma 2.5.4 (called in Game 2), ExtBasisLeft (Item 3, Lemma 2.5.4, SampleD (Item 4, Lemma 2.5.4 and RandBasis (Item 5, Lemma 2.5.4 can work well.
- By Lemma 2.5.11, $\beta_{\mathcal{F}} = (\frac{p^d - 1}{p-1} \cdot m)^\tau$. Remind that τ is the depth of circuits that compute the functions in the family $\mathcal{F}$ and $p < q$ is the upper bound for input values to the multiplication gates of these circuits.
- Parameters have also to fulfil Equation (5.4) for the correctness of the construction.

5.5 Lattice–based PE Construction from DFKHE

We specific a family of functions that will be used to construct puncturable encryption from DFKHE. First, we define a function $eq_{t^*} : \mathbb{Z}_q \to \mathbb{Z}_q$, satisfying that

$$eq_{t^*}(t) = \begin{cases} 0 & t^* \neq t, \\ 1 & otherwise. \end{cases} \tag{5.8}$$

and define

$$f_{t^*}(\mathbf{t}) := eq_{t^*}(t_1) + \cdots + eq_{t^*}(t_d), \quad \text{for } \mathbf{t} = (t_1, \cdots, t_d) \in \mathbb{Z}_q^d.$$

Then,

$$f_{t^*}(\mathbf{t}) = \begin{cases} 0 & t^* \neq t_i, \forall i \in [d], \\ \text{some } z \text{ such that } 0 < z \leq d < q & otherwise. \end{cases} \tag{5.9}$$

The family we will be working on is $\mathcal{F} := \{f_{t^*} : \mathbb{Z}_q^d \to \mathbb{Z}_q \mid t^* \in \mathbb{Z}_q\}$. By applying the generic framework in Section 5.3.1 to the lattice–based DFKHE given in Section 5.4.1 and modifying the resulting PE, we come up with the lattice–based PE construction $\Pi_{\mathsf{PE}} = \{\mathsf{PE.Setup}, \mathsf{PE.Enc}, \mathsf{PE.Pun}, \mathsf{PE.Dec}\}$ presented below:

- $(\mathsf{pk}, \mathsf{sk}_0) \leftarrow \mathsf{PE.Setup}(1^\lambda)$. For the input security parameter λ, do the following:

  1. Choose $n = n(\lambda)$, $q = q(\lambda)$ prime, and the maximum number of tags $d = d(\lambda)$ per a ciphertext such that $d < q$.
  2. Choose $m = \Theta(n \log q)$. The plaintext space is $\mathcal{M} := \{0,1\}^m$, $\mathcal{T} := \mathbb{Z}_q$.
  3. Let χ be a (B, ν)–bounded noise distribution for which the $(n, 2m, q, \chi)$–DLWE is hard. Set $\sigma_1 = \omega(\beta_{\mathcal{F}} \cdot \sqrt{\log m})$.
  4. Sample $(\mathbf{A}, \mathbf{T}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^n, 1^m, q)$, $\mathbf{U}, \mathbf{B}_1, \cdots, \mathbf{B}_d \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$.
  5. Take the gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$.
  6. Output $\mathsf{pk} = \{\mathbf{A}, \mathbf{G}, \mathbf{B}_1, \cdots \mathbf{B}_d, \mathbf{U}\}$ and $\mathsf{sk}_0 := \mathbf{T}_{\mathbf{A}}$.

- $\mathsf{ct} \leftarrow \mathsf{PE.Enc}(\mu, \mathsf{pk}, \{t_1, \cdots, t_d\})$. For the input consisting of (a plaintext μ, the public key pk and ciphertext tags $(t_1, \cdots, t_d) \in \mathcal{T}^d$), perform the following steps:

  1. Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\mathbf{e}_{\mathsf{out}}, \mathbf{e}_{\mathsf{in}} \leftarrow \chi^m$, $\mathbf{S}_1, \cdots, \mathbf{S}_d \xleftarrow{\$} \{-1, 1\}^{m \times m}$.
  2. Compute $\mathbf{e} \leftarrow (\mathbf{I}_m | \mathbf{S}_1 | \cdots | \mathbf{S}_d)^\top \mathbf{e}_{\mathsf{in}} = (\mathbf{e}_{\mathsf{in}}^\top, \mathbf{e}_1^\top, \cdots, \mathbf{e}_d^\top)^\top$.
  3. Form $\mathbf{H} \leftarrow [\mathbf{A} | t_1\mathbf{G} + \mathbf{B}_1 | \cdots | t_d\mathbf{G} + \mathbf{B}_d]$ and compute $\mathbf{c} = \mathbf{H}^\top \mathbf{s} + \mathbf{e} \in \mathbb{Z}_q^{(d+1)m}$, $\mathbf{c} = (\mathbf{c}_{\mathsf{in}}^\top, \mathbf{c}_1^\top, \cdots, \mathbf{c}_d^\top)^\top$, where $\mathbf{c}_{\mathsf{in}} = \mathbf{A}^\top \mathbf{s} + \mathbf{e}_{\mathsf{in}}$ and $\mathbf{c}_i = (t_i\mathbf{G} + \mathbf{B}_i)^\top \mathbf{s} + \mathbf{e}_i$ for $i \in [d]$.
  4. Compute $\mathbf{c}_{\mathsf{out}} \leftarrow \mathbf{U}^\top \mathbf{s} + \mathbf{e}_{\mathsf{out}} + \mu \lceil \frac{q}{2} \rceil$, output $(\mathsf{ct} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_1, \cdots, \mathbf{c}_d, \mathbf{c}_{\mathsf{out}}), (t_1, \cdots, t_d))$.

- $\mathsf{sk}_\eta \leftarrow \mathsf{PE.Pun}(\mathsf{sk}_{\eta-1}, t_\eta^*)$. For the input (a puncture key $\mathsf{sk}_{\eta-1}$ and a punctured tag $t_\eta^* \in \mathcal{T}$), do:

  1. Evaluate $\mathbf{B}_{f_{t_\eta^*}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f_{t_\eta^*}, (\mathbf{B}_k)_{k=1}^d)$.
  2. Compute $\mathbf{E}_{f_{t_\eta^*}} \leftarrow \mathsf{ExtBasisLeft}([\mathbf{A} | \mathbf{B}_{f_{t_1^*}} | \cdots | \mathbf{B}_{f_{t_{\eta-1}^*}} | \mathbf{B}_{f_{t_\eta^*}}], \mathbf{T}_{f_{t_{\eta-1}^*}})$.
  3. Compute $\mathbf{T}_{f_{t_\eta^*}} \leftarrow \mathsf{RandBasis}([\mathbf{A} | \mathbf{B}_{f_{t_1^*}} | \cdots | \mathbf{B}_{f_{t_{\eta-1}^*}} | \mathbf{B}_{f_{t_\eta^*}}], \mathbf{E}_{f_{t_\eta^*}}, \sigma_\eta)$.
  4. Output $\mathsf{sk}_\eta := \mathbf{T}_{f_{t_\eta^*}}$.

- $\mu / \perp := \mathsf{PE.Dec}(\mathsf{ct}, \mathbf{t}, (\mathsf{sk}_\eta, \{t_1^*, \cdots, t_\eta^*\}))$. For the input (a ciphertext $\mathsf{ct} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_1, \cdots, \mathbf{c}_d, \mathbf{c}_{\mathsf{out}})$, the associated tags $\mathbf{t} = (t_1, \cdots, t_d)$, a puncture key $\mathsf{sk}_\eta := \mathbf{T}_{f_{t_\eta^*}}$ and the associated punctured tags $\{t_1^*, \cdots, t_\eta^*\} \subset \mathcal{T}$), execute the following steps:

  1. If there exists $j \in [\eta]$ such that $f_{t_j^*}(\mathbf{t}) \neq 0$, then output $\perp$. Otherwise, go to Step 2.

2. Evaluate $\mathbf{B}_{f_{t_i^*}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(f_{t_i^*}, (\mathbf{B}_k)_{k=1}^d)$ for all $i \in [\eta]$.
3. Sample $\mathbf{R} \leftarrow \mathsf{SampleD}([\mathbf{A}|\mathbf{B}_{f_{t_1^*}}|\cdots|\mathbf{B}_{f_{t_\eta^*}}], \mathbf{T}_{f_{t_\eta^*}}, \mathbf{U}, \sigma_\eta)$.
4. Evaluate $\mathbf{c}_{f_{t_j^*}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}(f_{t_j^*}, (t_k, \mathbf{B}_k, \mathbf{c}_k)_{k=1}^d)$, for $j \in [\eta]$.
5. Compute $\bar{\mu} = (\bar{\mu}_1, \cdots, \bar{\mu}_m) \leftarrow \mathbf{c}_{\mathsf{out}} - \mathbf{R}^\top(\mathbf{c}_{\mathsf{in}}|\mathbf{c}_{f_{t_1^*}}|\cdots|\mathbf{c}_{f_{t_\eta^*}})$.
6. For $\ell \in [m]$, if $|\bar{\mu}_\ell| < q/4$ then output $\mu_\ell = 0$; otherwise, output $\mu_\ell = 1$.

Remark that the analysis has been done for the lattice–based DFKHE in Section 5.4.3 can be applied well to the lattice–based PE. For completeness, we state two main theorems for the lattice–based PE without proof as follows.

Theorem 5.5.1 (Correctness of Π_{PE})**.** *The proposed* Π_{PE} *is correct if Equation* (5.4) *holds assuming that* $t_j^* \neq t_k$ *for all* $(j,k) \in [\eta] \times [d]$.

Theorem 5.5.2 (Security of Π_{PE})**.** *The proposed PE* Π_{PE} *scheme is IND–sPUN–CPA secure thanks to the IND–sVAR–CPA of the underlying* Π_{DFKHE}.

5.6 Summary

This chapter revisited the notion of puncturable encryption (PE). We begin with the concept of delegatable fully key–homomorphic encryption (DFKHE). We design a DFKHE scheme from DLWE using some popular lattice tools. The construction enjoys selective indistinguishability against chosen plaintext attacks (IND–sVAR–CPA) security. We showed that one can construct puncturable encryption from the new primitive through a generic framework. We also developed a puncturable encryption construction from the generic framework offering the same security level as the underlying DFKHE.

For further research, we list here some works that are worthwhile to pursue: (1) designing puncturable ABE as in [PNXW18], but its security based on lattices, (2) constructing efficient puncturable forward–secure encryption schemes as proposed in[GM15], (3) investigating to build PE schemes offering constant puncture key size, and (4) developing PE schemes supporting an unlimited number of punctures.

In Chapter 6, we will concentrate on spatial encryption, which subsumes PE and others as its subclasses.

Chapter 6

Spatial Encryption over Lattices and More

Part of the content in this chapter appeared in Le et al. [LDSP22b]. The author of this thesis is the first and the corresponding author of [LDSP22b]. He contributed to finding the topic, the design of the cryptosystems, the security analysis of the results and the writing of the manuscript.

6.1 Overview

Spatial encryption (SE) is a sort of public–key encryption introduced by Boneh and Hamburg [BH08] at Asiacrypt 2008. In an SE system, encryption and decryption involve affine and vector objects such as affine points, vectors, affine subspaces and vector subspaces. For the formal definition of SE, please refer to Section 2.7.6. One has found several versatile applications of SE in constructing, e.g., (hierarchical) identity–based encryption ((H)IBE), broadcast (H)IBE, attribute–based encryption (ABE) and forward–secure cryptosystems (see, e.g., [BH08], [Ham11]).

In the literature, a generic framework for constructing SE is shown by Chen *et al.* [CLLW14]. The framework transforms hierarchical inner product encryption (HIPE) (Okamoto and Takashima [OT09] at Asiacrypt 2009), which is a variant of inner product encryption (IPE), into an SE. Applying the framework, one can achieve an SE system over lattices thanks to some existing lattice–based HIPE proposals. However, such lattice–based SE systems suffer from shortcomings, as shown in Section 1.2.1.

This chapter and [LDSP22b] revisit SE over lattices, aiming for a better construction than previous ones. We start with introducing a new primitive called *delegatable multiple inner product encryption* (DMIPE). Even though this primitive is also a delegatable variant of inner product encryption (IPE), HIPE and DMIPE are not the same.

In particular, a DMIPE ciphertext is produced together with an *attribute vector*, while a DMIPE decryption key can be generated from the master secret key. Alternatively, a DMIPE decryption key can also be obtained from other secret keys by delegating more *predicate vectors*. We show the equivalence of DMIPE and SE by presenting security–preserving conversions between them. As a proof of concept, we instantiate a DMIPE system whose security relies on the hardness of the decision learning with errors (DLWE) problem. Moreover, it enjoys the selective payload–hiding security in the standard model (SDM). At a high–level description, our lattice–based DMIPE construction utilises the lattice trapdoors and the lattice homomorphic evaluation on a family of inner product functions. Using the DLWE–based DMIPE design, we can achieve an SE system over lattices which is more compact in terms of sizes than those over lattices converted from HIPE (e.g., [ADCM12, Xag15]). We see that our (d–dimensional) lattice SE construction (from the lattice DMIPE) is more efficient, in terms of sizes, than SE obtained from $\Delta(d)$–HIPEs [ADCM12, Xag15] using the Chen *et al.* framework [CLLW14].

As a side product, we formally define allow–/deny–list encryption (ADE), which subsumes many advanced primitives, e.g., PE [GM15]. Furthermore, we show that we can implement some variants of ADE through SE.

To summarise, our main contributions in this chapter includes:

- a primitive, named delegatable multiple inner product encryption (DMIPE), presented in Section 6.2,
- an security–preserving equivalence of DMIPE and SE, demonstrated in Section 6.3 and Section 6.5;
- A construction of DMIPE (and hence SE) over lattices, given in Section 6.4,
- a formal definition and security notions for the allow–/deny–list encryption (ADE), included in Section 6.6; and
- a transformation from some ADE variants to SE, given in Section 6.6.2.

The primary technical tools used in this chapter are the dual Regev encryption framework (Section 2.7.3), the leftover hash lemma (Section 2.5.1), the lattice trapdoor (Section 2.5.3) (that will be merged in a general framework), the lattice homomorphic evaluations (Section 2.5.4) and some tools in Algebra (Section 6.3.1).

In the following, we give more details about the contributions and techniques/tools that will be presented later in this chapter.

DMIPE. Formally, DMIPE is defined by main algorithms DMIPE.Setup, DMIPE.Derive, DMIPE.Del, DMIPE.Enc, and DMIPE.Dec. Given a vector space $\mathcal{D}$ that supports the inner product operation (e.g., $\mathcal{D} = \mathbb{Z}_q^d$ for q prime). For a security parameter λ and a setup

parameter sp, the setup algorithm $\mathsf{DMIPE.Setup}(\lambda, \mathsf{sp})$ generates public parameters pp and a master secret key msk. The key generation algorithm $\mathsf{DMIPE.Derive}(\mathsf{pp}, \mathsf{msk}, \vec{V})$ takes public parameters pp, a master secret key msk and a list $\vec{V} \subset \mathcal{D}$ of vectors. It outputs a secret key $\mathsf{sk}_{\vec{V}}$ for $\vec{V}$. For public parameters pp, a secret key $\mathsf{sk}_{\vec{V}}$ for $\vec{V}$ and a predicate vector $\mathbf{v} \in \mathcal{D}$, the delegation algorithm $\mathsf{DMIPE.Del}(\mathsf{pp}, \mathsf{sk}_{\vec{V}}, \mathbf{v})$ returns a secret key for $\mathsf{sk}_{\vec{V}'}$, where $\vec{V}' = \vec{V} \cup \{\mathbf{v}\}$. The encryption algorithm $\mathsf{DMIPE.Enc}(\mathsf{pp}, \mu, \mathbf{x})$ outputs a ciphertext $\mathsf{ct}_{\mathbf{x}}$ for public parameters pp, a plaintext μ and an attribute vector $\mathbf{x} \in \mathcal{D}$. The decryption algorithm $\mathsf{DMIPE.Enc}(\mathsf{pp}, \mathsf{sk}_{\vec{V}}, \mathsf{ct}_{\mathbf{x}})$ either recovers the plaintext μ if $\langle \mathbf{x}, \mathbf{v} \rangle = 0$ for all $\mathbf{v} \in \vec{V}$ or returns $\perp$.

There is an important requirement for predicate vectors, say $\vec{V}$. They have to be linearly independent, i.e., no vector is a linear combination of two or more other vectors from $\vec{V}$. The requirement is necessary to ensure that there is no redundant vector in $\vec{V}$ when checking decryption conditions. Besides, the delegation of a decryption key for $\vec{V} \cup \mathbf{v}$ is possible if $\mathbf{v}$ is linearly independent of the existing predicate vectors in $\vec{V}$. Further details can be found in Section 6.2.

Let us illustrate a simple application of DMIPE. Consider a company where each officer/worker has a secret key allowing him/her to access internal documents. Assume that access is restricted depending on his/her role/department in the company. To this end, each document is encrypted with an attribute vector, and each person in the company is issued with a predicate vector. A private key for each person corresponds to a list of predicate vectors. Furthermore, a manager can use her/his key to generate a key for subordinates using delegation.

DMIPE is a generalisation of IPE that is more natural than HIPE. Compared to HIPE, the decryption hierarchy in DMIPE is more flexible for delegation.

Figure 6.1 below depicts the relation of SE, DMIPE, HIPE, and other primitives, such as HIBE, ABE, puncturable encryption (PE) [GM15], puncturable IBE (PIBE) [DSDR21], Dual–form PE (DFPE) [DRSS21] and several forward–secure cryptosystems (we generally write as "Forward Security" in Figure 6.1). Here, the arrow "$\mathsf{A} \rightarrow \mathsf{B}$" means that "we can construct B from A".

Details of DMIPE will be presented in Section 6.2.

Lattice–based DMIPE. At a high–level description, our lattice–based DMIPE design exploits the dual Regev encryption framework (Section 2.7.3), the lattice trapdoor mechanism and the lattice evaluation for inner product functions (see Lemma 6.4.1 and Lemma 6.4.3 for formal statements). The DMIPE design's security is based on the intractability of the DLWE problem.

Now, we sketch the lattice–based DMIPE construction. For appropriately chosen parameters n, m, d, q, σ_0, a public key (included in the public parameters pp) consists of matrices $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n \times dm}$, $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$ and $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, which is generated together with a σ_0–

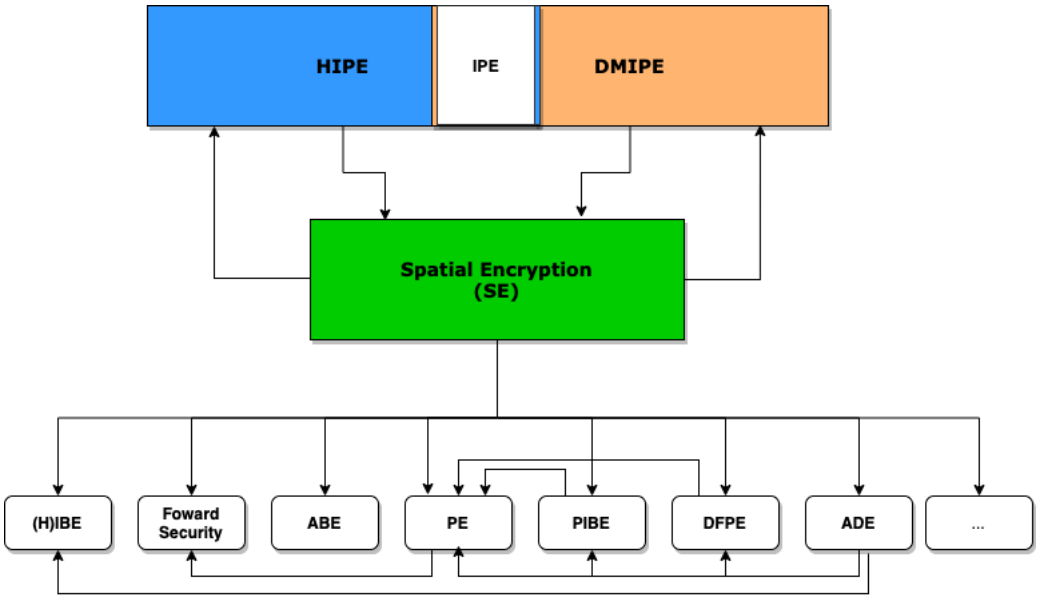

Figure 6.1: The relation of SE and other primitives.

Table 6.1: Comparison of our lattice–based d–dimensional SE with other SEs based on lattices.

d-dim. SE from	pk-size ($\ell := \lceil \log_r q \rceil$)	msk-size ($\ell := \lceil \log_r q \rceil$)	sk-size (k predicate vectors)	ct-size (h attribute vectors, m-bit plaintext)
Abdalla *et al.* [ADCM12] ($\Delta(d)$-HIPE)	$(d^2(\ell+1)\cdot\mathbb{Z}_q^{n\times m}$ $+2\cdot\mathbb{Z}_q^{n\times m}$	$1\cdot D_{\mathbb{Z}}^{m\times m}$	$1\cdot D_{\mathbb{Z}}^{km\times m}$	$(hd(\ell+1))\cdot\mathbb{Z}_q^{m}$ $+2\cdot\mathbb{Z}_q^{m}$
Xagawa [Xag15] ($\Delta(d)$-HIPE)	$(d^2+d)\cdot\mathbb{Z}_q^{n\times n\ell}$ $+2\cdot\mathbb{Z}_q^{n\times ml}$	$1\cdot D_{\mathbb{Z}}^{(m-n\ell)\times n\ell}$	$1\cdot D_{\mathbb{Z}}^{(m+(2k-1)n\ell)\times m}$ $+1\cdot D_{\mathbb{Z}}^{(m+(2k-1)n\ell)\times nk}$	$(h-1+hd)\cdot\mathbb{Z}_q^{n\ell}$ $+2\cdot\mathbb{Z}_q^{m}$
Ours (DMIPE)	$(d+2)\cdot\mathbb{Z}_q^{n\times m}$	$1\cdot D_{\mathbb{Z}}^{m\times m}$	$1\cdot D_{\mathbb{Z}}^{km\times m}$	$(d+2)\cdot\mathbb{Z}_q^{m}$

trapdoor $\mathbf{A}_{\sigma_0}^{-1}$. The trapdoor acts as the master secret key. To generate a key for a list $\vec{V}$ of k predicate vectors, we compute $\mathbf{B}_{\mathbf{v}_i} := \mathbf{B}\mathbf{H}_{\mathbf{v}_i}$ for each $\mathbf{v}_i \in \vec{V}$. Here, we use the public evaluation $\mathbf{H}_{\mathbf{v}_i} \leftarrow \mathsf{EvalF}^{\mathsf{IP}}(f_{\mathbf{v}_i}, \mathbf{B})$ from Lemma 6.4.3. The secret key $\mathsf{sk}_{\vec{V}}$ is the σ_0-trapdoor $\mathbf{A}_{\vec{V},\sigma_0}^{-1}$ for $\mathbf{A}_{\vec{V}}$ computed using the master secret key $\mathbf{A}_{\sigma_0}^{-1}$ via Lemma 6.4.1, where $\mathbf{A}_{\vec{V}} := [\mathbf{A}|\mathbf{B}_{\mathbf{v}_1}|\cdots|\mathbf{B}_{\mathbf{v}_k}]$. Regarding delegation, given a secret key $\mathsf{sk}_{\vec{V}} := \mathbf{A}_{\vec{V},\sigma_0}^{-1}$, one can update it to get a secret key for $\vec{V}' := \vec{V} \cup \{\mathbf{v}_{k+1}\}$ for any $\mathbf{v}_{k+1}$. This can be done by generating a trapdoor $\mathbf{A}_{\vec{V}',\sigma_0}^{-1}$ for $\mathbf{A}_{\vec{V}'} = [\mathbf{A}_{\vec{V}}|\mathbf{B}_{\mathbf{v}_{k+1}}]$ from $\mathbf{A}_{\vec{V},\sigma_0}^{-1}$. Again, $\mathbf{B}_{\mathbf{v}_{k+1}}$ is computed by evaluating $(f_{\mathbf{v}_{k+1}}, \mathbf{B})$ using $\mathsf{EvalF}^{\mathsf{IP}}$.

Encryption on an attribute vector $\mathbf{x}$ employs the dual Regev's style to produce a ciphertext $\mathsf{ct}_{\mathbf{x}} := (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_{\mathsf{mid}}, \mathbf{c}_{\mathsf{out}})$, in which $\mathbf{c}_{\mathsf{mid}} := \mathbf{s}^\top(\mathbf{B} - \mathbf{x}\otimes\mathbf{G}) + \mathbf{e}_{\mathsf{in}}^\top\mathbf{R} \in \mathbb{Z}_q^{md}$. Here $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$

is an LWE secret, $\mathbf{e}_{\mathsf{in}} \leftarrow \chi^m$ is an LWE error, and $\mathbf{R} \xleftarrow{\$} \{-1,0,1\}^{m\times md}$. For decrypting a ciphertext $\mathsf{ct}_{\mathbf{x}}$ using a secret key $\mathsf{sk}_{\vec{V}}$, we again exploit Lemma 6.4.3 to compute $\mathbf{c}_{\mathbf{v}_i} := \mathbf{c}_{\mathsf{mid}}\mathbf{H}_{\mathbf{v}_i}$ for all $\mathbf{v}_i \in \vec{V}$. Putting $\mathbf{c}_{\mathbf{v}_i}$'s, $\mathbf{c}_{\mathsf{in}}$, and $\mathbf{c}_{\mathsf{out}}$ together allows us to recover the underlying plaintext if $\langle \mathbf{x}, \mathbf{v}_i \rangle = 0 \pmod q$ for all $\mathbf{v}_i \in \vec{V}$. Otherwise, decryption fails.

The lattice–based DMIPE will be given in Section 6.4.

Equivalence of DMIPE and SE. We prove in Section 6.3 and Section 6.5 that DMIPE and SE are equivalent in the sense that we can establish *security–preserving* conversions between them. In particular, we can use DMIPE to construct SE, where SE inherits security from DMIPE and vice versa. It also means we can get a lattice–based SE from a lattice–based DMIPE. This way, our (d–dimensional) SE construction is more efficient in terms of sizes, than SE obtained from $\Delta(d)$–HIPEs [ADCM12, Xag15], according to the generic framework of Chen et al. [CLLW14]. Table 6.1 compares our SE construction with two other lattice–based SEs. All are d–dimensional ones.

ADE and the Construction from SE. ADE is, in fact, also a subclass of PrE, in which both predicates and attributes are *tags*. These tags are categorised into two lists: *allow list* contains positive tags and *deny list* – negative tags. Both ciphertexts and decryption keys are associated with these two kinds of tags. Further, ADE also supports the delegation mechanism called *puncturing*. Roughly saying, *negatively puncturing* is the delegation on negative tags, and this puncturing can revoke the decryption ability. In contrast, *positively puncturing* is delegation done on positive tags and allows decryption.

We will present a formal definition and security model for ADE. Moreover, we consider three versions of ADE: (i) standard ADE (sADE); (ii) inclusive ADE (iADE) and (iii) k–threshold ADE (k–tADE). We show that one can construct sADE and iADE from SE by applying appropriate encodings. This result will be detailed in Section 6.6. However, translating k–tADE to SE is an open problem.

The security notions for SE, DMIPE and even ADE inherit from those for PrE, which were introduced in [KSW08]. They include selective payload-hiding, selective attribute–hiding, adaptive payload–hiding, and adaptive attribute–hiding. We stress that, in this work, we concentrate on the selective payload–hiding security.

Figure 6.2 shows the roadmap of this chapter.

6.2 Delegatable Multiple Inner Product Encryption

This section introduces a new primitive called delegatable multiple inner product encryption (DMIPE). We do that by presenting DMIPE's syntax and security notions. A DMIPE ciphertext is produced together with a d–dimensional *ciphertext vector*, or *attribute vector*. Notice that a DMIPE decryption key associated with a list of d–dimensional *key vec-*

Summary [Section 6.7]
↑
Allow–/Deny–list Encryption from Spatial Encryption [Section 6.6]
↑
Constructing DMIPE from SE [Section 6.5]
↑
DMIPE Construction over Lattices [Section 6.4]
↑
Generic SE from DMIPE [Section 6.3]
↑
Delegatable Multiple Inner Product Encryption [Section 6.2]
↑
Overview [Section 6.1]

Figure 6.2: The roadmap of this chapter.

tors or *predicate vectors*. We assume that all vectors reside in the same space $\mathcal{D}$, which supports the inner product defined by $\langle \mathbf{a}, \mathbf{b} \rangle = a_1 b_1 + \cdots + a_d b_d \in \mathcal{D}$ where $\mathbf{a} := (a_1, \cdots, a_d)$, $\mathbf{b} := (b_1, \cdots, b_d) \in \mathcal{D}^d$.

We can see that DMIPE is a generalisation of IPE that is more natural than HIPE. DMIPE's decryption hierarchy is also more flexible for delegation than HIPE. Remind that Table 1.5 gives a syntax comparison between IPE, HIPE and DMIPE, while Figure 6.1 intuitively illustrates their relation with several other primitives.

6.2.1 Syntax

A DMIPE is a tuple of algorithms DMIPE.Setup, DMIPE.Derive, DMIPE.Del, DMIPE.Enc and DMIPE.Dec, which are formally defined as follows:

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{DMIPE.Setup}(1^\lambda, \mathsf{sp})$. DMIPE.Setup is the key setup algorithm. It is PPT. Its inputs are a security parameter λ and setup parameters sp. Its outputs are a public key pk and a master secret key msk.

- $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{msk}, \vec{V})$. DMIPE.Derive is the key derivation algorithm. It is PPT. Its inputs are a public key pk, a master secret key msk and a list of vectors $\vec{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$. Its output is a secret key $\mathsf{sk}_{\vec{V}}$ for $\vec{V}$.

- $\perp / \mathsf{sk}_{\vec{V}_2} \leftarrow \mathsf{DMIPE.Del}(\mathsf{pk}, \mathsf{sk}_{\vec{V}_1}, \mathbf{v}_{k+1})$. DMIPE.Del is key delegation algorithm. It is PPT. Its inputs are a public key pk, a secret key $\mathsf{sk}_{\vec{V}_1}$ for $\vec{V}_1 = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ and a vector $\mathbf{v}_{k+1}$. Its output is either a secret key $\mathsf{sk}_{\vec{V}_2}$ for $\vec{V}_2 := \vec{V}_1 \cup \{\mathbf{v}_{k+1}\}$ ($\mathbf{v}_{k+1}$ is linearly independent of $\vec{V}$) or $\perp$ (if $\mathbf{v}_{k+1}$ is not linearly independent of $\vec{V}$).

- $\mathsf{ct}_{\mathbf{x}} \leftarrow \mathsf{DMIPE.Enc}(\mathsf{pk}, \mu, \mathbf{x})$. DMIPE.Enc is the encryption algorithm. It is PPT. Its inputs are a public key, a plaintext μ and a vector $\mathbf{x}$. Its output is a ciphertext $\mathsf{ct}_{\mathbf{x}}$.

- $\perp/\mu := \mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{sk}_{\vec{V}}, \mathsf{ct}_{\mathbf{x}})$. $\mathsf{DMIPE.Dec}$ is the decryption algorithm. It is DPT. Its inputs are a ciphertext $\mathsf{ct}_{\mathbf{x}}$, a secret key $\mathsf{sk}_{\vec{V}}$. Its output is either a plaintext μ (if it succeeds) or $\perp$ (otherwise). Note that $\mathsf{DMIPE.Dec}$ is successful if $\vec{V} \cdot \mathbf{x} = \mathbf{0}$ (i.e., $\langle \mathbf{v}_i, \mathbf{x} \rangle = 0$ for all $\mathbf{v}_i \in \vec{V}$).

Here we require predicate vectors $\vec{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ to be linearly independent. The restriction is to make sure that no redundant vectors are in $\vec{V}$ making the decryption conditions well–defined. Of course, a decryption key for $\vec{V} \cup \mathbf{v}$ could be produced from the decryption for $\vec{V}$ if $\vec{V} \cup \mathbf{v}$ is also a set of linearly independent vectors.

6.2.2 Correctness

The correctness of DMIPE requires that: for any security parameters λ, any system parameters sp, any $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{DMIPE.Setup}(1^\lambda, \mathsf{sp})$, $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Del}(\mathsf{pk}, \mathsf{sk}_{\vec{V}'}, \mathbf{v})$ (where $\vec{V} = \vec{V}' \cup \{\mathbf{v}\}$) or $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{msk}, \vec{V})$,

- if $\langle \mathbf{v}, \mathbf{x} \rangle = \mathbf{0}$ for all $\mathbf{v} \in \vec{V}$ then

$$\Pr[\mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{sk}_{\vec{V}}, \mathsf{DMIPE.Del}(\mathsf{pk}, \mathsf{sk}_{\vec{V}'}, \mathbf{v})) = \mu] \geq 1 - \mathsf{negl}(\lambda),$$

- otherwise,

$$\Pr[\mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{sk}_{\vec{V}}, \mathsf{DMIPE.Del}(\mathsf{pk}, \mathsf{sk}_{\vec{V}'}, \mathbf{v})) = \mu] < \mathsf{negl}(\lambda),$$

over the randomness of the involved algorithms.

6.2.3 Security Notions

Similarly to SE, we can define the adaptive/selective payload–/attribute–hiding security for DMIPE. However, we only state the selective payload–hiding security. We present the security notion in Definition 6.2.1 and Figure 6.3. Note that in Figure 6.3, by "$\mathbf{x} \not\perp \vec{V}$", we mean "$\exists \mathbf{v} \in \vec{V} : \mathbf{x} \not\perp \mathbf{v}$".

Definition 6.2.1 (PAY–sATT–ATK security for DMIPE)**.** *Define the advantage of the adversary $\mathcal{A}$ in the game* $\mathsf{DMIPE}^{\mathsf{PAY–sATT–ATK}}_{\mathcal{A}}(\lambda, \mathsf{sp})$ *as*

$$\mathsf{Adv}^{\mathsf{PAY–sATT–ATK}}_{\mathcal{A},\ \mathsf{DMIPE}}(\lambda) := \left| \Pr[\mathsf{DMIPE}^{\mathsf{PAY–sATT–ATK}}_{\mathcal{A}}(\lambda, \mathsf{sp}) \Rightarrow 1] - \frac{1}{2} \right|.$$

We say that a DMIPE is PAY–sATT–ATK secure if, for any polynomial–time adversary $\mathcal{A}$, it holds that

$$\mathsf{Adv}^{\mathsf{PAY–sATT–ATK}}_{\mathcal{A},\ \mathsf{DMIPE}}(\lambda) \leq \mathsf{negl}(\lambda).$$

GAME $\mathsf{DMIPE}_{\mathcal{A}}^{\mathsf{PAY\text{-}sATT\text{-}ATK}}(\lambda, \mathsf{sp})$:

(where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$)

1. $\mathbf{x}^* \leftarrow \mathcal{A}(1^\lambda, \mathsf{sp})$;
2. $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{DMIPE.Setup}(1^\lambda, \mathsf{sp})$;
3. $(\mu_0^*, \mu_1^*) \leftarrow \mathcal{A}^{\mathsf{KQ}(\cdot),\ \mathsf{DQ}_1(\cdot,\cdot)}(\mathsf{pk})$;
4. $b \xleftarrow{\$} \{0,1\}$, $\mathsf{ct}^*_{\mathbf{x}^*} \leftarrow \mathsf{DMIPE.Enc}(\mathsf{pk}, \mathbf{x}^*, \mu_b^*)$;
5. $b' \leftarrow \mathcal{A}^{\mathsf{ct}^*_{\mathbf{x}^*},\ \mathsf{KQ}(\cdot),\ \mathsf{DQ}_2(\cdot,\cdot)}(\mathsf{pk})$; // **NOTE:** $\mathsf{DQ}_2(\vec{V}, \mathsf{ct}^*_{\mathbf{x}^*})$ with $\mathbf{x}^* \perp \vec{V}$ is not allowed.
6. If $b' = b$, return 1. Otherwise, return 0.

Queried Oracles:

- Key Oracle $\mathsf{KQ}(\vec{V})$ (allowed only if $\mathbf{x}^* \not\perp \vec{V}$): Run $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{msk}, \vec{V})$.
- Decryption Oracle $\mathsf{DQ}_1(\vec{V}, \mathsf{ct}_{\mathbf{x}})$ (allowed only if $\mathsf{ATK} \in \{\mathsf{CCA1}, \mathsf{CCA2}\}$): Run $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{msk}, \vec{V})$. Return the output of $\mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_{\vec{V}})$.
- Decryption Oracle $\mathsf{DQ}_2(\vec{V}, \mathsf{ct}_{\mathbf{x}})$ (allowed only if $\mathsf{ATK} = \mathsf{CCA2}$): Run $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{msk}, \vec{V})$. Return the output of $\mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_{\vec{V}})$.

Figure 6.3: Security game for DMIPE.

6.3 Generic SE Construction from DMIPE

First, we review some necessary concepts in affine geometry and linear algebra.

6.3.1 Selected Facts

Let $\mathbb{F}$ denote a field. A d–dimensional vector subspace $V \sqsubseteq \mathbb{F}^d$ having a basis $\mathbf{M} \in \mathbb{F}^{d \times m}$ can be represented as

$$V := \mathsf{span}(\mathbf{M}) = \{\mathbf{M}\mathbf{x} : \mathbf{x} \in \mathbb{F}^m\}.$$

As $\mathbf{M}$ is a basis for V then all rows of $\mathbf{M}$ are linearly independent. Whereas, a d–dimensional affine subspace W of $\mathbb{F}^d$ is defined as

$$W = \mathbf{y} + \mathsf{span}(\mathbf{M}) = \{\mathbf{y} + \mathbf{M}\mathbf{x} : \mathbf{x} \in \mathbb{F}^m\}, \tag{6.1}$$

for some $\mathbf{y} \in \mathbb{F}^d, \mathbf{M} \in \mathbb{F}^{d \times m}$.

Suppose that W is a d–dimensional affine subspace represented as in Equation (6.1). Then one can change W into a vector subspace defined as follows:

$$W = \mathsf{span}(\mathbf{M}') := \left\{ \mathbf{M}'\mathbf{x}' : \mathbf{x}' = \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix}, \mathbf{x} \in \mathbb{F}^{m+1} \right\}, \tag{6.2}$$

where $\mathbf{M}' := \begin{bmatrix} 1 & 0 \\ \mathbf{y} & \mathbf{M} \end{bmatrix} \in \mathbb{F}^{(d+1)\times(m+1)}$. Of course, the linear independence of $\mathbf{M}'$'s rows is guaranteed by that of $\mathbf{M}$'s rows. Equation (6.2) says that W is now a $(d+1)$–dimensional

vector subspace. We have shown that a d–dimensional affine SE can always be embedded into a $(d+1)$–dimensional linear SE. Therefore, from here on, by "SE" we always mean "linear SE" associated with vector subspaces and vectors over some field $\mathbb{F}$. For example, $\mathbb{F}$ can be $\mathbb{Z}_q$ for prime q.

Remark that for linear SE, we create a ciphertext using a vector $\mathbf{x}$, while we produce a decryption key with a vector space V. If the relation $\mathbf{x} \in V$ holds then decryption is successful. For the generic construction of SE from DMIPE, we have to map the "belonging to, $\in$" relation to the "orthogonal to, $\perp$" relation.

The following lemma allows deriving a basis for the given space's orthogonal complement.

Lemma 6.3.1 ([Coh96, Algorithm 2.3.7] and [CLLW14])**.** *There is an efficient algorithm, named* OCB *taking as input a vector space V to output a basis, denoted $\mathcal{B}^{\perp}(V)$, for the V's orthogonal complement $V^{\perp}$. In addition, the algorithm ensures that if $V_2 \sqsubseteq V_1$ then $\mathcal{B}^{\perp}(V_1) \subseteq \mathcal{B}^{\perp}(V_2)$.*

6.3.2 The Generic Construction

Let $\Pi_{\mathsf{DMIPE}} := (\mathsf{DMIPE.Setup}, \mathsf{DMIPE.Derive}, \mathsf{DMIPE.Del}, \mathsf{DMIPE.Enc}, \mathsf{DMIPE.Dec})$ be a DMIPE system. Using Π_{DMIPE}, an SE system, say $\Pi_{\mathsf{SE}} := (\mathsf{SE.Setup}, \mathsf{SE.Derive}, \mathsf{SE.Del}, \mathsf{SE.Enc}, \mathsf{SE.Dec})$ can be built up as below.

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{SE.Setup}(1^{\lambda}, \mathsf{sp})$. For input a security parameter λ, a system parameters sp, run $(\mathsf{dmipe.pp}, \mathsf{dmipe.msk}) \leftarrow \mathsf{DMIPE.Setup}(1^{\lambda}, \mathsf{sp})$ and set $\mathsf{pk} := \mathsf{dmipe.pp}$, and $\mathsf{msk} := \mathsf{dmipe.msk}$.

- $\mathsf{sk}_V \leftarrow \mathsf{SE.Derive}(\mathsf{pk}, \mathsf{msk}, V)$. For input a public key pk, the master secret key msk and a subspace V, perform:

  1. Run $\mathcal{B}^{\perp}(V) \leftarrow \mathsf{OCB}(V)$, and set $\vec{V} := \{\mathbf{v} : \mathbf{v} \in \mathcal{B}^{\perp}(V)\}$.
  2. Run $\mathsf{dmipe.sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{msk}, \vec{V})$, and set $\mathsf{sk}_V := \mathsf{dmipe.sk}_{\vec{V}}$.

- $\mathsf{sk}_{V_2} \leftarrow \mathsf{SE.Del}(\mathsf{pk}, \mathsf{sk}_{V_1}, V_2)$. For input a public key pk, secret key for subspace $\mathsf{sk}_{V_1} = \mathsf{dmipe.sk}_{\vec{V}}$ for V_1 and a subspace $V_2 \sqsubseteq V_1$, perform:

  1. Run $\mathcal{B}^{\perp}(V_1) \leftarrow \mathsf{OCB}(V_1)$, $\mathcal{B}^{\perp}(V_2) \leftarrow \mathsf{OCB}(V_2)$, and set $\vec{V}_1 := \{\mathbf{v} : \mathbf{v} \in \mathcal{B}^{\perp}(V_1)\}$, $\vec{V}_2 := \{\mathbf{v} : \mathbf{v} \in \mathcal{B}^{\perp}(V_2)\}$. Note that since $V_2 \sqsubseteq V_1$, $\vec{V}_1 \subseteq \vec{V}_2$.
  2. Suppose that $\vec{V}_2 \setminus \vec{V}_1 = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ for some $k \geq 1$. Set $\vec{V} \leftarrow \vec{V}_1$. For $i \in [k]$, run $\mathsf{dmipe.sk}_{\vec{V} \cup \{\mathbf{v}_i\}} \leftarrow \mathsf{DMIPE.Del}(\mathsf{pk}, \mathsf{dmipe.sk}_{\vec{V}}, \mathbf{v}_i)$, then set $\vec{V} \leftarrow \vec{V} \cup \{\mathbf{v}_i\}$.
  3. At this point, we reach $\vec{V} = \vec{V}_2$. Finally, output $\mathsf{sk}_{V_2} := \mathsf{dmipe.sk}_{\vec{V}_2}$.

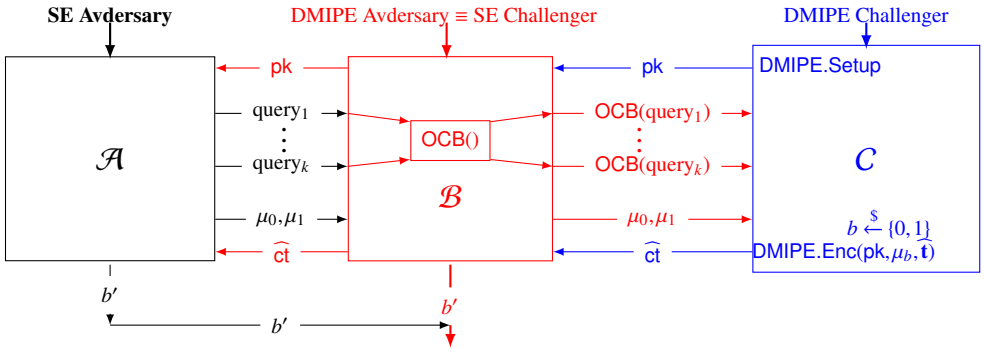

Figure 6.4: Reduction from DMIPE to SE.

Doing this makes it clear that the distribution of the private keys is independent of the path taken. Namely, the distribution for the key sk_{V_3} computed from sk_{V_2} is the same as that of sk_{V_3} computed from sk_{V_1} with $V_3 \sqsubseteq V_2 \sqsubseteq V_1$.

- $\mathsf{ct}_{\mathbf{x}} \leftarrow \mathsf{SE.Enc}(\mathsf{pk}, \mathbf{x}, \mu)$. For input the a public key pk, an attribute vector $\mathbf{x}$ and a plaintext μ, run $\mathsf{dmipe.ct}_{\mathbf{x}} \leftarrow \mathsf{DMIPE.Enc}(\mathsf{pk}, \mathbf{x}, \mu)$ and output a ciphertext $\mathsf{ct}_{\mathbf{x}} := \mathsf{dmipe.ct}_{\mathbf{x}}$.

- $\mu / \perp \leftarrow \mathsf{SE.Dec}(\mathsf{pk}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_V)$. For input the a public key pk, a ciphertext $\mathsf{ct}_{\mathbf{x}}$ and a secret key sk_V for a space V, return $\mu / \perp \leftarrow \mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_V)$.

6.3.3 Correctness

The correctness of SE is stated in Theorem 6.3.1.

Theorem 6.3.1. *The SE scheme* Π_{SE} *is correct provided the correctness of the underlying DMIPE scheme* Π_{DMIPE}.

Proof. The induced scheme Π_{SE} is correct following the equivalence of "$\mathbf{x} \in V$" and "$\mathbf{x} \perp \mathbf{v}$, for all $\mathbf{v} \in \mathcal{B}^{\perp}(V)$". □

6.3.4 Security Analysis

Theorem 6.3.2. *Suppose that* $\mathcal{A}$ *is the adversary that wins some security game (selective/adaptive payload–/attribute–hiding) for* Π_{SE}. *Then, there is an adversary* $\mathcal{B}$ *that wins the same security game for* Π_{DMIPE} *such that* $\mathsf{Adv}^{\mathsf{DMIPE}}_{\mathcal{B}} \geq \mathsf{Adv}^{\mathsf{SE}}_{\mathcal{A}}$.

Proof. We present a reduction in which the DMIPE adversary $\mathcal{B}$ plays as the SE challenger. It plays with $\mathcal{A}$ in the SE security game. The $\mathcal{B}$'s strategy is to simulate the security game environment for Π_{SE}. At the same time, $\mathcal{B}$ also communicates with the DMIPE challenger C. The reduction's main idea is as follows. At first, $\mathcal{B}$ forwards a

public key sent by C to $\mathcal{A}$. After that, any queries having received from $\mathcal{B}$ will be transformed using the algorithm OCB (in Lemma 6.3.1) before being forwarded to C. The challenge ciphertext for the SE security game is also produced by C then sent back to $\mathcal{A}$ by $\mathcal{B}$. Finally, $\mathcal{B}$ returns what $\mathcal{A}$ have returned.

We visualise the reduction in Figure 6.4 and describe it in detail below.

Setup. $\mathcal{B}$ hands $\mathcal{A}$ the a public key pk, which has been received from the DMIPE challenger C.

Query 1. For any $\mathcal{A}$,'s query, $\mathcal{B}$ first transforms using OCB to a query that is compatible with DMIPE, which is then sent to C. $\mathcal{B}$ answers the query of $\mathcal{A}$ by forwarding the C's response.

Challenge. Now $\mathcal{A}$ challenges by submitting two plaintexts and/or attribute vectors. At this point, $\mathcal{B}$ forwards the challenge to C. $\mathcal{B}$ sent back to $\mathcal{A}$ what C has responded.

Query 2. Similar to **Query 1** but is under some restrictions mentioned in security games for DMIPE and SE.

Output. $\mathcal{B}$ returns as his guess what $\mathcal{A}$ has just returned.

It is easy to see that $\mathcal{B}$ perfectly simulates the SE environment for $\mathcal{A}$. Moreover, what $\mathcal{B}$ received from $\mathcal{A}$ are also perfect in order for $\mathcal{B}$ to play with C in the DMIPE security game. Therefore, $\mathcal{B}$ can succeed in the DMIPE security game with a probability not less than that of $\mathcal{A}$ in the SE game. The proof follows. □

6.4 DMIPE Construction over Lattices

The DMIPE construction over lattices employs the lattice trapdoor reviewed in Section 2.5.3 and the lattice evaluation algorithms in Section 2.5.4. The construction will also involve a family of inner product functions. We will then define the family right now.

Let $\mathbf{v} \in \mathbb{Z}_q^d$ be a vector. We denote as $f_{\mathbf{v}} : \mathbb{Z}_q^d \to \mathbb{Z}_q$ an inner product indicated by $\mathbf{v}$. The function $f_{\mathbf{v}}$ will map any $\mathbf{x} \in \mathbb{Z}_q^d$ to $= \langle \mathbf{v}, \mathbf{x} \rangle \pmod q$. Formally,

$$f_{\mathbf{v}}(\mathbf{x}) := \langle \mathbf{v}, \mathbf{x} \rangle \pmod q.$$

It is well–known that one can represent the function as a circuit having only one addition gate; see [BGG+14, Section 4] for discussion.

For the DMIPE construction, we consider the max–absolute–value norm $\|\cdot\|_{\max}$ instead of Euclidean norm $\|\cdot\|$. We therefore use a modified version of lattice trapdoor which will be demonstrated in Section 6.4.1. Further, the lattice homomorphic evaluations will focus on the inner product functions as detailed in Section 6.4.2.

6.4.1 Modified Lattice Trapdoors

Let $n, m, q \in \mathbb{Z}^+$, $\sigma \in \mathbb{R}^+$ and let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ be a matrix. We have known in Section 2.5.3 that given a matrix $\mathbf{U} \in \mathbb{Z}_q^{n\times m'}$ (including zero matrices) for any positive integer m', there is an algorithm helps sample via discrete Gaussian distributions $(D_{\mathbb{Z}^m,\sigma})^{m'}$ over $\Lambda_q^{\mathbf{U}}(\mathbf{A})$. That is, the algorithm outputs a matrix, say $\mathbf{X} \sim (D_{\mathbb{Z}^m,\sigma})^{m'}$, such that $\mathbf{AX} = \mathbf{U} \pmod q$. In this chapter, such the algorithm is called *a σ–trapdoor* and denoted by $\mathbf{A}_\sigma^{-1}(\cdot)$. Then, we have $\mathbf{X} \leftarrow \mathbf{A}_\sigma^{-1}(\mathbf{U})$.

Slightly abusing, $\mathbf{A}_\sigma^{-1}$ is also called a σ–trapdoor for $\mathbf{A}$. Also, the publicly known constant trapdoor $\mathbf{T_G} \in \mathbb{Z}^{m\times m}$ for the gadget matrix $\mathbf{G}$ is denoted as $\mathbf{G}_{O(1)}^{-1}$ in this chapter. Following [BV16, Tsa19, KNYY20], we collect some selected standard results and algorithms on lattice trapdoors used for our design.

Lemma 6.4.1 ([Ajt96, GPV08, AP09, ABB10, CHKP10, MP12])**.** *The following facts hold for lattice trapdoors:*

1. *Let n, m, q be positive integers where $m = O(n\log q)$. There is an efficient algorithm* $\mathsf{TrapGen}(1^n, 1^m, q)$ *that takes (n, m, q) as input to generate a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ together with its trapdoor $\mathbf{A}_{\sigma_0}^{-1}$ satisfying that the distribution of $\mathbf{A}$ is negligibly far from $\mathcal{U}(\mathbb{Z}_q^{n\times m})$ with $\sigma_0 = \omega(n\log q\log n)$.*

2. *Given a trapdoor $\mathbf{A}_{\sigma_1}^{-1}$, one can compute $\mathbf{A}_{\sigma_2}^{-1}$ for any $\sigma_2 \geq \sigma_1$.*

3. *Given a trapdoor $\mathbf{A}_\sigma^{-1}$, one can compute $[\mathbf{A}|\mathbf{B}]_\sigma^{-1}$, $[\mathbf{B}|\mathbf{A}]_\sigma^{-1}$ for any matrix $\mathbf{B}$ having the same number of rows as $\mathbf{A}$.*

4. *Given the gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m'}$ with $m' \geq n\lceil\log q\rceil$, using its trapdoor $\mathbf{G}_{O(1)}^{-1}$ one can compute the trapdoor $[\mathbf{A}|\mathbf{AR}+\mathbf{G}]_\sigma^{-1}$ for all $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and $\mathbf{R} \in \mathbb{Z}^{m\times m'}$ and $\sigma = m \cdot \|\mathbf{R}\|_{\max} \cdot \omega(\sqrt{\log m})$.*

5. *For a trapdoor $\mathbf{A}_\sigma^{-1}$ and for any $\mathbf{U} \in \mathbb{Z}_q^{n\times m'}$, by Lemma 2.3.7,* $\Pr[\|\mathbf{A}_\sigma^{-1}(\mathbf{U})\|_{\max} \leq 12\sigma : \mathbf{x} \leftarrow D_{\mathbb{Z},\sigma}] \geq 1 - 2^{-100}$.

6.4.2 Lattice Homomorphic Evaluations for Inner Product Functions

In this section, we will modify to restrict the lattice homomorphic evaluation algorithms (Section 2.5.4) to the inner product functions. Moreover, we only concentrate inner product functions defined over $\mathbb{Z}_q$. Notice that [BV16, BTVW17, Tsa19] work with binary functions, i.e., $f : \{0,1\}^* \to \{0,1\}^*$ rather than functions over $\mathbb{Z}_q$ when using the lattice homomorphic evaluations. For example, the following lemma is from [Tsa19, Theorem 2].

Lemma 6.4.2 ([Tsa19, Theorem 2])**.** *There exist efficient deterministic algorithms* EvalF *and* EvalFX *such that for all* $n, q, d \in \mathbb{N}$ *and* $m = n\lceil \log q \rceil$*, for any depth-*$\ell$ *Boolean circuit* $f : \{0,1\}^d \to \{0,1\}^k$ *and for every* $\mathbf{x} \in \{0,1\}^d$*, for any matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times md}$*, the outputs* $\mathbf{H} \leftarrow \mathsf{EvalF}(f, \mathbf{B})$ *and* $\widehat{\mathbf{H}} \leftarrow \mathsf{EvalFX}(f, \mathbf{x}, \mathbf{B})$ *are both in* $\mathbb{Z}^{md \times mk}$ *and it holds that* $\|\mathbf{H}\|_{\max}, \|\widehat{\mathbf{H}}\|_{\max} \leq (2m)^\ell$ *and*

$$[\mathbf{B} - \mathbf{x} \otimes \mathbf{G}]\widehat{\mathbf{H}} = \mathbf{B}\mathbf{H} - f(\mathbf{x}) \otimes \mathbf{G} \ (\mathrm{mod}\ q),$$

where $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ *is the gadget matrix.*

Hence, Lemma 6.4.2 is not helpful to our work. Further, the norm bound of $\widehat{\mathbf{H}}$ and $\mathbf{H}$ in Lemma 6.4.2 are quite large than expected. The following lemma suffices for our DMIPE design.

Lemma 6.4.3 (Evaluation for Inner Product Functions)**.** *Let* $n, q, d \in \mathbb{N}$ *and* $m = n\lceil \log q \rceil$ *be positive integers. Let* $f_{\mathbf{v}} : \mathbb{Z}_q^d \to \mathbb{Z}_q$ *be any inner product function indicated by* $\mathbf{v} \in \mathbb{Z}_q^d$*. Let* $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ *be the gadget matrix. Then, there exists a DPT algorithm* $\mathsf{EvalF}^{\mathsf{IP}}$ *that on input the function* $f_{\mathbf{v}}$ *and any matrix* $\mathbf{B} \in \mathbb{Z}_q^{n \times md}$*, outputs a matrix* $\mathbf{H} \in \{0,1\}^{md \times m} \leftarrow \mathsf{EvalF}^{\mathsf{IP}}(f_{\mathbf{v}}, \mathbf{B})$*, satisfying that* $\|\mathbf{H}\|_{\max} \leq 1$ *and that for every* $\mathbf{x} \in \mathbb{Z}_q^d$*,*

$$[\mathbf{B} \pm \mathbf{x} \otimes \mathbf{G}]\mathbf{H} = \mathbf{B}\mathbf{H} \pm \langle \mathbf{v}, \mathbf{x} \rangle \cdot \mathbf{G} \ (\mathrm{mod}\ q). \tag{6.3}$$

Proof. We will show an instance of the matrix $\mathbf{H}$ that satisfies Equation (6.3). Let $\mathbf{v} = (v_1, \cdots, v_d) \in \mathbb{Z}_q^d$ be the vector corresponding to the inner product function $f_{\mathbf{v}}$. For $i \in [d]$, we define a matrix $\mathbf{H}_i$ as $\mathbf{H}_i := \mathbf{G}^{-1}(v_i \mathbf{G}) \in \{0,1\}^{m \times m}$. Here, $\mathbf{G}^{-1}$ is mentioned in Section 2.5.2. By definition of $\mathbf{G}^{-1}$, it holds that, $\mathbf{G}\mathbf{H}_i = v_i \mathbf{G}$ for all $i \in [d]$. Define,

$$\mathbf{H} := \begin{bmatrix} \mathbf{H}_1 \\ \vdots \\ \mathbf{H}_d \end{bmatrix} \in \{0,1\}^{md \times m}.$$

The following holds:

$$(\mathbf{x} \otimes \mathbf{G})\mathbf{H} = \sum_{i=1}^{d} x_i \mathbf{G}(\mathbf{G}^{-1}(v_i \mathbf{G})) = \sum_{i=1}^{d} x_i v_i \mathbf{G} = \langle \mathbf{v}, \mathbf{x} \rangle \cdot \mathbf{G}.$$

It implies that

$$[\mathbf{B} \pm \mathbf{x} \otimes \mathbf{G}]\mathbf{H} = \mathbf{B}\mathbf{H} \pm \langle \mathbf{v}, \mathbf{x} \rangle \cdot \mathbf{G} \ (\mathrm{mod}\ q).$$

Obviously, $\|\mathbf{H}\|_{\max} \leq 1$ as $\mathbf{H} \in \{0,1\}^{md \times m}$. □

In the next section, we present the DMIPE construction over lattices.

6.4.3 The Construction

First, we summarise the system parameters and give their descriptions in Table 6.2.

Table 6.2: System parameters in our lattice–based DMIPE construction.

Parameters	Definition
λ	security parameter
d	dimension of the DMIPE system
n	# row of matrices $\mathbf{G}$
q	system modulus
m	# column of matrices $\mathbf{G}$
B, ν	parameters for the bounded distribution χ used in DLWE
σ^*	Gaussian parameter in the underlying LWE
σ_0	Gaussian parameter used in the trapdoor

We will first give a brief description of the construction. A DMIPE public key pk includes matrices $\mathbf{A}, \mathbf{G}, \mathbf{B}, \mathbf{U}$. The DMIPE master secret key will be $\mathsf{msk} := \mathbf{A}_{\sigma_0}^{-1}$. Here, $\mathbf{G}$ is the gadget matrix, $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$ and its σ_0–trapdoor $\mathbf{A}_{\sigma_0}^{-1}$ are created by $\mathsf{TrapGen}$, $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n\times dm}$ and $\mathbf{U} \in \mathbb{Z}_q^{n\times m}$.

Let $\vec{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ be a list of predicate vectors. A key $\mathsf{sk}_{\vec{V}}$ associated with $\vec{V}$ is produced by for each $\mathbf{v}_i \in \vec{V}$, evaluating $\mathbf{H}_{\mathbf{v}_i} \leftarrow \mathsf{EvalF}^{\mathsf{IP}}(f_{\mathbf{v}_i}, \mathbf{B})$ (Lemma 6.4.3) and then setting $\mathbf{B}_{\mathbf{v}_i} := \mathbf{B}\mathbf{H}_{\mathbf{v}_i}$. Now, $\mathsf{sk}_{\vec{V}}$ for $\vec{V}$ is $\mathbf{A}_{\vec{V},\sigma_0}^{-1}$ which is a σ_0–trapdoor for $\mathbf{A}_{\vec{V}} := [\mathbf{A}|\mathbf{B}_{\mathbf{v}_1}|\cdots|\mathbf{B}_{\mathbf{v}_k}]$. Suppose that $\vec{V}'$ is a list of vectors satisfying $\vec{V} \subseteq \vec{V}'$. Then, from $\mathbf{A}_{\vec{V},\sigma_0}^{-1}$ we can delegate a key $\mathsf{sk}_{\vec{V}'}$ for $\vec{V}'$ by evaluating $\mathbf{H}_{\mathbf{v}_i} \leftarrow \mathsf{EvalF}^{\mathsf{IP}}(f_{\mathbf{v}_i}, \mathbf{B})$ and setting $\mathbf{B}_{\mathbf{v}_i} := \mathbf{B}\mathbf{H}_{\mathbf{v}_i}$ for all $\mathbf{v}_i \in \vec{V}' \setminus \vec{V}$. Similarly, the key $\mathsf{sk}_{\vec{V}'}$ is a σ_0–trapdoor for $\mathbf{A}_{\vec{V}'} := [\mathbf{A}|\mathbf{B}_{\mathbf{v}_1}|\cdots|\mathbf{B}_{\mathbf{v}_{k'}}]$, where $k' = |\vec{V}'|$.

A ciphertext $\mathsf{ct}_{\mathbf{x}}$ with respect to vector $\mathbf{x}$ on a plaintext $\mu \in \{0,1\}^m$ consists of ($\mathbf{c}_{\mathsf{in}}$, $\mathbf{c}_{\mathsf{mid}}$, $\mathbf{c}_{\mathsf{out}}$), in which $\mathbf{c}_{\mathsf{in}} := \mathbf{s}^\top\mathbf{A} + \mathbf{e}_{\mathsf{in}}^\top \in \mathbb{Z}_q^m$, $\mathbf{c}_{\mathsf{mid}} := \mathbf{s}^\top(\mathbf{B} - \mathbf{x}\otimes\mathbf{G}) + \mathbf{e}_{\mathsf{in}}^\top\mathbf{R} \in \mathbb{Z}_q^{md}$, $\mathbf{c}_{\mathsf{out}} := \mathbf{s}^\top\mathbf{U} + \mathbf{e}_{\mathsf{out}}^\top + \mu\cdot\lceil q/2\rceil \in \mathbb{Z}_q^m$. Here, $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\mathbf{R} \xleftarrow{\$} \{-1,1\}^{m\times md}$ and $\mathbf{e}_{\mathsf{in}}, \mathbf{e}_{\mathsf{out}} \leftarrow \chi^m$ for some distribution χ.

Using a secret key $\mathsf{sk}_{\vec{V}}$ to decrypt a ciphertext $\mathsf{ct}_{\mathbf{x}}$, one computes $\mathbf{c}_{\mathbf{v}_i} := \mathbf{c}_{\mathsf{mid}}\mathbf{H}_{\mathbf{v}_i}$ for all $\mathbf{v}_i \in \vec{V}$. Computing $\mathbf{c}_{\mathsf{out}} - [\mathbf{c}_{\mathsf{in}}|\mathbf{c}_{\mathbf{v}_1}|\cdots|\mathbf{c}_{\mathbf{v}_k}]\mathbf{W}$ enables regaining the underlying plaintext μ as long as $\langle\mathbf{x}, \mathbf{v}_i\rangle = 0 \,(\mathrm{mod}\ q)$, $\forall\mathbf{v}_i \in \vec{V}$.

In the following, we formally present the lattice–based DMIPE system. The system comprises algorithms $\mathsf{DMIPE.Setup}$, $\mathsf{DMIPE.Del}$, $\mathsf{DMIPE.Drive}$, $\mathsf{DMIPE.Enc}$ and $\mathsf{DMIPE.Dec}$.

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{DMIPE.Setup}(1^\lambda, 1^d)$. Taking as input a security parameter λ and a dimension d, $\mathsf{DMIPE.Setup}$ performs the following:

  1. Choose n, m, q according to λ, d. Also, choose a (B,ν)–bounded distribution χ

for the underlying LWE problem. We can take $\chi = D_{\mathbb{Z},\sigma^*}$ (for some $\sigma^* > 0$) which is a $(12\sigma^*, 2^{-100})$–bounded distribution.

2. Take the gadget matrix $\mathbf{G} \in \mathbb{Z}_q^{n\times m}$, and its publicly known trapdoor $\mathbf{G}_{O(1)}^{-1}$.
3. Select a Gaussian parameter σ_0.
4. Generate $(\mathbf{A}, \mathbf{A}_{\sigma_0}^{-1})$ using $\mathsf{TrapGen}(1^n, 1^m, q)$. Also, sample $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n\times m}$, $\mathbf{B} \xleftarrow{\$} \mathbb{Z}_q^{n\times md}$.
5. Output $\mathsf{pk} := (\mathbf{A}, \mathbf{G}, \mathbf{B}, \mathbf{U})$ as public key and $\mathsf{msk} := \mathbf{A}_{\sigma_0}^{-1}$ as master secret key.

- $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{msk}, \vec{V})$. Taking as input a public key pk, a master secret key msk and a list of d–dimensional vectors $\vec{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$, perform:

  1. Compute $\mathbf{H}_{\mathbf{v}_i} \leftarrow \mathsf{EvalF}(f_{\mathbf{v}_i}, \mathbf{B})$ and $\mathbf{B}_{\mathbf{v}_i} := \mathbf{B}\mathbf{H}_{\mathbf{v}_i}$ for each vector $\mathbf{v}_i$.
  2. Set $\mathbf{B}_{\vec{V}} := [\mathbf{B}_{\mathbf{v}_1}|\cdots|\mathbf{B}_{\mathbf{v}_k}]$ and $\mathbf{A}_{\vec{V}} := [\mathbf{A}|\mathbf{B}_{\vec{V}}]$.
  3. Calculate trapdoor $\mathbf{A}_{\vec{V},\sigma_0}^{-1}$ for $\mathbf{A}_{\vec{V}}$ (via Item 3 of Lemma 6.4.1) and return $\mathsf{sk}_{\vec{V}} := \mathbf{A}_{\vec{V},\sigma_0}^{-1}$.

- $\mathsf{sk}_{\vec{V}_2} \leftarrow \mathsf{DMIPE.Del}(\mathsf{pk}, \mathsf{sk}_{\vec{V}_1}, \mathbf{v}_{k+1})$. Taking as input a public key pk, a secret key $\mathsf{sk}_{\vec{V}_1} = \mathbf{A}_{\vec{V}_1,\sigma_0}^{-1}$ for a list $\vec{V}_1 = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$, and a vector $\mathbf{v}_{k+1} \notin \vec{V}_1$), $\mathsf{DMIPE.Del}$ performs the following steps:

  1. For all $i \in [k+1]$, compute $\mathbf{H}_{\mathbf{v}_i} \leftarrow \mathsf{EvalF}^{\mathsf{IP}}(f_{\mathbf{v}_i}, \mathbf{B})$ and $\mathbf{B}_{\mathbf{v}_i} := \mathbf{B}\mathbf{H}_{\mathbf{v}_i}$.
  2. Set $\mathbf{A}_{\vec{V}_2} := [\mathbf{A}|\mathbf{B}_{\mathbf{v}_1}|\cdots|\mathbf{B}_{\mathbf{v}_k}|\mathbf{B}_{\mathbf{v}_{k+1}}]$ with $\vec{V}_2 := \vec{V}_1 \cup \{\mathbf{v}_{k+1}\}$. Note that $\mathbf{A}_{\vec{V}_1} := [\mathbf{A}|\mathbf{B}_{\mathbf{v}_1}|\cdots|\mathbf{B}_{\mathbf{v}_k}]$.
  3. Compute trapdoor $\mathbf{A}_{\vec{V}_2,\sigma_0}^{-1}$ using the trapdoor $\mathbf{A}_{\vec{V}_1,\sigma_0}^{-1}$ (via Item 3 of Lemma 6.4.1) and output $\mathsf{sk}_{\vec{V}_2} := \mathbf{A}_{\vec{V}_2,\sigma_0}^{-1}$.

- $\mathsf{ct}_{\mathbf{x}} \leftarrow \mathsf{DMIPE.Enc}(\mathsf{pk}, \mu, \mathbf{x})$. Taking as input a public key pk, a plaintext $\mu := (\mu_1, \cdots, \mu_m) \in \{0,1\}^m$ and an attribute vector $\mathbf{x} \in \mathbb{Z}_q^d$, $\mathsf{DMIPE.Enc}$ does the following:

  1. Sample $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^n$, $\mathbf{R} \xleftarrow{\$} \{-1,1\}^{m\times md}$ and $\mathbf{e}_{\mathsf{in}}, \mathbf{e}_{\mathsf{out}} \leftarrow \chi^m$.
  2. Compute $\mathbf{c}_{\mathsf{in}} := \mathbf{s}^\top\mathbf{A} + \mathbf{e}_{\mathsf{in}}^\top \in \mathbb{Z}_q^m$, $\mathbf{c}_{\mathsf{mid}} := \mathbf{s}^\top(\mathbf{B} - \mathbf{x}\otimes\mathbf{G}) + \mathbf{e}_{\mathsf{in}}^\top\mathbf{R} \in \mathbb{Z}_q^{md}$, $\mathbf{c}_{\mathsf{out}} := \mathbf{s}^\top\mathbf{U} + \mathbf{e}_{\mathsf{out}}^\top + \mu\cdot\lceil q/2\rceil \in \mathbb{Z}_q^m$.
  3. Return ciphertext $\mathsf{ct}_{\mathbf{x}} := (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_{\mathsf{mid}}, \mathbf{c}_{\mathsf{out}})$.

- $\mu/\perp := \mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{sk}_{\vec{V}}, \mathsf{ct}_{\mathbf{x}})$. Taking as input a public key pk, secret key $\mathsf{sk}_{\vec{V}} := \mathbf{A}_{\vec{V}}^{-1}$ for $\vec{V} = (\mathbf{v}_1, \cdots, \mathbf{v}_k)$ and a ciphertext $\mathsf{ct}_{\mathbf{x}} := (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_{\mathsf{mid}}, c_{\mathsf{out}})$ with respect to $\mathbf{x} \in \mathbb{Z}_q^d$, $\mathsf{DMIPE.Dec}$ does the following:

  1. Evaluate $\mathbf{H}_{\mathbf{v}_i} \leftarrow \mathsf{EvalF}^{\mathsf{IP}}(f_{\mathbf{v}_i}, \mathbf{B})$ and $\mathbf{B}_{\mathbf{v}_i} := \mathbf{B}\mathbf{H}_{\mathbf{v}_i}$ for each vector $\mathbf{v}_i$.

2. Set $\mathbf{A}_{\vec{V}} := [\mathbf{A}|\mathbf{B}_{\mathbf{v}_1}|\cdots|\mathbf{B}_{\mathbf{v}_k}]$.
3. Calculate $\mathbf{W} \leftarrow \mathbf{A}^{-1}_{\vec{V},\sigma_0}(\mathbf{U})$, i.e., $\mathbf{A}_{\vec{V}}\mathbf{W} = \mathbf{U} \pmod q$.
4. For $i \in [k]$, compute $\mathbf{c}_{\mathbf{v}_i} := \mathbf{c}_{\mathsf{mid}}\mathbf{H}_{\mathbf{v}_i}$, i.e, $\mathbf{c}_{\mathbf{v}_i} = \mathbf{s}^\top(\mathbf{B}_{\mathbf{v}_i} + \langle \mathbf{v}_i, \mathbf{x}\rangle \cdot \mathbf{G}) + \mathbf{e}_{\mathsf{in}}^\top \mathbf{R}\mathbf{H}_{\mathbf{v}_i}$.
5. Calculate $\mu' := (\mu'_1, \cdots, \mu'_m) \leftarrow \mathbf{c}_{\mathsf{out}} - [\mathbf{c}_{\mathsf{in}}|\mathbf{c}_{\mathbf{v}_1}|\cdots|\mathbf{c}_{\mathbf{v}_k}]\mathbf{W}$.
6. For $i \in [m]$, return $\mu_i = 0$ if $|\mu'_i| < q/4$; return $\mu_i = 1$ otherwise.

6.4.4 Correctness

Theorem 6.4.1 (Correctness)**.** *Assume that the chosen parameters satisfy*

$$B + 12(mB + km^3B) \cdot \sigma_0 < q/4.$$

Then, the lattice–based DMIPE described in Section 6.4.3 is correct.

Proof. First, recall that $\mathbf{c}_{\mathbf{v}_i} = \mathbf{c}_{\mathsf{mid}}\mathbf{H}_{\mathbf{v}_i} = \mathbf{s}^\top(\mathbf{B}_{\mathbf{v}_i} - \langle \mathbf{v}_i, \mathbf{x}\rangle \cdot \mathbf{G}) + \mathbf{e}_{\mathsf{in}}^\top \mathbf{R}\mathbf{H}_{\mathbf{v}_i}$. Then, $\mathbf{c}_{\mathbf{v}_i} = \mathbf{s}^\top\mathbf{B}_{\mathbf{v}_i} + \mathbf{e}_{\mathsf{in}}^\top \mathbf{R}\mathbf{H}_{\mathbf{v}_i}$ if and only if $\langle \mathbf{v}_i, \mathbf{x}\rangle = 0$. Therefore, in the case that $\langle \mathbf{v}_i, \mathbf{x}\rangle = 0$ for all $\mathbf{v}_i \in \vec{V}$, we have

$$\mu' := \mathbf{c}_{\mathsf{out}} - [\mathbf{c}_{\mathsf{in}}|\mathbf{c}_{\mathbf{v}_1}|\cdots|\mathbf{c}_{\mathbf{v}_k}]\mathbf{W} = \mu \cdot \lceil q/2 \rceil + \mathbf{e}_{\mathsf{out}} + [\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_1}|\cdots|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_k}]\mathbf{W}.$$

Now, we need to bound $\|\mathbf{e}_{\mathsf{out}} + [\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}t\mathbf{H}_{\mathbf{v}_1}|\cdots|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_k}]\mathbf{W}\|_{\max}$. We have

$$\begin{aligned}
&\|\mathbf{e}_{\mathsf{out}} + [\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}t\mathbf{H}_{\mathbf{v}_1}|\cdots|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_k}]\mathbf{W}\|_{\max} \\
&\quad \le \|\mathbf{e}_{\mathsf{out}}\|_{\max} + [\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_1}|\cdots|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_k}]\mathbf{W}\|_{\max} \\
&\quad \le \|\mathbf{e}_{\mathsf{out}}\|_{\max} + (m\|\mathbf{e}_{\mathsf{in}}^\top\|_{\max} + km \max_{i\in[k]} \|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_i}\|\|_{\max}) \cdot \|\mathbf{W}\|_{\max} \\
&\quad \le \|\mathbf{e}_{\mathsf{out}}\|_{\max} + (m\|\mathbf{e}_{\mathsf{in}}^\top\|_{\max} + km^3\|\mathbf{e}_{\mathsf{in}}^\top\|_{\max} \cdot \|\mathbf{R}\|_{\max} \cdot \max_{i\in[k]} \|\mathbf{H}_{\mathbf{v}_i}\|_{\max}) \cdot \|\mathbf{W}\|_{\max}.
\end{aligned}$$

By Lemma 2.1.4, the second and the third inequality above hold. Now, because χ is (B, ν)–bounded then $\|\mathbf{e}_{\mathsf{out}}^\top\|_{\max} \le B$, $\|\mathbf{e}_{\mathsf{in}}^\top\|_{\max} \le B$. Moreover, $\|\mathbf{H}_{\mathbf{v}_i}\|_{\max} \le 1$ by Lemma 6.4.3 and $\|\mathbf{W}\|_{\max} \le 12\sigma_0$ by Item 5 of Lemma 6.4.1. Of course, $\|\mathbf{R}\|_{\max} \le 1$. Therefore,

$$\|\mathbf{e}_{\mathsf{out}} + [\mathbf{e}_{\mathsf{in}}^\top|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}t\mathbf{H}_{\mathbf{v}_1}|\cdots|\mathbf{e}_{\mathsf{in}}^\top\mathbf{R}\mathbf{H}_{\mathbf{v}_k}]\mathbf{W}\|_{\max} \le B + 12(mB + km^3B) \cdot \sigma_0.$$

If parameters fulfil $B + 12(mB + km^3B) \cdot \sigma_0 \le q/4$, then the system is correct. The proof follows. □

6.4.5 Security Analysis

We give the security of the DMIPE system in the following theorem.

Theorem 6.4.2 (Selective Payload–hiding Security)**.** *Under the hardness of the* $(n, 2m, q, \chi)$–DLWE *instance, the lattice–based* DMIPE *described in Section 6.4.3 is selectively payload–hiding secure against chosen plaintext attacks. Specifically, if there exists an adversary* $\mathcal{A}$ *that breaks the selective payload–hiding security of* Π_{DMIPE}*, then one can build a solver* $\mathcal{B}$ *that solves the* $(n, 2m, q, \chi)$ – DLWE *instance.*

Proof. We prove the theorem through hybrid games. The original selective payload-hiding security game is Game 0. The last game is Game 4 in which the adversary's advantage is zero. We will prove two consecutive games to be indistinguishable. In particular, we will show that Game 3 and Game 4 are indistinguishable owing to a reduction from DLWE. Now, let $\mathcal{W}_i$ be the event that the adversary wins in Game i. Our goal is to prove that $|\Pr[\mathcal{W}_0] - 1/2| = \mathsf{negl}(\lambda)$. Details are below.

Game 0. This is the original game $\mathsf{DMIPE}^{\mathsf{sel,CPA}}_{\mathsf{payload},\mathcal{A}}$ stated in Figure 6.3. In this game, the adversary releases a target attribute vector $\mathbf{x}^*$. Also, suppose that in the **Challenge** phase when producing the challenge ciphertext, the challenger generates the short matrix $\mathbf{R}^* \in \{-1, 1\}^{m \times md}$.

Game 1. This game resembles Game 0 except that generating the short matrix $\mathbf{R}^* \xleftarrow{\$} \{-1, 1\}^{m \times md}$ is now moved to the **Setup** phase, no longer in the **Challenge** phase.

Game 2. This game is modified from Game 1 in the way of generating $\mathsf{pk} := (\mathbf{A}, \mathbf{G}, \mathbf{B}, \mathbf{U})$. Everything else is unchanged. Specifically, $\mathbf{B}$ is not randomly sampled but set as

$$\mathbf{B} := \mathbf{A}\mathbf{R}^* + \mathbf{x}^* \otimes \mathbf{G} \in \mathbb{Z}_q^{n \times md}. \tag{6.4}$$

The matrix $\mathbf{A}$ is still created together with $\mathbf{A}_{\sigma_0}^{-1}$ by $\mathsf{TrapGen}(1^n, 1^m, q)$ and $\mathbf{U} \xleftarrow{\$} \mathbb{Z}_q^{n \times m}$. We know that, the challenge ciphertext $\mathsf{ct}^*_{\mathbf{x}^*}$ will consist of $(\mathbf{c}^*_{\mathsf{in}}, \mathbf{c}^*_{\mathsf{mid}}, \mathbf{c}^*_{\mathsf{out}})$. However, notice that in this game, by Equation (6.4), the component $\mathbf{c}^*_{\mathsf{mid}}$ will become

$$\mathbf{c}^*_{\mathsf{mid}} := \mathbf{s}^\top(\mathbf{B} - \mathbf{x}^* \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{in}}^\top \mathbf{R}^* = \mathbf{s}^\top(\mathbf{A}\mathbf{R}^*) + \mathbf{e}_{\mathsf{in}}^\top \mathbf{R}^* = \mathbf{c}^*_{\mathsf{in}} \mathbf{R}^*.$$

Game 3. This game is the same as Game 2, except that now the matrix $\mathbf{A}$ is sampled uniformly at random from $\mathbb{Z}_q^{n \times m}$. The challenger does not have $\mathbf{A}_{\sigma_0}^{-1}$ anymore. However, the challenger can use the trapdoor $\mathbf{G}_{O(1)}^{-1}$ for its response to the adversary's queries. Specifically, let $\vec{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$. The key queries $\mathsf{KQ}(\vec{V})$, is responded as below:

1. For each vector $\mathbf{v}_i$, the challenger computes $\mathbf{H}_{\mathbf{v}_i} \leftarrow \mathsf{EvalF}^{\mathsf{IP}}(f_{\mathbf{v}_i}, \mathbf{A}\mathbf{R}^*)$.
   and then sets $\mathbf{B}_{\mathbf{v}_i} := \mathbf{B}\mathbf{H}_{\mathbf{v}_i} = \mathbf{A}\mathbf{R}^*\mathbf{H}_{\mathbf{v}_i} + \langle \mathbf{v}_i, \mathbf{x}^* \rangle \cdot \mathbf{G}$.
2. The challenger sets the matrix $\mathbf{A}_{\vec{V}} := [\mathbf{A} | \mathbf{B}_{\mathbf{v}_1} | \cdots | \mathbf{B}_{\mathbf{v}_k}]$.

3. If for all $i \in [k]$, $\langle \mathbf{v}_i, \mathbf{x}^* \rangle = 0 \pmod q$, then the challenger aborts the query. Otherwise, there exists $i \in [k]$ so that $\langle \mathbf{v}_{i_0}, \mathbf{x}^* \rangle \neq 0 \pmod q$. Then, the challenger can use $\mathbf{G}_{O(1)}^{-1}$ to compute $[\mathbf{A}|\mathbf{A}\mathbf{R}^*\mathbf{H}_{\mathbf{v}_{i_0}} + \langle \mathbf{v}_{i_0}, \mathbf{x}^* \rangle \cdot \mathbf{G}]_\sigma^{-1}$ (via Item 4 of Lemma 6.4.1). After that, the challenger uses $[\mathbf{A}|\mathbf{A}\mathbf{R}^*\mathbf{H}_{\mathbf{v}_{i_0}} + \langle \mathbf{v}_{i_0}, \mathbf{x}^* \rangle \cdot \mathbf{G}]_\sigma^{-1}$ tp compute $[\mathbf{A}_{\vec{V}}]_\sigma^{-1}$ (via Item 3 of Lemma 6.4.1), and finally returns $\mathsf{sk}_{\vec{V}} \leftarrow [\mathbf{A}_{\vec{V}}]_\sigma^{-1}$.

Key delegation queries $\mathsf{KD}(\vec{V}, \mathbf{v})$ are also answered in the same way.

Game 4. This game slightly modifies Game 3. In this game, in the challenge ciphertext two components $\mathbf{c}_{\mathsf{in}}$ and $\mathbf{c}_{\mathsf{out}}$ are both uniform in $\mathbb{Z}_q^m$. The component $\mathbf{c}_{\mathsf{mid}}$ is kept unchanged. Then, it is easy to see that $|\Pr[\mathcal{W}_4] - 1/2| = 0$.

Our purpose is to prove that $|\Pr[\mathcal{W}_0] - 1/2| \leq \mathsf{negl}(\lambda)$. To do that, we will show the indistinguishability of the two consecutive games. This is done through the following lemmas.

Lemma 6.4.4. Game 1 *and* Game 0 *are perfectly the same in the view of the adversary; i.e.,*

$$\Pr[\mathcal{W}_1] = \Pr[\mathcal{W}_0].$$

Proof. By the construction, sampling $\mathbf{R}^*$ is independent of the view of the adversary. Therefore, sampling $\mathbf{R}^*$ at the **Setup** phase will not notice the adversary. □

Lemma 6.4.5. *In the view of the adversary $\mathcal{A}$,* Game 2 *and* Game 1 *are indistinguishable, i.e.,*

$$|\Pr[\mathcal{W}_1] - \Pr[\mathcal{W}_2]| \leq \mathsf{negl}(\lambda).$$

Proof. By the leftover hash lemma (Lemma 2.5.1), this lemma holds. □

Lemma 6.4.6. *In the view of the adversary $\mathcal{A}$,* Game 3 *and* Game 2 *are indistinguishable, i.e.,*

$$|\Pr[\mathcal{W}_2] - \Pr[\mathcal{W}_3]| \leq \mathsf{negl}(\lambda).$$

Proof. Note that Item 1 of Lemma 6.4.1 says that matrix $\mathbf{A}$ generated by TrapGen looks like random. Furthermore, both using the trapdoor $\mathbf{G}_{O(1)}^{-1}$ and using the trapdoor $\mathbf{A}_{\sigma_0}^{-1}$ return the same output distribution. However, choosing the Gaussian parameter σ in Step 3 of Hybrid 3 should be careful. Namely, we should choose

$$\begin{aligned}
\sigma &= m \cdot \|\mathbf{R}^*\mathbf{H}_{\mathbf{v}_{i_0}}\|_{\max} \cdot \omega(\sqrt{\log m}) \\
&\leq m^2 d \cdot \|\mathbf{R}^*\|_{\max} \cdot \|\mathbf{H}_{\mathbf{v}_{i_0}}\|_{\max} \cdot \omega(\sqrt{\log m}) \\
&\leq m^2 d \cdot \omega(\sqrt{\log m}).
\end{aligned}$$

□

Lemma 6.4.7. *In the view of the adversary $\mathcal{A}$,* Game 4 *and* Game 3 *are indistinguishable, i.e.,*

$$|\Pr[\mathcal{W}_3] - \Pr[\mathcal{W}_4]| \leq \mathsf{negl}(\lambda),$$

assuming the hardness of the $(n, 2m, q, \chi)$–DLWE *problem.*

Proof. We prove the lemma via a reduction. The key idea of the reduction is as follows. Suppose that $\mathcal{A}$ can distinguish Game 4 from Game 3. Then, we can construct a DLWE solver $\mathcal{B}$ exploiting $\mathcal{A}$. Specifically,

DLWE Instance. The DLWE solver $\mathcal{B}$ is required to solve an $(n, 2m, q, \chi)$–DLWE instance $(\mathbf{F}, \mathbf{c})$, where $\mathbf{F} = [\mathbf{A}|\mathbf{U}] \xleftarrow{\$} \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^{n\times m}$, and a vector $\mathbf{c} = (\mathbf{c}_{\mathsf{in}}, \mathbf{c}_{\mathsf{out}}) \in \mathbb{Z}_q^m \times \mathbb{Z}_q^m$. The goal of $\mathcal{B}$ is to decide the following two cases:

- **Case 1:** $\mathbf{c}$ is random in $\mathbb{Z}_q^{2m}$; or
- **Case 2:** $\mathbf{c}$ is LWE samples, i.e., $\mathbf{c}_{\mathsf{in}}^\top = \mathbf{s}^\top \mathbf{A} + \mathbf{e}_{\mathsf{in}}^\top$, $\mathbf{c}_{\mathsf{out}}^\top = \mathbf{s}^\top \mathbf{U} + \mathbf{e}_{\mathsf{out}}^\top$, for some random vector $\mathbf{s} \in \mathbb{Z}_q^n$ and $(\mathbf{e}_{\mathsf{out}}, \mathbf{e}_{\mathsf{out}}) \leftarrow \chi^m \times \chi^m$.

Initialise. $\mathcal{A}$ releases its target attribute vector $\mathbf{x}^* \in \mathbb{Z}_q^d$.

Setup. $\mathcal{B}$ now samples $\mathbf{R}^* \xleftarrow{\$} \{-1, 1\}^{m\times md}$ and then computes $\mathbf{B} := \mathbf{A}\mathbf{R}^* + \mathbf{x}^* \otimes \mathbf{G}$. After that, $\mathcal{B}$ sets the a public key $\mathsf{pk} = (\mathbf{A}, \mathbf{G}, \mathbf{B}, \mathbf{U})$ and master secret key in the same way as in Game 3. The public key pk will be sent to $\mathcal{A}$.

Query. For queries of $\mathcal{A}$, $\mathcal{B}$ follows Game 3 to respond.

Challenge. $\mathcal{A}$ submits two plaintexts μ_0^* and μ_1^* to $\mathcal{B}$. The challenge ciphertext will be computed as follows: $\mathcal{B}$ chooses a bit $b \xleftarrow{\$} \{0, 1\}$, then sets $\mathbf{c}_{\mathsf{in}}^* = \mathbf{c}_{\mathsf{in}}$, $\mathbf{c}_{\mathsf{mid}}^{*\top} \leftarrow \mathbf{c}_{\mathsf{in}}^{*\top} \mathbf{R}^*$ and $\mathbf{c}_{\mathsf{out}}^* \leftarrow \mathbf{c}_{\mathsf{out}} + \mu_b^* \lceil \frac{q}{2} \rceil$. At this point, the challenge ciphertext is computes $\mathsf{ct}_{\mathbf{x}^*} = (\mathbf{c}_{\mathsf{in}}^*, \mathbf{c}_{\mathsf{mid}}^*, \mathbf{c}_{\mathsf{out}}^*)$.

Output. Based on the result that $\mathcal{A}$ has returned, $\mathcal{B}$ will decide the DLWE instance.

Our reasoning is that Game 3 and Game 4 just differ in the way of generating the challenge ciphertext, which directly relates to the DLWE instance. We notice here that

- If $\mathbf{c}$ belongs to **Case 2**, then $\mathbf{c}_{\mathsf{in}}^\top = \mathbf{s}^\top \mathbf{A} + \mathbf{e}_{\mathsf{in}}^\top$, $\mathbf{c}_{\mathsf{out}}^\top = \mathbf{s}^\top \mathbf{U} + \mathbf{e}_{\mathsf{out}}^\top$. Hence, $\mathbf{c}_{\mathsf{mid}}^{*\top} = \mathbf{c}_{\mathsf{in}}^{*\top} \mathbf{R}^* = \mathbf{s}^\top \mathbf{A}\mathbf{R}^* + \mathbf{e}_{\mathsf{in}}^\top \mathbf{R}^* = \mathbf{s}^\top (\mathbf{B} - \mathbf{x}^* \otimes \mathbf{G}) + \mathbf{e}_{\mathsf{in}}^\top \mathbf{R}^*$, which is precisely the ones computed in Game 3.
- If $\mathbf{c}$ belongs to **Case 1**, then $\mathbf{c}_{\mathsf{in}}^*$, $\mathbf{c}_{\mathsf{out}}^*$ are uniformly random. Then $\mathsf{ct}_{\mathbf{x}^*}^*$ is exactly computed as in Game 4.

Therefore, if $\mathcal{A}$ can distinguish Game 3 and Game 4, then $\mathcal{B}$ can succeeds to decide the DLWE instance with probability not less than $\mathcal{A}$'s probability. This completes the proof for Lemma 6.4.7. □

From Lemmas 6.4.4–6.4.7, we conclude

$$\begin{aligned}\left|\Pr[\mathcal{W}_0]-\frac{1}{2}\right| &\leq |\Pr[\mathcal{W}_0]-\Pr[\mathcal{W}_1]|+|\Pr[\mathcal{W}_1]-\Pr[\mathcal{W}_2]| \\ &\quad +|\Pr[\mathcal{W}_2]-\Pr[\mathcal{W}_3]|+|\Pr[\mathcal{W}_3]-\Pr[\mathcal{W}_4]|+|\Pr[\mathcal{W}_4]-1/2| \\ &\leq \mathsf{negl}(\lambda).\end{aligned}$$

This completes the proof for Theorem 6.4.2. □

6.4.6 Setting Parameters

Parameters for the DMIPE construction in Section 6.4.3 can be heuristically chosen as follows:

- Choose a security parameter first and denote it by λ.
- Choose parameters such that the $(n, 2m, q, \chi)$–DLWE (in Lemma 6.4.7) is hard which follows Lemma 2.4.1. Specifically, we choose $n = n(\lambda)$, ϵ, ν, $q = q(n) \leq 2^n$, $m = \Theta(n \log q) = \mathsf{poly}(n)$, $\chi = \chi(n)$ such that χ is a (B, ν)–bounded for some $B = B(n)$ such that, $q/B \geq 2^{n^\epsilon}$. Note that in the literature, choosing practical parameters usually follows the "core–SVP hardness" methodology; see [ABDo20, Section 5.2.1].
- Choose $m > (n+1)\log q + \omega(\log n)$ (For Lemma 6.4.5; due to Lemma 2.5.1).
- Choose Gaussian parameter $\sigma_0 = \omega(n \log q \log n)$ (for TrapGen; due to Item 1 of Lemma 6.4.1).
- Choose Gaussian parameter $\sigma \geq m^2 d \cdot \omega(\sqrt{\log m})$ (for Game 3 to work; due to Lemma 6.4.6).
- Parameters should also satisfy the condition

  $$B + 12(mB + km^3 B) \cdot \sigma_0 < q/4,$$

  (for Correctness; due to Theorem 6.4.1).

6.5 Constructing DMIPE from SE

We have shown that there is a generic construction of SE from DMIPE. Reversely, in this section, we will prove that: one can also build a DMIPE from an SE. The primary tool for doing this is a transformation, named OVS, that maps a list of predicate vectors $\vec{V} = \{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ in the DMIPE manner to the (unique) orthogonal complement of the subspace generated by all vectors in $\vec{V}$, which acts as a vector subspace in the SE manner. That is,

$$\mathsf{OVS}(\vec{V}) := (\mathsf{span}(\mathbf{v}_1, \cdots, \mathbf{v}_k))^{\perp}.$$

Such a transformation guarantees that $\langle \mathbf{v}_i, \mathbf{x} \rangle = 0 \pmod{q}$ $\forall \mathbf{v}_i \in \vec{V}$ is equivalent to $\mathbf{x} \in \mathsf{OVS}(\vec{V})$. Additionally, the transformation also makes sure that if $\vec{V}_1 \subseteq \vec{V}_2$ then $\mathsf{OVS}(\vec{V}_2) \sqsubseteq \mathsf{OVS}(\vec{V}_1)$.

The construction works as follows.

- $(\mathsf{pk}, \mathsf{msk}) \leftarrow \mathsf{DMIPE.Setup}(1^\lambda, \mathsf{sp})$. Run $(\mathsf{se.pp}, \mathsf{se.msk}) \leftarrow \mathsf{SE.Setup}(1^\lambda, \mathsf{sp})$ and then output $\mathsf{pk} := \mathsf{se.pp}$, $\mathsf{msk} := \mathsf{se.msk}$.
- $\mathsf{sk}_{\vec{V}} \leftarrow \mathsf{DMIPE.Derive}(\mathsf{pk}, \mathsf{T}, \mathsf{msk}, \vec{V})$. Run $V \leftarrow \mathsf{OVS}(\vec{V})$, $\mathsf{se.sk}_V \leftarrow \mathsf{SE.Derive}(\mathsf{pk}, \mathsf{msk}, V)$ and then output $\mathsf{sk}_{\vec{V}} := \mathsf{se.sk}_V$.
- $\mathsf{ct}_{\mathbf{x}} \leftarrow \mathsf{DMIPE.Enc}(\mathsf{pk}, \mathbf{x}, \mu)$. Run $\mathsf{se.ct}_{\mathbf{x}} \leftarrow \mathsf{SE.Enc}(\mathsf{pk}, \mathbf{x}, \mu)$ and output $\mathsf{ct}_{\mathbf{x}} := \mathsf{se.ct}_{\mathbf{x}}$.
- $\mathsf{sk}_{\vec{V}_2} \leftarrow \mathsf{DMIPE.Del}(\mathsf{pk}, \vec{V}_1, \mathsf{sk}_{\vec{V}_1}, \mathbf{v})$. Compute $V_1 \leftarrow \mathsf{OVS}(\vec{V}_1)$, $V_2 \leftarrow \mathsf{OVS}(\vec{V}_1 \cup \{\mathbf{v}\})$, and then $\mathsf{se.sk}_{V_2} \leftarrow \mathsf{SE.Del}(\mathsf{pk}, \mathsf{se.sk}_{V_1}, V_2)$. (Note that $\mathsf{se.sk}_{V_1} = \mathsf{sk}_{\vec{V}_1}$.) Finally, output $\mathsf{sk}_{\vec{V}_2} := \mathsf{se.sk}_{V_2}$.
- $\mu/\perp \leftarrow \mathsf{DMIPE.Dec}(\mathsf{pk}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_V)$. Return the output $\mu/\perp \leftarrow \mathsf{SE.Dec}(\mathsf{pk}, \mathsf{ct}_{\mathbf{x}}, \mathsf{sk}_V)$.

The correctness of the DMIPE construction straightforwardly follows from the correctness of the corresponding SE. We can prove the security of the DMIPE construction in a similar way to that of Theorem 6.3.2.

6.6 Allow–/Deny–list Encryption from Spatial Encryption

Allow–/deny–list encryption (ADE) was introduced by Derler *et al.* [DRSS21] as a generalisation of IBE [Sha85], HIBE [GS02a], PE [GM15], DFPE [DRSS21], FuPE [DKL+18]. However, Derler *et al.* [DRSS21] did not state any syntax or security notions for ADE. Roughly stating, ADE is a sort of PKE that involves with *tags*, namely, *positive tags* included in an *allow list* and *negative tags* included in a *deny list*. These two

kinds of lists are involved in both ciphertexts and decryption keys through a delegation mechanism called *puncturing*. More specifically, ciphertexts can be generated together with mixed tags. Puncturing on negative tags will produce decryption keys revoked from the decryption ability. Contrarily, puncturing on positive tags will produce decryption keys allowed for decryption.

This section formally defines the syntax and security notions for ADE. Moreover, we also introduce three ADE versions: standard ADE (sADE), k–threshold ADE (k–tADE) and inclusive ADE (iADE). They are different according to the correctness requirements. We then present the encodings that help transform sADE and iADE to SE.

6.6.1 Framework of ADE

Denote λ to be a security parameter. Let $d = d(\lambda)$ (respectively, $a = a(\lambda)$) be the maximum number of negative tags per ciphertext (respectively, the the maximum number of positive tags) for the ADE system. Denote by $\mathcal{M} = \mathcal{M}(\lambda)$, $\mathcal{T}^{(+)} = \mathcal{T}^{(+)}(\lambda)$ and $\mathcal{T}^{(-)} = \mathcal{T}^{(-)}(\lambda)$ the plaintext space, the positive tag space and the negative tag space, respectively.

Syntax. ADE is a collection of the algorithms: key generation $\mathsf{ADE.Gen}$, encryption $\mathsf{ADE.Enc}$, negative puncturing $\mathsf{ADE.Npun}$, positive puncturing $\mathsf{ADE.Ppun}$, and decryption $\mathsf{ADE.Dec}$. They work as described below.

- $(\mathsf{pk}, \mathsf{sk}_{\emptyset}^{\emptyset}) \leftarrow \mathsf{ADE.Gen}(1^{\lambda}, 1^{a}, 1^{d})$. $\mathsf{ADE.Gen}$ is a key generation algorithm. It is PPT. Its inputs are a security parameter λ, a maximum number a of positive tags per ciphertext and a maximum number d of negative tags per ciphertext. Its output are a public key pk, and a (not punctured) initial secret key $\mathsf{sk}_{\emptyset}^{\emptyset}$.

- $\mathsf{sk}_{DL'}^{AL'_1 \cup AL'_2} \leftarrow \mathsf{ADE.Ppun}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'_1}, AL'_2, k)$. [a] $\mathsf{ADE.Ppun}$ is the positive puncturing algorithm. It is PPT. Its inputs are a public key pk, a previously punctured key $\mathsf{sk}_{DL'}^{AL'_1}$ for a set of positive tags $\emptyset \subseteq AL'_1 \subseteq \mathcal{T}^{(+)}$ and a set of negative tags $\emptyset \subseteq DL' \subseteq \mathcal{T}^{(-)}$, and a set of positive tags $AL'_2 \in \mathcal{T}^{(+)} \setminus AL'_1$. Its output is a new punctured key $\mathsf{sk}_{DL'}^{AL'_1 \cup \mathsf{AL'_2}}$.

- $\mathsf{sk}_{DL'_1 \cup DL'_2}^{AL'} \leftarrow \mathsf{ADE.Npun}(\mathsf{pk}, \mathsf{sk}_{DL'_1}^{AL'}, DL'_2)$. $\mathsf{ADE.Npun}$ is the negative puncturing algorithm. It is PPT. Its input are a public key pk, a previously punctured key $\mathsf{sk}_{DL'_1}^{AL'}$ for a set of positive tags $\emptyset \subseteq AL' \subseteq \mathcal{T}^{(+)}$, a set of negative tags $\emptyset \subseteq DL'_1 \subseteq \mathcal{T}^{(-)}$ and a set of negative tags $DL'_2 \subseteq \mathcal{T}^{(-)} \setminus DL'_1$. Its output is a new punctured key $\mathsf{sk}_{DL'_1 \cup DL'_2}^{AL'}$.

- $\mathsf{ct}_{DL}^{AL} \leftarrow \mathsf{ADE.Enc}(\mathsf{pk}, \mu, AL, DL)$. $\mathsf{ADE.Enc}$ is the encryption algorithm. It is PPT. Its inputs are a public key pk, a plaintext μ, a set of positive tags AL and a set of negative tags DL. Its output is a ciphertext ct_{DL}^{AL}.

[a] Here, note that k is only used in the k-tADE variant.

- $\mu/\perp := \mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'}, \mathsf{ct}_{DL}^{AL})$. $\mathsf{ADE.Dec}$ is the decryption algorithm. It is DPT. Its inputs are a public key pk, a secret key $\mathsf{sk}_{DL'}^{AL'}$ associated with $AL' \subseteq \mathcal{T}^{(+)}$ and $DL' \subseteq \mathcal{T}^{(-)}$, and a ciphertext ct_{DL}^{AL} associated with $AL \subseteq \mathcal{T}^{(+)}$ and $DL \subseteq \mathcal{T}^{(-)}$. Its output is either a plaintext μ (if decryption succeeds) or $\perp$ (otherwise).

Correctness and ADE Versions. We will define the correctness and classify ADE versions at the same time. Consider any $\lambda, a, d \in \mathbb{N}$, $\mu \in \mathcal{M}$, $\emptyset \subset AL, AL' \subseteq \mathcal{T}^{(+)}$, $\emptyset \subset DL, DL' \subseteq \mathcal{T}^{(-)}$, any $(\mathsf{pk}, \mathsf{sk}_{\emptyset}^{\emptyset}) \leftarrow \mathsf{ADE.Gen}(1^{\lambda}, 1^{a}, 1^{d})$, and any punctured key $\mathsf{sk}_{DL'}^{AL'}$ generated using any combination of $\mathsf{ADE.Npun}$, and $\mathsf{ADE.Ppun}$ on AL', DL'.

All versions require that the initial key $\mathsf{sk}_{\emptyset}^{\emptyset}$ is always able to successfully decrypt any ciphertext that was produced using the corresponding a public key pk. That is,

$$\Pr[\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{\emptyset}^{\emptyset}, \mathsf{ADE.Enc}(\mathsf{pk}, \mu, AL, DL)) = \mu] \geq 1 - \mathsf{negl}(\lambda).$$

However, when punctured, the additional correctness requirement varies for each version. Specifically,

- **Standard ADE (sADE).** If $(AL = AL') \wedge (DL \cap DL' = \emptyset)$ then

  $$\Pr[\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'}, \mathsf{ADE.Enc}(\mathsf{pk}, \mu, AL, DL)) = \mu] \geq 1 - \mathsf{negl}(\lambda).$$

  Otherwise,

  $$\Pr[\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'}, \mathsf{ADE.Enc}(\mathsf{pk}, \mathsf{pk}, \mu, AL, DL)) = \mu] \leq \mathsf{negl}(\lambda).$$

- **Inclusive ADE (iADE).** If $((AL' \subseteq AL) \wedge (DL \cap DL' = \emptyset))$ then

  $$\Pr[\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'}, \mathsf{ADE.Enc}(\mathsf{pk}, \mu, AL, DL)) = \mu] \geq 1 - \mathsf{negl}(\lambda).$$

  Otherwise,

  $$\Pr[\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'}, \mathsf{ADE.Enc}(\mathsf{pk}, \mu, AL, DL)) = \mu] \leq \mathsf{negl}(\lambda).$$

- **k–threshold ADE (k–tADE).** If $((|AL \cap AL'| \geq k) \wedge (DL \cap DL' = \emptyset))$, then

  $$\Pr[\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'}, \mathsf{ADE.Enc}(\mathsf{pk}, \mu, AL, DL)) = \mu] \geq 1 - \mathsf{negl}(\lambda).$$

  Otherwise,

  $$\Pr[\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}_{DL'}^{AL'}, \mathsf{ADE.Enc}(\mathsf{pk}, \mu, AL, DL)) = \mu] \leq \mathsf{negl}(\lambda).$$

Note that if $AL' \subseteq AL$ holds, then iADE should be sADE.

Security Notions of ADE Versions. We define the selective payload–hiding security for all ADE versions. The security notion is given in Definition 6.6.1 and Figure 6.5 below.

Definition 6.6.1 (PAY–sPUN–ATK for ADE)**.** *Define the advantage of the adversary $\mathcal{A}$ in the game* $\mathsf{ADE}^{\mathsf{PAY\text{–}sPUN\text{–}ATK}}_{\mathcal{A}}(\lambda, a, d)$ *as*

$$\mathsf{Adv}^{\mathsf{PAY\text{–}sPUN\text{–}ATK}}_{\mathcal{A},\ \mathsf{ADE}}(\lambda) := \left|\Pr[\mathsf{ADE}^{\mathsf{PAY\text{–}sPUN\text{–}ATK}}_{\mathcal{A}}(\lambda, a, d) \Rightarrow 1] - \frac{1}{2}\right|.$$

We say that an ADE is PAY–sPUN–ATK secure if, for any polynomial–time adversary $\mathcal{A}$,

$$\mathsf{Adv}^{\mathsf{PAY\text{–}sPUN\text{–}ATK}}_{\mathcal{A},\ \mathsf{ADE}}(\lambda) \leq \mathsf{negl}(\lambda).$$

GAME $\mathsf{ADE}^{\mathsf{PAY\text{–}sATT\text{–}ATK}}_{\mathcal{A}}(\lambda, a, d)$:

(where $\mathsf{ATK} \in \{\mathsf{CPA}, \mathsf{CCA1}, \mathsf{CCA2}\}$)

1. $(AL^*, DL^*) \leftarrow \mathcal{A}(1^\lambda, 1^a, 1^d)$;
2. $(\mathsf{pk}, \mathsf{sk}^{\emptyset}_{\emptyset}) \leftarrow \mathsf{ADE.Gen}(1^\lambda, 1^a, 1^d)$, $AL' \leftarrow \emptyset$, $DL' \leftarrow \emptyset$;
3. $(\mu^*_0, \mu^*_1) \leftarrow \mathcal{A}^{\mathsf{Punc}(\cdot,\cdot),\mathsf{DQ}(\cdot,\cdot)}(\mathsf{pk})$;
4. $b \xleftarrow{\$} \{0,1\}$, $\mathsf{ct}^{AL^*}_{DL^*} \leftarrow \mathsf{ADE.Enc}(\mathsf{pk}, \mu^*_b, AL^*, DL^*)$;
5. $b' \leftarrow \mathcal{A}^{\mathsf{Pun}(\cdot,\cdot),\mathsf{DQ}(\cdot)}(\mathsf{pk}, \mathsf{ct}^{AL^*}_{DL^*})$; // **NOTE:** Not allowed $\mathsf{DQ}(AL', DL', \mathsf{ct}^{AL^*}_{DL^*})$ with $(AL', DL') \in \mathsf{SUCC}(AL^*, DL^*)$.
6. If $b' = b$, return 1. Otherwise, return 0.

Queried Oracles:

- Puncturing Oracle $\mathsf{Pun}((AL', DL'))$ (It is only allowed if $(AL', DL') \notin \mathsf{SUCC}(AL^*, DL^*)$): Run $\mathsf{ADE.Ppun}$ and $\mathsf{ADE.Npun}$ in any order using $\mathsf{sk}^{\emptyset}_{\emptyset}$ to output $\mathsf{sk}^{AL'}_{DL'}$.
- Decryption Oracle $\mathsf{DQ}(AL', DL', \mathsf{ct}^{AL}_{DL})$ (allowed only if ATK=CCA): Run $\mathsf{ADE.Ppun}$ and $\mathsf{ADE.Npun}$ in any order using $\mathsf{sk}^{\emptyset}_{\emptyset}$ to get $\mathsf{sk}^{AL'}_{DL'}$. Finally, return the output of $\mathsf{ADE.Dec}(\mathsf{pk}, \mathsf{sk}^{AL'}_{DL'}, \mathsf{ct}^{AL}_{DL})$.

Define $\mathsf{SUCC}(AL^*, DL^*)$ **for ADE Variants:**

- **For sADE:**
  $\mathsf{SUCC}(AL^*, DL^*) := \{(AL', DL') : ((AL' = AL^*) \wedge (DL' \cap DL^* = \emptyset))\}$.
- **For k–tADE**:
  $\mathsf{SUCC}(AL^*, DL^*) := \{(AL', DL') : ((|AL' \cap AL^*| \geq k) \wedge (DL' \cap DL^* = \emptyset))\}$.
- **For iADE:**
  $\mathsf{SUCC}(AL^*, DL^*) := \{(A'L, DL') : ((AL' \subseteq AL^*) \wedge (DL' \cap DL^* = \emptyset))\}$.

Figure 6.5: Security game for ADE versions.

6.6.2 Transforming sADE and iADE to SE

For q prime, let $\mathcal{T}^{(-)}, \mathcal{T}^{(+)} \subset \mathbb{Z}_q$ be (finite) negative tag space and positive tag space, respectively. Suppose that $|\mathcal{T}^{(+)}| = a$ and $|\mathcal{T}^{(-)}| = d$ involved in the sADE (or iADE) version. For two pairs of allow list and deny list $(AL'_1, DL'_1), (AL'_2, DL'_2) \in \mathcal{T}^{(+)} \times \mathcal{T}^{(-)}$, we say $(AL'_1, DL'_1) \subseteq (AL'_2, DL'_2)$ if and only if $(AL'_1 \subseteq AL'_2) \wedge (DL'_1 \subseteq DL'_2)$.

The idea for transforming sADE and iADE to SE is as follows: Assume that $(AL', DL') \subseteq \mathcal{T}^{(+)} \times \mathcal{T}^{(-)}$ to be a pair of tag lists that is punctured on decryption keys. We encode the pair into a subspace V compatible with the SE syntax. Note that V is possibly affine. Accordingly, for any pair $(AL, DL) \subseteq \mathcal{T}^{(+)} \times \mathcal{T}^{(-)}$ of ciphertext tags, we encode it as a vector $\mathbf{v}$ satisfying that $\mathbf{v} \in V$ if and only if $(AL' \subseteq AL) \wedge (DL' \cap DL = \emptyset)$.

To perform that, we use the following encodings $\mathsf{EncodeInKey}$ and $\mathsf{EncodeInCipher}$:

- $W_{\mathsf{key}} \leftarrow \mathsf{EncodeInKey}(AL', DL')$. On input a pair $(AL', DL') \subseteq \mathcal{T}^{(+)} \times \mathcal{T}^{(-)}$, do the following:

  1. The allow list $AL' = \{p_1, \cdots, p_k\}$ is stick to a space of vectors beginning with $(p_1, \cdots, p_k)^\top$, namely

  $$W_{AL'} := \{(p_1, \cdots, p_k, x_{k+1}, \cdots, x_a)^\top : x_i \in \mathbb{Z}_q\} \subseteq \mathbb{Z}_q^a. \tag{6.5}$$

  One can easily check that if $AL'_1 \subseteq AL'_2$ then $W_{AL'_2} \sqsubseteq W_{AL'_1}$.

  2. The deny list DL' is stick to the space

  $$W_{DL'} := \mathsf{span}\{\mathbf{v}_x : x \in DL''\}, \tag{6.6}$$

  where $\mathbf{v}_x := (1, x, x^2, \cdots, x^{2d-1})$ and $DL'' := \mathcal{T}^{(-)} \setminus DL'$ is the complement of DL'.

  We observe that adding one more tag into DL' is equivalent to removing one tag from DL''. Then given $DL'_1 \subseteq DL'_2$ we have $W_{DL''_2} \sqsubseteq W_{DL''_1}$.

  3. Output the subspace W_{key} which is defined as

  $$W_{\mathsf{key}} := W_{AL'} \times W_{DL'}.$$

  Now, one can perform negative puncturing and positive puncturing of sADE and iADE via the delegation of SE.

- $\mathbf{x}_{\mathsf{ct}} \leftarrow \mathsf{EncodeInCipher}(AL, DL)$. On input a pair $(AL, DL) \subseteq \mathcal{T}^{(+)} \times \mathcal{T}^{(-)}$, do the following steps:

  1. Stick $AL = \{p_1, \cdots, p_k\}$ to vector $\mathbf{x}_{AL} := (p_1, \cdots, p_k, 0, \cdots, 0) \in \mathbb{Z}_q^a$.

Clearly, if $AL' \subseteq AL$ then $\mathbf{x}_{AL} \in W_{AL'}$. Here $W_{AL'}$ is defined as in Equation (6.5).

2. Encode the list DL as $\mathbf{x}_{DL} := \sum_{x \in DL} \mathbf{v}_x \in \mathbb{Z}_q^{2d}$, where $\mathbf{v}_x := (1, x, x^2, \cdots, x^{2d-1})$. It is observed that $\mathbf{x}_{DL} \notin W_{DL'}$ for any $DL \cap DL' \neq \emptyset$ (i.e., $DL \not\subseteq DL''$). Here $W_{DL'}$ is defined as in Equation (6.6).
3. Output vector $\mathbf{x}_{\mathsf{ct}} := (\mathbf{x}_{AL}, \mathbf{x}_{DL}) \in \mathbb{Z}_q^{a+2d}$.

Obviously, $\mathbf{x}_{\mathsf{ct}} \in W_{\mathsf{key}}$ if and only if $(\mathbf{x}_{AL}, \mathbf{x}_{DL}) \in W_{AL'} \times W_{DL'}$, which is equivalent to $(AL' \subseteq AL) \wedge (DL' \cap DL = \emptyset)$. Therefore, the correctness and the security of sADE and iADE are inherited from those of SE.

6.7 Summary

We have revisited the notion of spatial encryption (SE). We were motivated by the shortcomings of a generic framework that transforms a hierarchical inner product encryption (HIPE) system into an SE system. The framework will allow having lattice–based SE from the corresponding lattice–based HIPE. However, such an induced SE incurs a large key and ciphertext size.

To realise an SE over lattices with smaller sizes, we begin by introducing a primitive named delegatable multiple inner product encryption (DMIPE). DMIPE is also a generalisation of inner product encryption (IPE) but having equipped with a delegation mechanism. We proved that one could generically transform a DMIPE to get an SE and vice versa. In other words, there are "security notions–preserving" conversions between SE and DMIPE. We then instantiate a DMIPE in the lattice setting.

Following our generic framework, the SE construction over lattices obtained from the lattice DMIPE is better in size than the existing lattice–based SEs induced from HIPE.

We showed that the DMIPE over lattices (hence, the induced SE construction) offers selectively payload–hiding security in the standard model. Nevertheless, we believe that our lattice DMIPE construction can achieve the *selectively weak attribute–hiding* security which is defined by Agrawal *et al.* [AFV11]. Moreover, a possible technical idea to realise that might also be from Agrawal *et al.* [AFV11]. So this would be exciting work for the future.

Furthermore, an adaptively secure DMIPE (and hence SE) construction from lattices would be desired. Remark that in the recent work [KNYY20], an IPE enjoying adaptively security has been proposed. Therefore, adaptively secure DMIPE and SE constructions may be able to exploit the ideas of [KNYY20]. However, the delegation mechanism may be the most challenging thing in achieving adaptive security for DMIPE. Therefore, we leave it for further research in the future. In addition, a DMIPE and SE construction that

offer *attribute–hiding security* in the lattice setting should also be worthwhile for pursuing further research.

One more thing is that we could only convert sADE and iADE into SE via encodings specified in Section 6.6.2. However, finding the encodings for changing k–tADE into SE is still open. Seemingly, the idea of *threshold gates* (see [Ham11, Page 51]) might be helpful to do that. However, we think that instead of the original SE defined in this thesis, the doubly spatial encryption (DSE) (which is also an SE variant introduced in [Ham11]) or some other SE variants might be needed. Again, we leave this as further work.

Chapter 7

Conclusion and Future Work

In this chapter, we summarise the contributions of the thesis. We then sketch some fascinating directions for further research.

7.1 Summary of the Thesis

Lattice–based cryptography is rapidly developing with many breakthrough results. It promises to be an excellent source of cryptographic constructions, which are secure against quantum adversaries. Moreover, lattice–based cryptography still offers vast room for further developments and improvements.

This thesis investigates several selected topics of lattice–based cryptography. We exploit some well–known technical tools such as rejection sampling, forking lemma, lattice trapdoors, and lattice homomorphic evaluations, to list a few. Our primary research focus has been to develop new technical tools, propose novel primitives and improve the existing constructions. The following contributions are noteworthy.

- We have proposed the first forward–secure blind signatures over lattices. We have shown that they are secure against key exposure attacks.
- We have developed a trapdoor delegation for the LVV19 trapdoor. We have demonstrated that the trapdoor can be used to construct the first HIBE based on the DM-PLWE problem.
- We have introduced a new primitive called delegatable fully key–homomorphic encryption (DFKHE). From DFKHE, we can get puncturable encryption (PE) in the generic sense. Further, we have developed a concrete lattice-based PE by instantiation of DFKHE.
- We have revisited the notion of spatial encryption (SE). In particular, we have introduced the concept of delegatable multiple inner product encryption (DMIPE).

Moreover, we have shown a security–preserving equivalence between SE and DMIPE and then have presented a DMIPE over lattices. Additionally, we have formally defined allow–/deny–list encryption (ADE), which covers PE (among others) as a subclass. It turns out that one can get two versions of ADE from SE through appropriate encodings.

7.2 Future Work

We have pointed out some open problems/questions in each chapter. However, from a more generic perspective, there are some other exciting research directions that we can work on in the future. First, we would like to implement the lattice–based cryptosystems proposed in this thesis and experiment with their parameters to maximise efficiency and security. In particular, we would be able to evaluate other practical aspects such as performance and actual sizes of keys, signatures and ciphertexts. Additionally, we could compare our lattice–based construction with others. However, before doing that, we must consider cryptographic optimisation and integrate this concept into the revised design when implementing our systems. Furthermore, we should do a systematic investigation on choosing specific parameters for techniques/tools in lattices, such as trapdoors and homomorphic evaluations. Specifying hidden constants in parameter conditions/relations is one of the main jobs. The job may need intensive experiments to bound the value of the hidden constants. This investigation will help set concrete parameters in our lattice–based cryptosystems and follow–up works.

Second, the lattice trapdoors and the lattice homomorphic evaluations seem to be still far from being practical in terms of efficiency. Recently, we have seen some efforts to improve the efficacy of the lattice trapdoor mechanisms, including [GPR$^+$18, CGM19, BEP$^+$21]. One research direction for the future is to improve the practical aspects of these mechanisms.

Third, the principal purpose of designing cryptosystems based on lattices is their resistance against quantum adversaries. However, it seems to be insufficient to guarantee the post–quantum security of a lattice–based cryptosystem if the cryptosystem is just proved secure in the random oracle model (ROM). This is because a quantum adversary may access random oracles in quantum superposition as discussed in [BDF$^+$11, ABB$^+$17]. On the other hand, one can obtain the post–quantum security of a lattice–based cryptosystem if its security is proven in the standard model. Unfortunately, security in the standard model usually comes at a high cost in performance. A recently discussed alternative is security proofs in the quantum random oracle model (QROM) (e.g., [BDF$^+$11, ABB$^+$17, KLS18, LZ19, DFMS19, YZ21] to name a few). It is folklore that cryptosystems secure in ROM often offer better efficiency than ones secure in the standard model; see, e.g.,

[BR93] for further detail. However, this fact may no longer be true for QROM. At this point, we must determine whether QROM or the standard model is more suitable for practical purposes. The answer may vary among specific cryptosystems. In our future work, we intend to convert all cryptosystems proposed in this thesis into counterparts secure in QROM. By doing that, we can compare which model between the standard model and QROM provides better efficiency for each of our lattice cryptosystems.

Besides that, quantum adversaries may also have the ability to query quantumly to the signing/encryption/decryption oracle. Advanced cryptosystems over lattices that resist such quantum adversaries' power should be worth focusing on further in later works.

Bibliography

[AB09] Shweta Agrawal and Dan Boneh. Identity-Based Encryption from Lattices in the Standard Model. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, 2009. (Cited on pages 10, 99, and 108.)

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient Lattice (H)IBE in the Standard Model. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 LNCS, pages 553–572, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. (Cited on pages 4, 5, 8, 10, 11, 13, 33, 34, 35, 108, 113, and 143.)

[ABB+17] Erdem Alkim, Nina Bindel, Johannes Buchmann, Özgür Dagdelen, Edward Eaton, Gus Gutoski, Juliane Krämer, and Filip Pawlega. Revisiting Tesla in the Quantum Random Oracle Model. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10346 LNCS:143–162, 2017. (Cited on page 160.)

[ABDo20] Erdem Alkim, Joppe W Bos, Leo Ducas, and others. Frodo{KEM}: Learning with Errors Key Encapsulation Algorithm Specifications And Supporting Documentation, version 30 September, 2020. Technical report, 2020. (Cited on pages 129 and 151.)

[AD97] Miklos Ajtai and Cynthia Dwork. A Public-Key Cryptosystem with Equivalence. *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 284–293, 1997. (Cited on page 3.)

[ADCM12] Michel Abdalla, Angelo De Caro, and Karina Mochetti. Lattice-based hierarchical inner product encryption. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7533 LNCS:121–138, 2012. (Cited on pages 14, 15, 16, 17, 133, 135, and 136.)

[AFV11] Shweta Agrawal, David Mandell Freeman, and Vinod Vaikuntanathan. Functional encryption for inner product predicates from learning with errors.

Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 7073 LNCS:21–40, 2011. (Cited on pages 57 and 157.)

[Ajt96] M Ajtai. Generating Hard Instances of Lattice Problems (Extended Abstract). In *Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing*, STOC '96, pages 99–108, New York, NY, USA, 1996. ACM. (Cited on pages 2, 3, 5, 10, 34, and 143.)

[Ajt99] Miklós Ajtai. Generating Hard Instances of the Short Basis Problem. In *Automata, Languages and Programming, 26th International Colloquium, ICALP'99, Prague, Czech Republic, July 11-15, 1999, Proceedings*, pages 1–9, 1999. (Cited on page 10.)

[And02] Ross Anderson. Two remarks on public key cryptology. Technical Report, University of Cambridge, Computer Laboratory, 2002. (Cited on page 8.)

[AP09] Joël Alwen and Chris Peikert. Generating Shorter Bases for Hard Random Lattices. In *26th International Symposium on Theoretical Aspects of Computer Science, {STACS} 2009, February 26-28, 2009, Freiburg, Germany, Proceedings*, pages 75–86, 2009. (Cited on pages 34, 35, and 143.)

[AR00] Michel Abdalla and Leonid Reyzin. A New Forward-Secure Digital Signature Scheme. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, pages 116–129, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. (Cited on page 8.)

[AS15] Jacob Alperin-Sheriff. Short signatures with short public keys from homomorphic trapdoor functions. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9020, pages 236–255, 2015. (Cited on page 4.)

[BB11] Dan Boneh and Xavier Boyen. Efficient Selective-ID Secure Identity Based Encryption Without Random Oracles. *Journal of Cryptography*, 24(4:659–693, 2011. (Cited on pages 13 and 129.)

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7073 LNCS:41–69, 2011. (Cited on page 160.)

[BEP+21] Pauline Bert, Gautier Eberhart, Lucas Prabel, Adeline Roux-Langlois, and Mohamed Sabt. *Implementation of Lattice Trapdoors on Modules and Applications*, volume 12841 LNCS. 2021. (Cited on page 160.)

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8441 LNCS, pages 533–556, 2014. (Cited on pages 5, 8, 12, 13, 23, 29, 33, 34, 35, 40, 41, 111, 112, 113, and 142.)

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (Leveled) Fully Homomorphic Encryption Without Bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, pages 309–325, New York, NY, USA, 2012. ACM. (Cited on pages 4 and 5.)

[BH08] Dan Boneh and Michael Hamburg. Generalized identity based and broadcast encryption schemes. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5350 LNCS:455–470, 2008. (Cited on pages 13, 14, 15, 54, and 132.)

[BKM17] Dan Boneh, Sam Kim, and Hart Montgomery. Private Puncturable PRFs from Standard Lattice Assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 415–445, Cham, 2017. Springer International Publishing. (Cited on page 113.)

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical Hardness of Learning with Errors. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, pages 575–584, New York, NY, USA, 2013. ACM. (Cited on pages 4, 5, and 30.)

[BM99] Mihir Bellare and Sara K Miner. A Forward-Secure Digital Signature Scheme. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 431–448, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. (Cited on pages 8, 9, 45, and 59.)

[BN06] Mihir Bellare and Gregory Neven. Multi-signatures in the plain public-Key model and a general forking lemma. *Proceedings of the ACM Conference*

on Computer and Communications Security, pages 390–399, 2006. (Cited on page 66.)

[Boy10] Xavier Boyen. Lattice Mixing and Vanishing Trapdoors: A Framework for Fully Secure Short Signatures and More. In Phong Q Nguyen and David Pointcheval, editors, *Public Key Cryptography – PKC 2010*, pages 499–517, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. (Cited on page 4.)

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: a paradigm for designing efficient protocols. *1st ACM Conference on Computer and Communications Security*, (November 1993):62–73, 1993. (Cited on pages 41 and 161.)

[BTVW17] Zvika Brakerski, Rotem Tsabary, Vinod Vaikuntanathan, and Hoeteck Wee. Private Constrained PRFs (and More) from LWE. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10677 LNCS:264–302, 2017. (Cited on pages 5 and 143.)

[BV11a] Zvika Brakerski and Vinod Vaikuntanathan. Efficient Fully Homomorphic Encryption from (Standard) LWE. In *Proceedings of the 2011 IEEE 52Nd Annual Symposium on Foundations of Computer Science*, FOCS '11, pages 97–106, Washington, DC, USA, 2011. IEEE Computer Society. (Cited on page 5.)

[BV11b] Zvika Brakerski and Vinod Vaikuntanathan. Fully Homomorphic Encryption from Ring-LWE and Security for Key Dependent Messages. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 505–524, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. (Cited on page 5.)

[BV16] Zvika Brakerski and Vinod Vaikuntanathan. Circuit-ABE from LWE: Unbounded Attributes and Semi-adaptive Security. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016*, volume 9816, pages 363–384, Berlin, Heidelberg, 2016. Springer. (Cited on pages 5, 13, 29, 30, and 143.)

[CGM19] Yilei Chen, Nicholas Genise, and Pratyay Mukherjee. Approximate Trapdoors for Lattices and Smaller Hash-and-Sign Signatures. In Steven D Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 3–32, Cham, 2019. Springer International Publishing. (Cited on page 160.)

[Cha83] David Chaum. Blind Signatures for Untraceable Payments. In David Chaum, Ronald L Rivest, and Alan T Sherman, editors, *Advances in Cryptology*, pages 199–203, Boston, MA, 1983. Springer US. (Cited on pages 8, 9, 43, and 59.)

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A Forward-Secure Public-Key Encryption Scheme. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003*, pages 255–271, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on pages 10, 88, and 111.)

[CHK04] Ran Canetti, Shai Halevi, and Jonathan Katz. Chosen-Ciphertext Security from Identity-Based Encryption. In *Advances in Cryptology - {EUROCRYPT} 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 207–222, 2004. (Cited on page 13.)

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010. EUROCRYPT 2010. Lecture Notes in Computer Science*, volume 6110, pages 601–639, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on pages 4, 5, 8, 10, 11, 34, 35, 113, and 143.)

[CHN+16] Aloni Cohen, Justin Holmgren, Ryo Nishimaki, Vinod Vaikuntanathan, and Daniel Wichs. Watermarking Cryptographic Capabilities. In Daniel Wichs and Yishay Mansour, editors, *STOC 2016: Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 1115–1127, Cambridge, MA,USA, 2016. (Cited on page 11.)

[CHYC05] Sherman S M Chow, Hui Lucas Chi Kwong, Siu-Ming Yiu, and K P Chow. Forward-secure multisignature and blind signature schemes. *Applied Mathematics and Computation*, 168:895–908, 2005. (Cited on page 9.)

[CLLW14] Jie Chen, Hoonwei Lim, San Ling, and Huaxiong Wang. The relation and transformation between hierarchical inner product encryption and spatial encryption. *Designs, Codes, and Cryptography*, 71(2):347–364, 2014. (Cited on pages 14, 15, 132, 133, 136, and 140.)

[Coh96] Henri Cohen. *A Course in Computational Algebraic Number Theory*. Number Graduate texts in mathematics, 138. Springer, Berlin, 1996. (Cited on page 140.)

[Cop97] Don Coppersmith. Small solutions to polynomial equations, and low exponent RSA vulnerabilities. *Journal of Cryptology*, 10(4):233–260, 1997. (Cited on page 2.)

[CPS95] Jan L Camenisch, Jean-Marc Piveteau, and Markus A Stadler. Blind signatures based on the discrete logarithm problem. In Alfredo De Santis, editor, *Advances in Cryptology — EUROCRYPT'94*, pages 428–432, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg. (Cited on pages 8 and 9.)

[CW14] Jie Chen and Hoeteck Wee. Doubly spatial encryption from DBDH. *Theoretical Computer Science*, 543(C):79–89, 2014. (Cited on pages 14 and 15.)

[CZF12] Cheng Chen, Zhenfeng Zhang, and Dengguo Feng. Fully secure doubly-spatial encryption under simple assumptions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 7496 LNCS:253–263, 2012. (Cited on pages 14 and 15.)

[DCK03] Dang Nguyen Duc, Jung Hee Cheon, and Kwangjo Kim. A Forward-Secure Blind Signature Scheme Based on the Strong RSA Assumption. In Sihan Qing, Dieter Gollmann, and Jianying Zhou, editors, *Information and Communications Security*, pages 11–21, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on pages 8, 9, 45, and 46.)

[DF03] Yevgeniy Dodis and Nelly Fazio. Public Key Broadcast Encryption for Stateless Receivers. In Joan Feigenbaum, editor, *Digital Rights Management*, pages 61–80, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. (Cited on pages 10 and 88.)

[DFMS19] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Security of the Fiat-Shamir Transformation in the Quantum Random-Oracle Model. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11693 LNCS, pages 356–383. Springer Verlag, 2019. (Cited on page 160.)

[DGJ18] David Derler, Kai Gellert, and Tibor Jager. Bloom Filter Encryption and Applications to Efficient Forward-Secret 0-RTT Key Exchange. Cryptology ePrint Archive, Report 2018/199, 2018. (Cited on pages 11 and 12.)

[DJSS18] David Derler, Tibor Jager, Daniel Slamanig, and Christoph Striecks. Bloom filter encryption and applications to efficient forward-secret 0-RTT key exchange. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Lecture Notes*

in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 10822 LNCS, pages 425–455, Cham, 2018. Springer International Publishing. (Cited on pages 11 and 12.)

[DKL+18] David Derler, Stephan Krenn, Thomas Lorünser, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. Revisiting Proxy Re-encryption: Forward Secrecy, Improved Security, and Applications. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography – PKC 2018*, pages 219–250, Cham, 2018. Springer International Publishing. (Cited on pages 11, 14, 17, and 152.)

[DM14] Léo Ducas and Daniele Micciancio. Improved Short Lattice Signatures in the Standard Model. In Juan A Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 335–352, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. (Cited on page 4.)

[DOW92] W Diffie, P C V Oorschot, and M J Wiener. Authentication and authenticated key exchanges. *Des Codes Crypt*, 2:107–125, 1992. (Cited on page 8.)

[DRSS21] David Derler, Sebastian Ramacher, Daniel Slamanig, and Christoph Striecks. Fine-Grained Forward Secrecy : Allow-List / Deny-List Encryption and Applications. In *Financial Cryptography and Data Security 2021*, pages 1–22, 2021. (Cited on pages 14, 17, 134, and 152.)

[DSDR21] Priyanka Dutta, Willy Susilo, Dung Hoang Duong, and Partha Sarathi Roy. Puncturable Identity-Based Encryption from Lattices. In Joonsang Baek and Sushmita Ruj, editors, *Australasian Conference on Information Security and Privacy*, pages 571–589, Cham, 2021. Springer International Publishing. (Cited on page 134.)

[FS87] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194, Berlin, Heidelberg, 1987. Springer Berlin Heidelberg. (Cited on page 68.)

[FS90] Uriel Feige and Adi Shamir. Witness indistinguishable and witness hiding protocols. pages 416–426, 1990. (Cited on page 67.)

[Gen09] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *In Proc. STOC*, pages 169–178, 2009. (Cited on pages 4 and 5.)

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1294(k 2):112–131, 1997. (Cited on pages 3 and 5.)

[GHJL17] Felix Günther, Britta Hale, Tibor Jager, and Sebastian Lauer. 0-RTT Key Exchange with Full Forward Secrecy. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017*, pages 519–548, Cham, 2017. Springer International Publishing. (Cited on pages 11 and 12.)

[GM15] M D Green and I Miers. Forward Secure Asynchronous Messaging from Puncturable Encryption. In *2015 IEEE Symposium on Security and Privacy*, pages 305–320, 5 2015. (Cited on pages 11, 12, 14, 17, 53, 110, 131, 133, 134, and 152.)

[GPR+18] Kamil Doruk Gur, Yuriy Polyakov, Kurt Rohloff, Gerard W. Ryan, and Erkay Savas. Implementation and evaluation of improved Gaussian sampling for latice trapdoors. *Proceedings of the ACM Conference on Computer and Communications Security*, pages 61–71, 2018. (Cited on page 160.)

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for Hard Lattices and New Cryptographic Constructions. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 197–206, New York, NY, USA, 2008. ACM. (Cited on pages 4, 5, 8, 10, 11, 13, 28, 29, 32, 34, 35, 49, 113, and 143.)

[GS02a] Craig Gentry and Alice Silverberg. Hierarchical ID-Based Cryptography. In Yuliang Zheng, editor, *Advances in Cryptology — ASIACRYPT 2002*, pages 548–566, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. (Cited on pages 14, 50, and 152.)

[GS02b] Craig Gentry and Mike Szydlo. Cryptanalysis of the Revised NTRU Signature Scheme. In Lars R Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002*, pages 299–320, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. (Cited on pages 9 and 88.)

[GSW13a] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Ran Canetti and Juan A Garay, editors, *Advances*

in Cryptology – CRYPTO 2013, pages 75–92, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. (Cited on page 5.)

[GSW13b] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. Cryptology ePrint Archive, Report 2013/340, 2013. (Cited on page 33.)

[Gün90] Christoph G Günther. An Identity-Based Key-Exchange Protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *Advances in Cryptology — EUROCRYPT '89*, pages 29–37, Berlin, Heidelberg, 1990. Springer Berlin Heidelberg. (Cited on page 8.)

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9216(639554):503–523, 2015. (Cited on page 5.)

[Ham11] Mike Hamburg. Spatial Encryption. *PhD. Thesis*, (July), 2011. (Cited on pages 13, 14, 15, 54, 132, and 158.)

[HKLN20] Eduard Hauck, Eike Kiltz, Julian Loss, and Ngoc Khanh Nguyen. Lattice-based blind signatures, revisited. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 12171 LNCS, pages 500–529. Springer, 2020. (Cited on pages 61, 85, and 86.)

[HL02] Jeremy Horwitz and Ben Lynn. Toward hierarchical identity-based encryption. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2332:466–481, 2002. (Cited on pages 9 and 88.)

[HM96] Shai Halevi and Silvio Micali. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 201–215, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. (Cited on page 69.)

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph H Silverman. NTRU: A ring-based public key cryptosystem. In Joe P Buhler, editor, *Algorithmic Number Theory*, pages 267–288, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg. (Cited on page 3.)

[IR01] Gene Itkis and Leonid Reyzin. Forward-Secure Signatures with Optimal Signing and Verifying. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 332–354, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. (Cited on page 8.)

[JFC+10] Yu Jia, Kong Fanyu, Xiangguo Cheng, Hao Rong, Chen Yangkui, Li Xuliang, and Li Guowen. Forward-Secure Multisignature, Threshold Signature and Blind Signature Schemes. *Journal of Networks*, 5, 2010. (Cited on page 9.)

[Kil06] Eike Kiltz. {Chosen-Ciphertext Security from Tag-Based Encryption}. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 581–600, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. (Cited on page 13.)

[KLS18] Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiat-shamir signatures in the quantum random-oracle model. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10822 LNCS:552–586, 2018. (Cited on page 160.)

[KNYY20] Shuichi Katsumata, Ryo Nishimaki, Shota Yamada, and Takashi Yamakawa. Adaptively Secure Inner Product Encryption from LWE. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12493 LNCS:375–404, 2020. (Cited on pages 5, 143, and 157.)

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4965 LNCS(2006):146–162, 2008. (Cited on pages 14, 54, 57, and 136.)

[KTX08] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Concurrently Secure Identification Schemes Based on the Worst-Case Hardness of Lattice Problems. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008*, pages 372–389, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. (Cited on page 69.)

[Kuc10] Marcin Kucharczyk. Blind Signatures in Electronic Voting Systems. In Andrzej Kwiecień, Piotr Gaj, and Piotr Stera, editors, *Computer Networks*, pages 349–358, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. (Cited on pages 8 and 59.)

[LC05] Yeu-Pong Lai and Chin-Chen Chang. A simple forward secure blind signature scheme based on master keys and blind signatures. In *19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 1 (AINA papers)*, volume 2, pages 139–144, 2005. (Cited on page 9.)

[LDR+21] Huy Quoc Le, Dung Hoang Duong, Partha Sarathi Roy, Willy Susilo, Kazuhide Fukushima, and Shinsaku Kiyomoto. Lattice-based Signcryption with Equality Test in Standard Model. *Computer Standards & Interfaces*, 76(June 2021):103515, 2021. (Cited on page xvii.)

[LDS+20] Huy Quoc Le, Dung Hoang Duong, Willy Susilo, Ha Thanh Nguyen Tran, Viet Cuong Trinh, Josef Pieprzyk, and Thomas Plantard. Lattice Blind Signatures with Forward Security. In Joseph K Liu and Hui Cui, editors, *Information Security and Privacy*, pages 3–22, Cham, 2020. Springer International Publishing. (Cited on pages xvi, 9, 18, 59, 85, and 86.)

[LDSP20] Huy Quoc Le, Dung Hoang Duong, Willy Susilo, and Josef Pieprzyk. *Trapdoor Delegation and HIBE from Middle-Product LWE in Standard Model*, volume 2. Springer International Publishing, 2020. (Cited on pages xvi, 10, 11, 18, and 88.)

[LDSP22a] Huy Quoc Le, Dung Hoang Duong, Willy Susilo, and Josef Pieprzyk. Spatial Encryption Revisited: From Delegatable Multiple Inner Product Encryption and More. Cryptology ePrint Archive, Report 2022/095, 2022. (Cited on page 19.)

[LDSP22b] Huy Quoc Le, Dung Hoang Duong, Willy Susilo, and Josef Pieprzyk. Spatial Encryption Revisited: From Delegatable Multiple Inner Product Encryption and More. In Atluri Vijayalakshmi, Roberto Di Pietro, Jensen Christian D., and Meng Weizhi, editors, *Computer Security – ESORICS 2022*, pages 283–302, Cham, 2022. Springer International Publishing. (Cited on pages xvi, 19, and 132.)

[Len83] H. W. Lenstra. Integer Programming With a Fixed Number of Variables. *Mathematics of Operations Research*, 8(4):538–548, 1983. (Cited on page 2.)

[LLL82] Arjen K Lenstra, Hendrik W Lenstra, and Laszlo Lovasz. Factoring Polynomials with Rational Coefficients. In *Mathematische Annalen*, volume 261, 1982. (Cited on pages 2 and 5.)

[LM06] Vadim Lyubashevsky and Daniele Micciancio. Generalized Compact Knapsacks Are Collision Resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *Automata, Languages and Programming*, pages 144–155, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. (Cited on page 31.)

[LM09] Vadim Lyubashevsky and Daniele Micciancio. On bounded distance decoding, unique shortest vectors, and the minimum distance problem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5677 LNCS:577–594, 2009. (Cited on page 4.)

[LPR10] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On Ideal Lattices and Learning with Errors over Rings. In *Advances in Cryptology - (EUROCRYPT) 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, pages 1–23, 2010. (Cited on pages 4 and 5.)

[LPR13] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. A toolkit for ring-LWE cryptography. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 7881 LNCS, pages 35–54, 2013. (Cited on page 4.)

[LS15] Adeline Langlois and Damien Stehlé. Worst-case to average-case reductions for module lattices. *Designs, Codes and Cryptography*, 75(3):565–599, 6 2015. (Cited on pages 4 and 5.)

[LSK+19] Huy Quoc Le, Willy Susilo, Thanh Xuan Khuc, Minh Kim Bui, and Dung Hoang Duong. A Blind Signature from Module Latices. In *2019 IEEE Conference on Dependable and Secure Computing, DSC 2019 - Proceedings*, 2019. (Cited on pages xvii and 85.)

[LVD+21] Huy Quoc Le, Bay Vo, Dung Hoang Duong, Willy Susilo, Ngoc T. Le, Kazuhide Fukushima, and Shinsaku Kiyomoto. Identity-Based Linkable Ring Signatures From Lattices. *IEEE Access*, 9:84739–84755, 2021. (Cited on page xvi.)

[LVV19] Alex Lombardi, Vinod Vaikuntanathan, and Thuy Duong Vuong. Lattice Trapdoors and IBE from Middle-Product LWE. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography*, pages 24–54, Cham, 2019. Springer International Publishing. (Cited on pages xiii, 4, 5, 8, 10, 11, 23, 31, 32, 34, 36, 37, 38, 39, 88, 89, 90, and 99.)

[Lyu08a] Vadim Lyubashevsky. Lattice-based identification schemes secure under active attacks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4939 LNCS:162–179, 2008. (Cited on page 60.)

[Lyu08b] Vadim Lyubashevsky. *Towards practical lattice-based cryptography*. PhD thesis, UC San Diego, 2008. (Cited on pages 67 and 68.)

[Lyu09] Vadim Lyubashevsky. Fiat-Shamir with Aborts: Applications to Lattice and Factoring-Based Signatures. In *Advances in Cryptology - {ASIACRYPT} 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 598–616, 2009. (Cited on pages 4 and 68.)

[Lyu11] Vadim Lyubashevsky. Lattice Signatures Without Trapdoors. Technical report, Cryptology ePrint Archive, Paper 2011/537, 2011. (Cited on pages 78, 83, and 85.)

[Lyu12] Vadim Lyubashevsky. Lattice Signatures without Trapdoors. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, pages 738–755, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on pages 4, 29, 32, 63, and 78.)

[LZ19] Qipeng Liu and Mark Zhandry. Revisiting Post-quantum Fiat-Shamir. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11693 LNCS, pages 326–355. Springer Verlag, 2019. (Cited on page 160.)

[Mic01] Daniele Micciancio. Improving Lattice based cryptosystems using the Hermite Normal Form. In Joseph Silverman, editor, *Cryptography and Lattices Conference — CaLC 2001*, volume 2146 of *Lecture Notes in Computer Science*, pages 126–145, Providence, Rhode Island, 2001. Springer-Verlag. (Cited on page 3.)

[Mic07] Daniele Micciancio. Generalized compact knapsacks, cyclic lattices, and efficient one-way functions. *Computational Complexity*, 16(4):365–411, 2007. (Cited on page 3.)

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

Heidelberg, 2012. Springer Berlin Heidelberg. (Cited on pages 4, 5, 10, 11, 33, 34, 35, 36, 88, 90, 112, and 143.)

[MR04] Daniele Micciancio and Oded Regev. Worst-Case to Average-Case Reductions Based on Gaussian Measures. In *45th Symposium on Foundations of Computer Science (FOCS) 2004) 17-19 October 2004, Rome, Italy, Proceedings*, pages 372–381, 2004. (Cited on pages 3, 5, 26, 27, 28, and 29.)

[NIS16] NIST. NIST Post-Quantum Cryptography Standardization. *https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization.*, 2016. (Cited on page 1.)

[OT09] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 5912 LNCS, pages 214–231, 2009. (Cited on pages 14 and 132.)

[Pan01] Victor Y Pan. *Structured Matrices and Polynomials: Unified Superfast Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001. (Cited on page 90.)

[Pei09] Chris Peikert. Bonsai Trees (or, Arboriculture in Lattice-Based Cryptography). Cryptology ePrint Archive, Report 2009/359, 2009. (Cited on pages 4 and 30.)

[Pei16] Chris Peikert. A Decade of Lattice Cryptography. *Found. Trends Theor. Comput. Sci.*, 10(4):283–424, 3 2016. (Cited on pages 2 and 6.)

[PNXW18] T V X Phuong, R Ning, C Xin, and H Wu. Puncturable Attribute-Based Encryption for Secure Data Delivery in Internet of Things. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, pages 1511–1519, 4 2018. (Cited on page 131.)

[PS96] David Pointcheval and Jacques Stern. Provably secure blind signature schemes. In Kwangjo Kim and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '96*, pages 252–265, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. (Cited on pages 8, 9, and 59.)

[PS00] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 13(3):361–396, 2000. (Cited on pages 9, 65, and 85.)

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5157 LNCS(2006):554–571, 2008. (Cited on page 4.)

[Reg03] Oded Regev. New lattice based cryptographic constructions. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 407–416, 2003. (Cited on page 3.)

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *Proceedings of the 37th Annual {ACM} Symposium on Theory of Computing, Baltimore, MD, USA, May 22-24, 2005*, pages 84–93, 2005. (Cited on pages 3, 4, and 5.)

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM*, 56(6):84–93, 9 2009. (Cited on page 30.)

[RSSS17] Miruna Roșca, Amin Sakzad, Damien Stehlé, and Ron Steinfeld. Middle-Product Learning with Errors. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 283–297, Cham, 2017. Springer International Publishing. (Cited on pages 4, 5, and 31.)

[Rüc10] Markus Rückert. Lattice-based blind signatures. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6477 LNCS:413–430, 2010. (Cited on pages 9, 60, 61, 69, and 85.)

[SCJ+16] Smith-Tone, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta Daniel, Ray Perlner, and Daniel Smith-Tone. Report on Post-Quantum Cryptography. Technical report, National Institute of Standards and Technology, 2016. (Cited on page 1.)

[SDL20] Willy Susilo, Dung Hoang Duong, and Huy Quoc Le. Efficient Post-quantum Identity-based Encryption with Equality Test. In *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*, pages 633–640, 2020. (Cited on page xvii.)

[SDLP20] Willy Susilo, Dung Hoang Duong, Huy Quoc Le, and Josef Pieprzyk. Puncturable encryption: A generic construction from delegatable fully key-homomorphic encryption. *Lecture Notes in Computer Science (including*

subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 12309 LNCS:107–127, 2020. (Cited on pages xvi, 12, 18, 110, and 111.)

[Sha85] Adi Shamir. Identity-Based Cryptosystems and Signature Schemes. In George Robert Blakley and David Chaum, editors, *Advances in Cryptology*, volume 196 LNCS, pages 47–53, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg. (Cited on pages 9, 14, 50, 88, and 152.)

[Sho02] P.W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 11 2002. (Cited on page 1.)

[Sim10] Denis Simon. Selected applications of LLL in number theory. *Information Security and Cryptography*, 10:265–282, 2010. (Cited on page 2.)

[SSS+20] Shi Feng Sun, Amin Sakzad, Ron Steinfeld, Joseph K. Liu, and Dawu Gu. Public-Key Puncturable Encryption: Modular and Compact Constructions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12110 LNCS:309–338, 2020. (Cited on pages 11 and 12.)

[SSTX09] Damien Stehlé, Ron Steinfeld, Keisuke Tanaka, and Keita Xagawa. Efficient Public Key Encryption Based on Ideal Lattices. In *Advances in Cryptology - ASIACRYPT 2009, 15th International Conference on the Theory and Application of Cryptology and Information Security, Tokyo, Japan, December 6-10, 2009. Proceedings*, pages 617–635, 2009. (Cited on pages 4, 5, 31, and 32.)

[Tsa19] Rotem Tsabary. *Fully Secure Attribute-Based Encryption for t-CNF from LWE*, volume 11692 LNCS. Springer International Publishing, 2019. (Cited on pages 5, 143, and 144.)

[WCW+19] Jianghong Wei, Xiaofeng Chen, Jianfeng Wang, Xuexian Hu, and Jianfeng Ma. Forward-Secure Puncturable Identity-Based Encryption for Securing Cloud Emails. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11736 LNCS, pages 134–150, 2019. (Cited on page 14.)

[Xag15] Keita Xagawa. Improved (Hierarchical) Inner-Product Encryption from Lattices. *Full version of the paper appeared at PKC'13*, pages 235–252, 2015. (Cited on pages 14, 15, 16, 17, 133, 135, and 136.)

[YFDL04] Danfeng Yao, Nelly Fazio, Yevgeniy Dodis, and Anna Lysyanskaya. ID-Based Encryption for Complex Hierarchies with Applications to Forward Security and Broadcast Encryption. In *Proceedings of the 11th ACM Conference on Computer and Communications Security*, CCS '04, page 354–363, New York, NY, USA, 2004. Association for Computing Machinery. (Cited on pages 10 and 88.)

[YZ21] Takashi Yamakawa and Mark Zhandry. Classical vs Quantum Random Oracles. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part II*, pages 568–597, Berlin, Heidelberg, 2021. Springer-Verlag. (Cited on page 160.)

[ZC09] Muxin Zhou and Zhenfu Cao. Spatial encryption under simpler assumption. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5848 LNCS:19–31, 2009. (Cited on pages 14 and 15.)