ADDRESSING NEW CHALLENGES IN PUBLIC-KEY CRYPTOGRAPHY

ZUOXIA YU

PhD

The Hong Kong Polytechnic University

2020

The Hong Kong Polytechnic University

Department of Computing

Addressing New Challenges in Public-Key Cryptography

Zuoxia Yu

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

September 2019

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

Signed:

_____

Name: Zuoxia Yu

_____

Abstract

Public-key cryptography, introduced by Diffie and Hellman in 1976, has found numerous applications in reality. After years of development, public-key cryptography has been well-studied and is gradually becoming mature. However, the emerging of several exciting technologies in computer science, while bringing convenience to our daily life, also imposes new challenges to current public-key cryptographic systems deployed in practical applications. In this thesis, we focus on addressing some new challenges in two well-known representatives of public-key cryptography, namely, public-key encryption and digital signature. In particular, we aim at designing an encryption scheme suitable for flexible and efficient data-sharing in the cloud as well as enhancing its resilience against side-channel attacks. In addition, we investigate the real-world applicability of signature scheme in blockchain-based cryptocurrencies and explore how to strengthen the signature component of blockchain-based cryptocurrencies to be quantum safe.

More precisely, we present the following results:

- We present a new variant of public-key encryption named as cross-system proxy re-encryption, which could make data-sharing in the cloud flexible and efficient. It allows one to transform ciphertext of a large class of attribute-based encryption schemes (a variant of public-key encryption that supports fine-grained control over the decryption ability) into ciphertext of any public-key encryption scheme.

- We present a general framework for constructing attribute-based encryption schemes secure against side-channel attacks.

- We present the best possible statistical attack for tracing payers' identity in transactions of privacy-preserving blockchain-based cryptocurrencies based on ring signature. We also find that if some natural conditions are satisfied, our attack will not compromise security of the cryptocurrency. In this way, we identify a safe mode to use public-key cryptographic schemes in a blockchain-based cryptocurrency.

- We present the first lattice-based universal accumulator secure against quantum attacks. This primitive can be used to construct quantum safe dynamic group signature, a widely-used variant of digital signature that can be used to protect signers' identity.

Publications

Conference Paper

1. Rupeng Yang, Man Ho Au, Junzuo Lai, Qiuliang Xu, and **Zuoxia Yu**. Collusion Resistant Watermarking Schemes for Cryptographic Functionalities. In Proceedings of the 25th Annual International Conference on the Theory and Application of Cryptology and Information Security (Asiacrypt 2019), pages 371–398. Springer (2019).

2. Rupeng Yang, Man Ho Au, Zhenfei Zhang, Qiuliang Xu, **Zuoxia Yu**, William Whyte: Efficient Lattice-Based Zero-Knowledge Arguments with Standard Soundness: Construction and Applications. In Proceedings of the 39th Annual International Cryptology Conference (CRYPTO 2019), pages147-175. Springer (2019).

3. **Zuoxia Yu**, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. New Empirical Traceability Analysis of Cryptonote-style Blockchains. In Proceedings of the Financial Cryptography and Data Security 2019 (FC 2019), pages 133–149. Springer (2019).

4. **Zuoxia Yu**, Man Ho Au, Rupeng Yang, Junzuo Lai, Qiuliang Xu: Lattice-Based Universal Accumulator with Nonmembership Arguments. In Proceedings of the 23rd Australasian Conference on Information Security and Privacy (ACISP 2018), pages 502–519. Springer (2018).

5. Rupeng Yang, Man Ho Au, Qiuliang Xu, **Zuoxia Yu**: Decentralized Blacklistable Anonymous Credentials with Reputation. In Proceedings of the 23rd Australasian Conference on Information Security and Privacy (ACISP 2018), pages 720–738. Springer (2018).

6. **Zuoxia Yu**, Man Ho Au, Rupeng Yang, Junzuo Lai, Qiuliang Xu: Achieving Flexibility for ABE with Outsourcing via Proxy Re-Encryption. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS 2018), pages 659–672. ACM (2018).

7. Rupeng Yang, Man Ho Au, Junzuo Lai, Qiuliang Xu, **Zuoxia Yu**: Unforgeable Watermarking Schemes with Public Extraction. In Proceedings of the 11th International Conference on Security and Cryptography for Networks (SCN 2018), pages 63–80. Springer (2018).

8. **Zuoxia Yu**, Man Ho Au, Qiuliang Xu, Rupeng Yang, Jinguang Han: Leakage-Resilient Functional Encryption via Pair Encodings. In Proceedings of the 21st Australasian Conference on Information Security and Privacy (ACISP 2016), pages 443–460. Springer (2016).

Journal Paper

1. Rupeng Yang, Man Ho Au, Qiuliang Xu, **Zuoxia Yu**: Decentralized Blacklistable Anonymous Credentials with Reputation. Computers & Security 85: 353-371. Elsevier (2019).

2. **Zuoxia Yu**, Man Ho Au, Qiuliang Xu, Rupeng Yang, Jinguang Han: Towards Leakage-Resilient Fine-Grained Access Control in Fog Computing. Future Generation Computer Systems, 78:763–777. Elsevier (2018).

3. Rupeng Yang, Qiuliang Xu, Man Ho Au, **Zuoxia Yu**, Hao Wang, Lu Zhou: Position Based Cryptography with Location Privacy: A Step for Fog Computing. Future Generation Computer Systems, 78:799–806. Elsevier (2018).

Book Chapter

1. **Zuoxia Yu**, Man Ho Au, Rupeng Yang: Accountable Anonymous Credentials. Advances in Cyber Security: Principles, Techniques, and Applications, pages 49-68. Springer (2019).

Acknowledgements

Studying at the Hong Kong Polytechnic University is such a valuable and unforgettable experience.

First and foremost, I would like to thank my supervisor, Prof. Man Ho Au, for teaching me how to do research. I appreciate the countless discussions, helpful instructions as well as priceless suggestions, which help me on the right track of research. I have always admired his attitude and determination in discovering and solving problems, and have always tried to learnt from his ability of accurately abstracting the way of understanding the results. I am grateful for the freedom he gave me to explore my own research interests, and the opportunities he offered to participate at numerous conference, winter schools and research seminars. Besides research, his views on life really taught me a lot.

I would like to express my sincere thanks to the cryptography group at the Hong Kong Polytechnic University, including postgraduate students, Xingye Lu, Kang Li, Xiao Yang, Borui Gong, together with all previous and current post-docs, research assistants and project engineers. It is a pleasure to work with all of them, and thank them for making my work joyful, let alone those happy gatherings. I also thank my kind office mates, Jiaxin Chen and Kang Li, for those unexpected happiness in our daily life.

I wish to take this opportunity to express my heartful thanks to my master supervisor, Prof. Qiuliang Xu, who takes me to the field of cryptography. The results of this thesis emerged from the joint works with Prof. Man Ho Au, Prof. Qiuliang Xu, Prof. Junzuo Lai, Dr. Jiangshan Yu, Dr. Rupeng Yang, Wang Fat Lau and Dr. Jinguang Han. I thank them for all the wonderful discussions and for all of their kind help.

I would like to give the sincerest appreciation and gratitude to my dearest parents for their endless love, support, encouragement, and for everything. Without them, I would not be where I am today.

Last, but certainly not least, to my husband, Rupeng, thanks for always being there.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

PKE	Public-Key Encryption
ABE	Attribute-based Encryption
KP-ABE	Key-Policy Attribute-based Encryption
CP-ABE	Ciphertext-Policy Attribute-based Encryption
IBE	Identity-based Encryption
LR-ABE	Leakage-Resilient Attribute-based Encryption
DFA	Deterministic Finite Automata
SE	Spatial Encryption
DSE	Doubly Spatial Encryption
KP-DSE	Key-Policy Doubly Spatial Encryption
HIBE	Hierarchical Identity-Based Encryption
IPE	Inner Product Encryption
FE	Functional Encryption
COM	Commitment Scheme
FDGS	Fully Dynamic Group Signature
GM	Group Manager
TM	Tracing Manager
CH	Chameleon Hash Function
PE	Pair Encoding
LR-PE	Leakage-Resilient Pair Encoding
PMH	Perfectly Master-Key Hiding Property
SMH	Selectively Master-Key Hiding Property
CMH	Co-Selectively Master-Key Hiding Property
ZKAoK	Zero-Knowledge Arguments of Knowledge
NIZK	Non-Interactive Zero-Knowledge System
CML	Continuous Memory Leakage Model
LRC	Leakage-Resilient Cryptography
LR	Leakage-Resilient Property
PRE	Proxy Re-Encryption
CS-PRE	Cross-System Proxy Re-Encryption
RKG	Re-Encryption Key Generator
PKG	Public-Key Generator

RingCT	Ring Confidential Transaction
NM	Non-Membership
Acc	Accumulator
PPT	Probabilistic Polynomial Time
NP	Nondeterministic Polynomial Time
SIS	Short Integer Solution Problem
LWE	Learning With Error Problem
SD	Subgroup Decision Assumption
SD1	Subgroup Decision Assumption 1
SD2	Subgroup Decision Assumption 2
SD3	Subgroup Decision Assumption 3
IND-CPA	Chosen Plaintext Indistinguishability
IND-CCA	Chosen Ciphertext Indistinguishability
CPA	Chosen Plaintext Attack
CCA	Chosen Ciphertext Attack
C.S.	Closed Set

Chapter 1

Introduction

Public-key cryptography, introduced by Diffie and Hellman in [DH76], has been widely employed. Roughly speaking, it refers to the kind of cryptographic systems consisting of a pair of keys, i.e., public key and private key, where public key is disseminated publicly and private key is known only to the owner. Typical public-key cryptographic schemes mainly include public-key encryption (PKE) scheme and digital signature scheme. They almost appear everywhere in our daily life. For example, when you surf on the Internet, entering a link with prefix "https", public-key encryption schemes are used to encrypt all transcripts between you and the server. Another example, which appears recently, is that when you would like to transform some bitcoin to another user, you should use a signature scheme to identify your identity and authorize the validity of the transaction. In this thesis, we focus on some research issues raised recently in the area of public-key encryption and digital signature.

Public-Key Encryption. As a typical representative of public-key cryptography, public-key encryption scheme is introduced to ensure data confidentiality. It allows anyone to send some information to a receiver. Moreover, security of the PKE scheme ensures that no one except the designated receiver could obtain any information from the communication. Traditional PKE scheme only allows *one* designated receiver for each communication and a variant, called attribute-based encryption (ABE), supports multiple receivers. Roughly, in an ABE scheme, each user is assigned to some attributes and each message is encrypted under a policy. The functionality of ABE allows one to obtain the message if his attributes satisfy the policy and its security prevents one from learning any information about the message if his attributes do not satisfy the policy.

One important application of ABE schemes is to provide secure and fine-grained data sharing in the scenario of cloud computing. In a nutshell, cloud

computing allows people to use remote servers, aka “cloud”, to complete their computation or storage tasks. In this way, users of cloud computing services can conduct (heavy) computation tasks or store and share (massive) data in a flexible and economical way. To use an ABE scheme to share data, one first encrypts the data under some “policy”, then he puts the encrypted data on the cloud. Next, any one with the valid attributes is able to retrieve the data. A (usually overlooked) problem in an ABE-based data sharing system is "how to forward the data to others". At first glance, this requirement can be trivially satisfied via letting the user to download the ciphertext, decrypt it and send the plaintext to the target. However, as now people will outsource heavy computation and storage tasks to the cloud, they usually use a light device, e.g., a mobile phone, to deal with their data, which may be too resource-constrained to fully decrypt and store the data. Therefore, a better way to share data is to leave the encrypted data on the cloud, but transform them into a ciphertext under the public-key of the target user. In this way, we need an ABE scheme that allows transformation of its ciphertexts (e.g., from one policy to another) in a public way.

Besides, the development of technology enhances the adversary with new attack means to the security of pubic-key encryption schemes. Among them, side-channel attacks appear as a result of our ability in better controlling the hardware, including timing attacks [Koc96], power attacks [KJJ99], cold-boot attacks [HSH+09], etc, all of which will attempt to steal the internal (secret) state of a computation procedure. Recall that, cryptographic schemes were proved secure under the assumption that secret keys are perfectly protected against attackers. However, via launching side-channel attacks, partial information about the secret key will be revealed to the attacker and thus the previous proof will not be applicable. Even worse, many cryptographic schemes secure in traditional model are indeed broken by side-channel attacks. Hence, to enhance security of existing public-key encryption scheme, we also investigate the construction of ABE schemes secure against side-channel attacks.

Digital Signature. Digital signature scheme is proposed to guarantee the authenticity of a message, where secret key is employed to sign a message and generate a signature, associated public-key is used to verify the validity of a signature on a message. Since its introduction, signature scheme

has found numerous applications in practice. One of its famous applications proposed recently is blockchain-based cryptocurrencies, such as Bitcoin, Ethereum, etc. For instance, in each transaction of Bitcoin, signature is generated by the sender to authorize the validity of that transaction. While in some cases, one may hope that identities of users could also be hidden and to achieve this, the digital signature scheme used in Bitcoin should be replaced with a suitable privacy-preserving primitive, such as signature scheme supporting anonymity. As one of the well-known privacy-preserving primitives, ring signature [RST01, LSW06] is the signature scheme that allows a user to anonymously sign a message on behalf of a group of users. In light of this, Cryptonote protocol as well as those variants based on it, e.g., Monero, Bytecoin, etc., are designed to achieve anonymity through utilizing ring signature. Since ring signature scheme is provably secure, hence Cryptonote protocol as well as its variants, could provide good privacy. However, as shown in some recent works [MSH+18, KFTS17], most transactions in Monero, which use a linkable ring signature scheme to protect the payers' identity, are easy to trace. Hence, a question arises: why provably secure public-key cryptographic schemes fail to protect security of real-world applications and is there a safe mode to use linkable ring signature scheme in Monero?

In addition, the emergence of quantum computer raises new threats to those current cryptographic schemes based on traditional number theoretical assumptions. A quantum computer is referred to the kind of computer which performs computations utilizing the quantum-mechanical phenomena. The study of quantum computing can be dated back to 1980s when Feynman [Fey82] first expressed the idea of simulating quantum mechanism on computers. Since then, researchers have made great efforts in exploring both its existence and its usability. Especially, in 1994, Shor [Sho94] finds that it is able to solve both the factoring problem and the discrete logarithm problem efficiently by using a quantum computer. Besides, there are many breakthroughs towards implementing quantum computers in recent years, and it is estimated that practical quantum computers are likely to appear in a few decades.

It is widely known that modern public-key cryptography is built on the hardness of the factoring problem and that of the discrete logarithm problem. So, if practical quantum computers are realised, most existing cryptographic primitives based on traditional number theoretical assumptions will be totally broken. As a result, it is an urgent task to develop schemes

secure against quantum attacks. In this thesis, we also investigate how to enhance signature component used in cryptocurrency to be quantum safe. Specifically, we will focus on lattice-based cryptography and particularly, we will try to construct efficient privacy-preserving primitives (as just mentioned, they are variants of digital signature schemes) from lattice.

In summary, this thesis seeks to address the following questions:

- How to achieve flexible and efficient data sharing in the cloud.
- How to design secure public-key cryptographic primitives secure against side-channel attacks.
- Why provably secure public-key cryptographic schemes fail to protect security of real-world applications and is there a safe mode to use linkable ring signature scheme in Monero?
- How to enhance signature component used in cryptocurrency to be quantum safe.

We give answers to the foregoing questions. Specifically,

- To support efficient and flexible data sharing over cloud, we formalize the definition of cross-system proxy re-encryption, and propose a construction supporting transform the ciphertext of various ABE schemes to any target public-key encryption schemes.
- On addressing side-channel attacks, we introduce the first generic construction of ABE schemes secure against continuous memory leakage.
- To clarify under what condition privacy-preserving primitives can be safely used to protect users' privacy in a blockchain-based cryptocurrency, we present a new statistical attack for Monero (as an example). We prove that this attack is the best possible attack an adversary can launch (without attacking the underlying cryptographic primitives) and show that if some condition is satisfied, our attack will not compromise users' privacy. In this way, we identify a safe mode to use privacy-preserving primitives in a blockchain-based cryptocurrency.
- On addressing quantum attacks, we propose the first construction of universal accumulator in the lattice setting, which is a basic component for constructing many advanced privacy-preserving primitive, such as fully dynamic group signature.

Next, we elaborate the background and the contribution of each result from Section 1.1 to Section 1.4 and give the related works in Section 1.5.

1.1 Flexible ABE With Outsourcing

Outsourcing the decryption of ABE ciphertext is a promising method to tackle the question about how users can decrypt it efficiently. However, existing solutions require the type of the target ciphertext is determined at the setup phase of the outsourcing scheme. Accordingly, a remaining issue appears, namely, how to make the target cryptosystems (or the clients) to be versatile. Here the problem we wish to tackle is to transform an ABE ciphertext to any client who is using the same, or possibly different, public-key encryption system with the sender. The problem is of practical interest since it is hard to require all clients to use the same PKE, especially in the case of remote and cross-system data sharing. In addition, we also consider whether robust client-side decryption scheme can be adopted. This feature is not supported in the existing ABE with outsourcing.

We introduce cross-system proxy re-encryptions (CS-PRE), which is a new re-encryption paradigm in which a semi-trusted proxy converts a ciphertext of a source cryptosystem (Π_0) into a ciphertext of a target cryptosystem (Π). We formalize CS-PRE and present a construction that performs well in the following aspects. (1)*Versatility*: Π_0 can be any attribute-based encryption (ABE) within Attrapadung's pair encoding framework. Π can be any public-key encryption. Furthermore, the keys and public parameters can be generated independently. (2) *Compatibility*: CS-PRE does not modify the public parameters and keys of Π_0 and Π. Besides, input for the conversion is an ordinary ciphertext of Π_0. (3) *Efficiency*: The computational cost for re-encryption (resp. decryption) of the re-encrypted ciphertext is roughly the same as a decryption in Π_0 (resp. Π).

We prove that the construction is fully secure assuming Π_0 is secure in Attrapadung's framework and Π is IND-CPA secure. Furthermore, it remains secure when there are multiple target cryptosystems. As with other proxy re-encryption, CS-PRE enables flexible sharing of cloud data, as the owner can instruct the cloud server to re-encrypt his ciphertext to those for the intended recipient. In addition, it allows lightweight devices to enjoy access to remote data encrypted under powerful but possibly costly encryption, such as functional encryption, by utilizing the server's power in converting the ciphertext to a simpler encryption, such as RSA-OAEP. Finally,

instances of CS-PRE can be viewed as new proxy re-encryption schemes, such as a PRE supporting ABE for regular language to Hierarchical IBE or Doubly Spatial Encryption to lattice-based encryptions (e.g. NTRUCCA).

1.2 Leakage-Resilient Attribute-Based Encryption

On proposing construction of leakage-resilient cryptographic primitives, we introduce the first adaptively secure attribute-based encryption schemes in continual memory leakage model, which is a model that allows continuous leakage on both user and master secret keys.

The proposed generic framework for constructing leakage-resilient fully secure ABEs (LR-ABEs) results from leakage-resilient pair encoding, which is an extension of pair encoding presented in the recent work of Attrapadung. In this way, our framework simplifies the design and analysis of LR-ABEs into the design and analysis of predicate encodings.Moreover, we discover new adaptively secure LR-ABEs, including FE for regular languages, ABE for large universe and ABE with short ciphertext. Above all, leakage-resilient adaptively secure attribute-based encryption schemes can equip fog computing with higher security and fine-grained access control.

1.3 Applicability of Cryptographic Primitives in Blockchain

While the cascade effect attacks [MSH$^+$18] on the untraceability of Monero are circumvented by two approaches. The first one is to increase the minimum ring size of each input, from 3 (version 0.9.0) to 7 in the latest update (version 0.12.0). The second approach is introducing the ring confidential transactions with enhanced privacy guarantee. However, so far, no formal analysis has been conducted on the level of anonymity provided by the new countermeasures in Monero. In addition, since Monero is only an example of leading CryptoNote-style blockchains, the actual privacy guarantee provided by other similar blockchains in the wild remains unknown.

We propose a more sophisticated statistical analysis on CryptoNote-style cryptocurrencies. In particular, we introduce a new attack on the transaction untraceability, which we call *closed set* attack. We prove that our attack is optimal assuming that no additional information is given. In other words, in terms of the result, *closed set* attack is equivalent to the brute-force attack,

which exhausts all possible input choices and removes those that are impossible given the constraints imposed by the mixins of each transaction.

To verify the impact of our attack in reality, we conduct experiments on the top 3 CryptoNote-style cryptocurrencies, namely, Monero, Bytecoin and DigitalNote, according to their market capitalization. Since the computational cost of performing *closed set* attack is prohibitively expensive, we propose an efficient algorithm, called clustering algorithm, to (approximately) implement *closed set* attack. For Monero, out of the total of 23164745 transaction inputs (up to block 1541236), our clustering algorithm can further identify the real payer of 5752 transaction inputs. Indeed, by combining our clustering method with the cascade attack, we are able to identify the real coin spent in 16334967 transaction inputs, and reduce the mixin set (so the anonymity) of 1736530 inputs. Last but not least, through our combined attack, we are able to identify the real coin being spent in 74.25% Bytecoin inputs, and in 91.56% DigitalNote inputs.

In addition, we provide a theoretical analysis on the identified *closed set* attack, i.e., if every input in a CryptoNote-style blockchain has 3 mixins, and all mixins are sampled uniformly from all existing coins, the success rate of this attack is very small (about 2^{-19}). Given that *closed set* attack is equivalent to the best possible statistical attack, our findings provide two key insights. First, the current system configuration of Monero is secure against statistical attacks, as the minimum number of mixin is 6. Second, we identify a new factor in improving anonymity, that is, the number of unspent keys. Our analysis indicates that the number of mixins in an input does not need to be very large, if the percentage of unspent keys is high.

1.4 Quantum Safe Universal Accumulator

On addressing the quantum attacks, we introduce the first universal accumulator in lattice setting. Universal accumulator is a cryptographic primitive that provides a way to accumulate a set of elements into one. For each element accumulated, it can provide a short membership (resp. non-membership) witness to attest the fact that the element has been (resp. has not been) accumulated. When combined with a suitable zero-knowledge proof system, it can be used to construct many privacy-preserving applications. However, existing universal accumulator schemes are usually based on non-standard assumptions, e.g., the Strong RSA assumption and the

Strong Diffie-Hellman assumptions, and are not secure against quantum attacks.

The proposed lattice-based universal accumulator is based on standard lattice-based assumptions. The starting point is the lattice-based accumulator with Merkle-tree structure proposed by Libert et al. [LLNW16]. We present a novel method to generate short witnesses for non-accumulated members in a Merkle-tree, and give the construction of universal accumulator. Besides, we propose the first zero-knowledge arguments for the possession of the nonmembership witness of a value outside the accumulator in the lattice-based setting via the abstract Stern's protocol of Libert et al. [LLNW17]. Moreover, our proposed universal accumulator is useful for the construction of many privacy-preserving cryptographic primitives, such as group signature and anonymous credential.

1.5 Related Work

In this section, we give the related work of the thesis. Specifically, we focus on the development of some cryptographic primitives, including proxy re-encryption, attribute-based encryption and cryptographic accumulator. Moreover, we recall the concept of leakage-resilient cryptography and the existing attacks on Monero. The related works are given below.

Proxy Re-Encryption. Early PRE [BBS98, CH07, MNT10] considers bidirectional delegation, which allows the proxy to transform ciphertexts of the delegator to that of the delegatee and vice versa. This property, however, is not desirable in many scenarios, including our cloud storage settings, since the delegatee may not want to delegate his decryption rights to the delegator. Thus, another line of works [AFGH06, GA07, LV08, SC09] which considers unidirectional delegation has attracted more attentions recently.

A PRE scheme is *multi-hop* if the proxy can perform multiple consecutive re-encryptions on a ciphertext. Examples include [CH07, GA07, MNT10]. Otherwise, it is a *single-hop* PRE [ID03, AFGH06, GA07, LV08, SC09]. At a first glance, multi-hop PRE schemes appear to be more powerful and may even imply the later. However, the chain collusion attack presented by Shao and Cao [SC09] illustrated that a secure multi-hop PRE scheme may not remain secure in the single-hop setting. In practice, either single-hop or multi-hop PRE may be desirable and thus both types of PRE are being studied.

While a variety of PRE schemes have already been proposed, we observe that, as summarized in Figure 1.1, they fail in fulfilling those requirements. Even worse, some of them have some security issues preventing them from being deployed in practice. In particular, they face one or several of the following problems.

- Not Supporting Cross-System. Only few works attempt to construct PRE schemes supporting cross-system, and none of them supports "arbitrary target system".
- Poor Functionality. Many works only focus on ABE schemes with simple functionality, e.g. identity-based encryption (IBE) schemes, and ABE schemes supporting more advanced access control, e.g. doubly spatial encryption (DSE), have not been considered.
- Low Security Level. Some works only achieve a weak security level, namely, selective security.
- Insecurity. Some schemes are broken by particular attacks. For instance, Scheme 1 in [GA07], and schemes in [LAL+14, LAL+15, EMO10] can be compromised by the chain collusion attack presented in [SC09], which is effective for many single-hop PRE schemes. The main idea of this attack is as follows. In a single-hop PRE scheme, let A be the target of the adversary, and B, C be any two users, and consider the delegation from $A \xrightarrow{P_1} B$, $B \xrightarrow{P_2} C$, then the adversary corrupts proxy P_1, P_2 and user C. Next, the adversary attempts to recover the secret key of A from secret keys of parties he just corrupted. Note that this corruption behaviour is allowed in the security model of the single-hop PRE as it will not trivially break the security goal. Many schemes which apply a similar technique applied in designing multi-hop PRE schemes, e.g. those schemes we just mentioned, are vulnerable to this attack.
- Strong Assumptions. Some schemes establish their security in the random oracle model, which is quite controversial [CGH04], and some schemes need to involve a trusted central party to help generate the re-encryption key.

Leakage-Resilient Cryptography. Leakage-resilient cryptography was developed to address side-channel attacks. To model the additional capability in side-channel attacks, an attacker is allowed to submit an efficiently

single-hop						
Scheme	Functionality	Adaptive	Security	Standard Model	User-initiate cross-system	RKG -free
[GA07] Scheme1	IBE	✓	×		NA	✓
[Mat07]	PKE → IBE	NA	CPA	✓	†	✓
	IBE	✓	CPA	✓	NA	
[WWMO10]	IBE	✓	CPA		NA	
	IBE	✓	CCA		NA	
[THJ08]	IBE → IBE	✓	CPA		✓	✓
[DWQ+15]	IBBE→IBE		CPA	✓	✓	✓
[MD09]	ABE→IBE		CPA	✓	†	✓
[LAL+14]	DFA	✓	×	✓	NA	✓
[LAL+15]	ABE	✓	×	✓	NA	✓
[EMO10]	IBE	✓	×		NA	✓
[BGT15]	IPE	✓	CPA	✓	NA	
Ours	$* \to *$	✓	CPA	✓	✓	✓
multi-hop						
Scheme	Functionality	Adaptive	Security	Standard Model	User-initiate cross-system	RKG-free
[GA07] Scheme2	IBE	✓	CPA		NA	✓
[WCW10]	IBE	✓	CCA		NA	✓
[SC12]	IBE	✓	CCA	✓	NA	✓
[CT07]	IBE	✓	CPA	✓	NA	✓
	IBE	✓	×[SC09]	✓	NA	✓
[LCLS09]	ABE		CPA	✓	NA	✓

Table 1.1: Attribute-based PREs.

1. 'RKG' denotes the Re-encryption Key Generator, a trusted third party, might be operated by the same party as PKG. 'NA' means non-applicable. 'User-initiate cross-system' denotes the delegator can generate the re-encryption key only by himself, without the help of RKG or interacting with the delegatee.
2. '×' means that the scheme in that work cannot achieve its claimed security.
3.'†' denotes that scheme is not user-initiate cross-system. Namely, the delegator cannot initiate to generate the re-encryption key.
4. '$* \to *$' means that our construction supports a large class of attribute-based encryptions (e.g. IBE, ABE, DFA, SE, DSE etc.) to any CPA-secure public-key encryption scheme.

computable leakage function f to obtain the output of f on the current secret states of the cryptographic system. Many leakage models, differ in the restrictions imposed on f, have been proposed. Among them, the continual memory leakage (CML) model [BKKV10, DHLAW10] is believed to best describe those real world attacks. In this model, the entire lifetime of a scheme is divided into periods. At the end of each period, the secret state of the scheme is updated. The amount of leakage information in each time period is bounded, but the total amount of leakage during the lifetime of the scheme is unbounded. Since then, many cryptographic primitives have been designed in this model, include signatures [MTVY11], public-key encryption [DLWW11, YXZ+15], identity-based encryption (IBE) [YCZY12, LRW11], attribute-based encryption (ABE) [LRW11], multiparty computation [BGJK12], etc.

Attribute-based Encryption. We recall the notion of ABEs following the terminology of [Att14]. ABE is a new paradigm of public-key encryption that supports fine-grained access control policy. In ABEs, secret keys are associated with attributes $X \in \mathcal{X}$, ciphertexts are associated with attributes $Y \in \mathcal{Y}$, and a secret key can decrypt a ciphertext if and only if $R(X, Y) = 1$, where R is a predicate (access control policy) for attribute sets $\mathcal{X}$ and $\mathcal{Y}$. For example, IBE [Sha84, BF01] can be viewed as a kind of ABEs where R checks for equality. Achieving fully secure ABEs even in the leakage-free setting is difficult, since the attacker can choose the target after obtaining many decryption keys for arbitrary attributes of its choice. In other words, a successful security proof requires the construction of a simulator that can generate decryption keys for arbitrary attributes and at the same time, the ability to obtain information from the challenge ciphertext (by the attacker) should be useful for the simulator. This seems to pose a dilemma: if the simulator can generate all decryption keys, it can generate a key to decrypt the challenge ciphertext by itself and thus the attacker will be of no use. Indeed, many ABEs [BB04a, BB04b, BBG05, GPSW06, SW05] can be proven secure when the attacker has to declare the target, say Y^*, before seeing the public parameters. This avoids the dilemma since the simulator can be constructed in a way that it can only generate decryption keys associated with X as long as $R(X, Y^*) = 0$.

The dilemma was solved in 2009 by the seminal work of Waters [Wat09]. The main idea is to utilize two types of keys and ciphertexts, namely, normal and semi-functional, in the security proof. Both semi-functional keys

and ciphertext behave normally except that a semi-functional key cannot decrypt a semi-functional ciphertext. In the security proof, the simulator can only generate semi-functional keys and ciphertexts. Now the simulator cannot create a key to decrypt the challenge ciphertext itself. The remaining part of the dual system methodology is, roughly speaking, to prove that the behavior of any attacker is the same regardless whether or not it is given normal or semi-functional keys and ciphertext. This strategy, now commonly known as dual system methodology, allows the constructions of many ABEs [LW10, LW11, LOS+10, OT10, AL10, OT12, Lew12].

The usability of dual system methodology in leakage-resilient cryptography is first observed by Lewko et al. [LRW11].The main idea is that in a dual system security proof, the simulator is capable of generating arbitrary decryption keys (in semi-functional form). If it can generate a key, it can naturally allow leakage of these keys. Based on this idea, they are able to introduce leakage-resilient IBE, ABE and HIBE in CML.

Based on the observation that many existing ABEs constructed in the dual system paradigm exhibit very similar properties, Attrapadung [Att14] and Wee [Wee14] independently proposed unifying frameworks, which support modular design and analysis of ABEs. Both frameworks reduce the study of ABEs to the study of a simpler object called a pair encoding in [Att14] or predicate encoding in [Wee14]. Specifically, given encoding scheme P, one could convert P generically into a fully secure ABE following the framework. Recently, there are a number of enhancement to these frameworks, including one that works in prime order groups [Att15, CGW15].

Traceability Analysis of Monero. Recently, two independent and concurrent works [MMLN17, KFTS17][1] demonstrate that Monero transactions may be de-anonymized via statistical analysis. Specifically, they found that most inputs in Monero have very small number of mixins[2] and more than half inputs are paid without having any mixin. Those inputs without mixins can be trivially de-anonymized. Even worse, once a coin payed without mixin is chosen as a mixin in another transaction, the input of this transaction also faces a danger of being de-anonymized. Based on this simple yet

[1] An updated version [MSH+18] of [MMLN17] also appears recently, but both the method and the result for the traceability analysis are similar in these two works, thus we focus on the initial version.

[2] All other decoy coins in the input are called mixins.

vital observation, these two works adopt similar strategies to conduct empirical evaluations, which are based on the so-called “chain-reaction" analysis [MMLN17] or cascade effect [KFTS17]. Roughly speaking, the attacker first finds out all inputs with zero-mixin. As each located input is payed by merely one public-key, the public-key must be the real payer of the input. Since each public-key can only be used once in Monero, it is safe to delete these de-anonymized public-keys in mixins of the remaining inputs. This will lead to new zero-mixin inputs and the attack could be conducted repeatedly. According to the experiment results of [MMLN17, KFTS17], by Feb 2017, nearly 65% of transaction inputs are with zero-mixin, and the cascade effect can render another 22% of inputs traceable, i.e., nearly 87% of all Monero inputs are insecure when considering users' anonymity.

Having witnessed (and predicted) this type of attacks, Monero has proposed a few countermeasures. First, at version 0.9.0 (January 1, 2016), it releases a mandatory requirement that each transaction input should include at least 2 mixins. Subsequently, at version 0.10.0 (September 19, 2016), ring confidential transaction (Ring-CT), which aims at further enhancing privacy of users via hiding the transaction amount, is introduced. An added advantage of employing RingCTs is that all RingCT input must use outputs of RingCTs as its mixins, i.e., no public-key used before version 0.10.0 will be chosen as mixin for a RingCT input. Therefore, neither the chain-reaction attack nor the cascade attack works for RingCTs. Besides, after realizing the effect of the number of mixins, the minimum number of mixins gradually increases from 2 to 6 in version 0.12.0 (March 29, 2018).

Cryptographic Accumulator. First introduced by Benaloh and De Mare [BDM93], cryptographic accumulator provides a way to combine a set of values into one, and simultaneously offers a short witness for a given value which is accumulated. Since its introduction, accumulator has found numerous applications, including time-stamping [BDM93], membership testing [BDM93, LLX07], anonymous credential [AN11, CL02, LLX07, CKS09, LH10], group signature [TX03, LLX07, LLNW16], ring signature [LLNW16], fail-stop signature [BP97], anonymous authentication [DKNS04], anonymous attestation [LLX07], certificate revocation [GTH02], etc. Subsequently, many extensions have been introduced. Among them, Camenisch and Lysyanskaya [CL02] introduce the notion of dynamic accumulator which allows one to dynamically add and delete a value to and from the accumulator in a way that witnesses of existing elements can be updated efficiently. Later, Li

et al. [LLX07] propose universal accumulator which can also provide non-membership proof for an element which is not accumulated.

1.6 Sources and Organization of The Thesis

The contents of this thesis are mainly taken from the following papers, including:

- **Zuoxia Yu**, Man Ho Au, Rupeng Yang, Junzuo Lai, Qiuliang Xu: Achieving Flexibility for ABE with Outsourcing via Proxy Re-Encryption. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS 2018), pages 659–672. ACM (2018).
- **Zuoxia Yu**, Man Ho Au, Qiuliang Xu, Rupeng Yang, Jinguang Han: Towards Leakage-Resilient Fine-Grained Access Control in Fog Computing. Future Generation Computer Systems, 78:763–777. Elsevier (2018).
- **Zuoxia Yu**, Man Ho Au, Jiangshan Yu, Rupeng Yang, Qiuliang Xu, and Wang Fat Lau. New Empirical Traceability Analysis of Cryptonote-style Blockchains. In Proceedings of the Financial Cryptography and Data Security 2019 (FC 2019), pages 133–149. Springer (2019).
- **Zuoxia Yu**, Man Ho Au, Rupeng Yang, Junzuo Lai, Qiuliang Xu: Lattice-Based Universal Accumulator with Nonmembership Arguments. In Proceedings of the 23rd Australasian Conference on Information Security and Privacy (ACISP 2018), pages 502–519. Springer (2018).

The preliminaries of the thesis is given in Chapter 2. Chapters 3, 4, 5, 6 are dedicated to the foregoing contributions respectively. We conclude the thesis in Chapter 7. The appendices are used to show some tedious proofs of some theorems and lemmas claimed in the main body of the thesis.

Chapter 2

Preliminaries

In this chapter, we give the preliminaries that will be used in the following contents. Here, the purpose is to show some background information on the notations and definitions of some cryptographic primitives used latter. In particular, we will introduce assumptions on composite-order groups and on lattice, the syntax and security definition about ABE in standard model as well as in CML model, the definition about cryptographic primitives including pair encoding, zero-knowledge arguments, and universal accumulator. For readers who are familiar with the foregoing topics may skip this chapter.

2.1 Notations and Cryptographic Assumptions

2.1.1 Notations and Conventions

We use $r \xleftarrow{\$} R$ to denote that r is randomly and uniformly picked from a finite set R. $[n]$ is used to denote the integer set $\{1, 2, \cdots, n\}$. For any set R, its size is denoted as $|R|$. Also, for a distribution $\mathcal{D}$, we use $d \leftarrow \mathcal{D}$ to denote sampling d according to $\mathcal{D}$.

For a bit b, we use $\bar{b}$ to denote the negation of b. We write $negl(\cdot)$ to denote a negligible function. Let $\mathcal{R}$ be a binary relation, we use $\mathcal{L}_{\mathcal{R}}$ to denote the language characterized by $\mathcal{R}$.

Throughout this paper, we will use bold lower-case letters (e.g. $\mathbf{v}$) to delegate vectors, and use bold upper-case letters (e.g. $\mathbf{A}$) to delegate matrices. Note that we treat all vectors in this paper as column vectors, and all elements in vectors and matrices are integers unless otherwise stated. $\mathbf{0}^m$ denotes a zero vector with length m, $\mathbf{0}^{m \times n}$ denotes a zero matrix which has m rows and n columns. We use $\cdot$ to denote the dot product operation of vectors, and $*$ to denote the component-wise multiplication of vectors. For

a vector $\mathbf{v}$ of length n, we use $\|\mathbf{v}\|_1$ to denote its 1 norm, and we use $\mathbf{v}[i]$ to denote its ith element where $i \in [0, n-1]$.

Let $\mathbb{G}$ be a group, and $g \in \mathbb{G}$ is a group element. For $g \in \mathbb{G}$ and $\vec{k} = (k_1, k_2, \cdots, k_n) \in Z_N^n$, $g^{\vec{k}}$ denotes $(g^{k_1}, g^{k_2}, \cdots, g^{k_n})$.

Statistical Distance. Let X and Y be two random variables in a finite set S. The statistical distance between X and Y is defined as

$$\Delta(X, Y) = \frac{1}{2}\Sigma_{x \in S}|Pr[X = x] - Pr[Y = x]|.$$

We say that X and Y are ϵ-close if $\Delta(X, Y) \leq \epsilon$.

2.1.2 Composite-Order Bilinear Groups

We review the definition of composite-order bilinear group [BGN05]. Let $(\mathbb{G}, \mathbb{G}_\mathbb{T})$ denote bilinear groups of composite order $N = p_1 p_2 p_3$, where p_1, p_2 and p_3 are distinct primes, with an efficiently computable bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_\mathbb{T}$ satisfying the following properties.

- Non-degenerate. $e(g, h) \neq 1 \in \mathbb{G}_\mathbb{T}$ if $g, h \neq 1 \in \mathbb{G}$.
- Bilinear. $e(g^a, g^b) = e(g, g)^{ab}$ for any $g \in \mathbb{G}$ and $a, b \in Z_N$.

We use $\mathcal{G}(1^\lambda) \to (\mathbb{G}, \mathbb{G}_\mathbb{T}, e, N, p_1, p_2, p_3)$ to denote a bilinear group generator, where 1^λ is a security parameter. We recall some properties of a composite-order bilinear group. If $p|N$, there exists a subgroup $\mathbb{G}_p$ of $\mathbb{G}$ with order p. Also, for any $g \in \mathbb{G}_{p_i}$ and $h \in \mathbb{G}_{p_j}$, if $p_i \neq p_j$, then $e(g, h) = 1$. The later is sometimes referred to as orthogonality.

2.1.3 Assumptions

Subgroup Decision Assumptions (SD). We first re-cap some complexity assumptions related to composite-order group, which are presented in [Wat09, LW10]. Each of the SD assumptions starts with $\mathcal{G}(1^\lambda) \to (\mathbb{G}, \mathbb{G}_\mathbb{T}, e, N, p_1, p_2, p_3)$.

SD1: Given $D = (g_1 \xleftarrow{\$} \mathbb{G}_{p_1}, g_3 \xleftarrow{\$} \mathbb{G}_{p_3})$ and $T \in \mathbb{G}$, decides if $T = T_1 \xleftarrow{\$} \mathbb{G}_{p_1 p_2}$ or $T = T_2 \xleftarrow{\$} \mathbb{G}_{p_1}$.

SD2: Given $D = (g_1 \xleftarrow{\$} \mathbb{G}_{p_1}, Z_1 Z_2 \xleftarrow{\$} \mathbb{G}_{p_1 p_2}, g_3 \xleftarrow{\$} \mathbb{G}_{p_3}, W_2 W_3 \xleftarrow{\$} \mathbb{G}_{p_2 p_3})$ and $T \in \mathbb{G}$, decides if $T = T_1 \xleftarrow{\$} \mathbb{G}_{p_1 p_2 p_3}$ or $T = T_2 \xleftarrow{\$} \mathbb{G}_{p_1 p_3}$.

SD3: Given $D = (g_1 \xleftarrow{\$} \mathbb{G}_{p_1}, g_2 \xleftarrow{\$} \mathbb{G}_{p_2}, g_3 \xleftarrow{\$} \mathbb{G}_{p_3}, g_1^{\alpha} Y_2, g_1^{s} W_2)$ and $T \in \mathbb{G}$, decides if $T = T_1 = e(g_1, g_1)^{\alpha s}$ or $T = T_2 \xleftarrow{\$} \mathbb{G}_T$, where $W_2, Y_2 \xleftarrow{\$} \mathbb{G}_{p_2}$ and $\alpha, s \xleftarrow{\$} Z_N$.

The advantage of adversary $\mathcal{A}$ for problem instance SD_i is defined as follows:

$$Adv_{\mathcal{A}}^{SD_i}(1^{\lambda}) = |Pr[\mathcal{A}(D, T_1)] - Pr[\mathcal{A}(D, T_2)]|.$$

Then assumptions **SD1**, **SD2** and **SD3** for $\mathcal{G}$ assert that $Adv_{\mathcal{A}}^{SD_i}(1^{\lambda})$ is negligible for all probabilistic polynomial time adversary $\mathcal{A}$.

Lattice-Based Assumption. Here we review a hard problem in lattice-based setting, i.e., the short integer solution (SIS) problem.

Definition 2.1.1 (SIS [GPV08])**.** *The* $\mathsf{SIS}^{\infty}_{n,m,q,\beta}$ *problem is defined as follows: given uniformly random matrix* $\mathbf{A} \in Z_q^{n \times m}$*, find a non-zero vector* $\mathbf{Ax} \in Z^m$ *such that* $||\mathbf{x}||_{\infty} \leq \beta$ *and* $\mathbf{A} \cdot \mathbf{x} = 0 \bmod q$*.*

If $m, \beta = poly(n)$, and $q \geq \beta \cdot \tilde{\mathcal{O}}(\sqrt{n})$, then $\mathsf{SIS}^{\infty}_{n,m,q,\beta}$ problem is at least as hard as the worst-case lattice problem SIVP_{γ} for some $\gamma = \beta \cdot \tilde{\mathcal{O}}(\sqrt{nm})$ [GPV08]. In particular, the $\mathsf{SIS}^{\infty}_{n,m,q,1}$ problem is at least as hard as $\mathsf{SIVP}_{\tilde{\mathcal{O}}(n)}$, when $\beta = 1$, $q = \tilde{\mathcal{O}}(n)$, $m = 2n\lceil \log q \rceil$ [LLNW16].

Chernoff bound. We will need the Chernoff bound in our analysis. There are various forms of the Chernoff bound, here we use the one from [Goe15].

Lemma 2.1.1 (Chernoff Bounds)**.** *Let* $X = \sum_{i=1}^{n} X_i$*, where* $X_i = 1$ *with probability* p_i *and* $X_i = 0$ *with probability* $1 - p_i$*, and all* X_i *are independent. Let* $\mu = \mathbb{E}(X) = \sum_{i=1}^{n} p_i$*. Then*

$$\Pr[X \geq (1+\delta)\mu] \leq e^{-\frac{\delta^2}{2+\delta}\mu} \text{ for all } \delta > 0;$$

$$\Pr[X \leq (1-\delta)\mu] \leq e^{-\frac{\delta^2}{2}\mu} \text{ for all } 0 < \delta < 1.$$

2.2 Attribute-Based Encryptions

We review the definition of attribute-based encryptions, a generalisation of public-key encryption. We first recall the notion of predicate family in [Att14] which is used to describe the access control policy in attribute-based encryption. A **predicate family** $\mathcal{R} = \{R_k\}_{k \in N^c}$ for some constant $c \in N$, where

$R_k : \mathcal{X}_k \times \mathcal{Y}_k \to \{0,1\}$ is a predicate function that maps a pair of attributes (one in key space $\mathcal{X}_k$, the other one in ciphertext space $\mathcal{Y}_k$) to $\{0,1\}$. k is the index which specifies a description of predicate $R_k \in \mathcal{R}$. We require that the first entry of k specifies the domain of predicate function R_k, and write simply R_N to denote R_k whose domain is Z_N. R_N is *domain-transferable* if for p that divides N, then there exists projection maps $f_1 : \mathcal{X}_N \to \mathcal{X}_p$, $f_2 : \mathcal{Y}_N \to \mathcal{Y}_p$, such that for all $X \in \mathcal{X}_N$, $Y \in \mathcal{Y}_N$,

- Completeness: if $R_N(X,Y) = 1$, then $R_p(f_1(X), f_2(Y)) = 1$.
- Soundness:
  1. if $R_N(X,Y) = 0$, then $R_p(f_1(X), f_2(Y)) = 0$.
  2. if 1 does not hold, then there exists an algorithm $\mathcal{F}$ takes as input (X,Y), and outputs a non-trivial factor a, where $p|a$ and $a|N$.

An attribute-based encryption for predicate family $\mathcal{R}$ consists of four algorithms, namely, $(\texttt{Setup}, \texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$, defined below.

$\texttt{Setup}(1^\lambda, k) \to (\mathsf{PK}, \mathsf{MSK})$. This algorithm accepts a security parameter and an index k of predicate family $\mathcal{R}$, outputs the public parameter PK and the master secret key MSK. In the following, we assume the predicate for this ABE is $R : \mathcal{X} \times \mathcal{Y} \to \{0,1\}$.

$\texttt{KeyGen}(\mathsf{MSK}, \mathsf{PK}, X) \to SK$. On input master secret key MSK, public parameter PK and key attribute X chosen from predicate space $\mathcal{X}$, this algorithm outputs secret key SK for key attribute X.

$\texttt{Enc}(Y, M, \mathsf{PK}) \to CT$. On input a ciphertext attribute Y chosen from predicate space $\mathcal{Y}$, message M and public parameter PK, this algorithm outputs ciphertext CT of M for attribute Y.

$\texttt{Dec}(SK, CT) \to M$. On input secret key SK with attribute X and ciphertext CT with attribute Y, outputs a message M or $\perp$.

Correctness. For all valid index k, all $X \in \mathcal{X}_\mathsf{k}$, $Y \in \mathcal{Y}_\mathsf{k}$ such that $R_k(X,Y) = 1$ and all valid message M, the following equation holds:

$$\texttt{Dec}(\texttt{KeyGen}(\mathsf{MSK}, \mathsf{PK}, X), \texttt{Enc}(Y, M, \mathsf{PK})) = M, \tag{2.1}$$

where $(\mathsf{PK}, \mathsf{MSK})$ is the output of $\texttt{Setup}(1^\lambda, k)$.

Adaptive Security. For any stateful adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, we define advantage of $\mathcal{A}$ as follows:

$$Adv_{\mathcal{A}}^{ABE}(\lambda) = Pr\left[b = b' : \begin{array}{l} \textbf{Setup}:(\mathsf{MSK},\mathsf{PK}) \leftarrow \textbf{Setup}(\lambda,k); \\ \textbf{Phase 1}:(st,Y,M_0,M_1) \leftarrow \mathcal{A}_1^{\textbf{KeyGen}(\mathsf{MSK},\cdot)}(\mathsf{PK}); \\ \textbf{Challenge}:\begin{cases} b \xleftarrow{\$} \{0,1\}; \\ CT \leftarrow \textbf{Encrypt}(Y,M_b,\mathsf{PK}); \end{cases} \\ \textbf{Phase 2}:\mathcal{A}_2^{\textbf{KeyGen}(\mathsf{MSK},\cdot)}(CT,st); \\ \textbf{Guess}:b' \leftarrow \mathcal{A}_2; \end{array} \right] - \frac{1}{2},$$

with the restriction that all key queries X that $\mathcal{A}$ made to $\textbf{KeyGen}(\mathsf{MSK}, X)$ satisfies $R_k(X, Y) = 0$. An attribute-based encryption scheme is **adaptively secure** if for all PPT adversary $\mathcal{A}$, $Adv_{\mathcal{A}}^{ABE}(\lambda)$ is negligible.

Remark 2.2.1. *We note that the above definition also covers ordinary public-key encryption when* `KeyGen` *outputs* MSK *and any predicate R in the family $\mathcal{R}$ is of the form $\{1\} \times \{1\} \to \{1\}$. In this case, encryption is only with respect to public key* PK *and decryption requires the corresponding* MSK*. Note that since $R(X, Y) = 1$ for all X, Y, no key query is allowed.*

2.3 Secure ABE In CML Model

2.3.1 Security Definition

An attribute-based encryption scheme is adaptively secure in the continual memory leakage model [LRW11] if there is no PPT adversary $\mathcal{A}$ whose advantage is non-negligible in the following game.
$\textbf{Game}_{real}(\ell_{msk}, \ell_{sk})$:

1. **Setup Phase:** Challenger $\mathcal{C}$ runs $\textbf{Setup}(\lambda) \to (\mathsf{PK}, \mathsf{MSK})$, gives PK to $\mathcal{A}$, and sets $\mathcal{Q} = \varnothing$ and $\mathcal{T} = \{(0, \varepsilon, \mathsf{MSK}, 0)\}$, where $\mathcal{Q}$ denotes a subset of predicate attribute $\mathcal{X}$, and $\mathcal{T}$ denotes a set of tuples $(\mathcal{H}, \mathcal{X}, (\mathsf{MSK}$ or $SK), N)$, where $\mathcal{H}$ is the handles set, N denotes the number of leakage bit of key in this tuple.

2. **Phase 1:** In this phase, $\mathcal{A}$ can make the following queries, namely, create query, leakage query and reveal query.

   - **Create query:** $\mathcal{A}$ output a create query $\textbf{Create}(h, X)$, where h is the handle of a tuple in $\mathcal{T}$ whose key must be a master key, X can either be a predicate attribute or be the empty string ε. $\mathcal{C}$ initially

scans $\mathcal{T}$ to check whether the tuple with handle h is of the form $(h, \varepsilon, \mathsf{MSK}', L)$. If this tuple does not exist or is not in this form, $\mathcal{C}$ returns $\perp$ to $\mathcal{A}$. Otherwise, $\mathcal{C}$ runs **KeyGen**$(X, \mathsf{MSK}', \mathsf{PK}) \rightarrow K$ and adds tuple $(H+1, X, K, 0)$ to $\mathcal{T}$. K can be either a master key or a user key. $\mathcal{C}$ also needs to update the current handel $H \leftarrow H+1$.

- **Leakage query**: $\mathcal{A}$ makes a leakage query **Leak**(h, f) about a key with a handle h with a polynomial time computable function f with constant output. $\mathcal{C}$ first scans $\mathcal{T}$ to find this specified tuple with handle h, which is either with the form (h, X, SK_X, L) or $(h, \varepsilon, \mathsf{MSK}', L)$.
  - If $\mathcal{A}$ makes a **user key leakage query** for tuple (h, X, SK_X, L). $\mathcal{C}$ first checks whether $f(SK_X) + L \leq \ell_{sk}$. If yes, return $f(SK_X)$ to $\mathcal{A}$ and update L in this tuple with $L + f(SK_X)$. Else, return $\perp$ to $\mathcal{A}$.
  - If $\mathcal{A}$ makes a **master key leakage query** for tuple $(h, \varepsilon, \mathsf{MSK}', L)$, $\mathcal{C}$ first checks whether $f(\mathsf{MSK}') + L \leq \ell_{msk}$. If yes, return $f(\mathsf{MSK}')$ to $\mathcal{A}$ and update L in this tuple with $L + f(\mathsf{MSK}')$. Else, return $\perp$ to $\mathcal{A}$.
- **Reveal Query** $\mathcal{A}$ makes a reveal query **Reveal**(h). $\mathcal{C}$ scans $\mathcal{T}$ to find the corresponding tuple. If this handle refers to a master key, $\mathcal{C}$ returns $\perp$. Else, $\mathcal{C}$ returns the user secret key in tuple (h, X, SK, L) to adversary and add X to the set $\mathcal{Q}$.

3. **Challenge Phase:** $\mathcal{A}$ outputs two challenge messages M_0 and M_1 for a challenge attribute Y^* with the restriction that for all $X \in \mathcal{Q}$, $R_N(X, Y^*) \neq 1$. If no, return $\perp$, else $\mathcal{C}$ picks b $\xleftarrow{*} \{0, 1\}$, compute **Encrypt**$(Y^*, M_b, \mathsf{PK}) \rightarrow CT$ and returns CT to $\mathcal{A}$.

4. **Phase 2:** $\mathcal{A}$ can only make create query and reveal query for attribute X' with the restriction that $R_N(X', Y^*) \neq 1$. $\mathcal{C}$ return the corresponding result to $\mathcal{A}$.

5. **Guess Phase:** $\mathcal{A}$ output $b' \in \{0, 1\}$. If $b' = b$, then $\mathcal{A}$ successes.

The advantage of $\mathcal{A}$ in **Game**$_{real}$ is defined as following: **Adv**$_A(\lambda) = |Pr[b' = b] - \frac{1}{2}|$.

Remarks: Note that this security model is equivalent to the widely accepted continual leakage model if keys can be updated properly and no leakage

is allowed during the update process. We refer the reader to [LRW11] for more details.

2.3.2 A Lemma for Leakage-Resilient Analysis

The analysis of leakage-resilient property of our pair encoding scheme relies on the following lemma from [LRW11]. Let $\Delta(X_1, X_2)$ denote the statistical distance of two random variables.

Lemma 2.3.1. *Let $m \in N$, $m \geqslant 3$, and p be a prime. Let $\boldsymbol{\delta} \xleftarrow{\$} Z_p^m$, $\boldsymbol{\tau} \xleftarrow{\$} Z_p^m$, and $\boldsymbol{\tau}'$ be chosen uniformly and randomly from the set of vectors in Z_p^m which are orthogonal to $\boldsymbol{\delta}$ under dot product modulo p. Let $f : Z_p^m \to W$ be some function. Then:*

$$\Delta((\boldsymbol{\delta}, f(\boldsymbol{\tau})), (\boldsymbol{\delta}, f(\boldsymbol{\tau}'))) \leq \varepsilon,$$

as long as

$$|W| \leq 4 \cdot (1 - \frac{1}{p}) \cdot p^{m-2} \cdot \varepsilon^2.$$

2.4 The Pair Encoding Framework

In [Att14], Attrapadung introduced the pair encoding framework which states that an ABE scheme can be described by a simpler primitive called pair encoding which we review here. A Pair encoding scheme PE for predicate family $\mathcal{R} = \{R_k : \mathcal{X}_k \times \mathcal{Y}_k \to \{0,1\}\}_{k \in N^c}$ is a tuple of four deterministic algorithms (`Param`, `Enc1`, `Enc2`, `Pair`) which are given below.

`Param`$(\mathtt{k}) \to n$. This algorithm takes as input security parameter `k` and outputs the length of the public parameter $\vec{h}$ used in `Enc1` and `Enc2`.

`Enc1`$(X, N) \to (\vec{k}; m_2)$. On input attribute $X \in \mathcal{X}$, $N \in \mathbb{N}$, this algorithm outputs a sequence of polynomials $\{k_\ell\}_{\ell \in [m_1]}$, each polynomial k_ℓ is a linear polynomial in $\{\alpha\} \bigcup \{h_i\}_{i \in [n]} \bigcup \{r_j\}_{j \in [m_2]}$, where $\{r_j\}_{j \in [m_2]}$ is the randomness used in `Enc1`, and α is a variable.

`Enc2`$(Y, N) \to (\vec{c}; w_2)$. On input attribute $Y \in \mathcal{Y}$, $N \in \mathbb{N}$, this algorithm outputs a sequence of polynomials $\{c_\ell\}_{\ell \in [w_1]}$, each polynomial c_ℓ is a linear polynomial in $\{h_i\}_{i \in [n]} \bigcup \{s, s_j\}_{j \in [w_2]}$, where $\vec{s} = (s, s_1, s_2, \ldots, s_{w_2})$ is the randomness used in `Enc2`.

`Pair`$(X, Y, N) \to \vec{E}$. Take X, Y, N as input, output matrix $\vec{E} \in Z_N^{m_1 \times w_1}$.

Correctness. For any $N \in \mathbb{N}$, $X \in \mathcal{X}$ and $Y \in \mathcal{Y}$, $(\vec{k}, m_2) \leftarrow \texttt{Enc1}\ (X, N)$, $(\vec{c}; w_2) \leftarrow \texttt{Enc2}\ (Y, N)$, $\vec{E} \leftarrow \texttt{Pair}\ (X, Y, N)$, if $R(X, Y) = 1$,

$$\vec{k}\vec{E}\vec{c} = \alpha s.$$

Within this framework, an ABE scheme Π_0 = (Setup, KeyGen, Enc, Dec) for predicate family $\mathcal{R}$ can then be specified by a pair encoding scheme PE = (Param, Enc1, Enc2, Pair) for the same predicate family.

$\texttt{Setup}(1^{\breve{}}, \mathsf{k}) \rightarrow (\mathsf{PK}, \mathsf{MSK})$. Run $(\mathbb{G}, \mathbb{G}_T, e, N, p_1, p_2, p_3) \leftarrow \mathcal{G}(1^\lambda)$. Pick generators $g_1 \xleftarrow{\$} \mathbb{G}_{p_1}$, $g_3 \xleftarrow{\$} \mathbb{G}_{p_3}$. Then run $n \leftarrow \texttt{Param}(\mathsf{k})$, pick $\vec{h} \xleftarrow{\$} Z_N^n$, $\alpha \xleftarrow{\$} Z_N$. The public parameter $\mathsf{PK} = (g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\vec{h}})$, the master secret key is $\mathsf{MSK} = \alpha$.

$\texttt{KeyGen}(\mathsf{MSK}, \mathsf{PK}, X) \rightarrow SK_X$. Run $(\vec{k}; m_2) \leftarrow \texttt{Enc1}(X, N)$, choose $\vec{r} \xleftarrow{\$} Z_N^{m_2}$, $R_3 \xleftarrow{\$} \mathbb{G}_{p_3}^{m_1}$, compute

$$\mathsf{SK}_X = g_1^{\vec{k}(\alpha, \vec{h}, \vec{r})} \cdot R_3. \tag{2.2}$$

$\texttt{Enc}(\mathsf{PK}, Y, M) \rightarrow CT = (C_0, \vec{C}_1)$. Run $(\vec{c}; w_2) \leftarrow \texttt{Enc2}(Y, N)$, choose $\vec{s} = \{s, s_1, s_2, \ldots, s_{w_2}\} \xleftarrow{\$} Z_N^{w_2+1}$.

$$C_0 = M \times e(g_1, g_1)^{\alpha s} \qquad \vec{C}_1 = g_1^{\vec{c}(\vec{s}, \vec{h})}. \tag{2.3}$$

$\texttt{Dec}(SK_X, CT) \rightarrow M$. Obtain X and Y from SK_X and CT. If $R(X, Y) = 1$, run $\texttt{Pair}(X, Y) \rightarrow \vec{E}$. Then compute $e(SK^{\vec{E}}, \vec{C}_1) \rightarrow e(g_1, g_1)^{\alpha s}$ and obtain $C_0 / e(g_1, g_1)^{\alpha s} \rightarrow M$.

We also recall the security requirements of a pair encoding scheme, including perfectly master-key hiding (PMH), selectively master-key hiding security (SMH) and co-selectively master-key hiding security (CMH). As stated in [Att14], both SHM and CMH are computational and the later, but not the former, is implied by PMH.

Perfectly Master-Key Hiding Security. Let PE be a pair encoding scheme. For $N \in \mathbb{N}$, if $R(X, Y) = 0$, let $n \leftarrow \texttt{Param}(\mathsf{k})$, $(\vec{k}; m_2) \leftarrow \texttt{Enc1}(X, N)$, $(\vec{c}; w_2) \leftarrow \texttt{Enc2}(Y, N)$. If the following two distributions are identical, we say PE is perfectly master-key hiding (PMH).

$$\{c(\vec{s}, \vec{h}), \vec{k}(0, \vec{h}, \vec{r})\} \qquad \{c(\vec{s}, \vec{h}), \vec{k}(\alpha, \vec{h}, \vec{r})\},$$

where the probability is taken over $\vec{h} \xleftarrow{\$} Z_N^{n_1}, \alpha \xleftarrow{\$} Z_N, \vec{r} \xleftarrow{\$} Z_N^{m_3}, \vec{s} \xleftarrow{\$} Z_N^{w_2+1}$.
Computational Security (CMH, SMH). We recall the $\vec{Exp}_{\mathcal{G},b,\mathcal{A},T}(1^\lambda)$ to encompass these two security notions by using different types of Oracles ($\mathcal{O}^T = (\mathcal{O}^{T,1}, \mathcal{O}^{T,2})$, $T \in \{\text{SMH, CMH}\}$) for adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows: $\vec{Exp}_{\mathcal{G},b,\mathcal{A},T}(1^\lambda)$:

1. $(\mathbb{G}, \mathbb{G}_T, e, N, p_1, p_2, p_3) \leftarrow \mathcal{G}(1^\lambda)$;
2. $g_1 \xleftarrow{\$} \mathbb{G}_{p_1}, g_2 \xleftarrow{\$} \mathbb{G}_{p_2}, g_3 \xleftarrow{\$} \mathbb{G}_{p_3}$;
3. $\alpha \xleftarrow{\$} Z_N, n \leftarrow \texttt{Param}(\texttt{k}), \vec{h} \xleftarrow{\$} Z_N^n$;
4. $st \leftarrow A_1^{\mathcal{O}^{T,1}_{b,\alpha,\vec{h}}(\cdot)}(g_1, g_2, g_3)$;
5. $b' \leftarrow A_2^{\mathcal{O}^{T,2}_{b,\alpha,\vec{h}}(\cdot)}(st)$.

The advantage of $\mathcal{A}$ in the corresponding security game $\vec{Exp}_{\mathcal{G},b,\mathcal{A},T}(1^\lambda)$ is defined as follows:

$$Adv^T_{\mathcal{A}}(1^\lambda) = |Pr[\vec{Exp}_{\mathcal{G},0,\mathcal{A},T}(1^\lambda)] = 1| - |Pr[\vec{Exp}_{\mathcal{G},1,\mathcal{A},T}(1^\lambda)] = 1|.$$

The different types of Oracle $\mathcal{O}^T$ are defined as follows:

- **Selective Security (SMH).** $O^{SMH,1}$ can be queried once only while $O^{SMH,2}$ can be queried polynomial times.
  - $O^{SMH,1}_{b,\alpha,\vec{h}}(Y)$: Run $(\vec{c}; w_2) \leftarrow \texttt{Enc2}(Y, p_2)$, choose $\vec{s} \xleftarrow{\$} Z_{p_2}^{w_2+1}$, return $\vec{C} \leftarrow g_2^{\vec{c}(\vec{s},\vec{h})}$.
  - $O^{SMH,2}_{b,\alpha,\vec{h}}(X)$: If $R_{p_2}(X, Y) = 1$ return $\perp$. Run $(\vec{k};\ m_2) \leftarrow \texttt{Enc1}(X, p_2)$, choose $\vec{r} \xleftarrow{\$} Z_{p_2}^{m_2}$, return $\vec{k} \leftarrow \begin{cases} g_2^{\vec{k}(\alpha,\vec{h},\vec{r})} & b = 0 \\ g_2^{\vec{k}(0,\vec{h},\vec{r})} & b = 1 \end{cases}$.
- **Co-selective Security (CMH).** Both $O^{CMH,1}$ and $O^{CMH,2}$ can be queried once only.
  - $O^{CMH,1}_{b,\alpha,\vec{h}}(X)$: If X is empty, return $\perp$. Otherwise, run $(\vec{k}; m_2) \leftarrow \texttt{Enc1}(X, p_2)$, choose $\vec{r} \xleftarrow{\$} Z_{p_2}^{m_2}$, return $\vec{k} \leftarrow \begin{cases} g_2^{\vec{k}(\alpha,\vec{h},\vec{r})} & b = 0 \\ g_2^{\vec{k}(0,\vec{h},\vec{r})} & b = 1 \end{cases}$.
  - $O^{CMH,2}_{b,\alpha,\vec{h},\vec{x}}(Y)$: If $R_{p_2}(X, Y) = 1$, then return $\perp$. Otherwise, run $(\vec{c}; w_2) \leftarrow \texttt{Enc2}(Y, p_2)$, choose $\vec{s} \xleftarrow{\$} Z_{p_2}^{w_2+1}$, return $\vec{C} \leftarrow g_2^{\vec{c}(\vec{s},\vec{h})}$.

In this paper, we say that ABE Π is within the pair encoding framework if it can be written as a pair encoding scheme PE and is constructed following the above syntax. As a side note, [Att14] states that Π is adaptively secure if PE satisfies PMH (or CMH and SMH) under assumptions **SD1**, **SD2**, **SD3**.

2.5 CryptoNote-Style Cryptocurrencies

As a decentralized peer-to-peer network, cryptocurrency utilizes an append-only public ledger to record each user's account balance, which is actually a mapping between the user's public-key and an amount of currency. This public ledger is called as blockchain. Whenever an amount of currency is spent, a user broadcasts a digitally signed message called as transaction to the network. Only those transactions validated successfully can be grouped into a block by a miner, and this new block will be appended to the blockchain if it can be decided by the consensus protocol participated by all miners, which is also called as proof-of-work protocol. Since all information in the blockchain is publicly accessible, hence the double spending checking of a coin is easier under this condition. Specifically, each transaction consists of several inputs and outputs, where input consumes coins from the sender and output transfers coins to the receiver, while also maintains the total balance of coins. Besides, each transaction input should be an unspent previous transaction output.

CryptoNote Protocol. CryptoNote protocol [VS13] aims at providing a privacy enhanced cryptocurrency, with the following two properties:

- Untraceability: for any transaction, the real-spend should be anonymous among all the sets of outputs in an input;
- Unlinkability: it is impossible to prove that any two transactions were sent to a same user.

To achieve unlinkability, for each output in a transaction, CryptoNote uses a one-time random public-key as the destination, which is derived from receiver's public-key and sender's random data. In this way, only the receiver who holds the permanent secret key can redeem that output. Additionally, since each output can be uniquely identified by a public-key, we interchangeably use the term output and public-key throughout this paper. For the untraceability, CryptoNote adopts the ring signature, a cryptographic primitive that allows a user to anonymously sign a transaction

on behalf of a group of users, which is usually referred as a ring. Therefore, the real-spend will be hidden via the help of other outputs, which are called mixins. Obviously, for an input with n public-keys, the number of its mixin is n-1. An abstract structure of a general CryptoNote transaction is illustrated in **Figure** 2.1. We use $tx.in$ to denote an input of a transaction

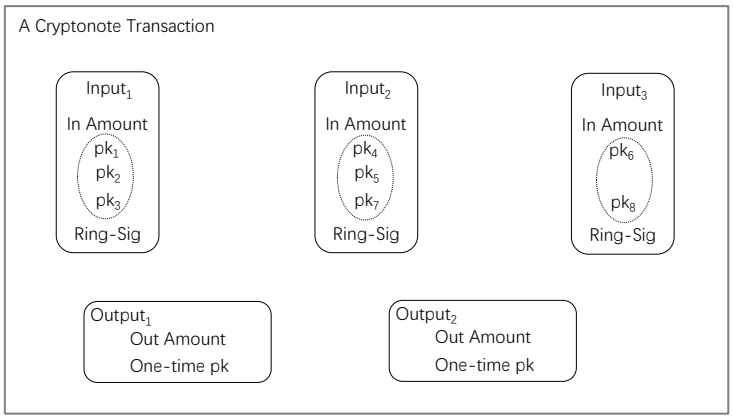


Figure 2.1: A General CryptoNote transaction.The transaction in this figure has three inputs and two outputs. Each input is composed of three parts, the input amount, a ring composed of several input keys, i.e., the one-time output keys of previous transactions, and a ring signature. While each output of the transaction is only composed of an output amount and a new one-time output key. In some variations of CryptoNote protocol, the amount of each input and output are hidden for stronger privacy, such as the ring confidential transaction defined in Monero.

tx, which is a set of public-keys $\{pk_1, pk_2, \ldots, pk_\ell\}$ used to create a ring signature. We also interchangeably call each input $tx.in$ of a transaction as a ring R throughout this paper. Specifically, we use $R = \{pk_1, pk_2, \ldots, pk_n\}$ to denote the transaction input including public-keys $pk_1, pk_2, \ldots, pk_n$.

2.6 Universal Accumulator

Universal accumulator is first proposed in [LLX07] and formalized by [DHS15]. Here we recall the scheme without trapdoor, and use $\mathsf{type} \in \{0, 1\}$ to indicate the given witness is a membership ($\mathsf{type} = 0$) or nonmembership ($\mathsf{type} = 1$) witness. The syntax of a universal accumulator is given below:

$\mathsf{Setup}(n) \rightarrow pp$. On input a security parameter n, this algorithm outputs the public parameter pp.

$\mathsf{Acc}_{pp}(R) \rightarrow \mathbf{u}$. On input an accumulated set $R = \{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{N-1}\}$ with size N, the algorithm outputs the accumulator value $\mathbf{u}$.

$\mathsf{Witness}_{pp}(\mathbf{d}, R, \mathsf{type}) \rightarrow w$ or $\perp$. The algorithm outputs a type of witness w for $\mathbf{d}$ according to the value of type. It outputs $\perp$ if $\mathbf{d} \notin R \wedge \mathsf{type} = 0$ or $\mathbf{d} \in R \wedge \mathsf{type} = 1$.

$\mathsf{Verify}_{pp}(\mathbf{d}, \mathbf{u}, w, \mathsf{type}) \rightarrow 0$ or 1. The algorithm outputs 1 if one of the following two cases happen:

1. type = 0 and w is a witness for $\mathbf{d} \in R$;
2. type = 1, and w is a witness for $\mathbf{d} \notin R$.

Otherwise, output 0.

Correctness. The correctness requires that for all $pp \leftarrow \mathsf{NM\text{-}Setup}(n)$, the following equations hold:

1. for all $\mathbf{d} \in R$, $\mathsf{Verify}_{pp}(\mathbf{d}, \mathsf{Acc}_{pp}(R), \mathsf{Witness}_{pp}(\mathbf{d}, R, 0), 0) = 1$;
2. for all $\mathbf{d} \notin R$, $\mathsf{Verify}_{pp}(\mathbf{d}, \mathsf{Acc}_{pp}(R), \mathsf{Witness}_{pp}(\mathbf{d}, R, 1), 1) = 1$.

Security Definition. A universal accumulator scheme defined above is secure if for all probabilistic polynomial-time adversary $\mathcal{A}$, the following equation holds:

$$Pr\left[\begin{array}{l} pp \leftarrow \mathsf{NM\text{-}Setup}(n); (R, \mathbf{d}^*, \mathbf{w}^*, \mathsf{type}) \leftarrow \mathcal{A}(pp) : \\ \quad \mathbf{d}^* \in R \wedge \mathsf{Verify}_{pp}(\mathbf{d}^*, \mathsf{Acc}_{pp}(R), \mathbf{w}^*, \mathsf{type} = 1) = 1 \\ \qquad\qquad\qquad or \\ \quad \mathbf{d}^* \notin R \wedge \mathsf{Verify}_{pp}(\mathbf{d}^*, \mathsf{Acc}_{pp}(R), \mathbf{w}^*, \mathsf{type} = 0) = 1 \end{array}\right] = negl(n),$$

where $negl(n)$ is a negligible function about n. In other words, the security of a universal accumulator requires that it is computationally infeasible to prove that a value d^* is not accumulated in the value $\mathbf{u}$ if it is or a value d^* is accumulated in the value $\mathbf{u}$ if it is not.

2.6.1 Accumulator For Nonmembership

Observe that a universal accumulator concerns two types of witness, one is the witness for membership and another is the witness for nonmembership, where the first part is the original definition of accumulator. We refer the

reader to **Definition1** in [DHS15] for the formal definition of accumulator (for membership). For the part about nonmembership, we separate the scheme for it as follows:

An accumulator for nonmembership is consisted of a tuple algorithms (NM-Setup, NM-Acc, NM-Witness, NM-Verify) given below:

NM-Setup$(n) \rightarrow pp$. This algorithm is used to generate the public parameter pp on the input security parameter n.

NM-Acc$_{pp}(R) \rightarrow \mathbf{u}$. On input a set $R = \{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{N-1}\}$ with size N, the algorithm outputs the accumulator value $\mathbf{u}$.

NM-Witness$_{pp}(\mathbf{d}, R) \rightarrow w$. For a set R and a value $\mathbf{d}$, if $\mathbf{d} \in R$, then outputs $\perp$. Otherwise, outputs a witness w for the fact that $\mathbf{d}$ is not accumulated in the output of NM-Acc$_{pp}(R)$.

NM-Verify$_{pp}(\mathbf{u}, \mathbf{d}, w) \rightarrow \{0, 1\}$. The algorithm outputs 1 if witness w can prove that $\mathbf{d}$ is not accumulated into $\mathbf{u}$. Otherwise, outputs 0.

Correctness. The correctness requires that for all $pp \leftarrow \texttt{Setup}(n)$, the following equation holds for all $\mathbf{d} \notin R$:

$$\text{NM-Verify}_{pp}(\text{NM-Acc}_{pp}(R), \mathbf{d}, \text{NM-Witness}_{pp}(\mathbf{d}, R)) = 1.$$

Security Definition. An accumulator for non-membership is secure if for all probabilistic polynomial-time adversary $\mathcal{A}$,

$$Pr[pp \leftarrow \texttt{Setup}(n); (L, d^*, \mathbf{w}^*) \leftarrow \mathcal{A}(pp) : d^* \in L \wedge \text{NM-Verify}_{pp}(\text{NM-Acc}_{pp}(L), d^*, \mathbf{w}^*) = 1] = negl(n),$$

where $negl(n)$ is a negligible function about n. In other words, the security says that it is computationally infeasible to prove that a value d^* is not accumulated in the value $\mathbf{u}$ if it is.

It is obviously that if we run the algorithms of accumulator and accumulator for nonmembership independently, then the combination of these two parts can give a universal accumulator. More precisely, let (M-Setup, M-Acc, M-Witness, M-Verify) be an accumulator scheme, and (Setup, NM-Acc, NM-Witness, NM-Verify) be an accumulator for nonmembership scheme, then a universal accumulator scheme (Setup, Acc,Witness,Verify) can be constructed as follows:

$\mathsf{Setup}(n)$. Run $pp_m \leftarrow \mathsf{M\text{-}Setup}(n)$, $pp_{nm} \leftarrow \mathtt{Setup}(n)$. Output $pp = (pp_m, pp_{nm})$.

$\mathsf{Acc}_{pp}(R)$. Run $\mathbf{u}_m \leftarrow \mathsf{M\text{-}Acc}_{pp_m}(R)$, $\mathbf{u}_{nm} \leftarrow \mathsf{NM\text{-}Acc}_{pp_{nm}}(R)$. Return $(\mathbf{u}_m, \mathbf{u}_{nm})$.

$\mathsf{Witness}_{pp}(\mathbf{d}, R, \mathsf{type})$. If $\mathsf{type} = 0$, run $w_m \leftarrow \mathsf{M\text{-}Witness}_{pp_m}(\mathbf{d}, R)$, and return w_m. Otherwise, run $w_{nm} \leftarrow \mathsf{NM\text{-}Witness}_{pp_{nm}}(\mathbf{d}, R)$, and return the output.

$\mathsf{Verify}_{pp}(\mathbf{d}, \mathbf{u}, w, \mathsf{type})$. If $\mathsf{type} = 0$, then recall $\mathsf{M\text{-}Verify}_{pp_m}(\mathbf{u}, \mathbf{d}, w)$, and return the output. Otherwise, run $\mathsf{NM\text{-}Verify}_{pp_{nm}}(\mathbf{u}, \mathbf{d}, w)$ and return the output.

Both the correctness and the security can be reduced to underlying primitives (accumulator and accumulator for nonmembership) straightforwardly, and we just omit the details here.

2.7 Zero-Knowledge Arguments of Knowledge

Zero-knowledge arguments of knowledge [GMR89] (ZKAoK) is an interactive protocol where a prover can convince the verifier that he possesses the witness for a statement in a NP relation without revealing any information about the witness. Moreover, we require it to have the following security properties [GMR89]:

Completeness. The prover can convince the verifier if he knows a witness testifying to the truth of the statement.

Soundness. A malicious prover cannot convince the verifier if the statement is false.

Zero-knowledge. A malicious verifier can know nothing but the statement is true from the proof.

Extractability. A probabilistic polynomial time extractor can extract the witness for a true statement from a convincing argument made by the prover.

In addition, as mentioned in [FS86], also known as Fiat-Shamir heuristic, a three round public-coin interactive ZKAoK can be transformed into a non-interactive one in the random oracle model. We refer the reader to [BR93] for the security analysis Fiat-Shamir heuristic.

2.7.1 Abstract Stern's Protocol

Here we recall the abstract Stern's protocol proposed by Libert et al. in [LLNW17], which is a type of ZKAoK system capturing the following relations. For all $i \in [n]$, assume n_i and $d_i \geq n_i$ be positive integers. For public matrices $\{\mathbf{P}_i \in Z_{q_i}^{n_i \times d_i}\}_{i\in[n]}$ and vectors $\mathbf{v}_i \in Z_{q_i}^{n_i}$, the prover argues the possession of mutually related integer vectors $\{\mathbf{x}_i \in \{-1,0,1\}^{d_i}\}_{i\in[n]}$ in a zero-knowledge manner, such that :

$$\forall i \in [1,n] : \mathbf{P}_i \cdot \mathbf{x}_i = \mathbf{v}_i \bmod q_i.$$

Let $d = d_1 + d_2 + \ldots + d_n$, and $\mathbf{x} = (\mathbf{x}_1||\mathbf{x}_2||\ldots||\mathbf{x}_n)$. Assume VALID is a subset of $\{-1,0,1\}^d$, and S be a finite set such that one can associate every $\pi \in S$ with a permutation T_π of d elements which satisfies the conditions (2.4), then we can get **Lemma** 2.7.1.

$$\begin{cases} \mathbf{x} \in \mathsf{VALID} \Longleftrightarrow T_\pi(\mathbf{x}) \in \mathsf{VALID}; \\ \textit{If } \mathbf{x} \in \mathsf{VALID} \textit{ and } \pi \textit{ is uniform in } S, \textit{ then } T_\pi(\mathbf{x}) \textit{ is uniform in } \mathsf{VALID}. \end{cases} \tag{2.4}$$

Lemma 2.7.1 (Theorem 1 in [LLNW17])**.** *The constructed abstract Stern's protocol shown in Figure 2.2 is a statistical* ZKAoK *with perfect completeness, soundness error* $2/3$*, and communication cost* $\mathcal{O}(\Sigma_{i=1}^{n} d_i \cdot \log q_i)$*. In particular:*

- *There exists an efficient simulator that, on input* $\{\mathbf{P}_i, \mathbf{v}_i\}_{i\in[1,n]}$*, outputs an accepted transcript which is statistically close to that produced by the real prover.*
- *There exists an efficient knowledge extractor that, on input a commitment CMT and 3 valid response (*RSP_1*,*RSP_2*, RSP*$_3$*) to all 3 possible values of the challenger Ch, outputs* $\mathbf{x}' = (\mathbf{x}'_1, \cdots, \mathbf{x}'_n) \in \mathsf{VALID}$ *such that* $\mathbf{P}_i \cdot \mathbf{x}'_i = \mathbf{v}_i \bmod q_i$ *for all* $i \in [1,n]$.

Therefore, to employ the abstract Stern's protocol to prove a statement, one needs to first transform the statement into the form of $\mathbf{P}_i \cdot \mathbf{x}_i = \mathbf{v}_i \bmod q_i$ with a specifically designed witness set VALID, then specify the set S and permutations of d elements $\{T_\pi, \pi \in S\}$ which can make conditions (2.4) hold. In this way, a ZKAoK can be constructed via the framework of abstract Stern's protocol.

1. **Commitment:** Prover P sample $\pi \xleftarrow{\$} S$, $\mathbf{r}_1 \xleftarrow{\$} Z_{q_1}^{d_1}$, ..., $\mathbf{r}_n \xleftarrow{\$} Z_{q_n}^{d_n}$, and computes $\mathbf{r} = (\mathbf{r}_1 || \ldots || \mathbf{r}_n)$, $\mathbf{z} = \mathbf{x} \boxplus \mathbf{r}$. Then P samples ρ_1, ρ_2, ρ_3 for commitment COM, then computes and sends CMT = $\{C_1, C_2, C_3\}$ to verifier V, where

$$C_1 = \mathsf{COM}(\pi, \{\mathbf{P}_i \cdot \mathbf{r}_i \bmod q_i\}_{i\in[1,n]}; \rho_1)$$
$$C_2 = \mathsf{COM}(T_\pi(\mathbf{r}); \rho_2)$$
$$C_3 = \mathsf{COM}(T_\pi(\mathbf{z}); \rho_3).$$

2. **Challenge:** Verifier V picks a uniformly random challenge $Ch \xleftarrow{\$} \{1, 2, 3\}$, and sends it to P.

3. **Response:** According to the Ch, P reveals different commitments via sending RSP in the following way:

   - $Ch = 1$: let $t_x = T_\pi(\mathbf{x})$, $t_r = T_\pi(\mathbf{r})$, RSP = $(t_x, t_r, \rho_2, \rho_3)$.
   - $Ch = 2$: let $\pi_2 = \pi$, $\mathbf{w} = \mathbf{z}$, RSP = $(\pi_2, \mathbf{w}, \rho_1, \rho_3)$.
   - $Ch = 3$: let $\pi_3 = \pi$, RSP = $(\pi_3, \mathbf{r}, \rho_1, \rho_2)$.

Verification: Once receiving RSP, verifier V checks as follows:

- $Ch = 1$: check that t_x is VALID, C_2 = $\mathsf{COM}(t_r; \rho_2)$, C_3 = $\mathsf{COM}(t_x \boxplus t_r; \rho_3)$.
- $Ch = 2$: parse $\mathbf{w} = (\mathbf{w}_1 || \ldots || \mathbf{w}_n)$, where $\mathbf{w}_i \in Z_{q_i}^{d_i}$ for all $i \in [1, n]$, then check that C_1 = $\mathsf{COM}(\pi_2, \{\mathbf{P}_i \cdot \mathbf{w}_i - \mathbf{v}_i \bmod q_i\}_{i\in[1,n]}; \rho_1)$, and C_3 = $\mathsf{COM}(T_{\pi_2}(\mathbf{w}); \rho_3)$.
- $Ch = 3$: parse $\mathbf{r} = (\mathbf{r}_1 || \ldots || \mathbf{r}_n)$, then check that C_1 = $\mathsf{COM}(\pi_3, \{\mathbf{P}_i \cdot \mathbf{r}_i \bmod q_i\}_{i\in[1,n]}; \rho_1)$, and C_2= $\mathsf{COM}(T_{\pi_3}(\mathbf{r}); \rho_2)$.

In each case, V outputs 1 if and only if all conditions hold.

Figure 2.2: Abstract Stern's protocol[LLNW17]. COM is a string commitment scheme proposed in [KTX08] with statistically hiding and computationally binding properties. '$\boxplus$' is the modular addition operator, such that $\mathbf{z} = ((\mathbf{x}_1 \boxplus \mathbf{r}_1) \bmod q_1) || \ldots || (\mathbf{x}_n \boxplus \mathbf{r}_n) \bmod q_n))$.

Chapter 3

Cross-System Proxy Re-Encryption

Outsourcing the decryption of ABE is introduced by Green et al. [GHW+11] to provide an efficient way to perform the decryption of the ciphertext of ABE. Specifically, they proposed a mechanism which allows a proxy to transform ABE ciphertext to a short Elgamal ciphertext which supports efficient decryption. However, in the aforementioned ABE with outsourcing, the type of the target ciphertext is determined at the setup of the outsourcing scheme. This leads to several problems.

- Firstly, the mechanism is not friendly for data sharing. For example, Alice is a staff at a university. While the university often provides daily information such as the notification of the Christmas party. Alice wants to forward them to her friend, but she does not want to complete the encryption operation on her mobile phone and hopes to outsource this task to her server on the original encrypted email. However, current outsourcing schemes cannot transform ciphertext into the one encrypted under other's public key.
- Secondly, the mobile phone may be infected with malware or suffered from side channel attacks. Part of the secret key embedded inside the phone may be stolen or tampered with. To guarantee data confidentiality, a robust client side decryption scheme should be used. Ideally, the target cryptosystem should offer resistance against side channel and related-key attacks. However, current outsourcing schemes do not support this.

To solve these problems, we revisit the task of ABE with outsourcing and observe that proxy re-encryption (PRE) may help. The concept of PRE was introduced by Blaze et.al. in [BBS98] to support delegation of decryption rights from one party to another with the help of a semi-trusted proxy holding a re-encryption key. In this way, ciphertexts of the delegator can be

transformed to ciphertexts of the delegatee by a proxy without the involvement of the former. Security of the PRE schemes requires that neither the proxy nor the delegatee alone could learn any information from ciphertexts of the delegator. This primitive has extensive pivotal applications, especially for efficient data storing and sharing [AFGH06, YWRL10, LWW14] in cloud computing and social networks [JMB11]. Especially, it was deployed by a company AtCipher in their product to enable secure cloud storage and data sharing.

Now, with the help of PRE schemes, we can transform the source ciphertext into the target ciphertext of a specific type of encryption scheme. For example, in order to access data encrypted under ABE schemes, Alice could send a re-encryption key to the cloud, then ask the cloud to transform the ABE ciphertext to a lightweight one, say, RSA ciphertext when needed; or ask the cloud to transform the ciphertext into the one encrypted under Bob's public key. Unfortunately, current existing constructions of PRE schemes cannot satisfy our requirement. Especially, there is no PRE scheme supporting flexible target scheme previously.

Chapter Organization. We start by an overview about our contribution and technique is given in **Section** 3.1. The formal definition of CS-PRE is presented in **Section** 3.2, and the construction of CS-PRE is shown in **Section** 3.3. Finally, the chapter ends with a discussion about the hardness of constructing CS-PRE in prime-order groups in **Section** 3.4.

3.1 Our Contribution in Constructing CS-PRE

We introduce and formalize the notion of cross-system proxy re-encryption (CS-PRE) which supports the transformation of the ciphertexts from the source cryptosystem to those in the target cryptosystem. Our construction of CS-PRE is versatile, supporting a large class of attribute-based encryptions for the source cryptosystem and does not impose any restriction on the target cryptosystem except that it must be IND-CPA secure. We prove that our proposal is fully secure in the standard model under static assumptions. The architecture of our system is shown in Figure 3.1.

The Techniques. The basic idea of our construction is similar to that in [GA07, LAL+15]. However, to support the cross-system property and to avoid the chain collusion attack, we encrypt the randomness used for hiding

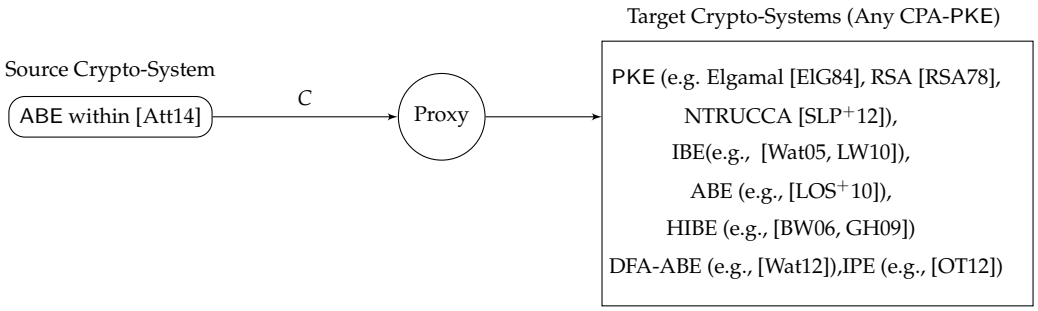


Figure 3.1: Workflow of our construction.The rectangle on the right means the set of all attribute-based schemes. ABE within [Att14] means those ABE schemes within the framework presented in [Att14]. PKE means the set of the original public-key encryption schemes. DFA-ABE means deterministic finite automata-based ABE for regular language.

the secret key of the original encryption scheme with another CPA-secure encryption scheme. To prove the security of the scheme, we devise an alternative proof sequence for re-encryption key queries for attributes matching the challenge ciphertext. Looking ahead, we make use of the subgroup decision assumptions and the IND-CPA security of the target encryption scheme to hide those keys computationally.

Two Application Scenarios. Here we also propose two applications of our cross-system proxy re-encryption schemes.

- For the case of efficiency, our constructed CS-PRE can be used to transform the ABE ciphertext into the ciphertexts of some lightweight public-key encryption schemes suitable for the portable devices. This case is very similar to outsourcing the decryption of ABE ciphertext. The architecture for this case is illustrated in Figure 3.2, where a cloud user may want to ask the cloud to help transform the ABE ciphertext put in the cloud to a lightweight public-key ciphertext.

- For the consideration of enhanced functionality, our constructed scheme can also be used to transform the ABE ciphertext into many different ciphertexts. The targeted ciphertexts can be of the same type of public-key encryption scheme but encrypted under different public keys, or of different types of public-key encryption schemes. This kind of transformation is done via the help of the different re-encryption keys. The architecture for this case is illustrated in Figure 3.3.

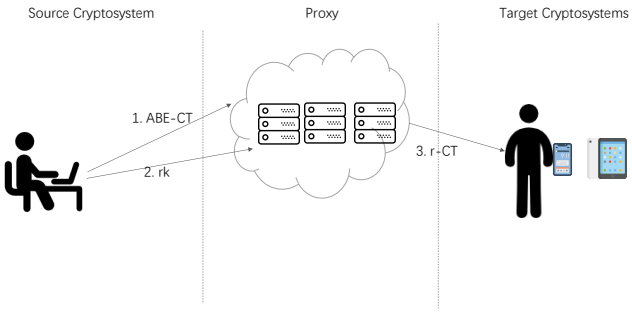


Figure 3.2: Use Case 1 of our CS-PRE.
"ABE-CT" denotes the ABE ciphertext, "rk" is the abbreviation for re-encryption key, and "r-CT" denotes the re-encryption ciphertext.

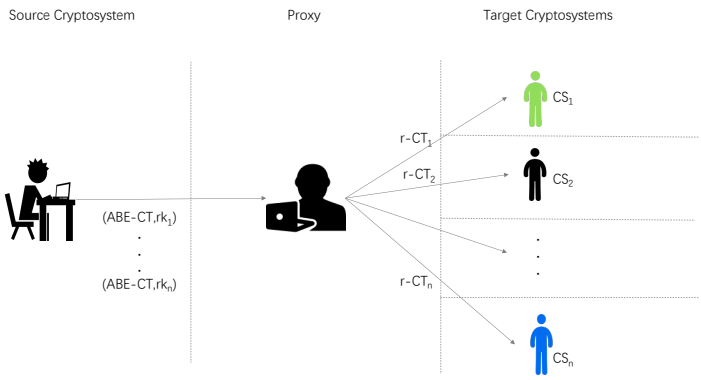


Figure 3.3: Use Case 2 of our CS-PRE.
"ABE-CT" denotes the ABE ciphertext, "rk" is the abbreviation for re-encryption key, and "r-CT" denotes the re-encryption ciphertext. "CS" is the short for cryptosystem.

3.2 Syntax of CS-PRE

Recall that in REMARK 2.2.1, we analyze the definition of ABE scheme can cover the definition of PKE. Hence, in this section, we directly consider the transformation between two ABE schemes. Consider two attribute-based encryption schemes, namely, Π_0 and Π_1. For notational convenience, we use Π_b.PK to denote the public parameter of an instance of Π_b for $b \in \{0,1\}$. In other words, $\Pi_b.\mathtt{KeyGen}(1^\lambda) \rightarrow (\Pi_b.\mathsf{PK}, \Pi_b.\mathsf{MSK})$. We slightly abuse the notation and use $\Pi_b.SK_X$ instead of $\Pi_b.SK_{\Pi_b.X}$ to denote a decryption key of attribute $\Pi_b.X$ in cryptosystem Π_b. A cross-system proxy re-encryption scheme CS-PRE for Π_0, Π_1 is a tuple of three algorithms (RE-KeyGen, RE-Enc, RE-Dec) whose definition is given below.

RE-KeyGen(Π_0.PK, $\Pi_0.SK_X$, Π_1.PK, $\Pi_1.Y$) $\rightarrow rk_{\Pi_0.X \rightarrow \Pi_1.Y}$. The algorithm outputs the re-encryption key $rk_{\Pi_0.X \rightarrow \Pi_1.Y}$, which can be used to convert any ciphertext whose attribute matches $\Pi_0.X$ in Π_0 to a ciphertext of attribute $\Pi_1.Y$ in Π_1.

RE-Enc(Π_0.PK, $rk_{\Pi_0.X \rightarrow \Pi_1.Y}$, $\Pi_0.CT$)$\rightarrow \Pi_1.CT^{(2)}$, which converts $\Pi_0.CT$ to ciphertext $\Pi_1.CT^{(2)}$ on input a re-encryption key.

RE-Dec(Π_0.PK, Π_1.PK, $\Pi_1.CT^{(2)}$, $\Pi_1.SK$) $\rightarrow M$.

Correctness. For all valid index k and k', all $\Pi_0.X \in \Pi_0.\mathcal{X}_\mathsf{k}$, $\Pi_0.Y \in \Pi_0.\mathcal{Y}_\mathsf{k}$, $\Pi_1.Y \in \Pi_1.\mathcal{Y}_{\mathsf{k}'}$, $\Pi_1.X \in \Pi_1.\mathcal{X}_{\mathsf{k}'}$, if $\Pi_0.R_\mathsf{k}(\Pi_0.X, \Pi_0.Y) = 1$ and $\Pi_1.R_{\mathsf{k}'}$ ($\Pi_1.X$, $\Pi_1.Y$) = 1, then for any ciphertext RE-Enc(Π_0.PK,$rk_{\Pi_0.X \rightarrow \Pi_1.Y}$, $\Pi_0.CT$) $\rightarrow$ $\Pi_1.CT^{(2)}$, the following equation holds,

$$\mathtt{RE\text{-}Dec}(\Pi_0.\mathsf{PK}, \Pi_1.\mathsf{PK}, \Pi_1.CT^{(2)}, \Pi_1.SK_X) = M,$$

where $rk_{\Pi_0.X \rightarrow \Pi_1.Y}$ is the output of $\mathtt{RE\text{-}KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_X, \Pi_1.\mathsf{PK}, \Pi_1.Y)$, and $\Pi_0.CT$ is computed by $\Pi_0.\mathtt{Enc}(\Pi_0.\mathsf{PK}, M, \Pi_0.Y)$.

3.2.1 Security Notion

We present a formal security model for the security requirement of CS-PRE. We allow the adversary to adaptively introduce new target cryptosystem and to obtain arbitrary decryption keys, master secret keys of the target cryptosystems as well as re-encryption keys subject to the constraint that none of which would allow the attacker to trivially win the security game.

Let Π_0 be an ABE scheme. Looking ahead, Π_0 acts as the source cryptosystem in the attack game. Let CS-PRE be the cross-system proxy re-encryption. Firstly, we define four oracles to model the capability of the attacker.

1. $\mathcal{O}_{\texttt{Setup}}(\Pi_i)$: run $\Pi_i.\texttt{Setup}(1^\lambda) \rightarrow (\Pi_i.\mathsf{PK}, \Pi_i.\mathsf{MSK})$, return the public key $\Pi_i.\mathsf{PK}$.

2. $\mathcal{O}_K^0$: This oracle allows the adversary to obtain decryption keys and re-encryption keys of Π_0. This oracle works as follows.

   - For a secret key query on $(\Pi_0.X)$. Run $\Pi_0.\texttt{KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.\mathsf{MSK}, \Pi_0.X) \rightarrow \Pi_0.SK_X$, return $\Pi_0.SK_X$.
   - For a re-encryption key query on $(\Pi_0.X, \Pi_i.Y, \Pi_i)$, run $\Pi_0.\texttt{KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.\mathsf{MSK}, \Pi_0.X) \rightarrow \Pi_0.SK_X$, compute $\texttt{RE-KeyGen}$ $(\Pi_0.\mathsf{PK}, \Pi_0.SK_X, \Pi_i.Y, \Pi_i.\mathsf{PK}) \rightarrow rk_{\Pi_0.X \rightarrow \Pi_i.Y}$, return $rk_{\Pi_0.X \rightarrow \Pi_i.Y}$.

3. $\mathcal{O}_K(\Pi_i, \Pi_i.X)$: run $\Pi_i.\texttt{KeyGen}(\Pi_i.\mathsf{PK}, \Pi_i.\mathsf{MSK}, \Pi_i.X) \rightarrow \Pi_i.SK_X$, return $\Pi_i.SK_X$.

4. $\mathcal{O}_{cor}(i)$: return $\Pi_i.\mathsf{MSK}$ where $i \neq 0$.

The adaptively chosen-plaintext game $\mathbf{Game}_{real}$ is defined as below.

- **SetupPhase** : $\mathcal{C}$ runs $\Pi_0.\texttt{Setup}(1^\lambda, \mathsf{k}) \rightarrow (\Pi_0.\mathsf{PK}, \Pi_0.\mathsf{MSK})$, then returns $\Pi_0.\mathsf{PK}$ to $\mathcal{A}$.
- **Phase1** : $\mathcal{A}$ can arbitrarily issue queries to the oracles defined above.
- **Challenge Phase** : $\mathcal{A}$ outputs two message (M_0, M_1) and a challenge ciphertext attribute $\Pi_0.Y^*$. $\mathcal{C}$ runs $\Pi_0.\texttt{Enc}(\Pi_0.Y^*, M_b, \Pi_0.\mathsf{PK}) \rightarrow \Pi_0.CT^*$, and returns $\Pi_0.CT^*$ to $\mathcal{A}$, where $\{0,1\} \xrightarrow{\$} b$. Otherwise, $\mathcal{C}$ returns $\perp$.
- **Phase2** : As in **Phase1**.
- **GuessPhase** : $\mathcal{A}$ outputs a guess b'. If $b' = b$, $\mathcal{A}$ wins.
- **Restrictions.** To prevent $\mathcal{A}$ from winning trivially, the following restrictions are imposed throughout the game.

  1. For all $\mathcal{O}_K^0(\Pi_0.X)$ queries made, $R_0(\Pi_0.X, \Pi_0.Y^*) = 0$.
  2. For all $\mathcal{O}_K(\Pi_i, \Pi_i.X)$ and $\mathcal{O}_K^0(\Pi_0.X, \Pi_i.Y, \Pi_i)$ queries, $R_0(\Pi_0.X, \Pi_0.Y^*)$ and $R_i(\Pi_i.X, \Pi_i.Y)$ cannot be 1 simultaneously.

3. For all $\mathcal{O}_{cor}(\Pi_i)$ and $\mathcal{O}_K^0(\Pi_0.X,\ \Pi_j.Y,\ \Pi_j)$ queries, if $R_0(\Pi_0.X,\ \Pi_0.Y^*) = 1, i \neq j$.

Informally speaking, $\mathcal{A}$ cannot ask for a key that is capable of decrypting the challenge ciphertext directly (restriction 1). If $\mathcal{A}$ obtains a re-encryption key from X^* (an attribute such that $R(X, Y^*) = 1$) to $\Pi_i.Y'$, it cannot ask for a key that can decrypt ciphertext for $\Pi_i.Y'$ (restriction 2) nor the master secret key for Π_i (restriction 3).

Definition 3.2.1 (Security). *A* CS-PRE *scheme is* ***fully chosen-plaintext secure*** *if for all PPT adversary $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2\}$, the advantage function $Adv_{\mathcal{A}}(1^\lambda)$ in the above game is negligible:*

$$Adv_{\mathcal{A}}(1^\lambda) = Pr\left[b = b' : \begin{array}{l} (\Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}) \leftarrow \Pi_0.\mathtt{Setup}(1^\lambda, k); \\ (st, M_0, M_1, \Pi_0.Y^*) \leftarrow \\ \qquad \mathcal{A}_1^{\mathcal{O}_K^0, \mathcal{O}_K, \mathcal{O}_{\mathtt{Setup}}, \mathcal{O}_{cor}}(\Pi_0.\mathsf{PK}); \\ \begin{cases} b \xleftarrow{\$} \{0,1\}; \\ \Pi_0.CT \leftarrow \Pi_0.\mathtt{Enc}(\Pi_0.Y^*, M_b, \Pi_0.\mathsf{PK}); \end{cases} \\ \mathcal{A}_2^{\mathcal{O}_K^0, \mathcal{O}_K, \mathcal{O}_{\mathtt{Setup}}, \mathcal{O}_{cor}}(\Pi_0.CT, st); \\ b' \leftarrow \mathcal{A}_2; \end{array} \right] - \frac{1}{2}.$$

For clarity, we have omitted restrictions in the above definition of advantage function.

3.3 Our Construction

In this section, we present our construction of CS-PRE for any ABE within the pair encoding framework discussed in Section 2.4 followed by its security analysis. We would like to reiterate that only the source cryptosystem needs to be within pairing encoding framework while the choice of the target cryptosystems does not have this requirement.

3.3.1 Overview of Our Construction

We observe that ABEs in the pair encoding framework [Att14] fit nicely into our construction idea. Informally speaking, the structure of the secret key

and ciphertext are of the form[1]:

$$SK_X = g_1^{\vec{k}(\alpha,\vec{r},\vec{h})} \qquad C_0 = M \times e(g_1,g_1)^{\alpha s} \qquad C_Y = g_1^{\vec{c}(\vec{s},\vec{h})}.$$

If key attribute X matches ciphertext attribute Y, i.e., $R(X,Y) = 1$, then the linear combination of $\vec{k}(\alpha,\vec{r},\vec{h})$ and $\vec{c}(\vec{s},\vec{h})$ can recover the value $e(g_1,g_1)^{\alpha s}$. In other words, there exists a matrix $\vec{E}$ such that $e(SK_X^{\vec{E}}, C_Y) = e(g_1,g_1)^{\alpha s}$, where SK_X and C_Y are both vectors of group elements. Note that this is what we need exactly. For any SK_X and any ciphertext C_0, C_Y, $e((SK_X^{\delta})^{\vec{E}}, C_Y) = e(g_1,g_1)^{\alpha s\delta}$ when $R(X,Y) = 1$.

Suppose $(\mathsf{enc}, \mathsf{dec})$ is the encryption and decryption algorithm of the target recipient, we can construct the re-encryption algorithm for any ABE in the above form as follows.

- Re-encryption key: Randomly pick δ, compute $rk_0 := SK_X^{H(\delta)}$ and $rk_1 := \mathsf{enc}(\delta)$, where H is a hash function that maps δ from the message space of enc to Z_N.
- Re-encryption: On input (C_0, C_Y), compute

  $$F := e(rk_0^E, C_Y) = e(g_1,g_1)^{\alpha s H(\delta)}.$$

  Output (C_0, F, rk_1).
- Re-Decryption: On input (C_0, F, rk_1), compute $\delta := \mathsf{dec}(rk_1)$. Output $C_0/F^{1/H(\delta)}$.

3.3.2 Our Construction of CS-PRE

We present our construction of CS-PRE, which allows the proxy server to convert a ciphertext from an ABE scheme Π_0 within the pair encoding framework to any other scheme Π. Since Π_0 is within the framework, it works in a bilinear group $(\mathbb{G}, \mathbb{G}_T, e, N)$. In the following, we assume $H : \{0,1\}^* \to \mathbb{Z}_N$ is a hash function whose output is uniformly distributed if its input is uniform. Notably, it is not hard to find such kind of hash function, e.g. KDF function in [CS03].

$\mathtt{RE\text{-}KeyGen}(\Pi_0.\mathsf{PK}, \Pi.\mathsf{PK}, \Pi_0.SK_X, \Pi.Y) \to rk_{\Pi_0.X \to \Pi.Y} := (\vec{rk}, \Pi.\vec{C})$.

[1]Note that this is a simplified description where we omit the $\mathbb{G}_{p_3}$-component of the keys and ciphertexts.

$\Pi.Y$ is the ciphertext attribute of Π, $\Pi_0.SK_X$ is a user secret key for attribute $\Pi_0.X$. Run $(\vec{k}; m_2) \leftarrow \texttt{Enc1}(\Pi_0.X, N)$, choose $\vec{r}_2 \xleftarrow{\$} Z_N^{m_2}$, $R_3 \xleftarrow{\$} \mathbb{G}_{p_3}^{m_1}$ and $\delta \xleftarrow{\$} \mathcal{M}_\Pi$.

$$\vec{rk} = (\Pi_0.SK_X \cdot R_3 \cdot g_1^{\vec{k}(0,\vec{h},\vec{r}_2)})^{H(\delta)}. \tag{3.1}$$

Run $\Pi.\texttt{Enc}(\Pi.\mathsf{PK}, \Pi.Y, \delta) \to \Pi.\vec{C}$, return $(\vec{rk}, \Pi.\vec{C})$.

$\texttt{RE-Enc}(\Pi_0.\mathsf{PK}, rk_{\Pi_0.X \to \Pi.Y}, \Pi_0.CT_Z, \Pi_0.X) \to \Pi.CT^{(2)}$. Suppose that $\mathsf{rk}_{\Pi_0.X \to \Pi.Y} = (\vec{rk}, \Pi.\vec{C})$, $\Pi_0.CT_Z = (\Pi_0.\mathcal{C}_{Z,0}, \Pi_0.\vec{C}_{Z,1})$ and R_0 $(\Pi_0.X, \Pi_0.Z) = 1$. Run $\texttt{Pair}(\Pi_0.X, \Pi_0.Z) \to \vec{E}$, compute

$$F = e(\vec{rk}^{\vec{E}}, \Pi_0.\vec{C}_{Z,1}) \tag{3.2}$$

Output $\Pi.CT^{(2)} = (\Pi_0.\mathcal{C}_{Z,0}, F, \Pi.\vec{C})$.

$\texttt{RE-Dec}(\Pi_0.\mathsf{PK}, \Pi.\mathsf{PK}, \Pi.CT^{(2)}, \Pi.SK_W) \to M$. If $R(W, Y) = 1$, then run $\Pi.\texttt{Dec}$ $(\Pi.\mathsf{PK}, \Pi.SK_W, \Pi.\vec{C}) \to \delta$. Return $\Pi_0.\mathcal{C}_{Z,0}/F^{\frac{1}{H(\delta)}}$.

3.3.3 Security Analysis

Assume adversary $\mathcal{A}$ makes q_1 and q_2 queries to $\mathcal{O}_K^0$ in **Phase1** and **Phase2** respectively. Also, let $\mathcal{A}$ issue q queries to $\mathcal{O}_{\texttt{Setup}}$. We have the following theorem regarding the security of our construction of CS-PRE.

Theorem 3.3.1. *Suppose the underlying pair encoding scheme* PE *for* Π_0 *is co-selectively and selectively master key hiding and assumptions* ***SD1***, ***SD2***, ***SD3*** *hold in* $\mathcal{G}$. *Also assume that the target cryptosystems* $\{\Pi_i\}_{i=1}^q$ *are chosen-plaintext secure. Then our construction satisfies Definition 3.2.1. More precisely, for any PPT adversary* $\mathcal{A}$, *there exists adversary* $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_6$ *who run nearly the same time as* $\mathcal{A}$, *such that for any* λ:

$$\begin{aligned}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) \leq\ & 2\boldsymbol{Adv}_{\mathcal{B}_1}^{SD1}(1^\lambda) + (4q_1 + 4q_2 + 1)\boldsymbol{Adv}_{\mathcal{B}_2}^{SD2}(1^\lambda) \\ & + q_1\boldsymbol{Adv}_{\mathcal{B}_3}^{CMH}(1^\lambda) + q_2\boldsymbol{Adv}_{\mathcal{B}_4}^{SMH}(1^\lambda) + \boldsymbol{Adv}_{\mathcal{B}_5}^{SD3}(1^\lambda) \\ & + 2q(q_1 + q_2)\boldsymbol{Adv}_{\mathcal{B}_6}^{\texttt{CPA}}(1^\lambda). \end{aligned} \tag{3.3}$$

Remark 2. The selectively master-key hiding security for PE used in our security proof is weaker than the original definition as mentioned in **Section** 2.4. Here we require that both $O^{SMH,1}$ and $O^{SMH,2}$ can be queried once only.

Semi-Functional Algorithm. The following algorithms will be used in the security proof of our generic construction.

$\texttt{SFSetup}(1^\lambda, \mathsf{k})$. This algorithm is nearly the same as $\texttt{Setup}(1^\lambda, \mathsf{k})$ defined in **Section** 2.4 except that it additionally outputs a generator $g_2 \xleftarrow{\$} \mathbb{G}_{p_2}$, $\hat{\vec{h}} \xleftarrow{\$} Z_N^n$. We call $\hat{\vec{h}}$ the semi-functional parameters.

$\texttt{SFEncrypt}(\Pi_0.\mathcal{Y}, M, \Pi_0.\mathsf{PK}, g_2, \hat{\vec{h}})$. This algorithm first runs $(\vec{c}; w_2) \leftarrow \texttt{Enc2}$ $(\Pi_0.\mathcal{Y}, N)$. Then choose $\vec{s}$= $(s, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$, $\hat{\vec{s}} \xleftarrow{\$} Z_N^{w_2+1}$, output the semi-functional ciphertext $\Pi_0.CT = (C_0, \vec{C}_1)$:

$$C_0 = e(g_1, g_1)^{\alpha s} \times M \qquad \vec{C}_1 = g_1^{\vec{c}(\vec{s},\vec{h})} \cdot g_2^{\vec{c}(\hat{\vec{s}},\hat{\vec{h}})}.$$

$\texttt{SFKeyGen}(\Pi_0.X, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}, g_2, type, \hat{\alpha}, \hat{\vec{h}})$. This algorithm runs $(\vec{k};\ m_2)$ $\leftarrow \texttt{Enc1}$ $(\Pi_0.X,\ N)$, picks $\hat{\vec{r}}$,$\vec{r} \xleftarrow{\$} Z_N^{m_2}$, $\vec{R}_3 \leftarrow Z_N^{m_1}$, then outputs one type of semi-functional secret key depending on the input type $\in \{1, 2, 3\}$.

$$\Pi_0.SK_X = \begin{cases} g_1^{\vec{k}(\alpha,\vec{h},\vec{r})} \cdot g_2^{\vec{k}(0,\hat{\vec{h}},\hat{\vec{r}})} \cdot g_3^{\vec{R}_3} & \text{if} \quad \mathsf{type} = 1 \\ g_1^{\vec{k}(\alpha,\vec{h},\vec{r})} \cdot g_2^{\vec{k}(\hat{\alpha},\hat{\vec{h}},\hat{\vec{r}})} \cdot g_3^{\vec{R}_3} & \text{if} \quad \mathsf{type} = 2 \\ g_1^{\vec{k}(\alpha,\vec{h},\vec{r})} \cdot g_2^{\vec{k}(\hat{\alpha},\vec{0},\vec{0})} \cdot g_3^{\vec{R}_3} & \text{if} \quad \mathsf{type} = 3 \end{cases}$$

$\texttt{SFRE-KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_X, \Pi_i.\mathcal{Y}, \Pi_i.\mathsf{PK}) \rightarrow rk_{\Pi_0.X \rightarrow \Pi_i.\mathcal{Y}}$. Run $(\vec{k}; m_2) \leftarrow \texttt{Enc1}$ $(\Pi_0.X,\ N)$, pick $\delta \xleftarrow{\$} \mathbb{G}_T, \vec{r}_2 \xleftarrow{\$} Z_N^{m_2}$, then compute

$$\vec{rk} = \begin{cases} (\Pi_0.SK_X^{type1} \cdot g_1^{\vec{k}(0,\vec{h},\vec{r}_2)})^{H(\delta)} & \text{if} \quad \mathsf{type} = 1 \\ (\Pi_0.SK_X^{type2} \cdot g_1^{\vec{k}(0,\vec{h},\vec{r}_2)})^{H(\delta)} & \text{if} \quad \mathsf{type} = 2 \\ (\Pi_0.SK_X^{type3} \cdot g_1^{\vec{k}(0,\vec{h},\vec{r}_2)})^{H(\delta)} & \text{if} \quad \mathsf{type} = 3 \end{cases}$$

$$\Pi_i.\vec{C} \leftarrow \Pi_i.\texttt{Enc}(\Pi_i.\mathsf{PK}, \Pi_i.\mathcal{Y}, \delta).$$

Notably, here the type of $\vec{rk}$ depends on the type of input $\Pi_0.SK_X$, and $\Pi_0.SK_X^{typei}$ means SK_X is a type-i secret user key.

Proof Overview. Here we first define a sequence of games.

Game$_{real}$: This is the real game as defined in **Section** 3.2.1.

Game$_{res}$: This game is nearly the same as **Game**$_{real}$, except that all attribute X queried to $\mathcal{O}_K^0$ is $R_{p_2}(\Pi_0.X, \Pi_0.Y^*) = 0$ rather than $R_N(\Pi_0.X, \Pi_0.Y^*)$ = 0.

Game$_0$: Run $\texttt{SFSetup}(1^\lambda, k)$ in the setup phase.

Modify $\Pi_0.CT^* \leftarrow \texttt{SFEncrypt}(\Pi_0.Y^*, M_b, \Pi_0.\mathsf{PK}, g_2, \hat{\vec{h}})$.

Game$_k$: For those key queries and re-encryption key queries to $\mathcal{O}_K^0$ made by $\mathcal{A}$, $\mathcal{C}$ answers the j-th query by generating the corresponding secret key as follows.

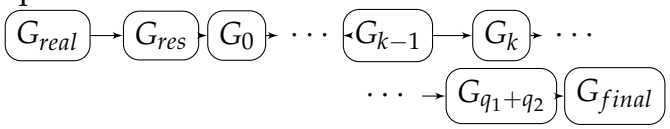
$\hat{\alpha}_j \leftarrow Z_N$,

$SK_{X_j} \leftarrow$

$$\begin{cases} SFKeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}, g_2, 3, \hat{\alpha}_j, \vec{0}, \vec{0}) & j < k \\ SFKeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}, g_2, 3, \hat{\alpha}_j, \vec{0}, \vec{0}) & j = k \\ KeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}) & j > k \end{cases}$$

Game$_{final}$: Modify $M \xleftarrow{\$} \mathcal{M}$, and $\Pi_0.CT^* \leftarrow \texttt{SFEncrypt}(\Pi_0.Y^*, M, \Pi_0.\mathsf{PK}, g_2, \hat{\vec{h}})$.

The proof structure is stated as follows.



Semi-Functional Key Invariance in Our Proof. In fact, the proof structure of our proof is similar as the proof in [Att14],but in our proof the simulator needs to handle the simulation for the re-encryption key query from $\Pi_0.X_k$ to $\Pi_i.Y_k$ in each phase. As a result, the original indistinguishability between **Game**$_{k-1}$ and **Game**$_k$ does not hold if we use the proof techniques as [Att14]. Then, for clarity of reading, we give the proof of the indistinguishability between any two adjacent games in Appendix A.1. In the following, we give the insight of proof of semi-functional key invariance.

As mentioned above, the main difference between our proof and the proof in [Att14] is that the adversary additionally has the ability to make query about re-encryption key from $\Pi_0.X_k$ to $\Pi_i.Y_k$ in each phase. As a result, the indistinguishability between **Game**$_{k-1}$ and **Game**$_k$ may be compromised. To solve this problem, we further divide the k-th query into one of the two cases, depending on whether the key attribute $\Pi_0.X_k$ in the k-th re-encryption key query can match the ciphertext attribute $\Pi_0.Y^*$ in the challenge ciphertext. We prove that in both cases, indistinguishability between **Game**$_{k-1}$ and **Game**$_k$ holds.

- Case 1: R($\Pi_0.X_k$, $\Pi_0.Y^*$) = 0. That is, secret key of attribute $\Pi_0.X_j$ cannot decrypt the challenge ciphertext. Then the security proof in this case is similar as the proof for the fully security of an ABE scheme.

Since the simulator in an ABE scheme can generate all keys which cannot decrypt the challenge ciphertext, it can also simulate the corresponding re-encryption key.

- Case 2: $R(\Pi_0.X_k, \Pi_0.Y^*) = 1$. In this case, secret key of attribute $\Pi_0.X_j$ can decrypt the challenge ciphertext. Additional technique is required to answer a re-encryption key query since simulator of an ABE scheme cannot generate the keys which can decrypt the challenge ciphertext. We leverage the fact that the simulator is not giving away this kind of keys in plain. Rather, it is masked by δ. which is encrypted in the target cryptosystem. The chosen-plaintext security of the target scheme can guarantee that δ is indistinguishable from a randomly and uniformly distributed value. The crucial point in the proof is that we need to prove a re-encryption key in which the secret key can decrypt the challenge ciphertext is indistinguishable from a re-encryption key whose underlying secret key cannot decrypt the challenge ciphertext. In this way, simulator of PREs can simulate this case.

Remark 3.3.1. *We remark that when the target* PKE *scheme* Π *has an enhanced security guarantee, e.g., leakage-resilience or relate-key attack resilience etc., the security of the resulting scheme can also be boosted (i.e. the* CS-PRE *will be resilient to the side-channel attacks or the related-key attacks to some extend).*

Besides, the computation cost of the re-encryption process and re-decryption process of our CS-PRE *schemeare roughly the same as the decryption of the source ciphertext and the decryption of the target ciphertext. The complexity of our proxy re-encryption is determined by the complexity of the decryption algorithms in source and target cryptosystems.*

3.4 Discussion

Compared with composite-order groups, schemes based on prime-order groups are preferable since they are more efficient. Recently, Chen et al. present two types of techniques [CW13, CGW15] for simulating composite-order groups in prime-order groups. Via their methods, many fully secure ABE schemes in prime-order groups ([CW13, CGW15, Att15]) have been presented. However, for the sake of efficiency and conciseness, only those essential properties of composite order groups are simulated. Therefore, their techniques are not applicable to our framework.

To see this, recall that in these techniques, the encoding parts, namely, the key encoding and the ciphertext encoding, are duplicated multiple times to simulate those properties of composite order groups, and in the decryption algorithm, intermediate results for all pairs of key encoding and ciphertext encoding will be summed up to recover the mask of the plaintext. These techniques, however, cannot be directly applied to our construction. Specifically, in our re-encryption algorithm, the re-encryption key is used to partially decrypt the ciphertext and that different intermediate results will be masked with different randomness and therefore cannot be summed up to recover the mask. Hence, we need to seek for new technique to construct attribute-based proxy re-encryption schemes in prime-order groups and leave this as an open problem.

Chapter 4

Leakage-Resilient Attribute-Based Encryption

In this section, we focus on strengthening the security of attribute-based encryption scheme to be secure against side-channel attacks, a new attacks appearing due to our ability in better controlling the hardware on which the cryptographic systems are deployed. In particular, we present the construction of attribute-based encryption schemes in the area of leakage-resilient cryptography, where the additional ability of side-channel attacker is modelled by queries of efficiently computable leakage function on secret states of the scheme. Here we consider the most realistic security model in leakage-resilient cryptography, namely, continual memory leakage model.

Chapter Organization. This chapter is started with an overview of our contribution in constructing leakage-resilient attribute-based encryptions in **Section** 4.1. We introduce the new proposed leakage-resilient pair encoding scheme in **Section** 4.2. The general framework of constructing LR-ABE is given in **Section** 4.3. Furthermore, **Section** 4.4 covers the general transformation to enhance pair encoding with leakage-resilient property. Finally, this chapter ends with the instantiations of our framework in **Section** 4.5.

4.1 Our Contributions in Constructing LR-ABE

In this work, we mainly consider how to construct functional encryption applicable to fog computing. Inspired by the techniques of [LRW11] and the framework of [Att14], we propose a framework for leakage-resilient adaptively secure ABEs. The main component of our framework is the definition of a new primitive that we called leakage-resilient pair encodings. Our framework involves two transformations which are illustrated in Figure 4.1:

- **Transformation from Pair Encodings to Leakage-Resilient Pair Encodings**. First, we define the notion of leakage-resilient pair encodings. Then, we present a transformation technique that turns a large class of pair encodings into leakage-resilient pair encodings.
- **Transformation from Leakage-Resilient Pair Encodings to Leakage-Resilient** ABEs. We show how fully secure ABEs in CML model can be obtained generically from leakage-resilient pair encodings.

Concrete ABEs as a result of our framework will be presented in Section 4.5 in detail. Looking ahead, our framework leads to improved schemes such as LR-IBE with tighter reductions and new schemes such as LR-ABE for regular languages, LR-ABE for large universe and LR-ABE with short-ciphertext, all with adaptive security in the standard model and the CML model.

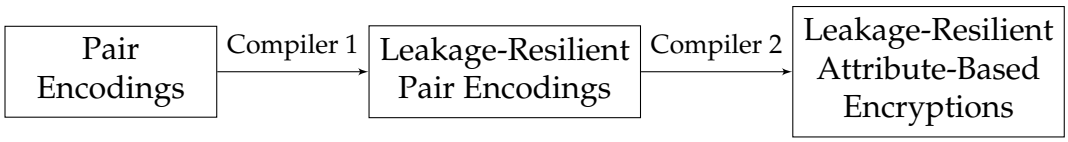


Figure 4.1: Main results of our paper

4.2 Leakage-Resilient Pair Encoding Scheme

We first define a notion known as leakage-resilient pair encoding scheme. Then we present a few definitions regarding its security requirements. Our definitions can be thought of as extending the definition of pair encodings [Att14] to capture leakage-resilience.

4.2.1 Syntax

A leakage-resilient pair encoding scheme for predicate family R is a tuple of four deterministic algorithms, (**Param**, **Enc1**, **Enc2**, **Pair**):

Param(λ) $\rightarrow (n_1, n_2)$. n_1 denotes the number of the public variables $\mathbf{h} = (h_1, h_2, \cdots, h_{n_1})$ used in encoding algorithms, n_2 denotes the number of another public variables $\mathbf{x} = (x_1, x_2, \cdots, x_{n_2})$ used in encoding algorithms.

Enc1$(X, N) \rightarrow$ **(**$\mathbf{k}_1$**,** $\mathbf{k}_2$; m_1, m_2, m_3**)**, where $X \in \mathcal{X}$, N, m_1, m_2 and $m_3 \in \mathbb{N}$. $\mathbf{k}_1$**(**α**,** $\mathbf{h}$**,** $\mathbf{x}$**,** $\mathbf{r}$**)** is a sequence of polynomials $\{k_{1,\ell}\}_{\ell\in[m_1]}$, each polynomial $k_{1,\ell}$ is in $\{\alpha\} \bigcup \{h_j\}_{j\in[n_1]} \bigcup \{x_k\}_{k\in[n_2]} \bigcup \{r_i\}_{i\in[m_3]}$. Meanwhile, $\mathbf{k}_2$**(**$\mathbf{h}$**,** $\mathbf{r}$**)** is a sequence of polynomials $\{k_{2,\ell}\}_{\ell\in[m_2]}$ and each $k_{2,\ell}$ is a polynomial in $\{h_j\}_{j\in[n_1]} \bigcup \{r_i\}_{i\in[m_3]}$. More precisely, this algorithm outputs two sets of coefficients:

$$\{C_\ell, C_{\ell,i}, C_{\ell,i,j}, C_{\ell,i,k}\}_{\ell\in[m_1],i\in[m_3],j\in[n_1],k\in[n_2]}$$

$$\{C'_{\ell',i}, C'_{\ell',i,j}\}_{\ell'\in[m_2],i\in[m_3],j\in[n_1]}$$

$$\mathbf{k}_{1,\ell}(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}) = C_\ell \alpha + (\sum_{i\in[m_3]} C_{\ell,i} \cdot r_i) + (\sum_{i\in[m_3],j\in[n_1]} C_{\ell,i,j} \cdot r_i h_j) + \sum_{\substack{i\in[m_3]\\k\in[n_2]}} C_{\ell,i,k} \cdot r_i x_k. \quad (4.1)$$

$$\mathbf{k}_{2,\ell'}(\mathbf{h}, \mathbf{r}) = \sum_{i\in[m_3]} C'_{\ell',i} \cdot r_i + \sum_{i\in[m_3],j\in[n_1]} C'_{\ell',i,j} \cdot r_i h_j. \quad (4.2)$$

The input attribute X of $Enc1$ could be empty string ε. In this case, algorithm **Enc1** also works in the same way as mentioned above. For any set of predicate attribute $\mathcal{X}$, we require that the outputs of this algorithm **Enc1** satisfy the regeneration property, meaning that for all **Enc1**$(X, N) \rightarrow (\mathbf{k}_1, \mathbf{k}_2)(X \in \mathcal{X})$ and **Enc1**$(\varepsilon, N) \rightarrow (\mathbf{k}_1^\varepsilon, \mathbf{k}_2^\varepsilon)$, the following property hold:

Given ($\mathbf{k}_1^\varepsilon(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}), \mathbf{k}_2^\varepsilon(\mathbf{h}, \mathbf{r})$) and non-empty attribute $X \in \mathcal{X}$, there exists an efficient algorithm that can compute $(\mathbf{k}'_1(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}'), \mathbf{k}'_2(\mathbf{h}, \mathbf{r}'))$ whose distribution is the same as $(\mathbf{k}_1, \mathbf{k}_2)$.

Enc2$(Y, N) \rightarrow (\mathbf{c} = (c_1, c_2, \cdots, c_{w_1}); w_2)$. $c = (c_1, c_2, \cdots, c_{w_1})$ is a set of polynomials output by this encoding scheme with coefficients in Z_N. We require that each c_ι is a polynomial in $\{h_j\}_{j\in[n_1]} \bigcup \{s_i\}_{i\in[w_2]} \bigcup \{x_k\}_{k\in[n_2]} \bigcup \{s\}$.

$$c_\iota(\mathbf{s}, \mathbf{x}, \mathbf{h}) = a_\iota s + \sum_{i\in[w_2]} a_{\iota,i} s_i + \sum_{k\in[n_2]} a'_{\iota,k} x_k s + \sum_{j\in[n_1]} a_{\iota,j} h_j s + \sum_{i\in[w_2],j\in[n_1]} a_{\iota,i,j} h_j s_i, \quad (4.3)$$

where $\mathbf{s} = (s, s_1, s_2, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$.

Pair$(X, Y, N) \rightarrow \mathbf{E}$. Takes as input X, Y, N, this algorithm outputs $\mathbf{E}$, which belongs to $Z_N^{(m_1+m_2)\times w_1}$.

Correctness. For all valid attribute X (X is not empty), we require that **Enc1**$(X, N) \rightarrow (\mathbf{k}_1, \mathbf{k}_2; m_1, m_2, m_3)$, **Enc2**$(Y, N) \rightarrow (\mathbf{c}; w_2)$, and **Pair**$(X, Y, N)$ $\rightarrow \mathbf{E}$. If $R(X, Y) = 1$, then

$$(\mathbf{k}_1, \mathbf{k}_2)^\top \mathbf{Ec} = \alpha s.$$

(ℓ_1, ℓ_2)-Leakage-Resilient Property (LR).
We say a pair encoding scheme $\mathbf{P}$ defined above is (ℓ_1, ℓ_2)-leakage-resilient if for all probabilistic polynomial time adversary $\mathcal{A}$, advantage of $\mathcal{A}$ in game $\mathbf{Exp}_{\mathcal{G},b,\mathcal{A},LR}(\lambda)$ is negligible. We denote it by:

$$Adv_{\mathcal{A}}^{LR}(\lambda) = |Pr[\mathbf{Exp}_{\mathcal{G},0,\mathcal{A},LR}(\lambda) = 1] - Pr[\mathbf{Exp}_{\mathcal{G},1,\mathcal{A},LR}(\lambda) = 1]| \leq negl(\lambda).$$

$\mathbf{Exp}_{\mathcal{G},b,\mathcal{A},LR}(\lambda)$:

1. $(\mathbb{G}, \mathbb{G}_T, e, N, p_1, p_2, p_3) \leftarrow \mathcal{G}(\lambda)$;
2. $g_1 \xleftarrow{\$} \mathbb{G}_{p_1}, g_2 \xleftarrow{\$} \mathbb{G}_{p_2}, g_3 \xleftarrow{\$} \mathbb{G}_{p_3}$;
3. $\alpha \xleftarrow{\$} Z_N, (n_1, n_2) \leftarrow \mathbf{Param}(\lambda), \mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}$;
4. $st \leftarrow A_1^{\mathcal{O}_{b,\alpha,\mathbf{h},\mathbf{x}}^{L}(\cdot)}(g_1, g_2, g_3)$;
5. $b' \leftarrow A_2^{\mathcal{O}_{b,\alpha,\mathbf{h},\mathbf{x}}^{2}(\cdot)}(st)$;

Here we require that both O^L and O^2 are queried once only. Let f be any polynomial time computable leakage function whose output size is no longer than ℓ_1 or ℓ_2.

- $O_{b,\alpha,\mathbf{h},\mathbf{x}}^{L}(X, f)$: run **Enc1**$(X, p_2) \rightarrow (\mathbf{k}_1, \mathbf{k}_2; m_1, m_2, m_3)$, choose $r \xleftarrow{\$} Z_{p_2}^{m_3}$, return *f(k)*, where

$$k \leftarrow \begin{cases} (g_2^{\mathbf{k}_1(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})}, g_2^{\mathbf{k}_2(\mathbf{h},\mathbf{r})}) & b = 0 \\ (g_2^{\mathbf{k}_1(0,\mathbf{h},\mathbf{x},\mathbf{r})}, g_2^{\mathbf{k}_2(\mathbf{h},\mathbf{r})}) & b = 1 \end{cases}$$

  Notably, the query X input to $\mathcal{O}^L$ could be empty, in this case $|f(k)| \leq \ell_1$. If $X \in \mathcal{X}$, then $|f(k)| \leq \ell_2$.

- $O_{b,\alpha,\mathbf{h},\mathbf{x}}^{2}(Y)$: run **Enc2**$(Y, p_2) \rightarrow (\mathbf{c}, w_2)$, choose $\mathbf{s} \xleftarrow{\$} Z_{p_2}^{w_2+1}$, return $\mathbf{C} \leftarrow g_2^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}$.

4.2.2 Security Definitions

Similar to the definitions of pair encodings in [Att14], we define perfectly master-key hiding security and computationally master-key hiding security for leakage-resilient pair encodings.

Perfectly Master-Key Hiding Security. Let **P** be a leakage-resilient pair encoding scheme. For $N \in \mathbb{N}$, if $R(X, Y) = 0$, let $\mathbf{Param}(\lambda) \to (n_1, n_2)$, $\mathbf{Enc1}(X, N) \to (\mathbf{k}_1, \mathbf{k}_2; m_1, m_2, m_3)$, $\mathbf{Enc2}(Y, N) \to (\mathbf{c}, w_2)$. If the following two distributions are identical, we say **P** is perfectly master-key hiding (PMH).

$$\{\mathbf{c}(\mathbf{s}, \mathbf{h}, \mathbf{x}), \mathbf{k}_1(0, \mathbf{h}, \mathbf{x}, \mathbf{r}), \mathbf{k}_2(\mathbf{h}, \mathbf{r})\} \quad \text{and} \quad \{\mathbf{c}(\mathbf{s}, \mathbf{h}, \mathbf{x}), \mathbf{k}_1(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}), \mathbf{k}_2(\mathbf{h}, \mathbf{r})\},$$

where the probability is taken over $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \alpha \xleftarrow{\$} Z_N, \mathbf{r} \xleftarrow{\$} Z_N^{m_3}, \mathbf{s} \xleftarrow{\$} Z_N^{w_2+1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}$.

Computational Security. We define selectively master-key hiding security (SMH) and co-selectively master-key hiding security (CMH) on a bilinear group generator $\mathcal{G}$ respectively as follows. We define a game template $\mathbf{Exp}_{\mathcal{G},b,\mathcal{A},T}(\lambda)$ to encompass these two security notions by using different types of Oracles ($\mathcal{O}^T = (\mathcal{O}^{T,1}, \mathcal{O}^{T,2})$, $T \in \{\text{SMH, CMH}\}$) for adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ as follows:

$\mathbf{Exp}_{\mathcal{G},b,\mathcal{A},T}(\lambda)$:

1. $(\mathbb{G}, \mathbb{G}_T, e, N, p_1, p_2, p_3) \leftarrow \mathcal{G}(\lambda)$;
2. $g_1 \xleftarrow{\$} \mathbb{G}_{p_1}, g_2 \xleftarrow{\$} \mathbb{G}_{p_2}, g_3 \xleftarrow{\$} \mathbb{G}_{p_3}$;
3. $\alpha \xleftarrow{\$} Z_N, (n_1, n_2) \leftarrow \mathbf{Param}(\lambda), \mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}$;
4. $st \leftarrow A_1^{\mathcal{O}^{T,1}_{b,\alpha,\mathbf{h},\mathbf{x}}(\cdot)}(g_1, g_2, g_3)$;
5. $b' \leftarrow A_2^{\mathcal{O}^{T,2}_{b,\alpha,\mathbf{h},\mathbf{x}}(\cdot)}(st)$

The advantage of $\mathcal{A}$ in the corresponding security game $\mathbf{Exp}_{\mathcal{G},b,\mathcal{A},T}(\lambda)$ is defined as follows:

$$Adv^T_{\mathcal{A}}(\lambda) = |Pr[\mathbf{Exp}_{\mathcal{G},0,\mathcal{A},T}(\lambda) = 1] - Pr[\mathbf{Exp}_{\mathcal{G},1,\mathcal{A},T}(\lambda) = 1]|.$$

The different types of Oracle $\mathcal{O}^T$ are defined as follows:

- **Selective Security (SMH).** $O^{SMH,1}$ can be queried once while $O^{SMH,2}$ can be queried polynomially many times.

– $O^{SMH,1}_{b,\alpha,\mathbf{h},\mathbf{x}}(Y)$: Run $\mathbf{Enc2}(Y, p_2) \rightarrow (\mathbf{c}, w_2)$, choose $\mathbf{s} \xleftarrow{\$} Z_{p_2}^{w_2+1}$, return $\mathbf{C} \leftarrow g_2^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}$.

– $O^{SMH,2}_{b,\alpha,\mathbf{h},\mathbf{x}}(X)$: If $R_{p_2}(X, Y) = 1$ or X is empty, return $\perp$. Run $\mathbf{Enc1}(X, p_2)$ $\rightarrow$ ($\mathbf{k}_1$,$\mathbf{k}_2$; m_1,m_2, m_3), pick $\mathbf{r} \xleftarrow{\$} Z_{p_2}^{m_3}$, return

$$k \leftarrow \begin{cases} (g_2^{\mathbf{k}_1(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})}, g_2^{\mathbf{k}_2(\mathbf{h},\mathbf{r})}) & b = 0 \\ (g_2^{\mathbf{k}_1(0,\mathbf{h},\mathbf{x},\mathbf{r})}, g_2^{\mathbf{k}_2(\mathbf{h},\mathbf{r})}) & b = 1 \end{cases}$$

- **Co-selective Security (CMH).** Both $O^{CMH,1}$ and $O^{CMH,2}$ can be queried once only.

  – $O^{CMH,1}_{b,\alpha,\mathbf{h},\mathbf{x}}(X)$: If X is empty, return $\perp$. Otherwise, run $\mathbf{Enc1}(X, p_2) \rightarrow (\mathbf{k}_1, \mathbf{k}_2; m_1, m_2, m_3)$, choose $\mathbf{r} \xleftarrow{\$} Z_{p_2}^{m_3}$, return

$$k \leftarrow \begin{cases} (g_2^{\mathbf{k}_1(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})}, g_2^{\mathbf{k}_2(\mathbf{h},\mathbf{r})}) & b = 0 \\ (g_2^{\mathbf{k}_1(0,\mathbf{h},\mathbf{x},\mathbf{r})}, g_2^{\mathbf{k}_2(\mathbf{h},\mathbf{r})}) & b = 1 \end{cases}$$

  – $O^{CMH,2}_{b,\alpha,\mathbf{h},\mathbf{x}}(Y)$: If $R_{p_2}(X, Y) = 1$, then return $\perp$. Otherwise, run $\mathbf{Enc2}(Y, p_2)$ $\rightarrow (\mathbf{c}, w_2)$, choose $\mathbf{s} \xleftarrow{\$} Z_{p_2}^{w_2+1}$, return $\mathbf{C} \leftarrow g_2^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}$.

We would like to remark that perfectly master-key hiding implies CMH but not SMH as in [Att14].

4.3 From Leakage-Resilient Pair Encoding to LR-ABE

In this section, we first present a generic construction of leakage-resilient attribute-based encryptions using leakage-resilient pair encodings as a building block. Then, we present a security analysis of our generic construction. Our generic construction is based on [Att14].

4.3.1 Generic construction

Let $\mathbf{P} = (Param, Enc1, Enc2, Pair)$ be a (ℓ_{msk}, ℓ_{sk})-leakage-resilient pair encoding scheme for predicate family $\mathcal{R}$. We construct a functional encryption FE for $\mathcal{R}$ as follows:

- **Setup**(λ, k): Run $(\mathbb{G}, \mathbb{G}_\mathbb{T}, e, N, p_1, p_2, p_3) \leftarrow \mathcal{G}(\lambda)$, pick generator $g_1 \xleftarrow{\$} \mathbb{G}_{p_1}$, $g_3 \xleftarrow{\$} \mathbb{G}_{p_3}$. Run $Param(\lambda) \rightarrow (n_1, n_2)$ and $Enc1(\varepsilon, N) \rightarrow$($\mathbf{k}_1^\varepsilon$, $\mathbf{k}_2^\varepsilon$;

m_1, m_2, m_3). Pick $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}$, $\mathbf{x} \xleftarrow{\$} Z_N^{n_2}$, $\alpha \xleftarrow{\$} Z_N$, $\mathbf{r} \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_1 \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_2 \xleftarrow{\$} Z_N^{m_2}$. The public key is $\mathsf{PK} = (g_1, g_3, e(g_1, g_1)^{\alpha}, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$, the master secret key is $\mathsf{MSK} = (\mathbf{K}_1, \mathbf{K}_2) = (g_1^{\mathbf{k}_1^{\varepsilon}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})} * g_3^{\mathbf{R}_1}, g_1^{\mathbf{k}_2^{\varepsilon}(\mathbf{h},\mathbf{r})} * g_3^{\mathbf{R}_2})$.

- **KeyGen** $(\mathsf{MSK}, X, \mathsf{PK})$: Run $Enc1(X, N) \rightarrow (\mathbf{k}_{X,1}, \mathbf{k}_{X,2}; m_1, m_2, m_3)$. Pick $\mathbf{r}' \xleftarrow{\$} Z_N^{m_3}$. Then compute user key as follows:

$$SK = (\mathbf{K}_1, \mathbf{K}_2) = (g_1^{\mathbf{k}_{X,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}+\mathbf{r}')} * g_3^{\mathbf{R}_1}, g_1^{\mathbf{k}_{X,2}(\mathbf{h},\mathbf{r}+\mathbf{r}')} * g_3^{\mathbf{R}_2}).$$

  Notably, SK can be computed from MSK due to the regeneration property of $\mathbf{P}$.

- **Encrypt**(Y, M, PK): Run $Enc2(Y, N) \rightarrow (\mathbf{c}; w_2)$. Pick $\mathbf{s} = (s, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$, and compute the ciphertext $CT = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = M \cdot e(g_1, g_1)^{\alpha s}, \qquad \mathbf{C}_1 = g_1^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}.$$

- **Decrypt**(CT, SK): Obtain X, Y from SK and CT. Check whether $R(X, Y) = 1$. If yes, run $Pair(X, Y, N) \rightarrow \mathbf{E}$. Next compute $e((K_1, K_2)^{\mathbf{E}}, C_1)$ to get the blinding factor $e(g_1, g_1)^{\alpha s}$ and get the plaintext M.

$$M = \frac{C_0}{e((K_1,K_2)^{\mathbf{E}}, C_1)}.$$

Semi-Functional Algorithms These algorithms will be used in the security proof of our generic construction.

SFSetup(λ, k) This algorithm is nearly the same as **Setup**(λ, k) except that it additionally outputs a generator $g_2 \xleftarrow{\$} \mathbb{G}_2$, $\hat{\mathbf{h}} \xleftarrow{\$} Z_N^{n_1}$ and $\hat{\mathbf{x}} \xleftarrow{\$} Z_N^{n_2}$. We call $\hat{\mathbf{h}}$ and $\hat{\mathbf{x}}$ semi-functional parameters.

SFEncrypt$(Y, M, \mathsf{PK}, g_2, \hat{\mathbf{h}}, \hat{\mathbf{x}})$: This algorithm first run $Enc2(Y) \rightarrow (\mathbf{c}; w_2)$. Then pick $\mathbf{s} = (s, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$, $\hat{\mathbf{s}} \xleftarrow{\$} Z_N^{w_2+1}$, output the semi-functional ciphertext $CT = (\vec{C}_1, C_0)$:

$$\vec{C}_1 = g_1^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})} * g_2^{\mathbf{c}(\hat{\mathbf{s}},\hat{\mathbf{h}},\hat{\mathbf{x}})}.$$
$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M.$$

SFKeyGen$(X, \mathsf{MSK}, \mathsf{PK}, g_2, type, \hat{\alpha}, \hat{\mathbf{h}}, \hat{\mathbf{x}})$ Note that, X is a predicate attribute (or an empty string). In both case, run algorithm $Enc1(X, N) \rightarrow (\mathbf{k}_{X,1}$, $\mathbf{k}_{X,2}$; m_1, m_2, m_3). Next pick $\hat{\mathbf{r}} \xleftarrow{\$} Z_N^{m_3}$. Then output one type of

semi-function secret key (or master key) depending on the input type $t \in \{1, 2, 3\}$.

$$\mathbf{K} = \begin{cases} g_1^{\mathbf{k}_{X,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})} * g_2^{\mathbf{k}_{X,1}(0,\hat{\mathbf{h}},\hat{\mathbf{x}},\hat{\mathbf{r}})} * g_3^{\mathbf{R}_{3,1}}, g_1^{\mathbf{k}_{X,2}(\mathbf{h},\mathbf{r})} * g_2^{\mathbf{k}_{X,2}(\hat{\mathbf{h}},\hat{\mathbf{r}})} * g_3^{\mathbf{R}_{3,2}} & t = 1 \\ g_1^{\mathbf{k}_{X,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})} * g_2^{\mathbf{k}_{X,1}(\hat{\alpha},\hat{\mathbf{h}},\hat{\mathbf{x}},\hat{\mathbf{r}})} * g_3^{\mathbf{R}_{3,1}}, g_1^{\mathbf{k}_{X,2}(\mathbf{h},\mathbf{r})} * g_2^{\mathbf{k}_{X,2}(\hat{\mathbf{h}},\hat{\mathbf{r}})} * g_3^{\mathbf{R}_{3,2}} & t = 2 \\ g_1^{\mathbf{k}_{X,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})} * g_2^{\mathbf{k}_{X,1}(\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})} * g_3^{\mathbf{R}_{3,1}}, g_1^{\mathbf{k}_{X,2}(\mathbf{h},\mathbf{r})} * g_2^{\mathbf{k}_{X,2}(\mathbf{0},\mathbf{0})} * g_3^{\mathbf{R}_{3,2}} & t = 3 \end{cases}$$

4.3.2 Security Proof of Our Generic Construction

Here we describe two ways to prove the security of our generic construction. If the underlying leakage-resilient pair encoding scheme can achieve computational security (SMH and CMH), we give a tighter reduction with cost of $\mathcal{O}(q_1)$. On the other hand, if the underlying leakage-resilient pair encoding scheme is perfectly master-key hiding[1], we achieve a reduction with cost of $\mathcal{O}(q_1 + q_2)$. Here q_1 and q_2 are the numbers of queries made in Phase 1 and Phase 2 respectively.

Theorem 4.3.1. *Assume that the (ℓ_{msk}, ℓ_{sk})-leakage-resilient pair encoding scheme $\mathbf{P}$ is selectively and co-selectively master-key hiding in $\mathcal{G}$ and subgroup assumption SD1, SD2, SD3 holds in $\mathcal{G}$. Then* ABE *constructed above from $\mathbf{P}$ is (ℓ_{msk}, ℓ_{sk})-leakage-resilient and adaptively secure in continual memory leakage model. For any PPT adversary $\mathcal{A}$, there exists adversary $\mathcal{B}_1, \mathcal{B}_2, \cdots, \mathcal{B}_6$ who runs nearly the same time as $\mathcal{A}$, such that for any λ:*

$$\begin{aligned} \boldsymbol{Adv}_{\mathcal{A}}^{\mathsf{ABE}}(\lambda) \leq 2\boldsymbol{Adv}_{\mathcal{B}_1}^{SD1}(\lambda) + (2q_1 + 3)\boldsymbol{Adv}_{\mathcal{B}_2}^{SD2}(\lambda) + q_1\boldsymbol{Adv}_{\mathcal{B}_3}^{CMH}(\lambda) \\ + q_1\boldsymbol{Adv}_{\mathcal{B}_4}^{LR}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}_5}^{SMH}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}_6}^{SD3}(\lambda) \end{aligned} \quad (4.4)$$

Proof Outline. Here we only give the definition of a sequence of games used in our security proof. The indistinguishability between adjacent games are given in **Appendix** A.2.

- **Game**$_{real}$: This is the real game as defined in the continual memory leakage model.
- **Game**$_{res}$: This game is exactly the same as $Game_{real}$ except that we require all attribute X contained in reveal queries made by $\mathcal{A}$ is $R_{p_2}(X, Y^*) = 0$ rather than $R_N(X, Y^*) = 0$.

[1]Recall that perfectly master-key hiding only implies computational co-selective security but not computational selective security.

- **Game**$_0$: Run **SFSetup**(λ, k) in the setup phase. Run $CT^* \leftarrow$ **SFEncrypt** $(Y^*, M_b, \mathsf{PK}, g_2, \hat{\mathbf{h}}, \hat{\mathbf{x}})$.

- **Game**$_{k,1}$: For all **Create** queries made by $\mathcal{A}$ in **Phase 1**, answer the j-th **Create** query by using the corresponding key as follows:
$$\hat{\alpha_j} \leftarrow Z_N, k_j \leftarrow \begin{cases} \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 3, \hat{\alpha_j}, \mathbf{0}, \mathbf{0}) & if \quad j < k \\ \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 1, 0, \hat{\mathbf{h}}, \hat{\mathbf{x}}) & if \quad j = k \\ \textbf{KeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}) & if \quad j > k \end{cases}$$
Note that if X_j is null, then $\mathcal{A}$ is making a query about master key.

- **Game**$_{k,2}$: For all **Create** queries made by $\mathcal{A}$ in **Phase 1**, answer the j-th **Create** query by generating the corresponding key as follows:
$$\hat{\alpha_j} \leftarrow Z_N, k_j \leftarrow \begin{cases} \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 3, \hat{\alpha_j}, \mathbf{0}, \mathbf{0}) & if \quad j < k \\ \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 2, \hat{\alpha_j}, \hat{\mathbf{h}}, \hat{\mathbf{x}}) & if \quad j = k \\ \textbf{KeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}) & if \quad j > k \end{cases}$$
Note that if X_j is null, then $\mathcal{A}$ is making a query about master key.

- **Game**$_{k,3}$: In this Game, for all **Create** queries made by $\mathcal{A}$ in **Phase 1**, answer the j-th **Create** query by generating the corresponding key as follows:
$$\hat{\alpha_j} \leftarrow Z_N, k_j \leftarrow \begin{cases} \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 3, \hat{\alpha_j}, \mathbf{0}, \mathbf{0}) & if \quad j < k \\ \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 3, \hat{\alpha_j}, \mathbf{0}, \mathbf{0}) & if \quad j = k \\ \textbf{KeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}) & if \quad j > k \end{cases}$$
Note that if X_j is null, then $\mathcal{A}$ is making a query about master key.

- **Game**$_{2,1}$: For all **Create** queries made by A in **Phase 2**, generate the corresponding key as follows:
$$k_j \leftarrow \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 1, 0, \hat{\mathbf{h}}, \hat{\mathbf{x}}).$$

- **Game**$_{2,2}$: For all **Create** queries made by A in **Phase 2**, generate the corresponding key as follows:
$$k_j \leftarrow \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 2, \hat{\alpha}, \hat{\mathbf{h}}, \hat{\mathbf{x}}).$$

- **Game**$_{2,3}$: For all **Create** queries made by A in **Phase 2**, generate the corresponding key as follows:
$$k_j \leftarrow \textbf{SFKeyGen}(X_j, \mathsf{MSK}, \mathsf{PK}, g_2, 3, \hat{\alpha}, \mathbf{0}, \mathbf{0}).$$

- **Game**$_{final}$: $M \leftarrow \mathcal{M}$, $CT^* \leftarrow$ **SFEncrypt**$(Y^*, M, \mathsf{PK}, g_2, \hat{\mathbf{h}}, \hat{\mathbf{x}})$.

□

Theorem 4.3.2. *Assuming the (ℓ_{msk}, ℓ_{sk})-leakage-resilient pair encoding scheme P is perfectly master-key hiding and subgroup assumption SD1, SD2, SD3 holds in $\mathcal{G}$. Then* ABE *constructed above is (ℓ_{msk}, ℓ_{sk})-leakage-resilient and adaptively secure in continual memory leakage model. For any PPT adversary $\mathcal{A}$, there exists adversary $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}_3$, $\mathcal{B}_4$ who runs nearly the same time as $\mathcal{A}$, such that for any λ:*

$$\boldsymbol{Adv}_{\mathcal{A}}^{ABE}(\lambda) \leq 2\boldsymbol{Adv}_{\mathcal{B}_1}^{SD1}(\lambda) + (2(q_1+q_2)+1)\boldsymbol{Adv}_{\mathcal{B}_2}^{SD2}(\lambda) + q_1\boldsymbol{Adv}_{\mathcal{B}_3}^{LR}(\lambda) + \boldsymbol{Adv}_{\mathcal{B}_4}^{SD3}(\lambda) \quad (4.5)$$

Proof outline. The proof for **Theorem** 4.3.2 is very similar to **Theorem** 4.3.1, except that we handle **Create** key queries made in **Phase 2** one by one. More precisely, instead of changing the type of results of all **Create** key queries in **Phase 2** simutaneously, in each game, we only change the type of the result of one **Create** key query. Besides, the indistinguishability between semi-functional type-1 key and semi-functional type-2 key is based on the perfectly master-key hiding security of the underlying pair encoding. Hence, the proof is completed. □

4.4 From Pair Encoding to Leakage-Resilient Encoding

In this section, we show how to construct leakage-resilient encodings from pair encodings that satisfy some additional constraints. We would like to remark that these additional conditions are not overly restrictive. In fact, a large number of pair encoding schemes in [Att14] satisfies the requirements.

4.4.1 Extending the Definition of Attrapadung's Pair Encoding to Support Encoding of Empty Attribute

We extend the definition of pairing encoding schemes in [Att14] to support encoding for the empty attribute ε. Specifically, a pair encoding scheme $\mathbf{P} = (\mathbf{Param}, \mathbf{Enc1}, \mathbf{Enc2}, \mathbf{Pair})$ for predicate family $\mathcal{R}$ is defined as follows:

- $\mathbf{Param}(\lambda) \to n_1$, pick $\mathbf{h} = (h_1, h_2, \cdots, h_{n_1}) \xleftarrow{\$} Z_N^{n_1}$.
- $\mathbf{Enc1}(X, N)$:

– If X is empty, $\mathbf{Enc1}(\varepsilon, N) \rightarrow (\mathbf{k}_1^{\varepsilon}, \mathbf{k}_2^{\varepsilon}; 1, 1, 0)$ and $\mathbf{k}_1^{\varepsilon}(\alpha, \mathbf{h}, \mathbf{r}) = \alpha$, $\mathbf{k}_2^{\varepsilon}(\mathbf{h}, \mathbf{r}) = 0$.

– If $X \in \mathcal{X}$, then $\mathbf{Enc1}(X, N) \rightarrow (\mathbf{k}_1, \mathbf{k}_2; m_1, m_2, m_3)$, where $X \in \mathcal{X}$. Let $\mathbf{r} = (r_1, r_2, \cdots, r_{m_3}) \xleftarrow{\$} Z_N^{m_3}$. $\mathbf{k}_1 = \{k_{1,\ell}\}_{\ell \in [m_1]}$ is a set of polynomials, each one is in $\{\alpha\} \bigcup \{h_i\}_{i \in [n_1]} \bigcup \{r_j\}_{j \in [m_3]}$. $\mathbf{k}_2 = \{k_{2,\ell'}\}_{\ell' \in [m_2]}$ is a set of polynomials and each polynomial is in $\{h_i\}_{i \in [n_1]} \bigcup \{r_j\}_{j \in [m_3]}$. That is,

$$k_{1,\ell} = b_\ell \alpha + \sum_{i \in [n_1], j \in [m_3]} b_{\ell,i,j} h_i r_j. \tag{4.6}$$

$$k_{2,\ell'} = \sum_{j \in [m_3]} b_{\ell',j} r_j. \tag{4.7}$$

Remark 1: The output of $Enc1(X)$ is divided into two sets $\mathbf{k}_1$ and $\mathbf{k}_2$, and the concatenation of these two sets ($\mathbf{k}_1$, $\mathbf{k}_2$) is exactly the same as the output of **Enc1** as defined in [Att14]. Moreover, it is obvious that given $(\mathbf{k}_1^{\varepsilon}, \mathbf{k}_2^{\varepsilon})$ and any attribute $X \in \mathcal{X}$, we can generate an encoding ($\mathbf{k}_1$, $\mathbf{k}_2$) for X which is properly distributed.

- $\mathbf{Enc2}(X, N) \rightarrow (\mathbf{c}; w_2)$. Let $\mathbf{s} = (s, s_1, s_2, \cdots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$. $\mathbf{c} = \{c_\ell\}_{\ell \in [w_1]}$ is a set of polynomials, each one is in $\{s, s_k\}_{k \in [w_2]} \bigcup \{h_i\}_{i \in [n_1]}$.

$$c_\ell = t_\ell s + \sum_{k \in [w_2], i \in [n_1]} t_{\ell,k,i} s_k h_i + \sum_{k \in [w_2]} t_{\ell,k} s_k + \sum_{i \in [n_1]} t'_{\ell,i} h_i s. \tag{4.8}$$

- $\mathbf{Pair}(X, Y, N) \rightarrow \mathbf{E}$, where $\mathbf{E} \in Z_N^{(m_1+m_2) \times w_1}$.

4.4.2 Generic Transformation of Pair Encodings to Leakage-Resilient Pair Encodings

Let **P = (Param, Enc1, Enc2, Pair)** be a pair encoding supporting empty attribute. Below we show how to construct a leakage-resilient pairing encoding scheme $\mathbf{P}'$= ($\mathbf{Param}'$, $\mathbf{Enc1}'$, $\mathbf{Enc2}'$, $\mathbf{Pair}'$).

- $\mathbf{Param}'(\lambda)$: Run $\mathbf{Param}(\lambda) \rightarrow n_1$, pick $\mathbf{h} = (h_1, h_2, \cdots, h_{n_1}) \xleftarrow{\$} Z_N^{n_1}$ and choose a proper parameter $n = \mathcal{O}(\lambda)$. Pick $\mathbf{x} = (x_1, x_2, \cdots, x_n) \xleftarrow{\$} Z_N^n$ and output (n_1, n).

- $\mathbf{Enc1}'(X, N)$: Run $\mathbf{Enc1}(X, N) \rightarrow (\mathbf{k}_1(\alpha, \mathbf{h}, \mathbf{r}), \mathbf{k}_2(\mathbf{h}, \mathbf{r}); m_1, m_2, m_3)$, then choose $(y_1, y_2, \cdots, y_n) \xleftarrow{\$} Z_N^n$ and set $\mathbf{r}' = (y_1, y_2, \cdots, y_n, \mathbf{r})$.

$$\mathbf{k}_1'(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}') = (y_1, y_2, \cdots, y_n, \mathbf{k}_1(\alpha, \mathbf{h}, \mathbf{r})) + (\overbrace{0, 0, \cdots, 0}^{n}, \sum_{i \in [n]} -x_i y_i, \overbrace{0, \cdots, 0}^{m_1 - 1}). \quad (4.9)$$

$$\mathbf{k}_2'(\mathbf{h}, \mathbf{r}') = \mathbf{k}_2(\mathbf{h}, \mathbf{r}). \quad (4.10)$$

  Outputs $(\mathbf{k}_1'(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}'), \mathbf{k}_2'(\mathbf{h}, \mathbf{r}'); m_1 + n, m_2, n + m_3)$.

- $\mathbf{Enc2}'(Y, N)$: Run $\mathbf{Enc2}(Y, N) \rightarrow (\mathbf{c}(\mathbf{s}, \mathbf{h}) = (c_1, c_2, \cdots, c_{w_1}); w_2)$, then

$$\mathbf{c}'(\mathbf{s}, \mathbf{h}, \mathbf{x}) = (x_1 c_1, x_2 c_1, \cdots, x_n c_1, \mathbf{c}(\mathbf{s}, \mathbf{h})). \quad (4.11)$$

  Output $\mathbf{c}'(\mathbf{s}, \mathbf{h}, \mathbf{x})$.

- $\mathbf{Pair}'(X, Y, N)$: Runs $Pair(X, Y, N) \rightarrow \mathbf{E}$, then construct and output matrix $\mathbf{E}'$ defined as follows.

$$\mathbf{E}' = \left(\begin{array}{cccccc|c} 1 & 0 & 0 & 0 & \cdots & 0 & \\ 0 & 1 & 0 & 0 & \cdots & 0 & \\ 0 & 0 & 1 & 0 & \cdots & 0 & \mathbf{0}^{n \times w_1} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \\ 0 & 0 & 0 & 0 & \cdots & 1 & \\ \hline & & & & & & \\ & & \mathbf{0}^{(m_1+m_2)\times n} & & & & \mathbf{E} \\ & & & & & & \end{array}\right).$$

Theorem 4.4.1. $\mathbf{P}'$ *is a* $(\ell_{msk} = (n-1-2c)log(p_2), \ell_{sk} = (n-1-2c)log(p_2))$*-leakage-resilient pair encoding scheme, for some positive constant* c *such that* p_2^{-c} *is negligible, if* $\mathbf{P}$ *satisfies the following conditions:*

1. *The first row of the reconstruction matrix* $\boldsymbol{E}$ *of* $\mathbf{P}$ *is* $(\overbrace{1, 0, 0, \cdots, 0}^{w_1})$.
2. *For any attribute* X*, run* $\boldsymbol{Enc1}(X, N) \rightarrow (\boldsymbol{k}_1, \boldsymbol{k}_2; m_1, m_2, m_3)$*. Then the two distributions* $(\boldsymbol{k}_1(0, \boldsymbol{h}, \boldsymbol{r}), \boldsymbol{k}_2(\boldsymbol{h}, \boldsymbol{r}))$ *and* $(\boldsymbol{k}_1(\alpha, \boldsymbol{h}, \boldsymbol{r}), \boldsymbol{k}_2(\boldsymbol{h}, \boldsymbol{r}))$ *are identical except that*

$$\boldsymbol{k}_{11}^{\alpha} = \boldsymbol{k}_{11}^{0} + \alpha,$$

where $\boldsymbol{k}_{11}^{\alpha}$ is the first polynomial of $\boldsymbol{k}_1(\alpha, \boldsymbol{h}, \boldsymbol{r})$, $\boldsymbol{k}_{11}^{0}$ is the first polynomial of $\boldsymbol{k}_1(0, \boldsymbol{h}, \boldsymbol{r})$, and $\alpha \xleftarrow{\$} Z_N, \boldsymbol{h} \xleftarrow{\$} Z_N^{n_1}, \boldsymbol{r} \xleftarrow{\$} Z_N^{m_3}$.

Proof. In order to prove the above theorem, we will argue the correctness, regeneration property and the leakage-resilient property of $\mathbf{P}'$. Moreover, we will argue that $\mathbf{P}'$ preserves the original security properties (PMH, CMH, SMH) of $\mathbf{P}$.

Correctness. It is straightforward to verify the correctness of $\mathbf{P}'$ due to the correctness $\mathbf{P}$ and that the first row of the reconstruction matrix for $\mathbf{P}$ is $(1, 0, \cdots, 0)$.

Regeneration Property. Here we argue that given $(\mathbf{k}_1^{\varepsilon'}(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}), \mathbf{k}_2^{\varepsilon'}(\mathbf{h}, \mathbf{r}))$ and an attribute $X \in \mathcal{X}$, we can generate properly distributed $(\mathbf{k}_1'(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}'), \mathbf{k}_2'(\mathbf{h}, \mathbf{r}'))$. From our above definition, it is not hard to see that $\mathbf{k}_1^{\varepsilon'}(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r})$ is in the form $(y_1, y_2, \cdots, y_n, \alpha - \sum_{i\in[n]} x_i y_i)$, where $\mathbf{r} = (y_1, y_2, \cdots, y_n)$. Then run $\mathbf{Enc1}(X, N) \rightarrow (\mathbf{k}_1(0, \mathbf{h}, \mathbf{r}''), \mathbf{k}_2(\mathbf{h}, \mathbf{r}''); m_1, m_2, m_3)$, where $\mathbf{r}'' \xleftarrow{\$} Z_N^{m_3}$. Next picks $(y_1', y_2', \cdots, y_n') \xleftarrow{\$} Z_N^n$ and compute

$$\mathbf{k}_1'(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}') = (\mathbf{k}_1^{\varepsilon'}(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}), \overbrace{0, 0, \cdots, 0}^{m_1-1}) + (y_1', y_2', \cdots, y_n', -\sum_{i\in[n]} x_i y_i', \overbrace{0, 0, \cdots, 0}^{m_1-1}) + (\overbrace{0, 0, \cdots, 0}^{n}, \mathbf{k}_1(0, \mathbf{h}, \mathbf{r}'')),$$

where $\mathbf{r}' = (y_1 + y_1', y_2 + y_2', \cdots, y_n + y_n', \mathbf{r}'')$. $\mathbf{k}_2'(\mathbf{h}, \mathbf{r}') = \mathbf{k}_2(\mathbf{h}, \mathbf{r}'')$. We would like to remark that this is still efficiently computable in group operations.

Lemma 4.4.1. *The resulting pair encoding scheme $\mathbf{P}'$ constructed above is ($\ell_{msk} = (n-1-2c)log(p_2)$, $\ell_{sk} = (n-1-2c)log(p_2)$)-leakage-resilient.*

Proof. Assuming there exists PPT algorithm $\mathcal{A}$ whose advantage in $\mathbf{Exp}_{\mathcal{G},b,LR}(1^\lambda)$ is non-negligible. Then we will build PPT algorithm $\mathcal{B}$ who will distinguish between distributions $(\vec{\delta}, f(\vec{\tau}))$ and $(\vec{\delta}, f(\vec{\tau}'))$ with non-negligible advantage. This will yield a contradiction since these two distributions are statistically indistinguishable. $\mathcal{B}$ simulates the $\mathbf{Exp}_{\mathcal{G},b,LR}(1^\lambda)$ for $\mathcal{A}$ as follows: Run $(\mathbb{G}, \mathbb{G}_{\mathbb{T}}, e, N, p_1, p_2, p_3) \leftarrow \mathcal{G}(\lambda)$, then pick generators $g_1 \xleftarrow{\$} \mathbb{G}_{p_1}$, $g_2 \xleftarrow{\$} \mathbb{G}_{p_2}$, $g_3 \xleftarrow{\$} \mathbb{G}_{p_3}$ and give (g_1, g_2, g_3) to $\mathcal{A}$.

At some point, $\mathcal{A}$ will make $\mathcal{O}^L$ query for X^*. Note that here we only consider either X^* is empty or X^* matches the challenge predicate Y^* input to O^2 (namely the predicate in the challenge ciphertext). Otherwise,

leakage security is guaranteed by the perfect master-key hiding property or co-selective property of P'.

Once $\mathcal{A}$ makes a leakage query about (X^*, f), $\mathcal{B}$ will not create this key but instead encode the leakage function $\mathcal{A}$ asks as a single polynomial-time computable function f' whose output size is limited. $\mathcal{B}$ receives its own challenge instance $(\vec{\delta}, f'(\vec{\Gamma}))$, where $\vec{\Gamma} = (\Gamma_1, \Gamma_2, \cdots, \Gamma_n, \Gamma_{n+1})$ is distributed as either $\vec{\tau}$ or $\vec{\tau'}$, and use $f'(\vec{\Gamma})$ to answer all of $\mathcal{A}$'s leakage queries. $\mathcal{B}$ can do this by fixing all other variables in the challenge key, and works as follows:

$\mathcal{B}$ runs $Enc1(X, N) \rightarrow (\mathbf{k}_1, \mathbf{k}_2; m_1, m_2, m_3)$, computes $(\mathbf{k}_1(0, \mathbf{h}, \mathbf{r}), \mathbf{k}_2(\mathbf{h}, \mathbf{r}))$ ($\boldsymbol{h} \xleftarrow{\$} Z_N^{n_1}$, $\boldsymbol{r} \xleftarrow{\$} Z_N^{m_3}$), then implicitly sets $(\mathbf{k}_1'(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}'), \mathbf{k}_2'(\mathbf{h}, \mathbf{r})) = \vec{\Gamma}'$ and $\mathbf{K}^* = g_2^{\vec{\Gamma}'}$, where

$$\vec{\Gamma}' = (\Gamma_1, \Gamma_2, \cdots, \Gamma_n, \Gamma_{n+1}, \overbrace{0, 0, \cdots, 0}^{m_1+m_2-1}) + (\overbrace{0, 0, \cdots, 0}^{n}, \mathbf{k}_1(0, \mathbf{h}, \mathbf{r}), \mathbf{k}_2(\mathbf{h}, \mathbf{r})).$$

At some point, $\mathcal{A}$ makes an oracle query about $\mathcal{O}^2$ for challenge predicate Y^*. $\mathcal{B}$ constructs challenge ciphertext $C = g_2^{\mathbf{c}'(s,\mathbf{h},\mathbf{x})}$ as follows: run $Enc2(Y^*, N) \rightarrow (\mathbf{c}; w_2)$, and computes $\mathbf{c}(\mathbf{s}, \mathbf{h}) = (c_1, c_2, \cdots, c_{w_1})$ ($\boldsymbol{s}$= (s, s_1,s_2, s_3, $\cdots$, s_{w_2}) $\xleftarrow{\$} Z_N^{w_2+1}$) , then set the cipertext for Y^* to be $\mathbf{C}^* = g_2^{\mathbf{c}'(\mathbf{s},\mathbf{h},\mathbf{x})} = g_2^{\vec{\delta}'}$, where

$$\vec{\delta}' = (\delta_1 c_1 \delta_{n+1}^{-1}, \delta_2 c_1 \delta_{n+1}^{-1}, \cdots, \delta_n c_1 \delta_{n+1}^{-1}, \mathbf{c}(\mathbf{s}, \mathbf{h})).$$

Here δ_{n+1}^{-1} means the inverse of δ_{n+1} in Z_{p_2}. If $\delta_{n+1} = 0$, $\mathcal{B}$ aborts this simulation and randomly guess the distribution of Γ. However, this happens with negligible probability. Observe that $\mathcal{B}$ perfectly simulates the corresponding environment for $\mathcal{A}$ as it implicitly sets:

$$\alpha = \delta_{n+1}^{-1} \sum_{i \in [n]} \Gamma_i \delta_i + \Gamma_{n+1}.$$

- Case1: $\vec{\Gamma}$ is orthogonal to $\vec{\delta}$. It is clear that $\mathcal{B}$ perfectly simulates $\mathbf{Exp}_{\mathcal{G},0,LR}(1^\lambda)$ for $\mathcal{A}$;
- Case 2: $\vec{\Gamma}$ is not orthogonal to $\vec{\delta}$. $\mathcal{B}$ simulates $\mathbf{Exp}_{\mathcal{G},1,LR}(1^\lambda)$ for $\mathcal{A}$.

□

The following lemma states that our transformation preserves the master-key hiding property. The proof is straightforward and is thus omitted.

Lemma 4.4.2. *If P is perfectly (resp. selectively, co-selectively) master-key hiding, P' is also perfectly (resp. selectively, co-selectively) master-key hiding.*

□

4.5 Instantiations of Our Framework

Based on existing pair encodings, we achieve a number of new schemes and schemes with better properties. This is summarised in Table 4.1.

Table 4.1: New schemes/constructions from our Framework

Predicates	Adaptively Secure?	Leakage -Resilient?	Constructions	Underlying pair Encoding in [Att14]
IBE	✓	×	[LW10, Att14]	
	✓	✓ (CML)	[LRW11], this work (tighter reduction)	Scheme 1
FEs for regular language	×	×	[Wat12]	
	✓	×	[Att14]	
	✓	✓ (CML)	this work (new construction)	Scheme 3
KP-ABE for large universe	×	×	[RW13]	
	✓	×	[Att14]	
	✓	✓ (CML)	this work (new construction)	Scheme 4
KP-ABE with short ciphertext	×	×	[ALDP11]	
	✓	×	[Att14]	
	✓	✓ (CML)	this work (new construction)	Scheme 5
KP-DSE	×	×	[Ham11]	
	✓	×	[Att14]	
	✓	✓ (CML)	this work (new construction)	Scheme 6

Chapter 5

Traceability Analysis of CryptoNote-Style Blockchains

Since the introduction of Bitcoin in 2009 [Nak08], numerous distributed cryptocurrencies have been proposed. Nonetheless, most of existing cryptocurrencies are not designed to provide strong privacy protection. For instance, several works [RH11, MPJ+13, RS13] showed Bitcoin, currently the most popular and largest cryptocurrency, is vulnerable against the deanonymization attacks.

To address this problem, cryptocurrencies with stronger privacy guarantees are attracting more and more attentions. Among them, CryptoNote-style cryptocurrencies are one of the noteworthy efforts. The CryptoNote protocol was first introduced in [VS13], with a focus on protecting the privacy and anonymity of the electronic cash. Since its introduction, many variations utilizing this protocol are gradually proposed, including Bytecoin, Boolberry, Dashcoin, DigitalNote, Monero, etc. Similar to many other distributed cryptocurrencies, CryptoNote also adopts the notion of transaction to represent the process of spending coins. Each transaction contains several inputs and outputs. The total amount of coins consumed in the inputs and the total amount of coins transferred to the outputs should be equal. Besides, each transaction should be signed by the sender to authorize the transfer, by using the private key associated to the public-key (address)[1] of a to-be-spent coin. Here, a ring signature [RST01, LSW06] scheme is adopted to guarantee the privacy of the real-spend of each input, which is a cryptographic primitive that allows a user to anonymously sign a message on behalf of a group of users. Therefore, the identity of the real-spend is hidden. All other decoy coins in the input are called mixins.

[1]Throughout this work, we interchangeably use the term coin, output and the public-key.

However, in practice, CryptoNote-style cryptocurrencies fall short from achieving their claimed anonymity. Recently, two independent and concurrent works [MMLN17, KFTS17][2] demonstrate that Monero transactions may be de-anonymized via statistical analysis. According to the experiment results of [MMLN17, KFTS17], by Feb 2017, nearly 65% of transaction inputs are with zero-mixin, and the cascade effect can render another 22% of inputs traceable, i.e., nearly 87% of all Monero inputs are insecure when considering users' anonymity. Having witnessed (and predicted) this type of attacks, Monero has proposed a few countermeasures. Firstly, the minimum number of mixins gradually increases from 2 in version 0.9.0 (January 1, 2016)to 6 in version 0.12.0 (March 29, 2018). Secondly, at version 0.10.0 (September 19, 2016), ring confidential transaction (Ring-CT), which aims at further enhancing privacy of users via hiding the transaction amount, is introduced.

There is no doubt that known attacks are circumvented by Monero, but the initial problem still exists, which also threatens all CryptoNote-style currencies. That is, can anonymity of users be well-protected with a current small ring size, i.e., the countermeasures for known attacks? A related question is how to theoretically analyze the security level achieved by those cryptocurrencies adopting the ring signature for untraceability. Besides, to the best of our knowledge, there is no empirical analysis on other CryptoNote-style currencies. So, a natural question is whether these systems in practice provide a similar guarantee.

Chapter Organization. We start by an overview of contributions in **Section** 5.1. The definition of *closed set* attacks is introduced in **Section** 5.2. Considering the inefficiency of *closed set* attacks, we propose an approximate but efficient algorithm called clustering in **Section** 5.3. The experiment results on top three CryptoNote-style cryptocurrencies are given in **Section** 5.4. This chapter is concluded with **Section** 5.5 about our observations and recommendations.

[2] An updated version [MSH+18] of [MMLN17] also appears recently, but both the method and the result for the traceability analysis are similar in these two works, thus we focus on the initial version.

5.1 Our Contributions

In the following sections of the chapter, we give answers to the above questions. First, we show that the current countermeasures to resist known attacks make Monero a good system to provide anonymity. However, on the negative side, we show other CryptoNote-style protocols are still suffering from the same type of attacks. In fact, our combined attacks are much more effective on ByteCoin and DigitalNote, as we can de-anonymize up to 91.56% transactions in the chain of DigitalNote.

We introduce a new attack on the untraceability of CryptoNote-style currencies called *closed set* attack. This attack is based on the fact that n transaction inputs will and must use n distinct public-keys as real-spend, since each public-key can only be redeemed once. A set of inputs is called a *closed set* if the number of inputs equals to the number of distinct public-keys included. Hence, we can deduce that all public-keys included in a *closed set* must be mixins in other inputs outside of this *closed set*. In this way, the searching for *closed set*s will be helpful to track the real-spend of some other inputs. Different from cascade effect attack which relies on the "chain-reaction analysis" due to zero-mixin inputs, *closed set* attack conducts further traceability without relying on any previous traceable inputs.

The contributions of this work can be divided into the following aspects:

1. We introduce *closed set* attack on the untraceability of CryptoNote-style blockchains, and prove that *closed set* attack is *optimal*. In particular, it could get the minimal mixin for every input, i.e., it deletes all public-keys payed elsewhere in a mixin, identical to the results of brute-force attack.

2. We verify the impact of our attack via performing experiments on actual blockchain data, where we pick the top 3 CryptoNote-style currencies by market capitalization, i.e., Monero, Bytecoin and DigitalNote. As the *closed set* attack is too expensive to run due to its high complexity, we propose an efficient algorithm, namely, clustering algorithm, to (approximately) implement *closed set* attack. We give a lower bound of our clustering algorithm in implementing the *closed set* attack. Specifically, we prove that our algorithm can find all *closed set* of size 5.

   For Monero, up to block 1541236, the cascade attack can find the real-spend of 16329215 transaction inputs, which account for 70.492% of all

inputs. While our clustering attack can further identify the real-spend of 5752 transaction inputs out of the remaining 6835530 transaction inputs, accounting for 0.084% of all untraceable inputs after the cascade attack. During this process, 3017 distinct *closed set*s are found. For Bytecoin and DigitalNote, we provide the first work on analyzing them. The experiment results of these three currencies are given in **Table** 5.1.

Table 5.1: Experiment Results. All inputs considered in this work are non-coinbase transaction unless specific stated. Total(%) denotes the total traceable rate, while *Cas*. denotes the percentage of those inputs traced by cascade attack among all traceable inputs, and *Clu*. represents the percentage of inputs contributed by clustering attack. No. of C.S. denotes the total number of *closed set*s found by the clustering algorithm.

Coin	total Blocks	total Inputs	Total(%)	%		No. of C.S.
				Cas.	*Clu*.	
Monero	1541236 (30.03.2018)	23164745	70.516	99.96	0.04	3017
Bytecoin	1586652 (03.08.2018)	45663011	74.25	99.763	0.237	5912
DigitalNote	699748 (13.08.2018)	8110602	91.56	99.9993	0.0007	38

3. In addition, we also provide a theoretical analysis on the existence of *closed set*. We find that if all inputs have 3 mixins and all mixins are uniformly distributed, with all but a very small probability (about 2^{-19}), there will not exist any *closed set*. Our analysis suggests that the usage rate of outputs is closely related to the anonymity of Monero. Moreover, if we can guarantee that the probability of choosing an unspent key as mixin is 25%, then the number of mixins of each input could be as small as 3 to render brute-force attack ineffective. Furthermore, to help users obtaining an improved anonymity guarantee, we plan to release our algorithm as an online service for users to check if an output is included in a *closed set*.

5.2 Closed Set Attack

In this section, we introduce our *closed set* attack. All attacks considered in this section only assume the access to the transactions in the blockchain of a CryptoNote-style currency, without any further active ability. This assumption is valid since all transactions on the blockchains are publicly accessible.

We prove that our proposed *closed set* attack is equivalent to the currently known optimal attack, i.e., brute-force attack. Looking ahead, brute-force attack will traverse all possible assignments of payers of all inputs and delete those with conflict data. Both our attack and the brute-force attack return perfect results, where the set of candidates for the real payer of each input is minimum.

5.2.1 Brute-Force Attack

Brute-force attack is an attack that tries all possible sequences of distinct public-keys to test whether it is valid for the assignments of the real-spends for all transaction inputs. While a sequence of public-keys is valid if it satisfies requirements given below:

- the size of the sequence equals to the number of total transaction inputs in the dataset;
- all public-keys included in the sequence are distinct;
- for all index i of that sequence, the i-th public-key in the sequence belongs to the corresponding i-th input in the dataset.

In other words, brute-force attack is the process of searching for all valid sequences among the permutations of all public-keys with specific length according to the above requirements. We call all elements included in index i ($i \leq$ no. of all inputs) of all valid sequences as the candidates for the real-spend of the i-th transaction input. Therefore, the resulted valid sequences are the combinations of the possible real-spend of each transaction input. Besides, if a transaction input only has one candidate for the real payer, then the candidate must be its payer.

It is not hard to see that brute-force attack is a perfect attack which can find out all possible real-spends for each transaction input. Assume that there are n distinct keys and m transaction inputs in our dataset, and without loss of generality, n is larger than m. Let $\mathbb{A}_n^m$ denote the number of permutations of m elements among n elements. The number of valid sequences after the execution of brute-force attack is $(\mathbb{A}_n^m - |\mathsf{Conflicts}|)$, where $\mathsf{Conflicts}$ denotes the set of deleted permutations which fail to the above requirements.

5.2.2 Our Attack

Although the aforementioned brute-force attack is perfect, the complexity is quite expensive, which is $\mathcal{O}(n!)$. Considering the inefficiency and impracticability of brute-force attack, we propose a new attack, called *closed set*, which is more efficient while providing the same result.

The proposed *closed set* attack is based on the observation that if the number of distinct public-keys included in a set of transaction inputs equals to the number of the inputs of the set, then we can deduce that each public-key included must be a real-spend of a certain input in this set. This observation is due to the fact that each public-key (a.k.a., output of previous transaction) can only be redeemed once. Such kind of set of inputs is called *closed set* throughout this paper. Since each public-key included in a *closed set* must be spent inside the *closed set*, then it is safe to remove these keys from all other inputs outside the *closed set*. In this way, the finding of a *closed set* has at least two significant impacts. Firstly, it will render other inputs become traceable after removing public-keys of a *closed set*. Secondly, the average size of the inputs will decrease, which is helpful for further operation.

The *closed set* attack is an iteration process that finds out all possible *closed set*s from the transaction inputs, removes public-keys included, and finds those traceable inputs. Compared with the previous cascade effect attack presented by [KFTS17] or "chain-reaction" analysis by [MMLN17], the *closed set* attack can render more inputs traceable. More precisely, cascade effect attack utilizes the fact that the zero-mixins inputs will affect the traceability of other inputs who pick those public-keys of them as mixins. In other words, this attack bases on the set of previous traceable inputs to track the remaining anonymous ones, while our attack can start from any anonymous input.

To better explain our attack, we give a brief example below. Here we consider four inputs included in transactions $\{tx_i\}_{i\in[4]}$ and assume that there are four distinct public-keys $\{pk_j\}_{j\in[4]}$ included in the input sets of them, i.e.,

$$
\begin{aligned}
tx_1.in &= \{pk_1, pk_2, pk_3\};\\
tx_2.in &= \{pk_2, pk_3\};\\
tx_3.in &= \{pk_1, pk_3\};\\
tx_4.in &= \{pk_1, pk_2, pk_3, pk_4\}.
\end{aligned}
$$

Note that, there must exist no other transaction input who is only composed

of public-keys among $\{pk_j\}_{j\in[4]}$. Otherwise, the design principle of Monero that one output can only be redeemed once will be broken. While the original cascade attack [MMLN17, KFTS17] does not work here, since there exists no 0-mixin input.

Although we can not make all aforementioned inputs traceable, but we can trace the real-spend of one of them. Specifically, consider the set $S = \{tx_i.in\}_{i\in[3]}$. Among that, the union set of all distinct public-keys included is $\{pk_1, pk_2, pk_3\}$. Clearly, the size of S equals to the number of distinct public-keys included in it such that it is a *closed set*. Since each output can be spent once only, then the output pk_j ($j \in [3]$) must be a real-spend in a certain $tx_j (j \in [3])$. In this way, we can deduce that the real-spend of tx_4 must be pk_4.

A Naive Implementation. A naive method to find all *closed set*s is to visit all possible subsets of transaction inputs. For each visited subset, we check whether it is a *closed set* by comparing the number of inputs and the number of distinct public-keys included in it. If yes, we further conduct the removing and tracing operations triggered by this *closed set*. Otherwise, continue the process until all subsets have been visited. Due to the space limitation, we give this algorithm below.

Algorithm 1 Subset-Searching Algorithm

Let DataSet be the set of all transaction inputs in the blockchain.
Let ℓ be the size of current subset, and $\ell \geq 2$.
Cascade-Effect(Dataset).
while $\ell \leq |DataSet|$ **do**
    Let $Set_\ell \subseteq DataSet$ be the set of all inputs, s.t., the size of each input is smaller than ℓ.
    Let $\{Subset_{\ell,j}\}$ be all subsets of Set_ℓ with size ℓ, where $j \in \mathcal{C}^{\ell}_{|Set_\ell|}$, and each $Subset_{\ell,j} = \{R_1, R_2, \ldots, R_\ell | \ \forall i \in [\ell], R_i \in Set_\ell\}$
    for $j = 1$ to $\mathcal{C}^{\ell}_{|Set_\ell|}$ **do**

        if $Subset_{\ell,j}$ is a *closed set* **then**
            Remove$(Subset_{\ell,j}) \rightarrow flag$
            if flag == true **then**
                find traceable inputs
            end if
        end if

    end for
    goto while with ℓ++
end while

Theoretically, this algorithm can find all *closed set* included in the set of transaction inputs. However, it is expensive to implement in reality, since the complexity of traversing all subsets of inputs is $\subseteq(2^m)$, where m is the total number of all transaction inputs included in the blockchain. For instance, until the block 1541236 of Monero, the number of inputs in all blocks is 6835530 after removing all inputs of ring size 1 and those affected via cascade effect. Starting from 6835530 inputs, at the step of searching subsets of size 5, the complexity of the algorithm will become $\mathcal{O}(2^{100})$.

Analysis of Closed Set Attack. We prove that our *closed set* attack is optimal. In other words, our *closed set* attack is equivalent to the currently known perfect attack, namely, brute-force attack. Specifically, we prove that after the execution of our *closed set* attack, each transaction input is the set of the possible candidates of real-spends for it found by brute-force attack. This is to say, for each input, each public-key remaining inside must belong to a valid sequence found by brute-force attack. The analysis is concluded by the following theorem.

Theorem 5.2.1. *The aforementioned closed set attack is equivalent to the brute-force attack. In other words, for any set of transactions, the impact of our attack on it is identical to the impact of brute-force attack.*

Before giving the proof, we first introduce some notations used throughout the proof. Assume m be the total number of transaction inputs, and n be the total number of all distinct public-keys included in all inputs. We use S to denote the sequence of distinct public-keys, and we call it as sequence for short. Moreover, $S[i]$ represents the *i-th* ($i \in [|S|]$) member in S. Let $\mathbb{A}_n^m$ be the set of all sequence of n public-keys taken m at a time without duplication. We use R to denote a set of public-keys, which is also known as a transaction input, i.e., $R = \{pk_1, \ldots, pk_\ell\}$. Additionally, $\boldsymbol{\mathcal{R}}$ is used to denote the set of all transaction inputs, namely, $\boldsymbol{\mathcal{R}} = \{R_1, R_2, \ldots, R_m\}$.

Definition 5.2.1. *Assume $\mathcal{L}$ be a relation. For any sequence $S \in \mathbb{A}_n^m$, and any set of inputs $\boldsymbol{\mathcal{R}} = \{R_1, R_2, \ldots, R_m\}$, we say $(S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$ if and only if $\forall i \in [m], S[i] \in R_i$.*

Definition 5.2.2. *For a set of rings $\boldsymbol{\mathcal{R}} = \{R_1, R_2, \ldots, R_m\}$, if for any $Q \subseteq [m]$, we have*

$$|\cup_{i \in Q} R_i| \geq |Q|, \tag{5.1}$$

then we call $\boldsymbol{\mathcal{R}}$ as a good set.

Lemma 5.2.1. *For any good set $\boldsymbol{\mathcal{R}} = \{R_1, R_2, \ldots, R_m\}$, there exists a sequence S such that $(S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$.*

Proof. We use proof by induction. Firstly, in the case of $m = 1$, $\boldsymbol{\mathcal{R}} = \{R_1\}$, $|R_1| \geq 1$. Hence, there must exist a sequence S, such that $(S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$.

While for the case $m \geq 2$, we assume that the above lemma holds for good set with size $m = x$, then we show that it would also be true for good set with size $m = x + 1$. Let $\boldsymbol{\mathcal{R}}' = \{R_1, R_2, \ldots, R_x, R_{x+1}\}$ be a good set with size $(x + 1)$, and $\boldsymbol{\mathcal{R}} = \{R_1, R_2, \ldots, R_x\}$ be a subset of $\boldsymbol{\mathcal{R}}'$. Since $\boldsymbol{\mathcal{R}}'$ is a good set, we can easily deduce that $\boldsymbol{\mathcal{R}}$ is also a good set. To complete the part when $m = x + 1$, we need to utilize the following Claim.

Claim 5.2.3. *Suppose $R_m = \{pk_1, pk_2, \ldots, pk_k\}$ be the last input in $\boldsymbol{\mathcal{R}}'$, where $m = x + 1$, and $\boldsymbol{\mathcal{R}}'$, $\boldsymbol{\mathcal{R}}$ are defined above. For set $\boldsymbol{\mathcal{R}}$, there exists $j \in [k]$ such that $\boldsymbol{\mathcal{R}}^{(j)} = \{R_1 - \{pk_j\}, R_2 - \{pk_j\}, \ldots, R_x - \{pk_j\}\}$ is also a good set.*

Proof. We exploit proof by contradiction. Assume that for all $j \in [k]$, any $\boldsymbol{\mathcal{R}}^{(j)}$ is not good set. In the remainder of this proof, we use $R_i^{(j)}$ to denote the set $R_i - \{pk_j\}$, and obviously $R_i^{(j)} \in \boldsymbol{\mathcal{R}}^{(j)}$.

Let $\mathsf{Bad}_j \subseteq [x]$ be a set such that

$$|\cup_{i \in \mathsf{Bad}_j} R_i^{(j)}| \leq |\mathsf{Bad}_j| - 1. \tag{5.2}$$

Meanwhile, as $\boldsymbol{\mathcal{R}}$ is good, hence we can get

$$|\cup_{i \in \mathsf{Bad}_j} R_i| \geq |\mathsf{Bad}_j|. \tag{5.3}$$

Since $R_i^{(j)} = R_i - \{pk_j\}$, we can get that

$$|\cup_{i \in \mathsf{Bad}_j} R_i| \leq |\cup_{i \in \mathsf{Bad}_j} R_i^{(j)}| + 1. \tag{5.4}$$

Inequality (5.2) holds due to the assumption that any $\boldsymbol{\mathcal{R}}^{(j)}$ is not good set, where $j \in [k]$. Hence, there must exist a set of input's index Bad_j such that those inputs selected by Bad_j do not form a good set. Since $\boldsymbol{\mathcal{R}}$ is a good set, hence for any Bad_j, $|\cup_{i \in \mathsf{Bad}_j} R_i| \geq |\mathsf{Bad}_j|$. The reason why inequality (5.4) holds can be divided into two cases. On one case, if for all $i \in \mathsf{Bad}_j$, R_i doesn't include public-key pk_j, then we can get that $|\cup_{i \in \mathsf{Bad}_j} R_i| = |\cup_{i \in \mathsf{Bad}_j} R_i^{(j)}|$, hence $|\cup_{i \in \mathsf{Bad}_j} R_i| < |\cup_{i \in \mathsf{Bad}_j} R_i^{(j)}| + 1$. On the other case, if there exists

at least one R_i ($i \in \mathsf{Bad}_j$) including pk_j, then we can get that $|\cup_{i\in\mathsf{Bad}_j} R_i| = |\cup_{i\in\mathsf{Bad}_j} R_i^{(j)}| + 1$.

Considering the combination of inequalities (5.2, 5.3, 5.4,), it is not hard to get

$$|\cup_{i\in\mathsf{Bad}_j} R_i| = |\mathsf{Bad}_j|, \textit{and } pk_j \in \cup_{i\in\mathsf{Bad}_j} R_i. \tag{5.5}$$

Until now, for each $j \in [k]$, we find a bad indexes' set Bad_j such that the inputs set composed by those inputs whose index is in Bad_j is not a good set.

Then consider the set

$$U_h = \cup_{j\in[h]} \mathsf{Bad}_j, \tag{5.6}$$
$$V_h = \cup_{i\in U_h} R_i, \tag{5.7}$$

where $0 \leq h \leq k$.

When h becomes $(h+1)$, the size variances of these two sets are given below:

$$|U_{h+1}| = |U_h| + |\mathsf{Bad}_{h+1}| - |Repeat_{h+1}|$$
$$|V_{h+1}| \leq |V_h| + |\cup_{i\in\mathsf{Bad}_{h+1}} \mathsf{R}_i| - |\cup_{i\in\mathsf{Repeat}_{h+1}} \mathsf{R}_i|,$$

where $Repeat_{h+1} \subseteq \mathsf{Bad}_{h+1}$ is the set of replicate indexes introduced by Bad_{h+1}.

Recall that $\boldsymbol{\mathcal{R}}$ is a good set, hence

$$|Repeat_{h+1}| \leq |\cup_{i\in\mathsf{Repeat}_{h+1}} \mathsf{R}_i|,$$

as $Repeat_{h+1}$ is a subset of $[x]$.

Combining with the facts that $|U_0| = |V_0| = 0$, and $|\mathsf{Bad}_{h+1}| = |\cup_{i\in\mathsf{Bad}_{h+1}} \mathsf{R}_i|$, finally, we can get that

$$|U_k| \geq |V_k|.$$

Besides, since $\{pk_1, pk_2, \ldots, pk_k\} \subseteq V_k$, then we set

$$U = U_k \cup \{m\}$$
$$V = V_k \cup R_m.$$

Then $|V| = |V_k|$, and $|U|= |U_k|$ +1. In this way, we can deduce that

$$|U| > |V|. \tag{5.8}$$

However, inequality (5.8) contradicts the fact that $\boldsymbol{\mathcal{R}}'$ is a good set. □

Based on the above **Claim** 5.2.3, we know that there must exist a $j^* \in [k]$ such that $\boldsymbol{\mathcal{R}}^{(j^*)} = \{R_1 - \{a_{j^*}\}, R_2 - \{a_{j^*}\}, \ldots, R_x - \{a_{j^*}\}\}$ is a good set. According to our assumption, there exists a sequence S_x of public-keys with length x such that $(S_x, \boldsymbol{\mathcal{R}}^{(j^*)}) \in \mathcal{L}$, where $a_{j^*} \notin S_x$ since each input included in $\boldsymbol{\mathcal{R}}^{(j^*)}$ does not include a_{j^*}.

Next, we append S_x with a_{j^*} which becomes $S_x||a_{j^*}$. Then for set $\boldsymbol{\mathcal{R}}$, we can get that $(S_x||a_{j^*}, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$, since $\boldsymbol{\mathcal{R}}$ can be equivalently written as $\boldsymbol{\mathcal{R}}^{(j^*)} \cup \{R_m\}$. Then we complete the part of the proof when $m = x + 1$, and also complete the proof of this lemma. □

Let $\boldsymbol{\mathcal{R}} = \{R_1, R_2, \ldots, R_m\}$ be the original dataset, and assume $PKS = \{pk_1, pk_2, \ldots, pk_n\}$ be the set of all public-keys included in the inputs. Then for all $i \in [m]$, $R_i \subseteq PKS$. Let $\boldsymbol{\mathcal{R}}' = \{R'_1, R'_2, \ldots, R'_m\}$ be the set of inputs which has been removed all public-keys included in the *closed set*. For simplicity, we use $\boldsymbol{\mathcal{R}}' = Closed\_Set\_Attack\ (\boldsymbol{\mathcal{R}})$ to denote the process that our *closed set* attacks algorithm takes $\boldsymbol{\mathcal{R}}$ as input, and returns the result $\boldsymbol{\mathcal{R}}'$.

Based on the aforementioned definitions, we can get the following facts after the execution of our *closed set* attacks.

1. **Fact 1:** for all $i \in [m]$, $R'_i \subseteq R_i$;

2. **Fact 2:** for any removed public-key $pk \in R_i - R'_i$, there exists a set $T \subseteq [m]$ such that

$$|T| = |\cup_{j \in T} R_j| \wedge pk \in (\cup_{j \in T} R_j) \wedge i \notin T.$$

3. **Fact 3:** for any $pk \in R'_i$, and any set $T \subseteq [m]$ such that $|T| = |\cup_{j \in T} R_j|$, then

$$i \in T \qquad or \qquad pk \notin (\cup_{j \in T} R'_j).$$

Additionally, for any set $\boldsymbol{\mathcal{R}}$, if there exists a set $T \subseteq [m]$ that $|T| = |\cup_{j \in T} R_j|$, then for any public-key sequence S that $(S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$, we have

the following fact, and name it as **Fact 4**.

$$\{S[j]\}_{j\in T} = \cup_{j\in T} R_j.$$

Fact 4 states that for any sequence S and set $\boldsymbol{\mathcal{R}}$ satisfying the relation $\mathcal{L}$, and for any found closed set $\{R_j \in \boldsymbol{\mathcal{R}}\}_{j\in T}$, the $j\text{-}th(j \in T)$ element of sequence S, i.e., $S[j]$, must be a public-key included in $\cup_{j\in T} R_j$.

Corollary 5.2.4. *For any set $\boldsymbol{\mathcal{R}}$, if there exists a set $T \subseteq [m]$ that $|T| = |\cup_{j\in T} R_j|$, then for any sequence S that $(S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$, we have*

$$\{S[j]\}_{j\notin T} \cap \cup_{j\in T} R_j = \emptyset.$$

This is a corollary of **Fact 4**. According to our algorithm, once finding a closed set, we will remove all public-keys included in this *closed set* from the remaining transaction inputs. In addition, since every public-key can only be redeemed once, hence each public-key included in the *closed set* can only be redeemed by a certain input of this closed set. For all other inputs outside the *closed set*, its real input must not be any public-key of the *closed set*. As we use S to denote the sequence of the real-input of each transaction, hence for any index $j \notin T$, $S_j \notin (\cup_{j\in T} R_j)$.

Analysis of Correctness. To analyze the correctness of our *closed set* attack, we give the following theorem, which states the fact that our attack algorithm will not affect the real-spend of each input. In other words, any candidate of the real-spend of each input will remain the same after running our attack algorithm.

Theorem 5.2.2. *For any set $\boldsymbol{\mathcal{R}}$, and the $\boldsymbol{\mathcal{R}}' = Closed\_Set\_Attack(\boldsymbol{\mathcal{R}})$, for any sequence S, we have the following equivalent relationship,*

$$(S, \boldsymbol{\mathcal{R}}') \in \mathcal{L} \Longleftrightarrow (S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}.$$

Proof. The proof consists of two parts.

Part 1: $(S, \boldsymbol{\mathcal{R}}') \in \mathcal{L} \Longrightarrow (S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$ For any sequence S such that $(S, \boldsymbol{\mathcal{R}}') \in \mathcal{L}$, we have that for any $i \in [m]$, we have $S[i] \in R'_i$. Since $R'_i \subseteq R_i$ according to the aforementioned requirements, we get $S[i] \in R_i$ for all $i \in [m]$. In this way, we prove that $(S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$.

Part 2: $(S, \boldsymbol{\mathcal{R}}') \in \mathcal{L} \Longleftarrow (S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$ This part is proved by contradiction. Assume there exists a sequence S such that $(S, \boldsymbol{\mathcal{R}}) \in \mathcal{L}$ but $(S, \boldsymbol{\mathcal{R}}') \notin \mathcal{L}$. Then we have that $\exists i^* \in [m]$ such that $S[i^*] \in R_{i^*}$ and $S[i^*] \notin R'_{i^*}$.

We assume that $S[i^*] = pk^*$. There is no doubt that pk^* is removed from R_{i^*} by our *closed set* attack algorithm, i.e., $pk^* \in R_{i^*} - R'_{i^*}$. According to **Fact** 2, we get that there exists a set $T \subseteq [m]$, such that

$$|T| = |\cup_{j \in T} R_j| \wedge pk^* \in (\cup_{j \in T} R_j) \wedge i^* \notin T. \tag{5.9}$$

Hence, we can get $\exists i^* \in T$, but $\{S[i^*]\} \in (\cup_{j \in T} R_j)$ (i), which contradicts **Corollary** 5.2.4. This completes the proof of our correctness. □

Analysis of Optimality. Next we argue that after the execution of our *closed set* attack algorithm, i.e., $\boldsymbol{\mathcal{R}}' = Closed\_Set\_Attack(\boldsymbol{\mathcal{R}})$, for any input $R' \in \boldsymbol{\mathcal{R}}'$, every public-key included in it might be the real-spend. There is no extra useless public-key which can be removed from R'_i, where $i \in [m]$. In other words, any remaining public-key in R'_i $(\forall i \in [m])$, there is a sequence returned by the brute-force attack whose i-th element is that.

Theorem 5.2.3. *For any* $\boldsymbol{\mathcal{R}} = \{R_1, R_2, \ldots, R_m\}$, *let* $\boldsymbol{\mathcal{R}}' = Closed\_Set\_Attack(\boldsymbol{\mathcal{R}})$, *where* $\boldsymbol{\mathcal{R}}' = \{R'_1, R'_2, \ldots, R'_m\}$. *Then for any* $i \in [m]$, *and any* $pk^* \in R'_i$, *there exists a sequence* S *that*

$$S[i] = pk^* \wedge (S, \boldsymbol{\mathcal{R}}') \in \mathcal{L}.$$

Proof. We adopt the proof by contradiction. Assume there does not exist such sequence S^* among the result of brute-force attack for a public-key pk^*, where $pk^* \in R'_{i^*}$. In other words, for any sequence S satisfying the relation $(S, \boldsymbol{\mathcal{R}}') \in \mathcal{L}$, we have $S[i^*] \neq pk^*$. Then we consider the following two cases.

1. pk^* is contained in a *closed set* $\cup_{j \in T} R'_j$. In this case, i^* must belong to T^*, otherwise, pk^* will be removed from R'_{i^*} according to our attack algorithm. Without loss of generality, we assume the *closed set* $\{R'_j\}_{j \in T^*}$ is an indivisible *closed set* among all *closed set* including R'_{i^*}. As $\{R'_j\}_{j \in T^*}$ is indivisible, which means that any subset of this *closed set* will not be a *closed set*. Hence for any subset of $\{R'_j\}_{(j \in T^* \wedge j \neq i^*)}$, it will not be a *closed set*, which can be written as

$$|\cup_{j \in T'} R'_j| > |T'|, \tag{5.10}$$

where $T' \subseteq T^* \setminus \{i^*\}$.

Next we delete pk^* from each input R'_j, where $j \in T'$. Then inequality (5.10) will become

$$|(\cup_{j \in T'} R'_j) \setminus \{pk^*\}| \geq |T'|. \tag{5.11}$$

According to **Lemma** 5.2.1, we get that there exists a sequence S^* such that $(S^*, \{R'_j\}_{j \in (T^* \setminus \{i^*\})}) \in \mathcal{L}$, and $pk^* \notin S^*$. Whereas, if we combine S^* with pk^*, we can find a valid sequence $S^* \cup \{pk^*\}$ for $\{R'_j\}_{j \in T^*}$, namely, $(S^* \cup \{pk^*\}, \{R'_j\}_{j \in T^*}) \in \mathcal{L}$. However, this contradicts the assumption that there does not exist a sequence S, such that $(S, \boldsymbol{\mathcal{R}}') \in \mathcal{L}$ and $S[i^*] = pk^*$.

2. pk^* is not included in any *closed set*. We can deduce that $\boldsymbol{\mathcal{R}}'$ is not a *closed set* since pk^* is included in $\boldsymbol{\mathcal{R}}'$. Then for any subset $T \subseteq ([m] \setminus \{i^*\})$, we get that

$$|\cup_{j \in T} R'_j| > |T|. \tag{5.12}$$

Next we remove pk^* from each input R'_j whose index is included in T, i.e., $j \in T$. In this way, inequality (5.12) becomes

$$|\cup_{j \in T} (R'_j \setminus \{pk^*\})| \geq |T|. \tag{5.13}$$

According to **Lemma** 5.2.1, we get that there exists a sequence S^* such that $(S^*, \{R'_j\}_{j \in ([m] \setminus \{i^*\})}) \in \mathcal{L}$, and $pk^* \notin S^*$. Whereas, if we combine S^* with pk^*, we can find a valid sequence $S^* \cup \{pk^*\}$ for $\{R'_j\}_{j \in T^*}$, namely, $(S^* \cup \{pk^*\}, \{R'_j\}_{j \in [m]}) \in \mathcal{L}$. However, this contradicts the assumption that there does not exist a sequence S, such that $(S, \boldsymbol{\mathcal{R}}') \in \mathcal{L}$ and $S[i^*] = pk^*$.

This completes the proof. □

In summary, we show that for each transaction input, its candidates for real-spend found by brute-force attack and our *closed set* attack are identical. That is to say that our proposed *closed set* attack is equivalent to brute-force attack, both returns perfect results.

5.2.3 On The Existence of Closed Set: A Theoretical Perspective

As mentioned before, the *closed set* attack is optimal as brute-force attack. This is to say, we can conclude that anonymity of inputs cannot be reduced if no *closed set* exists. In this section, we estimate the probability that there exists at least one *closed set* in an ideal scenario, namely, all inputs have a (small) constant number of mixins and all mixins are selected uniformly from all keys.

More concretely, we consider a scenario that

- There are $6 \cdot 2^{20}$ inputs, with $6 \cdot 2^{20}$ real-spend public-keys;
- There are also additional 25% (i.e. $2 \cdot 2^{20}$) unspent public-keys;
- Each input has 3 mixins;
- Each mixin is sampled uniformly from all $8 \cdot 2^{20}$ keys;

where the first two conditions come from the real data of Monero after cascade attack, and the third condition is based on the fact that the average ring size after the cascade attack is 4.62.

Lemma 5.2.2. *With all but a small probability 2^{-19}, there does not exist any closed set in the above dataset if all inputs have 3 mixins and all mixins are sampled uniformly from all keys.*

Proof. Before analyzing the probability, we first define the notion of "dirty key" and show a relation between it and the existence of *closed set*.

Definition 5.2.5. *We define dirty key recursively:*

- *An unspent key is a dirty key.*
- *If an input chooses a dirty key as its mixin, then the real-spend key of this input is also a dirty key.*

We call an unspent key a level-0 dirty key and call a dirty-key a level-i dirty key if there exists a level-$(i-1)$ dirty key in its mixin and there does not exists a level-j dirty key in its mixin for $j < i-1$.

Lemma 5.2.3. *A closed set does not include a dirty key.*

Proof. First, as all keys in a *closed set* must be spent in one input in this *closed set*, a level-0 dirty key, i.e., an unspent key, cannot be included in a *closed set*.

Now, we assume there does not exist level-i dirty key in a *closed set*, and prove there does exist level-$(i+1)$ dirty key in a *closed set*. Also, as all keys in a *closed set* must be spent in one input in this *closed set*, a *closed set* with a key pk must include the input that pk is its real-spend. Thus, a *closed set* including a level-$(i+1)$ dirty key must also include a level-i dirty key, which contradicts the assumption.

That completes the proof of Lemma 5.2.3. □

Next, we bound the probability that there exists a *closed set* in the aforementioned scenario. The main idea is to show that those 25% unspent key, a.k.a., level-0 dirty key, will ultimately pollute all keys with a large probability.

Lemma 5.2.4. *With all but a small probability ($1/2^{19}$), all $8 \cdot 2^{20}$ keys are dirty keys.*

More precisely, let $\mathcal{K}$ be all $8 \cdot 2^{20}$ keys, including all real-spent and unspent keys, let $\mathcal{D}_i$ be the set of level-i dirty keys and let $\mathcal{K}_i = \mathcal{K} - \bigcup_{j=0}^{i-1} \mathcal{D}_j$ (We also use $\mathcal{K}_i$ to denote inputs using keys in $\mathcal{K}_i$ as its real-spent key for simplicity of notation), then with all but a small probability, namely, $1/2^{19}$, we have:

1. *At least $1/3$ of all public-keys in $\mathcal{K}_1$ are level-1 dirty keys.*
2. *At least $1/2$ of all public-keys in $\mathcal{K}_2$ are level-2 dirty keys.*
3. *At least $3/4$ of all public-keys in $\mathcal{K}_3$ are level-3 dirty keys.*
4. *At least $31/32$ of all public-keys in $\mathcal{K}_4$ are level-4 dirty keys.*
5. *All keys in $\mathcal{K}_5$ are either level-5 dirty keys or level-6 dirty keys.*

Proof. Recall that a key becomes level-i dirty key if it includes a level-$(i-1)$ dirty key in the mixin of its real-spent input. Thus, it is sufficient to analyse how many inputs will be affected by a level-$(i-1)$ dirty key.

First, for each of the $6 \cdot 2^{20}$ inputs, the probability that all three mixins are chosen from real-spent keys is $27/64$. Then by the Chernoff bound, the probability that there are more than two thirds inputs, namely, 2^{22}, does not include a level-0 dirty key is:

$$\Pr[X \geq (1 + \frac{94}{162}) \cdot \frac{27}{64} \cdot 6 \cdot 2^{20}] \leq e^{-2^{18}}.$$

Thus, the probability $\mathfrak{p}_1$ that less than $1/3$ real-spend keys, a.k.a., keys in $\mathcal{K}_1$, are level-1 dirty key is no more than $e^{-2^{18}}$.

Next, let $\mathfrak{p}_i$ be the conditional probability that less than q_i fraction of keys in $\mathcal{K}_i$ are level-i dirty key for $i \in [2,4]$ (under condition that statement 1 to $i-1$ enumerated in the description of Lemma 5.2.4 holds), where $q_2 = 1/2$, $q_3 = 3/4$, $q_4 = 31/32$. Via a similar analysis, we have $\mathfrak{p}_2 \leq e^{-2^{17}}$, $\mathfrak{p}_3 \leq e^{-2^{16}}$ and $\mathfrak{p}_4 \leq e^{-2^{13}}$.

Now, under the condition that statement 1 to 4 holds, which fail with a probability that no more than $\mathfrak{p}_1 + \mathfrak{p}_2 + \mathfrak{p}_3 + \mathfrak{p}_4 < e^{-2^{12}}$, we prove statement 5. As statement 1 to 4 holds, there are at most 2^{14} inputs remaining in $\mathcal{K}_5$, all of which will use keys in $\mathcal{K}_4$, which is at least 32 times larger than $\mathcal{K}_5$, as its mixins. Next, we analyze the probability $\mathfrak{p}_5$ that there exists key in $\mathcal{K}_5$ that is neither a level-5 dirty key nor a level-6 dirty key. First, for each input, the mixins can be viewed as sampled uniformly from all keys from $\mathcal{K}_4$, thus the probability that all three mixins are chosen from $\mathcal{K}_5$ is $1/2^{15}$. Next, we consider the following two cases:

- There are less than 2^{11} keys in $\mathcal{K}_5$. Then, by the union bound, the probability that there exists 4 inputs that chooses mixins from $\mathcal{K}_5$ is less than $\binom{2^{11}}{4}/2^{15\cdot 4} \leq 2^{-20}$. That is to say, with probability $1 - 2^{-20}$, at most 3 keys are in $\mathcal{K}_6$. Since each input needs to choose three other distinct keys as its mixins, all these three keys in $\mathcal{K}_6$ must be level-6 dirty keys.

- There are at least 2^{11} but at most 2^{14} keys in $\mathcal{K}_5$. Then, by the union bound, the probability that there exists 9 inputs that chooses mixins from $\mathcal{K}_5$ is less than $\binom{2^{14}}{9}/2^{15\cdot 9} \leq 2^{-27}$. That is to say, with probability $1 - 2^{-27}$, at most 8 keys are in $\mathcal{K}_6$. For each input in $\mathcal{K}_6$, the mixins can be viewed as sampled uniformly from all keys from $\mathcal{K}_5$, which has at least 2^{11} keys, thus the probability that all three mixins are chosen from $\mathcal{K}_6$ is $2^9/2^{33} = 2^{-24}$. Then, by the union bound, the probability that there exists at least one key that is not polluted by level-5 dirty keys is at most $8 \cdot 2^{-24} = 2^{-21}$.

To summarize, with probability $1 - 2^{-20}$, all keys in $\mathcal{K}_5$ are either level-5 dirty keys or level-6 dirty keys under condtion that statement 1 to 4 holds. Therefore, the probability that all 5 statements hold is at least $1 - (\mathfrak{p}_1 + \mathfrak{p}_2 + \mathfrak{p}_3 + \mathfrak{p}_4 + \mathfrak{p}_5) \geq 1 - 2^{-19}$. □

By combining Lemma 5.2.3 and Lemma 5.2.4, we can conclude that with all but a small probability 2^{-19}, there does not exist a close set in Monero if

all inputs have 3 mixins and all mixins are sampled uniformly from all keys. To some degree, we believe this analysis give the reason why our clustering algorithm can only find 3017 *closed set*s.

□

5.3 Our Clustering Algorithm

Considering the impracticability of subset-based algorithm mentioned above, here we introduce an approximate but efficient algorithm for searching *closed sets*, which is named as clustering algorithm. Looking ahead, although the clustering algorithm is just an approximate algorithm, we show that the provably lower bound of the size of *closed set* found by it is 5. In other words, for all *closed set* with size smaller than 5 can be found. Besides, we also conduct experiments and find that our clustering algorithm achieves a better result during the actual running.

Intuition of Our Clustering Algorithm. Recall that, the main feature of a *closed set* is that it embraces the same number of transaction inputs and distinct public-keys. Hence, our target should be finding a set of inputs with the above characteristics. To do so, one intuitive way is forming a set from a certain input, then absorb other input which is helpful to achieve a *closed set*. A key challenge is how to select other rings or what is the selection criteria?

We observe that since the ultimate target is to make two numbers about this set equal, it is possible to select rings based on the consequence of adding an input into a set. For instance, assuming the set being considered now is called S, which is initialized by input R. Whenever an input R' is added into S, the possible consequences can be divided into the following three cases:

- Case 1. If all public-keys included in R' are a subset of all public-keys contained in S, then for set S, the number of included transaction inputs is increased by one, and the number of distinct public-keys remains the same. Thus, the insertion of R' will certainly increase the possibility of S becoming a *closed set*. We call such kind of input as useful input.

- Case 2. If the insertion of R' will only introduce one distinct public-key to S, then the insertion of this input will not change the current

relationship between the number of distinct public-keys and the number of inputs included in S. This kind of input extends the public-key set of S, which maybe helpful for absorbing other inputs. We call such kind of input as uncertain input.

- Case 3. If the insertion of R' will introduce two or more distinct public-keys to S, then the number of inputs will only be increased by one, but the number of public-keys will be increased by 2 or more. As this does not help our analysis at all, we call such kind of input as bad input.

Above all, if we only pick the relatively useful and uncertain inputs to a set, then we can find a *closed set* faster with high probability.

Definition of Cluster. A cluster $Clus$ is defined as a set of inputs, namely, $Clus = \{R_1, R_2, \ldots, R_n\}$. Each cluster represents a set PK_Clus, which is defined as $PK\_Clus = \bigcup_{R \in Clus} R$. In other words, PK_Clus is the set used to collect all distinct public-keys included in the inputs in the cluster $Clus$.

The *distance* from an input to a cluster is defined as the number of public-keys included in the input but not in the cluster. The formal definition of it is given below:

$$Dist(R, Clus) = Dist(R, PK\_Clus) = |R| - |PK\_Clus \cap R|,$$

where R is the input considered to be added, and $Clus$ is a cluster with public-keys set PK_Clus. Notably, this definition is not symmetric. According to our definition, the distance from an input to a cluster, i.e., $Dist(R, Clus)$, is different with the distance from a cluster to an input, i.e., $Dist(Clus, R)$.

For instance, consider the cluster $Clus$ and the input R composed as follows:

$$Clus = \{\{pk_1, pk_2\}, \{pk_1, pk_3\}, \{pk_2, pk_4\}\},$$
$$R = \{pk_1, pk_3, pk_5\}.$$

Obviously, the public-key set of $Clus$ is $PK\_Clus = \{pk_1, pk_2, pk_3, pk_4\}$. The size of R is 3, and number of common public-keys are 2. Hence, according to our definition, the distance from R to $Clus$ is $Dist(R, Clus) = Dist(R, PK\_Clus) = 3 - 2 = 1$. So, if we add R into $Clus$, then only one new public-key, i.e., pk_5, will be introduced in $Clus$.

Starting from a specific input, the construction of a cluster is a dynamic process of searching for other qualified inputs. To clarify how an input can

be added into a cluster, we associate each cluster with a distance. More precisely, we say a cluster $Clus$ with distance 1, if for all inputs which can be added into this cluster, it satisfies that $Dist(R, Clus) \leq 1$. Actually, for a cluster, insertion of an input into it will cause changes to it. Hence, we should always adopt the present cluster to calculate the distance from an input to it. The algorithm for constructing a cluster from a certain input is given in **Algorithm** 2.

Algorithm 2 $Cluster\_Form$(R)

1: Start with an input R, and define the cluster as $Clus = \{R\}$
2: Let $DataSet$ be all transaction inputs in the blockchain
3: **for** each $R'(\neq R) \in DataSet$ **do**
4: **if** $Dist(R', Clus) \leq 1$ **then**
5: $Clus = Clus \cup \{R'\}$
6: **end if**
7: **end for**
8: **return** $Clus$

For each cluster, we use two additional parameters to check whether it is a *closed set*. One is the number of inputs included in it, the other one is the number of distinct public-keys included. Formally, if the number of inputs equals to the number of distinct public-keys included in a cluster, we say that this cluster is a *closed set*. Besides, in some cases, a *closed set* may contain other sub-*closed set*. To find all *closed set*s, whenever we get a *closed set* via this algorithm, we further conduct a sub-*closed set* searching operation. An important observation we discovered is that if a public-key only appears once in a *closed set*, then it must be the real spend of the input including it. For simplicity, we utilize this method to test whether there exists sub-*closed set* inside a *closed set*, since the complexity of brute-forcing all subsets of this *closed set* is quite large.

Next we introduce the clustering algorithm for all clusters with distance 1. The main idea is that we repeatedly pass over all the transaction inputs via numerous iterations. In each iteration, the algorithm picks an input and uses it to initialize a cluster $Clus$. Then we run the constructing cluster algorithm (Algorithm 2) to add proper inputs into $Clus$. Next the algorithm checks whether the resulted cluster is a *closed set*. If not, continue with the next iteration. Otherwise, before continuing with the next iteration, the algorithm should finish the following operations. Remove all public-keys contained in this cluster from the remaining inputs, and find the set of traceable ones. Afterwards, we check whether the current *closed set* includes a

public-key such that it only appears in one input. If yes, we further de-anonymize inputs inside a *closed set*.

The algorithm of searching for all clusters with distance 1 from all transaction inputs in the blockchain is given in **Algorithm** 3. Notably, all rings considered in our algorithm are anonymous. Once finding the real-spend of an input, we will not do any operation on that input. Besides, our algorithm concentrates on resulted data after the execution of cascade effect attack. Hence, in **Algorithm** 3, we abuse the concept, where a cascade effect algorithm is first invoked.

Algorithm 3 Clustering Algorithm

```
1: Let DataSet be all transaction inputs in the blockchain.
2: Cascade-Effect(Dataset)
3: Flag = true
4: while Flag == true do
5:     Flag = false
6:     for each R ∈ DataSet do
7:         Clus_Form(R) → Clus
8:         if Clus is a closed set then
9:             Remove(Clus) → Flag
10:            if Flag == true then
11:                find traceable inputs
12:                check whether rings inside Clus are traceable
13:            end if
14:            end if
15:        end for
16:    end while
```

Analysis of accuracy. The accuracy of the clustering algorithm is analyzed through the following lemma, which gives a lower bound of the clustering algorithm. This is to say that all *closed set*s with size smaller than 5 can be found after the execution of the clustering algorithm.

Theorem 5.3.1. *After the execution of our clustering algorithm with searching distance 1, all indivisible closed sets with size at most 5 can be returned by our algorithm.*

Proof. Here we only consider whether our algorithm can find all *closed set*s which do not contain any sub-*closed set*. We call such kind of *closed set* as indivisible *closed set*.

The reason for this can be divided into two aspects. On one hand, the search result of a *closed set* with sub-*closed set* relies on the result of its indivisible sub-*closed set*s. Without loss of generality, assume a *closed set* CoS_1

can be divided into two parts, one is an indivisible sub-*closed set* named as $Cos_{1,1}$, another part is a cluster $Clus_{1,2}$. In one case, if we can not find the sub-*closed set* $Cos_{1,1}$, then consider the search result for $Cos_{1,1}$ will become a smaller instance. While in the other case, if we can find sub-*closed set* $Cos_{1,1}$, then we will remove all public-keys contained in $Cos_{1,1}$ from $Clus_{1,2}$. The remaining part of $Clus_{1,2}$ become a smaller *closed set* again, and its search result will affect whether the algorithm can find CoS_1. We can repeat the above process to find the smallest indivisible *closed set*. On the other hand, if an indivisible *closed set* is found by our algorithm, and if the super *closed set* of it exists, our algorithm will always return it.

We use proof by contradiction. Assume that there exists a *closed set* CoS with size $N \leq 5$, which cannot be detected by our algorithm. Based on the definition of the *closed set*, there must be N inputs and N distinct public-keys included in CoS. Since CoS already contains all related inputs, we only need to consider those clusters initialized by the inputs included in CoS. In each resulted clusters, we only consider the existence of those inputs included in CoS. Because for any other input outside CoS, if it can be added into any cluster aforementioned, it only introduces at most one public-key, which has no effect on whether or not the part about CoS is a *closed set*.

In order to pick the so-called largest cluster, our selection criteria is stated as follows. If there exists one and only one input in CoS whose size is maximum, then we pick the cluster initialized by this input. Otherwise, we compare the size of all generated clusters, and pick the one who has the biggest number of inputs inside. We denote the chosen cluster as $Clus^*$. Suppose $Clus^*$ has m transactions and n distinct public-keys. Since we have assumed that all resulted clusters are not *closed set*s, hence $m < n$.

Obviously, $Clus^*$ cannot contain all inputs of CoS, otherwise, it is a *closed set*. For convenience, we call those inputs in CoS but not in $Clus^*$ as outsider inputs. Observe that any outsider input must have minimum distance 2 with $Clus^*$, otherwise it will be included into. In other words, the outsider inputs must occupy two distinct public-keys among all distinct public-keys in CoS. Then we can get that $n + 2 \leq N \leq 5$, which can be simplified as $n \leq 3$. Until now, the possible value for n can only be 2 or 3. Next we consider these two cases respectively.

- Case 1: $n = 2$. Since $m < n$, hence we can get that $m = 1$. This means that the $Clus^*$ only consists of one input with two distinct public-keys. Without loss of generality, we denote them as $\{pk_1, pk_2\}$. Recall the selection criteria of the so-called largest cluster. If there exist one and

only one input in *Cos* whose size is the biggest, then we pick the cluster initialized by it as $Clus^*$. Otherwise, we pick the cluster who has the biggest number of inputs in it. In other words, the size of all outsider inputs cannot be larger than 2. Since the distance from each outsider input to $Clus^*$ should be at least 2, then the public-keys pk_1 and pk_2 will never appear in any outsider input. However, in this situation, the combination of the input pk_1, pk_2 and all outsider inputs are not a *closed set*. which contradicts the fact that *Cos* is a *closed set*.

- Case 2: $n = 3$. As we require that $N \leq 5$, and $n + 2 \leq N$. Hence, we can get that $N = 5$. As $Clus^*$ has m inputs, then the number of outsider inputs is $5 - m$. Besides, the number of distinct public-keys included in the outsider inputs is 2, since 3 public-keys are included in $Clus^*$. As all clusters are not *closed set*s according to our assumptions, and consider the relationship between the number of outsider inputs and the number of distinct public-keys included in them. We can get $5 - m < 2$, which states that $m > 3$. Whereas $Clus^*$ requires that $m < n = 3$. Until now, we get a contradiction in this case.

Above all, we show that, for any atom *closed set* with size smaller than 5, our clustering algorithm can certainly find it.

In summary, we can prove that our algorithm can guarantee the successful searching for all *closed set*s whose size is smaller than 5. However, it is just a lower bound. Since during the actual running, we find that our algorithm also works well for searching those *closed set* with size more than 5.

□

Analysis of complexity. Assume the total number of transaction inputs included in the blockchain is N. The number of iterations in our algorithm is $\subseteq(N)$. Suppose the average length of an input is ℓ. While in each iteration, in the worst case, we calculate $\mathcal{O}(\ell N)$ times distance between all inputs and the current clusters. Therefore, in the worst case, the complexity is $\subseteq(\ell N^2)$.

5.4 Experiment Result

To test the level of anonymity achieved by the CryptoNote-style currencies, as well as the estimation of the probability of the existence of *closed set*s in reality, we implement our clustering algorithm in C++, and the program is

executed on a computer with 3.1 GHz Intel Core i5 Processor, 16 GB RAM and 256GB SSD storage disk. Notably, here we only analyze the top three CryptoNote-style currencies according to their market capitalizations [coi], i.e., Monero, Bytecoin and DigitalNote. For all these three currencies, we export all related data directly from the corresponding blockchain database via modifying its source codes.

5.4.1 Analysis of Monero

As there are two pioneering works [MMLN17, KFTS17] considering cascade effect attacks on the untraceability of Monero transactions, we mainly concentrate on the analysis of the anonymity of those data after the known attacks.

Dataset Collection. We collect all blocks in the Monero blockchain from the first block (18th April 2014) up to block 1541236 (30th March 2018). Additionally, all related data is directly exported from the blockchain database via modifying the source code of Monero [mon]. Our dataset in total contains 4153307 transactions. Among them, 2612070 are non-coinbase transactions, which are composed of 23164745 transaction inputs in total, and 25126033 distinct public-keys are involved. Notably, throughout this paper, we only consider those non-coinbase transactions unless otherwise stated.

Experiment Results. In **Table** 5.2, we give the result of the clustering algorithm on the aforementioned dataset. As it turns out, a total of 16334967 inputs become traceable. Specifically, 16329215 inputs are traceable due to the cascade effect attack, and the remaining inputs, i.e., 5752 in total, are traced by the finding of *closed set*. Total of 70.52% of Monero transaction inputs are traceable. While for the dataset after the cascade effect attack, only 0.084% inputs can be further traced.

Besides, a total of 6829778 transaction inputs are still untraceable. For all these remaining inputs, we give the frequency of number of mixins before and after the execution of clustering algorithm in **Figure** 5.1.

The clustering algorithm also finds 3017 distinct *closed set*s, whose size vary from 2 to 55, and include a total of 7478 distinct public-keys. As we mentioned before, these 7478 public-keys must be the real-spend of a certain input contained in these *closed set*. In other words, we can deduce that they are spent although we do not know which concrete transaction they

Table 5.2: Our Results. The first column denotes the number of mixins used to calculate the frequency. The second column named as "Total" denotes the total number of inputs with the corresponding mixins in the dataset. Column "Deducible" counts the number of inputs where the real spend can be deduced. The following two columns denote the number of inputs deduced by the titled attack. The last column counts the percentage of the traceable inputs among the total value with in the same row. The last row counts the total value of all values in a column.

No. of mixins	Total	Deducible	Cascade Effect	Clustering Algorithm	(%)
0	12209675	12209675	12209675	0	100
1	707786	625641	625264	377	88.39
2	4496490	1779134	1776192	2942	39.57
3	1486593	952855	951984	871	64.10
4	3242625	451959	451230	729	13.94
5	319352	74186	73980	206	23.23
6	432875	202360	202100	260	46.75
7	21528	4296	4282	14	19.96
8	30067	3506	3490	16	11.66
9	17724	2178	2162	16	12.29
$\geq$10	200030	29177	28856	321	14.59
Total	23164745	16334967	16329215	5752	70.52

are used. However, it is useless for the anonymity if any other new input picking public-keys from them. We call these keys included in *closed set*s as useless public-keys.

One may wonder there is a discrepancy between probability of 2^{-19} for finding *closed set* and the existence of 3017 *closed set*s found during the experiment. This is due to the fact that our analysis assumes mixins are chosen uniformly and that each input has 3 mixins. However, in practice, sampling distributions and number of mixins of all inputs are not uniform. This will increase slightly the probability of finding *closed set*.

5.4.2 Analysis of Bytecoin

We provide the first work on analyzing the untraceability of Bytecoin via cascade effect attack and clustering attack.

Dataset Collection. We collect all blocks in the Bytecoin blockchain from block 1 (4 July 2012) to block 1586652 (3 August 2018). A total of 3782566 non-coin based transactions is contained in this dataset, and there are altogether 45663011 transaction inputs included. Additionally, a total of 48613764 distinct public-keys are involved.

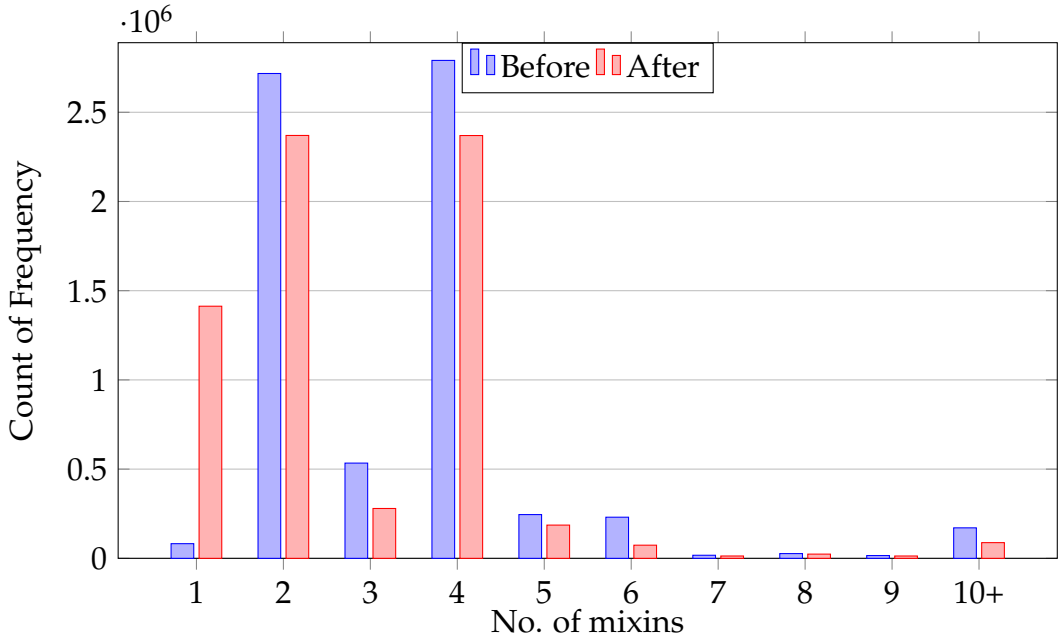


Figure 5.1: Frequency of number of mixins of those anonymous inputs before and after the execution of clustering algorithm.

Experiment Results. The experiment result on Bytecoin dataset is summarized in **Table** 5.3. More specifically, a total of 33902808 Bytecoin transaction inputs become traceable, counting for 74.25% of all inputs considered in our dataset. Among them, 28591486 inputs are zero-mixin inputs, and the cascade effect caused by them further makes 5231107 inputs become traceable, which covers 99.763% of the total traceable inputs. Besides, our clustering algorithm traces another 80215 transaction inputs from the remaining ones, which counts to 0.68% of those untraceable inputs after the cascade effect attacks. There are a total of 5912 *closed set*s found, whose size vary from 2 to 55.

5.4.3 Analysis of DigitalNote

We also provide the first work on analyzing the untraceability of DigitalNote.

Dataset Collection. We collect all 633548 non-coin based transactions included in the block 1 (31 May 2014) up to block 699748 (13 August 2018) in the DigitalNote blockchain. A total of 8110602 inputs are included in the aforementioned transactions, and 8396472 distinct public-keys are involved.

Table 5.3: Our Analysis Results on Bytecoin

No. of mixins	Total	Deducible	Cascade Effect	Clustering Algorithm	(%)
0	28591486	28591486	28591486	0	100
1	5751268	3281500	3240142	41358	57.06
2	2840745	1133602	1112648	20954	39.91
3	1442133	261197	260298	899	18.11
4	2516851	276237	275172	1065	10.98
5	617041	59922	59493	429	9.71
6	3145092	270355	255156	15199	8.60
7	388759	26434	26160	274	6.80
8	81504	1231	1220	11	1.51
9	65379	397	389	8	0.61
$\geq$10	222753	447	429	18	0.2
Total	45663011	33902808	33822593	80215	74.25

Experiment Results. The experiment result of DigitalNote is given in **Table** 5.4. Specifically, 91.56% of all transaction inputs in our dataset is traceable, while 60.39% of them is without any mixin. Besides, the cascade attack further contributes 39.60% of those traceable inputs. Our clustering algorithm makes 49 additional inputs traceable, which covers 0.007% of the untraceable inputs after the cascade effect attacks , with the help of 38 found *closed set*s.

Table 5.4: The traceability of DigitalNote

No. of mixins	Total	Deducible	Cascade Effect	Clustering Algorithm	(%)
0	4484726	4484726	4484726	0	100
1	2087295	1847151	1847132	19	88.49
2	1194410	895480	895472	8	74.97
3	129700	101872	101872	0	78.54
4	6225	4362	4358	4	70.07
5	193669	85941	85939	2	44.38
6	3071	1840	1837	3	59.92
7	844	442	440	2	52.38
8	1686	856	853	3	50.77
9	1288	682	681	1	52.95
$\geq$10	7688	2684	2677	7	34.91
Total	8110602	7426036	7425987	49	91.56

5.5 Observations and Recommendations

In this section, we give our observations and recommendations according to the experiment results.

- **Observation 1**: *The usage rate of outputs is an important factor for the anonymity of CryptoNote-style currencies.* The usage rate of outputs refers to the percentage of public-keys that have been spent. This can be easily calculated by using the total amount of inputs in the dataset over the total number of distinct outputs (i.e., public-keys), as each output can only be redeemed once. As we mentioned in **Section** 5.2.3, those unspent public-keys play an important role in preventing the formation of a *closed set*. Hence, it is fair to say that, to some degree, decreasing the usage rate will improve anonymity.

- **Observation 2**: *Closed sets are closely related to the anonymity of inputs.* In this work, we have shown that finding *closed set*s could help identify real-spends or decrease the ring size (so the level of anonymity) of those inputs. Although the probability of the existence of a *closed set* is not high, but *closed set*s do exist and threaten the anonymity of inputs.

- **Recommendation 1:** *Decreasing the usage rate of outputs by generating more outputs.* Recall that a lower usage rate of outputs is beneficial to the anonymity of Monero inputs. Hence, to decrease the usage rate of outputs, we recommend users to additionally generate some outputs with 0 amount, which can make the unspent output set larger.

- **Recommendation 2:** *Do not pick the useless mixin.* Take the Monero as an example, our clustering algorithm has found 3017 distinct *closed set*s, which contain 7478 distinct public-keys. These 7478 public-keys must be the real-spend of a certain input contained in these *closed set*s. Hence, for any newly generated input, picking these keys as mixin will not improve anonymity. So, we recommend users to not pick these useless mixins. However, for an ordinary user, it is difficult to determine whether an output is contained in *closed set*s or not. Thus, we plan to release our algorithm as an online service for users to check the *closed set*s, to avoid selecting those useless mixins.

Chapter 6

Lattice-Based Universal Accumulator

Universal accumulator [LLX07] provides a way to combine a set of values into one, and simultaneously offers a short membership witness for a given value which is accumulated and a short nonmembership witness for a given value which is not accumulated. Compared with traditional accumulator, universal accumulator is suitable for the case where nonmembership witness is desirable, as in the following example.

Suppose there is an online forum, where only legitimate users can post messages. Once a current legitimate user misbehaves, the forum manager can flag this user with a label “malicious" and forbid his or her right to post for a while, such as one day. To do this, the forum manager can maintain a list of malicious users, and update it every day. Certainly, registration before the first access of each user is needed. Then for any user who wish to post a message on this forum, besides proof of membership, he or she also needs to provide proof that he/she is not on the list of malicious users.

However, until now, the realizations of universal accumulator are mainly based on two types of non-standard number theoretic assumptions. The first type [LLX07] relies on the group of hidden order, such as Strong RSA assumption. The schemes based on this assumption usually have short public parameter but only permit primes to be accumulated. The second type [ATSM09, DT08] bases on bilinear map assumptions, including Strong Diffie-Hellman assumption. While there exists some hash-based constructions of universal accumulator [BLL00, BLL02, CHKO08], but the adoption of hash tree structure made them hardly compatible with efficient zero-knowledge proof. Without a suitable zero-knowledge proof for proving various facts about the accumulated values, they would not be as useful as the aforementioned accumulators.

To the best of our knowledge, there is no construction of lattice-based

universal accumulator. As lattice-based cryptography is promising in the post-quantum era due to its attractive properties including strong security from the worst-case hard problem, presumed resistance to quantum attacks [P+16], we design a lattice-based construction of universal accumulator with compatible zero-knowledge proofs.

Chapter Organization. For the following contents of this chapter, we start with a brief introduction about our contribution and technique used in **Section** 6.1. The proposed construction of universal accumulator appears in **Section** 6.2. Finally, this chapter is concluded with **Section** 6.3 about the application of our accumulator.

6.1 Our Contribution and Overview of Our Idea

The contribution of this work can be summarized as follows:

- *The first construction of lattice-based universal accumulator.* We propose the first lattice-based universal accumulator, which can provide a short witness for both an accumulated value and for a non-accumulated value.
- *The first zero-knowledge arguments of nonmembership in the lattice-based setting.* We introduce zero-knowledge arguments of knowledge (ZKAoK) for proving the possession of the nonmembership witness of a non-accumulated value.

Overview of Our Idea. Our Merkle-tree based accumulator considered accumulated set which is sorted. Then for any value not in the accumulated set, it must belong to an open interval formed by two adjacent values in the set. Then we pick the sibling paths of the two sibling leaves (denoting the two interval boundary values) to the root in the tree to be witness. In order to show that a given value is not accumulated, we need to prove two things in zero-knowledge: (1) the given value is between two sibling leaves in the witness; (2) the knowledge of a hash chain (via the method introduced in [LLNW16]). While the above approach appears to be very similar to the lattice-based Merkle-tree accumulator [LLNW16], the construction of the Merkle-tree in our paper is different to prevent revealing relationships between the given nonmember value and member values.

6.2 Lattice-Based Universal Accumulator

In this section, we present our construction of a universal accumulator, that is, an accumulator with membership and nonmemberhship proof. Our starting point is the accumulator from Libert et al. [LLNW16]. Here we show how to create nonmembership proof. For completeness, we separate the part of accumulator for nonmembership from universal accumulator, and give its definition in Section 2.6.1.

Throughout this section, we work with these positive integers, n, q, k, and m, where n is used as security parameter, q is $\tilde{\mathcal{O}}(n^{1.5})$, $k = \lceil \log q \rceil$, and $m = 2nk$.

Besides, for any vector $\mathbf{v} \in Z_q^n$, and its **binary representation** $\mathsf{bin}(\mathbf{v}) \in \{0,1\}^{nk}$, we have $\mathbf{G} \cdot \mathsf{bin}(\mathbf{v}) = \mathbf{v}$, where matrix $\mathbf{G}$ is defined as follows:

$$\mathbf{G} = \begin{Bmatrix} 1\ 2\ 2^2 \dots\ 2^{k-1} & & & \\ & 1\ 2\ 2^2 \dots\ 2^{k-1} & & \\ & & \ddots & \\ & & & 1\ 2\ 2^2 \dots 2^{k-1} \end{Bmatrix} \in Z_q^{n \times nk}.$$

In order to assign a unique value for each binary vector with length nk, we define the notion of **integer value**. The **integer value** $\mathsf{Int}(\mathbf{v})$ of a binary vector $\mathsf{bin}(\mathbf{v}) \in \{0,1\}^{nk}$ is computed as

$$\mathsf{Int}(\mathbf{v}) = (1\ 2\ 2^2\ 2^3\ \dots 2^{(nk-1)}) \cdot \mathsf{bin}(\mathbf{v}),$$

where we label $(1\ 2\ 2^2\ 2^3\ \dots 2^{(nk-1)})$ as $\mathbf{G}'$ in the following contents.

6.2.1 Our Construction of Accumulator for Nonmembership

In this section, we give our solution for nonmembership witness via constructing a Merkle-tree with $2^{\ell+1}$ leaves, where ℓ is a positive integer. Similar to [LLNW16], the construction of Merkle-tree is based on a family of lattice-based collision-resilient hash function $\mathcal{H} = \{h_{\mathbf{A}} | \mathbf{A} = [\mathbf{A}_0 || \mathbf{A}_1], \mathbf{A}_0, \mathbf{A}_1 \in Z_q^{n \times nk}\}$, mapping from $\{0,1\}^{nk} \times \{0,1\}^{nk}$ to $\{0,1\}^{nk}$. For any $(\mathbf{u}_0, \mathbf{u}_1) \in \{0,1\}^{nk} \times \{0,1\}^{nk}$, $h_{\mathbf{A}}(\mathbf{u}_0, \mathbf{u}_1) = \mathsf{bin}(\mathbf{A}_0 \cdot \mathbf{u}_0 + \mathbf{A}_1 \cdot \mathbf{u}_1 \bmod q) \in \{0,1\}^{nk}$.

Our construction of accumulator for nonmembership consists of four algorithms. Besides, for any input accumulated set S with size $N = 2^\ell - 1$, two auxiliary nodes are additionally chosen, denoted as $\mathbf{F}_{irst}$ and $\mathbf{L}_{ast}$.

$\mathtt{Setup}(n)$. Pick $\mathbf{A} \xleftarrow{\$} Z_q^{n\times m}$, $\mathbf{F}_{irst} = \mathbf{0}^{nk}$, and $\mathbf{L}_{ast} = \mathbf{1}^{nk}$. Then output $pp = \{\mathbf{A}, \mathbf{F}_{irst}, \mathbf{L}_{ast}\}$.

$\mathsf{NM\text{-}Acc}_{pp}(S)$. The algorithm takes input an accumulated set $S=\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, where each element $\mathbf{x}_i \in \{0,1\}^{nk} \setminus \{\mathbf{0}^{nk}, \mathbf{1}^{nk}\}$ ($i \in [1, N]$), and proceeds as follows:

1. **Sort Inputs**. First sort S in ascending order via the corresponding integer value $\mathsf{Int}(\mathbf{x}_j)$ (within 2^{nk}) of each element $\mathbf{x}_j$, and let ($\mathbf{x}'_1$, $\ldots$, $\mathbf{x}'_N$) be the sorting result.
2. **Assign Values**. Let ($\mathbf{u}_0$, $\mathbf{u}_1$, $\mathbf{u}_2$, $\ldots$, $\mathbf{u}_{2^{\ell+1}-1}$) be $2N+2 = 2^{\ell+1}$ variables. Then we assign value for each variable as follows:
   - $\mathbf{u}_0 = \mathbf{F}_{irst}$;
   - for $j = 1$ to N, $\mathbf{u}_j = \mathbf{x}'_j$;
   - for $j = N+1$ to $2N$, $\mathbf{u}_j = \mathbf{x}'_{j-N}$;
   - $\mathbf{u}_{2^{\ell+1}-1} = \mathbf{L}_{ast}$.

   In addition, for each $j \in [0, (2^{\ell+1}-1)]$, let $(j_1, j_2, \ldots, j_{\ell+1})$ be its binary representation string, then $\mathbf{u}_j = \mathbf{u}_{j_1,j_2,\ldots,j_{\ell+1}}$.
3. **Construct Tree**. Then construct a tree with depth $(\ell+1)$ based on the leaves ($\mathbf{u}_0$, $\mathbf{u}_1$, $\mathbf{u}_2$, $\ldots$, $\mathbf{u}_{2^{\ell+1}-1}$).
   - At any depth $i \in [1, \ell]$, for each $(b_1, b_2, \ldots, b_i) \in \{0,1\}^i$, each node $\mathbf{u}_{b_1,b_2,\ldots,b_i}$ is defined as

$$u_{b_1,b_2,\ldots,b_i} = h_{\mathbf{A}}(\mathbf{u}_{b_1,b_2,\ldots,b_i,0}, \mathbf{u}_{b_1,b_2,\ldots,b_i,1});$$

   - At depth 0, the root node is $\mathbf{u} = h_{\mathbf{A}}(\mathbf{u}_0, \mathbf{u}_1)$.

The algorithm outputs the nonmembership accumulator value $\mathbf{u}$.

$\mathsf{NM\text{-}Witness}_{pp}(S, \mathbf{d})$, where $\mathbf{d} \notin S$.

Let $\mathsf{Int}(\mathbf{d})$ be the integer value of $\mathbf{d}$. First find two **sibling** leaves ($\mathbf{u}_{b_1,\ldots,b_\ell,0}$, $\mathbf{u}_{b_1,\ldots,b_\ell,1}$) in the tree such that

$$\mathsf{Int}(\mathbf{u}_{b_1,b_2,\ldots,b_\ell,0}) < \mathsf{Int}(\mathbf{d}) < \mathsf{Int}(\mathbf{u}_{b_1,b_2,\ldots,b_\ell,1}).$$

Then return the witness for $\mathbf{d}$ as follows.

$$w = ((b_1, b_2, \ldots, b_\ell), (\mathbf{u}_{b_1,b_2,\ldots,b_\ell,0}, \mathbf{u}_{b_1,b_2,\ldots,b_\ell,1}, \mathbf{u}_{b_1,b_2,\ldots,\bar{b}_\ell}, \ldots, \mathbf{u}_{b_1,\bar{b}_2}, \mathbf{u}_{\bar{b}_1})) \in \{0,1\}^\ell \times (\{0,1\}^{nk})^{\ell+2}.$$

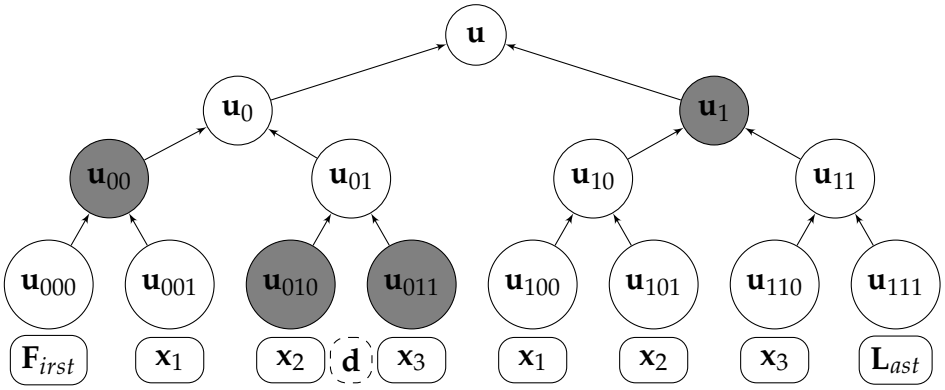


Figure 6.1: A Merkle-tree with 2^3 leaves, which accumulates the data blocks in the set S= $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ with ascending integer values, into an accumulator value $\mathbf{u}$. In addition, the bit string (01) and the gray nodes consist the witness for a node $\mathbf{d}$, which is not accumulated in $\mathbf{u}$, and $\mathsf{Int}(\mathbf{x}_2) < \mathsf{Int}(\mathbf{d}) < \mathsf{Int}(\mathbf{x}_3)$.

$\mathsf{NM\text{-}Verify}_{pp}(\mathbf{u}, \mathbf{d}, w)$. Assume the witness w is of the form $w = ((b_1, b_2, \ldots, b_\ell), (\mathbf{w}_{\ell,1}, \mathbf{w}_{\ell,2}, \mathbf{w}_\ell, \mathbf{w}_{\ell-1}, \ldots, \mathbf{w}_1))$.

- First check whether $\mathsf{Int}(\mathbf{w}_{\ell,1}) < \mathsf{Int}(\mathbf{d}) < \mathsf{Int}(\mathbf{w}_{\ell,2})$.
- If yes, then compute $\mathbf{v}_\ell = h_{\mathbf{A}}(\mathbf{w}_{\ell,1}, \mathbf{w}_{\ell,2})$, and

$$\forall i \in \{\ell-1, \ell-2, \ldots, 1, 0\} : \mathbf{v}_i = \begin{cases} h_{\mathbf{A}}(\mathbf{v}_{i+1}, \mathbf{w}_{i+1}) & if \quad b_{i+1} = 0 \\ h_{\mathbf{A}}(\mathbf{w}_{i+1}, \mathbf{v}_{i+1}) & if \quad b_{i+1} = 1 \end{cases}.$$

Finally, the algorithm returns 1 if $\mathbf{v}_0$ equals $\mathbf{u}$. Otherwise, returns 0.

Then we give an example of a tree with 2^3 leaves, where the size of the accumulated set is 3, and denote the set as $S = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$. For simplicity, we assume that elements in S are in ascending order. Then the tree is shown in Figure 6.1.

Correctness. The correctness of the above construction requires that for any binary string $\mathbf{d} \in \{0,1\}^{nk} \backslash \{0\ldots0, 1\ldots1\}$, $\mathbf{d} \notin S$, and $\mathbf{u} \leftarrow \mathbf{A}cc_{pp}(S)$, computes witness $w \leftarrow \mathsf{NM\text{-}Witness}_{pp}(S, \mathbf{d})$, $\mathsf{NM\text{-}Verify}_{pp}(\mathbf{u}, \mathbf{d}, w) = 1$. We also argue that for any $\mathbf{d}$, its witness w is unique in the above Merkle-tree.

Since set S is sorted via the integer value of each element in $\mathsf{NM\text{-}Acc}_{pp}$ algorithm, here we directly assume that $S = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ be a sorted binary string set, and each element inside is different. We use interval $\mathbf{I}_i$ to denote the open interval $(\mathsf{Int}(\mathbf{x}_{i-1}), \mathsf{Int}(\mathbf{x}_i))$, which is illustrated in Figure 6.2. Observe that value $\mathsf{Int}(\mathbf{d})$ must fall into one and only one interval $\mathcal{I}_i$ in Figure 6.2 since $\mathbf{d} \notin S$. Then the corresponding elements in S, namely $\mathbf{x}_i$ and

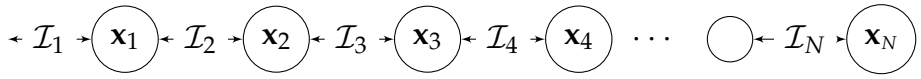


Figure 6.2: Illustration of Correctness

$\mathbf{x}_{i+1}$, constitute the first two nodes (sibling leaves) in the witness. Since we require them to be sibling leaves in the tree T, then we choose the corresponding sibling leaves $\mathbf{u}_j$ and $\mathbf{u}_{j+1}$ based on $\mathcal{I}_i$. If i is even, then choose $\mathbf{u}_{N+i-1}$ and $\mathbf{u}_{N+i}$, and the siblings of each node in the path from them to root to be witness. Otherwise, picks $\mathbf{u}_{i-1}$ and $\mathbf{u}_i$, and the siblings of each node in the path from them to root.

Regarding the security of our construction, we have the following theorem.

Theorem 6.2.1. *Under the hardness of* SIS *problem, the construction for nonmembership witness presented above is secure.*

Proof. Assume that there exists an adversary $\mathcal{B}$ who can break the security of the above accumulator scheme. Then we can construct another algorithm which can break the collision-resilient property of the hash function h used in the scheme, whose hardness is based on the SIS problem.

Given the public parameter $pp = (\mathbf{A}, \mathbf{F}_{irst}, \mathbf{L}_{ast})$ output by $\mathtt{Setup}(n)$, $\mathcal{B}$ outputs $(S^*, \mathbf{d}^*, w^*)$ such that $\mathbf{d}^* \in S^*$, and algorithm $\mathsf{NM\text{-}Verify}_{pp}(\mathsf{NM\text{-}Acc}_{pp}(S^*), \mathbf{d}^*, w^*) = 1$, where w^* is in the form $((b_1^*, b_2^*, \ldots, b_\ell^*), (w_{\ell,1}^*, w_{\ell,2}^*, w_\ell^*, w_{\ell-1}^*, \ldots, w_1^*))$.

Since $\mathsf{NM\text{-}Verify}_{pp}(\mathsf{NM\text{-}Acc}_{pp}(S^*), d^*, w^*) = 1$, hence $\mathsf{Int}(w_{\ell,1}^*) < \mathsf{Int}(\mathbf{d}^*) < \mathsf{Int}(w_{\ell,2}^*)$, which implies that $d^* \neq \mathbf{w}_{\ell,1}^*$ and $d^* \neq \mathbf{w}_{\ell,2}^*$. Let $\mathbf{v}_\ell^*, \mathbf{v}_{\ell-1}^*, \mathbf{v}_{\ell-2}^*, \ldots, \mathbf{v}_0^*$ be the path computed by algorithm $\mathsf{NM\text{-}Verify}_{pp}$. We also set $\mathbf{v}_{\ell,0}^* = w_{\ell,1}^*$, and $\mathbf{v}_{\ell,1}^* = w_{\ell,2}^*$. Then $\mathbf{v}_0^*$ must be equal to $\mathbf{u}$.

Next we construct the Merkle-tree T^* based on the sorted set S^*,$\mathbf{F}_{rist}$, and $\mathbf{L}_{ast}$. Notably, each node in T^* is represented via $\mathbf{u}_i$. Recall that $(b_1^*, b_2^*, \ldots, b_\ell^*)$ is the bit string contained in w^*. Let $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,0}$, $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,1}$, $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*}$, $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_{\ell-1}^*}, \ldots, \mathbf{u}_{b_1^*}, \mathbf{u}$ be the path from leaves $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,0}$, $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,1}$ to root $\mathbf{u}$. Notably, $\mathbf{d}^*$ must be equal to either $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,0}$ or $\mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,1}$ since $\mathbf{d}^* \in S^*$. In this way, we get two paths, they are

$$\begin{aligned}
&\mathit{Path1}: \mathbf{v}_{\ell,0}^*, \mathbf{v}_{\ell,1}^*, \mathbf{v}_\ell^*, \mathbf{v}_{\ell-1}^*, \mathbf{v}_{\ell-2}^*, \ldots, \mathbf{v}_1^*, \mathbf{v}_0^* \\
&\mathit{Path2}: \mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,0}, \mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,1}, \mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*}, \mathbf{u}_{b_1^*,b_2^*,\ldots,b_{\ell-1}^*}, \ldots, \mathbf{u}_{b_1^*}, \mathbf{u}
\end{aligned}$$

Comparing *Path1* and *Path2*, we can find the smallest integer $k \in [1, \ell+1]$ such that $\mathbf{v}_k^* \neq \mathbf{u}_{b_1^*,b_2^*,\ldots,b_k^*}$. Notably, in the case $k = \ell + 1$, we mean either $\mathbf{v}_{\ell,0}$

$\neq \mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,0}$ or $\mathbf{v}_{\ell,1} \neq \mathbf{u}_{b_1^*,b_2^*,\ldots,b_\ell^*,1}$ or both. In this way, we find a collision for $h_{\mathbf{A}}$ for $\mathbf{v}_{k-1}^*$. □

6.2.2 Zero-Knowledge Argument of Knowledge of Nonmembership Witness

In this section, we give a ZKAoK to prove the possession of the nonmembership witness of a non-accumulated value. More specifically, given common inputs ($pp = (\mathcal{A}, \mathbf{F}_{irst}, \mathbf{L}_{ast})$, $\mathbf{u}$), prover $\mathcal{P}$ convinces verifier $\mathcal{V}$ that he has ($\mathbf{d}$, w) such that NM-Verify$_{pp}$ ($\mathbf{u}$, $\mathbf{d}$, w) = 1. The relevant relation is defined as $\mathcal{R}_{nm}$:

$$\mathcal{R}_{nm} = \left\{ \begin{array}{l} ((pp, \mathbf{u}) \in (Z_q^{n\times m} \times 0^{nk} \times 1^{nk} \times \{0,1\}^{nk}); \\ \qquad \mathbf{d} \in \{0,1\}^{nk}, w \in \{0,1\}^{\ell} \times (\{0,1\}^{nk})^{\ell+2}) : \\ \qquad\qquad \text{NM-Verify}_{pp}(\mathbf{u}, \mathbf{d}, w) = 1 \end{array} \right\}.$$

Overview of Our Argument. Assume w is of the form ((b_1, b_2, …, b_ℓ), ($\mathbf{w}_{\ell,1}$, $\mathbf{w}_{\ell,2}$, $\mathbf{w}_{\ell}$, $\mathbf{w}_{\ell-1}$, …, $\mathbf{w}_1$)). Observe that for any ($\mathbf{d}$, w), algorithm NM-Verify$_{pp}$($\mathbf{u}$, $\mathbf{d}$, w) = 1 if and only if the following two requirements being satisfied:

1. The integer value of $\mathbf{d}$ belongs to the open interval $(\mathsf{Int}(\mathbf{w}_{\ell,1}), \mathsf{Int}(\mathbf{w}_{\ell,2}))$, namely
$$\mathsf{Int}(\mathbf{w}_{\ell,1}) < \mathsf{Int}(\mathbf{d}) < \mathsf{Int}(\mathbf{w}_{\ell,2}). \tag{6.1}$$

2. The path computed by NM-Verify$_{pp}$($\mathbf{u}$, $\mathbf{d}$, w) satisfies $\mathbf{v}_0 = \mathbf{u}$, and
$$\begin{array}{l} \mathbf{v}_\ell = h_{\mathcal{A}}(\mathbf{w}_{\ell,1}, \mathbf{w}_{\ell,2}), \\ \forall i \in \{\ell-1, \ell-2, \ldots, 1, 0\} : \mathbf{v}_i = \left\{ \begin{array}{lll} h_{\mathcal{A}}(\mathbf{v}_{i+1}, \mathbf{w}_{i+1}) & if & b_{i+1} = 0 \\ h_{\mathcal{A}}(\mathbf{w}_{i+1}, \mathbf{v}_{i+1}) & if & b_{i+1} = 1 \end{array} \right. . \end{array} \tag{6.2}$$

Roughly speaking, our proof can be reduced to proving the above two requirements in zero-knowledge. Before going into details, we first give a brief sketch about the techniques used in our proof. Based on the observation that if we can adjust each requirement into the form of $\mathbf{P}_i \cdot \mathbf{x}_i = \mathbf{v}_i \bmod q_i$, and define the valid set and permutation set for the two requirements satisfying the condition(2.4), then we can use the abstract Stern's protocol [LLNW17] to get the zero-knowledge arguments protocol. For the

first requirement, we observe that for any vector $\mathbf{u}$, $\mathbf{v} \in \{0,1\}^{nk}$, if $\mathsf{Int}(\mathbf{u}) < \mathsf{Int}(\mathbf{v})$, then there is one and only one binary vector $\mathbf{diff} \in \{0,1\}^{nk}$, such that $\mathsf{Int}(\mathbf{v})$ - $\mathsf{Int}(\mathbf{u}) - \mathsf{Int}(\mathbf{diff}) = 1 \mod (2q^n)$. This part can also be used as range proof of integer values. For the second requirement, we need to provide membership proof to sibling leaves $\mathbf{w}_{\ell,1}$ and $\mathbf{w}_{\ell,2}$, which can utilize the technique of membership proof presented by Libert et al. in [LLNW16] based on modulus q.

In the following contents, we first transform the above requirements into the linear form $\mathbf{P} \cdot \mathbf{x} = \mathbf{v} \bmod q$, then define the corresponding valid set and permutation set for the abstract Stern's protocol.

Transformation of Requirement (6.1). Observe that for any vector $\mathbf{v} \in \{0,1\}^{nk}$, its integer value is within the set $\{0, 1, 2, 3, \ldots, 2^{nk}\text{-}1\}$. Then for any three vectors $\mathbf{v}_1$, $\mathbf{v}_2$, $\mathbf{v}_3 \in \{0,1\}^{nk}$, if $\mathsf{Int}(\mathbf{v}_1) - \mathsf{Int}(\mathbf{v}_2) - \mathsf{Int}(\mathbf{v}_3) = 1 \bmod (2q^n)$, then we can get that $\mathsf{Int}(\mathbf{v}_1) > \mathsf{Int}(\mathbf{v}_2) \bmod q^n$. According to this observation, we can equivalently rewrite condition (6.1) to be

$$\begin{cases} \mathsf{Int}(\mathbf{w}_{\ell,1}) < \mathsf{Int}(\mathbf{d}) \mod q^n \\ \qquad\qquad \Leftrightarrow \mathsf{Int}(\mathbf{d}) - \mathsf{Int}(\mathbf{w}_{\ell,1}) - 1 = \mathsf{Int}(\mathbf{diff}_1) \mod 2q^n; \\ \mathsf{Int}(\mathbf{d}) < \mathsf{Int}(\mathbf{w}_{\ell,2}) \mod q^n \\ \qquad\qquad \Leftrightarrow \mathsf{Int}(w_{\ell,2}) - \mathsf{Int}(\mathbf{d}) - 1 = \mathsf{Int}(\mathbf{diff}_2) \mod 2q^n, \end{cases} \tag{6.3}$$

where vectors $\mathbf{diff}_1$, $\mathbf{diff}_2 \in \{0,1\}^{nk}$ are binary vectors of the differences.

Since for any binary vector $\mathbf{v}$, we have $\mathsf{Int}(\mathbf{v}) = \mathbb{G}' \cdot \mathbf{v}$, then requirement (6.1) can be equivalently rewritten as

$$\begin{cases} \mathbb{G}' \cdot \mathbf{d} - \mathbb{G}' \cdot \mathbf{w}_{\ell,1} - \mathbb{G}' \cdot \mathbf{diff}_1 = 1 \mod 2q^n; \\ \mathbb{G}' \cdot \mathbf{w}_{\ell,2} - \mathbb{G}' \cdot \mathbf{d} - \mathbb{G}' \cdot \mathbf{diff}_2 = 1 \mod 2q^n. \end{cases} \tag{6.4}$$

Transformation of Requirement (6.2). Before going into details, we first recall some notations and techniques introduced in [LLNW16].

- B_m^{nk} is used to denote the set of all vectors in $\{0,1\}^m$ with hamming weight nk. Besides, we denote S_m the set of all permutations of all m elements.
- Let $\mathsf{ext}(b, \mathbf{v})$ denote the vector $\mathbf{z} \in \{0,1\}^{2i}$ of the form $\mathbf{z} = \begin{pmatrix} \bar{b} \cdot \mathbf{v} \\ b \cdot \mathbf{v} \end{pmatrix}$, where $\mathbf{v} \in \{0,1\}^i$ ($i \in \{nk, m\}$), and $b \in \{0,1\}$.

- For any $b \in \{0,1\}$, and for any $\pi \in S_m$, let $F_{b,\pi}$ be the permutation on vector $\mathbf{z} = \begin{pmatrix} \mathbf{z}_0 \\ \mathbf{z}_1 \end{pmatrix} \in \{0,1\}^{2m}$ with two blocks of size m, which is defined as $F_{b,\pi} = \begin{pmatrix} \pi(\mathbf{z}_b) \\ \pi(\mathbf{z}_{\bar{b}}) \end{pmatrix}$.

Next, via the same transformation strategy presented in [LLNW16], the second requirement (6.2) can be equivalently rewritten to be

$$\begin{cases} \mathcal{A} \cdot \begin{pmatrix} \mathbf{w}_{\ell,1} \\ \mathbf{w}_{\ell,2} \end{pmatrix} - \mathbb{G} \cdot \mathbf{v}_\ell = \mathbf{0} \bmod q; \\ \forall i \in [1,\ell] : \mathbf{z}_i = \mathsf{ext}(b_i, \mathbf{v}_i), \mathbf{y}_i = \mathsf{ext}(\bar{b}_i, \mathbf{w}_i); \\ \forall i \in [1,\ell-1] : \mathcal{A} \cdot \mathbf{z}_{i+1} + \mathcal{A} \cdot \mathbf{y}_{i+1} - \mathbb{G} \cdot \mathbf{v}_i = \mathbf{0} \bmod q; \\ \mathcal{A} \cdot \mathbf{z}_1 + \mathcal{A} \cdot \mathbf{y}_1 = \mathbb{G} \cdot \mathbf{u} \bmod q. \end{cases} \tag{6.5}$$

Until now, $\mathsf{NM\text{-}Verify}(\mathbf{u}, \mathbf{d}, w)$ =1 equals the equations (6.4) and (6.5) hold. Beside the above transformations, extension technique presented in [LNSW13] is also needed, which does the follows :

- Matrix extension: $\mathcal{A} = [\mathcal{A}_0 || \mathcal{A}_1]$ is modified to be $\mathcal{A}^* = [\mathcal{A}_0 \,||\, \mathbf{0}^{n \times nk} \,||\, \mathcal{A}_1 || \,\mathbf{0}^{n \times nk}]$, $\mathbb{G}$ is modified to be $\mathbb{G}^* = [\mathbb{G} || \mathbf{0}^{n \times nk}]$, and $\mathbb{G}'$ is modified to be $\mathbb{G}^{''} = [\mathbb{G}' || \mathbf{0}^{1 \times nk}]$.

- Vector extension: all $\mathbf{w}_{\ell,0}, \mathbf{w}_{\ell,1}, \ldots, \mathbf{w}_1, \mathbf{v}_\ell, \mathbf{v}_{\ell-1}, \ldots, \mathbf{v}_1$, $\mathbf{d}$, $\mathbf{diff}_1$, $\mathbf{diff}_2$ are extended into $\mathbf{w}^*_{\ell,0}$, $\mathbf{w}^*_{\ell,1}$, …, $\mathbf{w}^*_1$, $\mathbf{v}^*_\ell$, $\mathbf{v}^*_{\ell-1}$, …, $\mathbf{v}^*_1$, $\mathbf{d}^*$, $\mathbf{diff}^*_1$, $\mathbf{diff}^*_2 \in \mathsf{B}^{nk}_m$ respectively. For each vector, this is done by appending it with a binary vector of length nk with the restriction that the resulted vector's Hamming weight is nk.

Then equations (6.4) and (6.5) can be equivalently written as follows:

$$\begin{cases} \mathbb{G}'' \cdot \mathbf{d}^* - \mathbb{G}'' \cdot \mathbf{w}^*_{\ell,1} - \mathbb{G}'' \cdot \mathbf{diff}^*_1 = 1 \mod 2q^n, \\ \mathbb{G}'' \cdot \mathbf{w}^*_{\ell,2} - \mathbb{G}'' \cdot \mathbf{d}^* - \mathbb{G}'' \cdot \mathbf{diff}^*_2 = 1 \mod 2q^n, \\ \mathcal{A}^* \cdot \begin{pmatrix} \mathbf{w}^*_{\ell,1} \\ \mathbf{w}^*_{\ell,2} \end{pmatrix} - \mathbb{G}^* \cdot \mathbf{v}^*_\ell = \mathbf{0} \mod q, \\ \forall i \in [1,\ell] : \mathbf{z}_i = \mathsf{ext}(b_i, \mathbf{v}^*_i), \mathbf{y}_i = \mathsf{ext}(\bar{b}_i, \mathbf{w}^*_i), \\ \forall i \in [1,\ell-1] : \mathcal{A}^* \cdot \mathbf{z}_{i+1} + \mathcal{A}^* \cdot \mathbf{y}_{i+1} - \mathbb{G}^* \cdot \mathbf{v}^*_i = \mathbf{0} \mod q; \\ \mathcal{A}^* \cdot \mathbf{z}_1 + \mathcal{A}^* \cdot \mathbf{y}_1 = \mathbb{G} \cdot \mathbf{u} \bmod q. \end{cases} \tag{6.6}$$

Upon the above preparation, the interactive protocol can be summarized as follows.

Common inputs: Matrices $\mathbb{G}''$, $\mathcal{A}^*$, $\mathbb{G}^*$, $\mathbb{G}$, and vector $\mathbf{u}$.

Prover's inputs: $(\mathbf{diff}_1^*, \mathbf{diff}_2^*, \mathbf{d}^*)$, $(b_1, \ldots, b_\ell)$,$(\mathbf{w}_{\ell,1}^*, \mathbf{w}_{\ell,2}^*, \mathbf{w}_\ell^*, \ldots, \mathbf{w}_1^*)$, $(\mathbf{v}_\ell^*, \ldots, \mathbf{v}_1^*)$, $(\mathbf{z}_\ell, \ldots, \mathbf{z}_1)$, $(\mathbf{y}_\ell, \ldots, \mathbf{y}_1)$

Prover's goal: prove the following things in a zero-knowledge manner.(1) $\mathbf{w}_{\ell,1}^*, \mathbf{w}_{\ell,2}^* \in \mathsf{B}_m^{nk}$; (2) for all $i \in [1, \ell]$, $\mathbf{v}_i^*, \mathbf{w}_i^* \in \mathsf{B}_m^{nk}$, and $\mathbf{z}_i = \mathsf{ext}(b_i, \mathbf{v}_i^*)$, $\mathbf{y}_i = \mathsf{ext}(\bar{b}_i, \mathbf{w}_i^*)$; (3) equations (6.6) hold.

Let $\mathbf{x} = (\mathbf{diff}_1^* \parallel \mathbf{diff}_2^* \parallel \mathbf{d}^* \parallel \mathbf{w}_{\ell,1}^* \parallel \mathbf{w}_{\ell,2}^* \parallel \mathbf{v}_\ell^* \parallel \mathbf{z}_\ell \parallel \mathbf{y}_\ell \parallel \ldots \parallel \mathbf{v}_1^* \parallel \mathbf{z}_1 \parallel \mathbf{y}_1)$. Next, we specify the definition of set VALID, set S and the associated permutation T_π for $\mathbf{x}$ which satisfy conditions (2.4).

Let VALID be the set of all vectors in $\{0,1\}^{5m+5m\ell}$ with the same form of vector $\mathbf{x}$, where

- $\mathbf{diff}_1^*, \mathbf{diff}_2^*, \mathbf{d}^*, \mathbf{w}_{\ell,1}^*, \mathbf{w}_{\ell,2}^*, \mathbf{v}_\ell^*, \mathbf{v}_{\ell-1}^* \ldots, \mathbf{v}_1^* \in \mathsf{B}_m^{nk}$;
- for all $j \in [1, \ell]$ $(\mathbf{z}_i \in (\mathsf{B}_m^{nk} \times \mathbf{0}^m) \wedge \mathbf{y}_i \in (\mathbf{0}^m \times \mathsf{B}_m^{nk}))$ or $(\mathbf{z}_i \in (\mathbf{0}^m \times \mathsf{B}_m^{nk}) \wedge \mathbf{y}_i \in (\mathsf{B}_m^{nk} \times \mathbf{0}^m))$.

The set S as well as the permutation $\{T_\pi : \pi \in S\}$ is defined as follows:

- $S = \overbrace{S_m \times S_m \times \ldots \times S_m}^{(5+2\ell)}$, where S_m is the set of all permutations for m elements.
- For each $\pi = (\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \ldots, \pi_{5+2\ell}) \in S$, where each $\pi_i \in S_m$ $(i \in [1, 5+2\ell])$, and for each $\mathbf{x} = (\mathbf{diff}_1, \mathbf{diff}_1, \mathbf{d}, \mathbf{w}_{\ell,1}, \mathbf{w}_{\ell,2}, \mathbf{v}_\ell, \mathbf{z}_\ell, \mathbf{y}_\ell, \mathbf{v}_{\ell-1}, \ldots, \mathbf{z}_1, \mathbf{y}_1)$, where $\mathbf{diff}_1$, $\mathbf{diff}_1$, $\mathbf{d}$, $\mathbf{w}_{\ell,1}$, $\mathbf{w}_{\ell,2}$, $\mathbf{v}_\ell$, $\mathbf{v}_{\ell-1}$, $\ldots$, $\mathbf{v}_1$ are with length m, and each other vector has length $2m$. For $\mathbf{z}_i \in \mathbf{x}$, we denote it as $\mathbf{z}_i = \begin{pmatrix} \mathbf{z}_{i,1} \\ \mathbf{z}_{i,2} \end{pmatrix}$, where $\mathbf{z}_{i,1}$ and $\mathbf{z}_{i,2}$ have m elements respectively. We denote $\mathbf{y}_i = \begin{pmatrix} \mathbf{y}_{i,1} \\ \mathbf{y}_{i,2} \end{pmatrix}$ similarly. The main technique used in the follows is that each pair of $\mathbf{v}_i$ and $\mathbf{z}_i$ shares an identical permutation. Pick b_ℓ, $b_{\ell-1}, \ldots, b_1 \xleftarrow{\$} \{0,1\}$.

$$T_\pi(\mathbf{x}) = \pi_1(\mathbf{diff}_1) \parallel \pi_2(\mathbf{diff}_2) \parallel \pi_3(\mathbf{d}) \parallel \pi_4(\mathbf{w}_{\ell,1}) || \pi_5(\mathbf{w}_{\ell,2}) \parallel \pi_6(\mathbf{v}_\ell) \parallel \begin{pmatrix} \pi_6(\mathbf{z}_{\ell,(1+b_\ell)}) \\ \pi_6(\mathbf{z}_{\ell,(2-b_\ell)}) \end{pmatrix} \parallel \begin{pmatrix} \pi_7(\mathbf{y}_{\ell,(1+b_\ell)}) \\ \pi_7(\mathbf{y}_{\ell,(2-b_\ell)}) \end{pmatrix} \parallel \pi_8(\mathbf{v}_{\ell-1}) \parallel \begin{pmatrix} \pi_8(\mathbf{z}_{\ell-1,(1+b_{\ell-1})}) \\ \pi_8(\mathbf{z}_{\ell-1,(2-b_{\ell-1})}) \end{pmatrix} \parallel \begin{pmatrix} \pi_9(\mathbf{y}_{\ell-1,(1+b_{\ell-1})}) \\ \pi_9(\mathbf{y}_{\ell-1,(2-b_{\ell-1})}) \end{pmatrix} \parallel \ldots\ldots \parallel \begin{pmatrix} \pi_{5+2\ell}(\mathbf{y}_{1,1+b_1}) \\ \pi_{5+2\ell}(\mathbf{y}_{1,2-b_1}) \end{pmatrix}.$$

Thanks to the useful equivalences introduced in [LLNW16], which state that

- For any vector $\mathbf{v} \in \{0,1\}^m$, and $\pi \in S_m$, we have

$$\mathbf{v} \in B_m^{nk} \iff \pi(\mathbf{v}) \in B_m^{nk};$$

- For any vector $\mathbf{v}, \mathbf{w} \in \{0,1\}^m$, $c, b \in \{0,1\}$, $\pi, \phi \in S_m$, we have

$$\begin{aligned} \mathbf{z} = \mathsf{ext}(c, \mathbf{v}) \wedge \mathbf{v} \in B_m^{nk} &\iff F_{b,\pi}(\mathbf{z}) = \mathsf{ext}(c \oplus b, \pi(\mathbf{v})) \wedge \pi(\mathbf{v}) \in B_m^{nk} \\ \mathbf{y} = \mathsf{ext}(\bar{c}, \mathbf{w}) \wedge \mathbf{w} \in B_m^{nk} &\iff F_{\bar{b},\phi}(\mathbf{y}) = \mathsf{ext}(c \oplus b, \phi(\mathbf{w})) \wedge \phi(\mathbf{w}) \in B_m^{nk}. \end{aligned}$$

We can get that $\mathbf{x} \in \mathsf{VALID}$ if and only if $T_\pi(\mathbf{x}) \in \mathsf{VALID}$. Besides, if π is uniformly chosen from S, then $T_\pi(\mathbf{x})$ is uniformly distributed in VALID. In this way, we can run the abstract Stern's protocol [LLNW17] to prove the knowledge of $\mathbf{x}$ satisfying all requirements stated in Prover's goal.

6.3 Application of Our Accumulator

As an independent interest, we give one potential application of our above proposed accumulator, i.e. fully dynamic group signature. Unlike a static group signature, a fully dynamic group signature should enable the users dynamical joining and user revocation. Our idea is that we construct two Merkle-trees in our constructed dynamic group signature scheme, one is for membership proof and another one is for non-membership proof. In the following, we call the Merkle-tree used for membership proof as T_1, and the Merkle-tree used for nonmembership proof as T_2. We first give a brief introduction on how to construct the group signature.

Firstly, group manager computes enough number of chameleon hash values, and use them as the leaves to construct the Merkle-tree T_1. Once a user is joining, group manager opens a non-designed chameleon hash values to be the user's public key. Notably, in our scheme, the joining operation of a user won't affect the root value of T_1, and once a chameleon hash value is open, it won't change forever.

As mentioned before, T_2 is the tree whose leaves are all users who have been revoked, which is constructed via the method presented in **Section** 6.2. When group manger wants to revoke a user at some time, he can just add this user to be a leaf in T_2, and this process needs to reconstruct T_2.

Then any member wants to produce a group signature, he needs to give two types of proofs. The first one is to prove that he is a member in T_1, this can be done via utilizing the technique presented in [LLNW16]. The second type of proof is to prove that he is not a member in the second tree T_2 via our technique. Then if these two parts are both valid, we say the signature is valid.

Recently, Ling et al. [LNWX17] present a fully dynamic group signature from updatable Merkle-tree accumulator where the cost of adding and deleting element is logarithmic size in the number of group member. In addition, their scheme is also lattice-trapdoor-free. Looking ahead,in our scheme, via the help of the chameleon hash function, the complexity of adding a node is $\mathcal{O}(1)$, which needs to utilize a trapdoor of the chameleon hash function. While every time the group manager issue a new revoked list, he needs to reconstruct the second accumulator (based on revoked members) for nonmembership proof. Hence the cost of deleting is the cost for constructing a Merkle-tree for the revoked set, which is worse than [LNWX17]. Besides, the signature size of our scheme is not as compact as [LNWX17]. However, we argue that our fully dynamic group signature fits for the scenario that user's status frequently changes (either be valid or revoked) in different time period, and the revoked list periodically updates.

6.3.1 Definition of Fully Dynamic Group Signature

We recall the definition of the fully dynamic group signature (FDGS)[BCC$^+$16, LNWX17]. A fully dynamic group signature (FDGS) is composed of the following polynomial time algorithms.

$\mathsf{GS.Setup}(\lambda) \rightarrow \mathsf{pp}$. This algorithm takes the security parameter as input and outputs the public parameter pp.

$\langle \mathsf{GS.Keygen}_{\mathsf{GM}}(\mathsf{pp}), \mathsf{GS.Keygen}_{\mathsf{TM}}(\mathsf{pp}) \rangle$. This is an interactive protocol between algorithms $\mathsf{GS.Keygen}_{\mathsf{GM}}(\mathsf{pp})$ and $\mathsf{GS.Keygen}_{\mathsf{TM}}(\mathsf{pp})$ run by group manager GM and tracing manager TM respectively. If this protocol completes successfully, algorithm $\mathsf{GS.Keygen}_{\mathsf{GM}}(\mathsf{pp})$ outputs a group manager key pair $(\mathsf{msk}, \mathsf{mpk})$. The algorithm $\mathsf{GS.Keygen}_{\mathsf{TM}}(\mathsf{pp})$ outputs a key pair $(\mathsf{tsk}, \mathsf{tpk})$. Set the group public key $\mathsf{gpk} = (\mathsf{pp}, \mathsf{mpk}, \mathsf{tpk})$.

$\mathsf{GS.UKgen}(\mathsf{pp}) \rightarrow (\mathsf{upk}, \mathsf{usk})$. On input the public parameter pp, this algorithm outputs a user's key pair $(\mathsf{upk}, \mathsf{usk})$.

$\langle \mathsf{Join}(\mathsf{Info}_\tau, \mathsf{gpk}, \mathsf{upk}, \mathsf{usk}); \mathsf{Issue}(\mathsf{Info}_\tau, \mathsf{msk}, \mathsf{upk}) \rangle$. This is an interactive protocol between group manager GM and user. Upon successful completion, this user becomes an element of the group, and is assigned an identifier uid. Algorithm Join also stores the secret signing key usk in $\mathsf{gsk}[\mathsf{uid}]$. In addition, algorithm Join adds the registration information to registration table **reg** with index uid.

$\mathsf{GS.Revoke}(\mathsf{gpk}, \mathsf{msk}, \mathsf{Info}_{\tau_{curr}}, S, \mathbf{reg}) \rightarrow \mathsf{Info}_{\tau_{new}}$. This algorithm is done by group manager GM to revoke users, update group information, and advance the epoch. On input the group master public key gpk, group master secret key msk, current group information $\mathsf{Info}_{\tau_{curr}}$, a set S of active users, registration table **reg**, the algorithm removes users in S from the group, generates new group information $\mathsf{Info}_{\tau_{new}}$, and may update the registration table **reg**. If there is no change to the group information, this algorithm outputs $\perp$. This algorithm aborts if any $\mathsf{uid} \in S$ has not run the Join protocol.

$\mathsf{GS.Sign}(\mathsf{gpk}, \mathsf{gsk}[\mathsf{uid}], \mathsf{Info}_\tau, M) \rightarrow \Sigma$. This algorithm generates a signature Σ for Message M by user uid. If the user uid is not an active user in time epoch τ, this algorithm outputs $\perp$.

$\mathsf{GS.Verify}(\mathsf{gpk}, \mathsf{Info}_\tau, M, \Sigma) \rightarrow 0$ or 1. This deterministic algorithm checks the validity of signature Σ at epoch τ, and outputs one resulted bit.

$\mathsf{GS.Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{Info}_\tau, \mathbf{reg}, M, \Sigma) \rightarrow (\mathsf{uid}, \Pi_{trace})$. This algorithm is run by tracing manager TM to open the signing user uid for signature Σ. Besides, this algorithm also generates a proof Π_{trace} to indicate that the tracing result is correct. It returns $\perp$ if it fails to trace to a group member.

$\mathsf{GS.Judge}(\mathsf{gpk}, \mathsf{uid}, \mathsf{Info}_\tau, \Pi_{trace}, M, \Sigma) \rightarrow 0$ or 1. This algorithm is used to check the validity of the proof Π_{trace}.

Correctness. The correctness of FDGS require that the signatures generated by an honest and active, and non-revoked users can always be accepted by the GS.Verify algorithm. Besides, an honest tracing manager can always identify the signer of the valid signatures mentioned above, and the proof for tracing output by GS.Trace algorithm can be accepted by GS.Judge algorithm.

Security. The security of FDGS mainly contains the following requirements as mentioned in [BCC+16]:
Anonymity requires that it is infeasible for any probabilistic polynomial time adversary to distinguish two signatures generated by two active group members which are chosen by adversary, even if the adversary can corrupt any user and fully corrupt the group manager by generating her key. In addition, adversary can also access the GS.Trace oracle for any other signatures except the challenge ones.
Non-Frameability requires that it is infeasible for any probabilistic polynomial time adversary to generate a signature which can be accepted by GS.Verify algorithm, and can be traced to an honest user who did not produce it. Even if the adversary corrupts the group manager, tracing manager and all other user in this system.
Traceability ensures that the adversary cannot produce a signature that cannot be traced back to an active user of the group at the chosen epoch, where adversary can corrupt any user and the tracing manager.
Tracing Soundness ensures that the adversary cannot produce a valid signature that traces back to two different active users of the group even if the adversary can corrupt the group manager, tracing manager, and all members of the group.

For simplicity of reading, here we only given an informal description of those security definition. We refer the reader to [BCC+16] for the formal definitions of correctness and security requirements of FDGS.

6.3.2 Our Construction

In this section, we show how to construct a lattice-based dynamic revocable group signature based on our accumulator and argument systems.

Let n be the security parameter, and let $N = 2^\ell - 1$ be the maximum number of users contained in the group. The constructed group signature scheme uses n, m, q, k as defined in Section 6.2. Let $\kappa = \omega(\log n)$ be the parameter that determines the number of protocol repetitions. To utilize the ℓ-bits Regev's encryption scheme, we set prime modulus $p = \tilde{\mathcal{O}}(n^{1.5})$, parameter $m_E = 2(n+\ell)\lceil \log p \rceil$, and a LWE error distribution $\chi = D_{\mathbb{Z},2\sqrt{n}}$.

$\mathsf{GS.Setup}(\lambda) \rightarrow \mathsf{pp}$. This algorithm takes the security parameter λ as input and specifies the following parameters:

- Recall that n is public parameter, let $q = \tilde{\mathcal{O}}(n^{1.5})$ and $k = \lceil \log q \rceil$.

- Set the maximum number of group member to $N = 2^{\ell} - 1$.
- Matrix dimension of accumulator is $m = 2nk$. Picks the two hash matrixs for two accumulators as $\mathbf{A}= \mathbf{A}_0||\mathbf{A}_1$, $\mathbf{A}'= \mathbf{A}'_0||\mathbf{A}'_1 \in Z_q^{n\times m}$.
- Let $m' = O(n \log q)$, and let $m_{ch} = nk + \mu + m'$ be the matrix dimension for chameleon hash. As stated in [CHKP10], the randomness space is $\mathcal{R} = \{\mathbf{r} \in \mathbb{Z}^{m'} : ||r|| \leq s \cdot \sqrt{m'}\}$ with Distribution space $D_{\mathbb{Z}^{m'},s}$. The message space is $\mathcal{M} = \{0,1\}^{nk+\mu}$, and range is $\mathcal{Y} = Z_q^n$.
- Let $p = \tilde{\mathcal{O}}(n^{1.5})$ be prime modulus for Regev's encryption scheme, parameter $m_E = 2(n+\ell)\lceil \log p \rceil$. Let $\beta= \sqrt{n} \cdot \omega(\log n)$, and χ be a β-bounded noise distribution.
- A hash function $\mathcal{H}_{FS}$: $\{0,1\}^* \to \{1,2,3\}^{\kappa}$, which is modeled as a random oracle in the Fiat-Shamir transformations, where $\kappa= \omega(\log n)$.
- Let $\mathsf{COM} : \{0,1\}^* \times \{0,1\}^m \to Z_q^n$ be the string commitment scheme presented in [KTX08], which will be used in our zero-knowledge argument of knowledge system.
- Picks a uniformly random matrix $\mathbf{A}_{uk} \in Z_q^{n\times m}$, which will be used in generation of user's key pair.
- Set $\mathbf{F}_{irst} = \overbrace{(00\ldots0)}^{nk}$, and $\mathbf{L}_{ast} = \overbrace{(11\ldots1)}^{nk}$.

Then this algorithm outputs the public parameters

$$\mathsf{pp} = \{n, q, N, k, m, \mu, m_{ch}, p, m_E, \beta, \chi, \kappa, \mathcal{H}_{FS}, \mathsf{COM}, \mathcal{R}, \mathcal{M}, \mathcal{Y}, \mathbf{A}, \mathbf{A}', \mathbf{A}_{uk}, \mathbf{F}_{irst}, \mathbf{L}_{ast}\}.$$

$\langle \mathsf{GS.Keygen_{GM}}(\mathsf{pp}), \mathsf{GS.Keygen_{TM}}(\mathsf{pp}) \rangle$. In this interactive protocols, the group manager GM and the tracing manager TM generate the keys and public group information as follows:

- $\mathsf{GS.Keygen_{GM}}(\mathsf{pp})$ run by GM:
  - Pick $\mathsf{msk} \xleftarrow{\$} \{0,1\}^m$, and computes $\mathsf{mpk} = \mathbf{A}_{uk} \cdot \mathsf{msk} \bmod q$. Same as in [BCC+16], msk is needed when a party want to edit the group information board, which is visible to everyone.
  - Run $\mathsf{CH.KeyGen}(n) \to (\mathbf{A}_{ch}, t_{ch})$, whose message space, randomness space and range is defined as those in pp.

- $\mathsf{GS.Keygen}_{\mathsf{TM}}(\mathsf{pp})$ run by TM: choose $\mathbf{B} \xleftarrow{\$} Z_q^{n \times m_E}$. For each $i \in \{1,2\}$, sample $\mathbf{S}_i \xleftarrow{\$} \chi^{n \times \ell}$, $\mathbf{E}_i \xleftarrow{\$} \chi^{\ell \times m_E}$, and compute $\mathbf{P}_i = \mathbf{S}_i^T \cdot \mathbf{B} + \mathbf{E}_i$. TM sets $\mathsf{tpk} = (\mathbf{B}, \mathbf{P}_1, \mathbf{P}_2)$ and $\mathsf{tsk} = (\mathbf{S}_1, \mathbf{E}_1)$.
- Then TM sends tpk to GM. Next GM initializes the public group information:
  - Picks $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N \xleftarrow{\$} \{0,1\}^{nk} / \{00\ldots0, 11\ldots1\}$. Let **reg** be the registration table in the form ($\mathbf{reg}[0][1]$, $\mathbf{reg}[0][2]$, $\mathbf{reg}[0][3]$, $\mathbf{reg}[1][1]$, $\mathbf{reg}[1][2]$, $\mathbf{reg}[1][3]$, …, $\mathbf{reg}[N][1]$,$\mathbf{reg}[N][2]$, $\mathbf{reg}[N][3]$) be a public table, where $\mathbf{reg}[i][1] = \mathbf{y}_i$, $\mathbf{reg}[i][2] = \mathbf{0}^{nk}$, $\mathbf{reg}[i][3] = 0$ for $i \in [1, N]$, while $\mathbf{reg}[0][1] = \mathbf{F}_{irst}$, $\mathbf{reg}[i][2] = \mathbf{0}^{nk}$, $\mathbf{reg}[i][3] = 0$. Looking ahead, $\mathbf{reg}[i][1]$ is used to record the chameleon hash value of public key of user i, and $\mathbf{reg}[i][2]$ is used to record the public key of user i, and $\mathbf{reg}[i][3]$ is used to record the joining epoch of user i.
  - Build a Merkle-tree T basing on nodes $\mathbf{F}_{irst}$, $\mathbf{reg}[1][1]$, $\mathbf{reg}[2][1]$, …, $\mathbf{reg}[N][1]$. Assume the root node for this tree is $\mathbf{u}_m$.
  - Set the counter of joined user $c = 0$.

  Then group manager GM outputs the group public key $\mathsf{gpk} = (\mathsf{pp}, \mathsf{mpk}, \mathcal{A}_{ch}, \mathsf{tpk})$, and the group information as $\mathsf{Info} = (0, \mathbf{u}_m, \emptyset, \emptyset, \emptyset)$, while keeping tree T and c for himself.

$\mathsf{GS.UKgen}(\mathsf{pp}) \rightarrow (\mathsf{upk}, \mathsf{usk})$. For each user who want to join the group, he will adopt this algorithm to generate his own key pair via sampling $\mathbf{x} \xleftarrow{\$} \{0,1\}^m$, then computing $\mathbf{p} = \mathsf{bin}(\mathbf{A}_{uk} \cdot \mathbf{x}) \bmod q$. Set $\mathsf{usk} = \mathbf{x}$, $\mathsf{upk} = \mathbf{p}$.

$\langle \mathsf{Join}(\mathsf{Info}_\tau, \mathsf{gpk}, \mathsf{upk}, \mathsf{usk}); \mathsf{Issue}(\mathsf{Info}_\tau, \mathsf{msk}, \mathsf{upk}) \rangle$. Assume a user with key pair $(\mathsf{upk}, \mathsf{usk}) = (\mathbf{p}, \mathbf{x})$ wants to join the group in the time epoch τ, he sends his public key $\mathbf{p}$ to GM. Then if GM accepts this request, the following operations will be done:

1. GM set the identifier to this user as $\mathsf{uid} = \mathsf{bin}(c) \in \{0,1\}^\ell$, and computes $\mathsf{bin}(\tau) \in \{0,1\}^\mu$. In addition, GM runs algorithm $\mathbf{r}_{ch} \leftarrow \mathsf{CH}.\mathcal{H}^{-1}(t_{ch}, y_c, (\mathbf{p} \,||\mathsf{bin}(\tau)))$. Then returns $(\mathsf{uid}, \mathsf{bin}(\tau), \mathbf{r}_{ch})$it to user. Then user set his signing key as $\mathsf{gsk}[c] = (\mathsf{bin}(c), \mathsf{bin}(\tau), \mathbf{p}, \mathbf{x}, \mathbf{r}_{ch})$.
2. GM updates the group information:
   - Register the user to table $\mathbf{reg}[c][2] = \mathbf{p}$ and $\mathbf{reg}[c][3] = \tau$.
   - Update the counter $c = c + 1$.

$\mathsf{GS.Revoke}(\mathsf{gpk}, \mathsf{msk}, \mathsf{Info}_{\tau_{curr}}, S, \mathbf{reg}) \rightarrow \mathsf{Info}_{\tau_{new}}$. Let S be the set of public key of those revoked users, $S = \{\mathbf{reg}[i_1][1], \mathbf{reg}[i_2][1], ldots, \mathbf{reg}[i_\iota][1]\}$, where $\iota \in [1, N]$, and $i_1, \ldots, i_\iota \in$ [1, N]. Then GM to generate another Merkle-tree T_2 runs algorithm $\mathbf{u}_{nm} \leftarrow \mathsf{NM\text{-}Acc}_{pp}(S)$ via matrix $\mathcal{A}'$, $\mathbf{F}_{irst}$, and $\mathbf{L}_{ast}$. Notably, as we require that the number of input nodes for algorithm $\mathsf{NM\text{-}Acc}_{pp}$ should be $2^\eta - 1$ for any integer η, while the size of S may not satisfy it in some cases. To handle these expected events, we can simply choose some auxiliary nodes $\mathbf{0}^{nk}$ and $\mathbf{1}^{nk}$ to meet the above requirement.

Let $\omega_{m,j}$ denote the witness for the fact that the chameleon hash value of a user's public key is accumulated into the value $\mathbf{u}_m$, and $\omega_{nm,i}$ denote the witness for the fact that a user's public key is not accumulated into the value $\mathbf{u}_{nm}$. Next GM update the group information to be

$$\mathsf{Info}_{\tau_{new}} = (\tau_{new}, \mathbf{u}_m, \mathbf{u}_{nm}, \{\omega_{m,j}\}_j, \{\omega_{nm,j}\}_j)$$

Looking ahead, a verifier only needs to download the first 2ℓ bits to obtain $\mathbf{u}_m, \mathbf{u}_{nm}$ to verify a signature at epoch τ. While a signer only needs to download the witness for himself.

$\mathsf{GS.Sign}(\mathsf{gpk}, \mathsf{gsk}[\mathsf{uid}], \mathsf{Info}_\tau, M) \rightarrow \Sigma$. In order to sign a message M at epoch τ_s, signer j with secret signing key $\mathsf{gsk}[j] = (\mathsf{bin}(j), \mathsf{bin}(\tau), \mathbf{p}, \mathbf{x}, \mathbf{r}_{ch}))$ first needs to check whether there are two witnesses (one for membership proof in the group member set, and another one for non-membership proof in the revoked set) in Info_τ. If no, then return $\perp$. Otherwise, downloads $\mathbf{u}_m$, $\mathbf{u}_{nm}$, and two witness

$$\begin{aligned}
\omega_m &= (\mathsf{bin}(j), (\mathbf{w}_\ell^m, \ldots, \mathbf{w}_1^m)), \\
\omega_{nm} &= (\mathsf{bin}(j'), (\mathbf{w}_{\ell,1}^{nm}, \mathbf{w}_{\ell,2}^{nm}, \mathbf{w}_\ell^{nm} \ldots, \mathbf{w}_1^{nm})),
\end{aligned}$$

then proceeds as follows:

1. Using Regev's encryption scheme to encrypt bit string $\mathsf{bin}(j) \in \{0,1\}^\ell$. This is done by sampling $\mathbf{r}_{E,i} \xleftarrow{\$} \{0,1\}^{m_E}$ for each $i \in \{1,2\}$, and computing

$$\begin{aligned}
\mathbf{c}_i &= (\mathbf{c}_{i,1}, \mathbf{c}_{i,2}) \\
&= (\mathbf{B} \cdot \mathbf{r}_{E,i} \bmod p, \mathbf{P}_i \cdot \mathbf{r}_{E,i} + \lceil \frac{p}{2} \rfloor \cdot \mathsf{bin}(j) \bmod p) \in Z_p^n \times Z_p^\ell.
\end{aligned}$$

2. Generation a NIZKAoK Π_m to demonstrate the possession of a valid tuple

$$(\mathbf{x}, \mathbf{p}, \mathbf{y}_j, \mathsf{bin}(j), \mathsf{bin}(j'), \mathsf{bin}(\tau), \mathbf{bin}(\tau_s), \mathbf{r}_{E,1}, \mathbf{r}_{E,2}, \mathbf{r}_{ch}, (\mathbf{w}_\ell^m, \mathbf{w}_{\ell-1}^m, \ldots, \mathbf{w}_1^m), (\mathbf{w}_{\ell,1}^{nm}, \mathbf{w}_{\ell,2}^{nm}, \mathbf{w}_\ell^{nm} \ldots, \mathbf{w}_1^{nm})),$$

such that

(a) $\mathsf{Verify}_{\mathcal{A}}(\mathbf{u}_m, \mathbf{y}, (\mathsf{bin}(j), \mathbf{w}_\ell^m, \mathbf{w}_{\ell-1}^m, \ldots, \mathbf{w}_1^m)) = 1 \bmod q$;

(b) $\mathsf{NM\text{-}Verify}_{\mathcal{A}'}(\mathbf{u}_{nm}, \mathbf{p}, (\mathsf{bin}(j'), \mathbf{w}_{\ell,1}^{nm}, \mathbf{w}_{\ell,2}^{nm}, \mathbf{w}_\ell^{nm} \ldots, \mathbf{w}_1^{nm})) = 1 \bmod q$;

(c) $\mathbf{A}_{uk} \cdot \mathbf{x} = \mathbf{p} \bmod q$;

(d) $\mathbf{A}_{ch} \cdot (\mathbf{p} || \mathsf{bin}(\tau), \mathbf{r}_{ch}) = \mathbf{y}_j \bmod q$;

(e) $\mathbf{c}_1$ and $\mathbf{c}_2$ are both correct encryptions of $\mathsf{bin}(j)$ with randomness $\mathbf{r}_{E,1}$, $\mathbf{r}_{E,2}$.

(f) $\tau \leq \tau_s$.

Notably, statement (2a) and (2e) can be proved by the protocols presented in [LLNW16], and statement (2b) can be covered by our protocols presented in Section 6.2.2. Obviously, statements (2c) and (2d) can be covered by Stern's-like protocol, such as the technique presented in [LNSW13]. The proof for the statement (2f) is presented in Section 6.2.2. Besides, we can also apply the abstraction Stern's protocol [LLNW17] to get a unifying ZKAoK for all above requirements.

Then we can apply Fiat-Shamir heuristic to achieve non-interactive property, the above protocol is needed to repeat κ times to achieve negligible soundness error simultaneously. Let $\Pi_{\mathsf{FDGS}} = (\{CMT_i\}_{i=1}^{\kappa}, CH, \{RSP\}_{i=1}^{\kappa})$, where

$$CH = \mathcal{H}_{\mathsf{FS}}(M, \{CMT_i\}_{i=1}^{\kappa}, \mathcal{A}, \mathcal{A}', \mathcal{A}_{uk}, \mathcal{A}_{ch}, \mathbf{u}_m, \mathbf{u}_{nm}, \mathbf{B}, \mathbf{P}_1, \mathbf{P}_2, \mathbf{c}_1, \mathbf{c}_2) \in \{1,2,3\}^{\kappa}.$$

3. Finally, signer outputs the signature

$$\Sigma = (\Pi_{\mathsf{FDGS}}, \mathbf{c}_1, \mathbf{c}_2).$$

$\mathsf{GS.Verify}(\mathsf{gpk}, \mathsf{Info}_\tau, M, \Sigma) \rightarrow 0$ or 1. Parse $\Sigma into(\Pi_{\mathsf{FDGS}}, \mathbf{c}_1, \mathbf{c}_2)$, then do the following:

- Get $\mathbf{u}_m$ and $\mathbf{u}_{nm}$ from Info_τ.
- If $CH \neq \mathcal{H}_{\mathsf{FS}}(M, \{CMT_i\}_{i=1}^{\kappa}, \mathcal{A}, \mathcal{A}', \mathcal{A}_{uk}, \mathcal{A}_{ch}, \mathbf{u}_m, \mathbf{u}_{nm}, \mathbf{B}, \mathbf{P}_1, \mathbf{P}_2, \mathbf{c}_1, \mathbf{c}_2)$, return 0.
- Run the Verification phase of the abstract Stern's protocol, similar as in Figure (2.2). Return 0 if any of the conditions are not satisfied.
- If all the above conditions hold, return 1.

$\mathsf{GS.Trace}(\mathsf{gpk}, \mathsf{tsk}, \mathsf{Info}_\tau, \mathbf{reg}, M, \Sigma) \rightarrow (\mathsf{uid}, \Pi_{trace})$. First parse tsk into the form $(\mathbf{S}_1, \mathbf{E}_1)$, the do the following:

1. Decrypt $\mathbf{c}_1 = (\mathbf{c}_{1,1}, \mathbf{c}_{1,2})$ to get a bit string $\mathbf{b} \in \{0,1\}^\ell$ through computing $\lfloor \mathbf{c}_{1,2} - \mathbf{S}_1^T \cdot \mathbf{c}_{1,1}/(p/2) \rceil$.
2. Checks that whether there is a witness in Info_τ containing $\mathbf{b}$. If yes, continue. Otherwise, return $\perp$.
3. Let $b' \in [1, N]$ be the decimal value of $\mathbf{b}$, checks whether $\mathbf{reg}[b'][2] = \mathbf{0}^{nk}$. If yes, return $\perp$.
4. Next generate a ZKAoK via Abstract Stern's protocol [LLM+16] to prove the possession of $\mathbf{S}_1$ and $\mathbf{E_1}$,$\mathbf{z} \in \mathbb{Z}^\ell$ such that

$$\begin{cases} ||\mathbf{S}_1||_\infty \leq \beta; ||\mathbf{E}_1||_\infty \leq \beta; ||\mathbf{z}||_\infty \leq \lceil p/5 \rceil \\ \mathbf{S}_1^T \cdot \mathbf{B} + \mathbf{E}_1 = \mathbf{P}_1 \bmod p; \\ \mathbf{c}_{1,2} - \mathbf{S}_1^T \cdot \mathbf{c}_{1,1} = \mathbf{z} + \lfloor p/2 \rfloor \cdot \mathbf{b} \bmod p. \end{cases}$$

   Similarly, via applying on Fiat-Shamir heuristic, we can get the proof $\Pi_{trace} = (\{CMT_i\}_{i=1}^{\kappa}, CH, \{RSP\}_{i=1}^{\kappa})$, where

$$CH = \mathcal{H}_{\mathsf{FS}}(M, \{CMT_i\}_{i=1}^{\kappa}, \mathsf{gpk}, \mathsf{Info}_\tau, M, \sigma, \mathbf{b}) \in \{1,2,3\}^\kappa.$$

5. Set $\mathsf{uid} = \mathbf{b}$, and output $(\mathsf{uid}, \Pi_{trace})$

$\mathsf{GS.Judge}(\mathsf{gpk}, \mathsf{uid}, \mathsf{Info}_\tau, \Pi_{trace}, M, \Sigma) \rightarrow 0$ or 1. This algorithm is used to verify the validity of Π_{trace} through the Verification phase of the corresponding ZKAoK protocol, If all conditions hold, return 1. Otherwise, return 0.

Correctness. The correctness of the scheme mentioned above is guaranteed with overwhelming probability via the correctness of Regev's encryption scheme, the completeness of the abstract Stern's protocol, and the correctness of chameleon hash function.

More precisely, a signature Σ produced by an honest and active group member must be accepted by the GS.Verify algorithm. This is because this user can always find a valid witness satisfying those requirements (2a to 2f) in the GS.Sign algorithm.

For GS.Trace algorithm, Regev's encryption scheme can guarantee that we can find $\mathsf{bin}(j)$ with overwhelming probability. Then the completeness of abstract Stern's protocol can ensure the correctness of proof Π_{trace}.

Security. The security of the above scheme is guaranteed by the following theorem.

Theorem 6.3.1. *Suppose the* ZKAoK *protocols used in above fully dynamic group signature are simulation-sound, $\mathcal{H}_{FS}$ is modeled as random oracle,* CH *is a secure chameleon hash function, the above scheme satisfies the security requirements group signature, namely anonymity, non-frameability, traceability, and tracing soundness, under the* $\mathsf{LWE}_{n,p,\chi}$ *and* $\mathsf{SIS}^{\infty}_{n,m,q,1}$ *assumptions.*

Combined with the security of the constructed universal accumulator as well as the security of chameleon hash, the proof of the following security properties is similar with the proof given in [LNWX17].

Chapter 7

Conclusion

This thesis aims at addressing some new challenges for the public-key cryptographic primitives. In particular, the contributions of this thesis is summarized as follows.

- In Chapter 3, we focus on enhancing the functionality of public-key encryption schemes suitable for data-sharing on cloud. Specifically, we give the construction of cross-system proxy re-encryption scheme (CS-PRE). The proposed construction could transform the ciphertext of any attribute-based encryption scheme within Attrapadung's framework into the ciphertext of any public-key encryption scheme. Besides versatility, our proposed CS-PRE embraces compatibility where CS-PRE does not need to modify the parameters of both source and target schemes, and efficiency where the computation cost of the re-encryption process is nearly the same as the decryption cost in source and target schemes.

- To strengthen attribute-based encryption secure against side-channel attacks, we propose the general construction of leakage-resilient attribute-based encryption scheme secure in continuous memory leakage model in Chapter 4.

- The analysis about anonymity achieved by Cryptonote-style cryptocurrencies is given in Chapter 5. Specifically, we propose an optimal statistical analysis on the traceability of this kind of cryptocurrencies. In addition, we give theoretical and practical experiment analyses on the impact of our attack.

- In Chapter 6, we give the first lattice-based universal accumulator scheme, which can be used to construct lattice-based fully dynamic group signature scheme. Besides, we give the first lattice-based zero-knowledge arguments for the possession of the non-membership witness of any value outside the accumulated set.

Open Problems. Many interesting problems are still open. One of them is how to construct cross-system proxy re-encryption scheme in prime-order groups. Besides, for the analysis of the anonymity of privacy-preserving cryptocurrencies, whether there exist other attacks which utilise some additional information, such as information related with users' IP address, that could easily track the identity of the real-payer.

On addressing quantum attacks, most existing lattice-based (universal) accumulator schemes are based on the Merkle-tree structure, which provides an elegant solution for current constructions. However, it may be hard to extend such structure to the construction of many advanced cryptographic primitives from accumulators, such as lattice-based vector commitment. To solve this problem, we wonder whether it is possible to find other constructing method for the lattice-based accumulator, which also enjoys those good algebraic properties like those in traditional number theory.

Appendix A

Proofs of Theorems

A.1 Proof of Theorem 3.3.1

Lemma A.1.1. $|\boldsymbol{G}_{real}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{res}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}^{SD1}_{\mathcal{B}}(1^\lambda) + \boldsymbol{Adv}^{SD2}_{\mathcal{B}}(1^\lambda)$.

Proof. Assume there is an adversary $\mathcal{A}$ that can submit a query $\Pi_0.X$ such that $R_{p_2}(\Pi_0.X, \Pi_0.Y^*) = 1$ but $R_N(\Pi_0.X, \Pi_0.Y^*) = 0$. We show how to construct an adversary $\mathcal{B}$ that breaks Assumption **SD1** or Assumption **SD2**. Given $(g_1 \xleftarrow{\$} \mathbb{G}_{p_1}, Z_3 \xleftarrow{\$} \mathbb{G}_{p_3}, N, \mathbb{G}, \mathbb{G}_{\mathbb{T}}, e)$, $\mathcal{B}$ simulates $Game_{real}$ for $\mathcal{A}$. Once $\mathcal{A}$ submits a query $\Pi_0.X$ such that $R(\Pi_0.X, \Pi_0.Y^*) = 1 \bmod p_2$ but $R(\Pi_0.X, \Pi_0.Y^*) = 0 \bmod N$ with non-negligible probability, $\mathcal{B}$ use the corresponding algorithm $\mathcal{F}$ to find a factor a, such that $p_2|a$ and $a|N$. Let $b = \frac{N}{a}$. Then we consider two cases:

- Case 1: $p_1|b$. $\mathcal{B}$ will break Assumption **SD1** by verifying g_1^b is the identity and then check whether T^b is identity. If yes, then $T \xleftarrow{\$} G_{p_1}$, else $T \xleftarrow{\$} G_{p_1p_2}$.
- Case 2: $a = p_1p_2, b = p_3$. $\mathcal{B}$ will break Assumption **SD2**. As $\mathcal{B}$ is additionally given $(Z_1Z_2 \xleftarrow{\$} G_{p_1p_2}, Z_3 \xleftarrow{\$} G_{p_3}, W_2W_3 \xleftarrow{\$} G_{p_2p_3})$, it will check $(Z_1Z_2)^a$ and Z_3^b are both identity group elements. Then it will verify whether $e(W_2W_3, T^b)$ is identity, if yes, $T \xleftarrow{\$} G_{p_1p_3}$; else $T \xleftarrow{\$} G_{p_1p_2p_3}$.

□

Lemma A.1.2. $|\boldsymbol{G}_{res}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_0\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}^{SD1}_{\mathcal{B}}(1^\lambda)$.

Proof. If the advantage of $\mathcal{A}$ between $\mathbf{G}_{res}$ and $\mathbf{G}_0$ is non-negligible, then we can build an algorithm $\mathcal{B}$ that breaks Assumption **SD1**. On input a problem instance $(\mathbb{G}, \mathbb{G}_{\mathbb{T}}, N, e, g_1, g_3, T)$ for Assumption **SD1**, $\mathcal{B}$ needs to decide whether $T \xleftarrow{\$} \mathbb{G}_{p_1}$ or $T \xleftarrow{\$} \mathbb{G}_{p_1p_2}$. $\mathcal{B}$ works as follows:

1. **Setup Phase.** $\mathcal{B}$ runs $\mathtt{Param}(\mathsf{k}) \rightarrow n$ and picks $\vec{h} \xleftarrow{\$} Z_N^n$, $\alpha \xleftarrow{\$} Z_N$. Then $\mathcal{B}$ computes $\Pi_0.\mathsf{PK} = (g_1, g_3, e(g_1,g_1)^{\alpha}, g_1^{\vec{h}})$, $\Pi_0.\mathsf{MSK} = \alpha$, and sends $\Pi_0.\mathsf{PK}$ to $\mathcal{A}$.

2. **Phase 1.** For all queries to $\mathcal{O}_K^0$ made by $\mathcal{A}$ in this phase, $\mathcal{B}$ works as follows:

   - Case1 : $\mathcal{A}$ makes a secret key query about key attribure $\Pi_0.X$, $\mathcal{B}$ runs $Enc1(\Pi_0.X, N) \rightarrow (\vec{k}; m_2)$, picks $\vec{r} \xleftarrow{\$} Z_N^{m_2}$, $\vec{R} \xleftarrow{\$} Z_N^{m_1}$, computes

   $$\Pi_0.SK_X = g_1^{\vec{k}(\alpha,\vec{h},\vec{r})} \cdot g_3^{\vec{R}}$$

   returns $\Pi_0.SK_X$ to $\mathcal{A}$.

   - Case2 :$\mathcal{A}$ makes a re-encryption key query from key attribure $\Pi_0.X$ to $\Pi_i.Y$, which is a ciphertext attribute of Π_i. $\mathcal{B}$ first generates $\Pi_0.SK_X$ as in Case 1, then run $\mathtt{RE\text{-}KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_X, \Pi_i.Y, \Pi_i.\mathsf{PK}) \rightarrow rk_{\Pi_0.X \rightarrow \Pi_i.Y,}$ and returns it to $\mathcal{A}$.

3. **Challenge Phase** In this phase, $\mathcal{A}$ outputs two messages $M_0, M_1 \in \mathbb{G}_T$ and challenge attribute $\Pi_0.Y^*$. $\mathcal{B}$ picks $b \xleftarrow{\$} \{0,1\}$, runs $Enc2(\Pi_0.Y^*, N) \rightarrow (\mathbf{c}; w_2)$, picks $\mathbf{s} = (s, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$, and computes the challenge ciphertext $\Pi_0.CT = (C_0, \mathbf{C}_1)$.

$$C_0 = M_b \times e(T, g_1)^{\alpha s}, \qquad \mathbf{C}_1 = T^{\mathbf{c}(\mathbf{s},\mathbf{h})}. \tag{A.1}$$

4. **Phase 2.** $\mathcal{B}$ answers key queries (secret key or re-encryption key queries) made by $\mathcal{A}$ as in **Phase 1**.

5. **Guess Phase.** $\mathcal{A}$ outputs a guess bit $b^{'}$. If $b^{'} = b$, then $\mathcal{A}$ wins.

We claim that $\mathcal{B}$ properly simulates $\mathbf{Game}_{res}$ or $\mathbf{Game}_0$ for $\mathcal{A}$.

- If $T \xleftarrow{\$} G_{p_1}$, $\mathcal{B}$ properly simulates $\mathbf{Game}_{res}$ as the challenge ciphertext generated by $\mathcal{B}$ is distributed identically as normal ciphertexts. Without loss of generatity, we assume that $T = g_1^{t_1}$, where $t_1 \xleftarrow{\$} Z_{p_1}$. Then we observe that the challenge ciphertext is

$$C_0 = e(g_1, g_1)^{t_1 \alpha s} \times M_b \qquad C_1 = g_1^{t_1 \mathbf{c}(\mathbf{s},\mathbf{h})} = g_1^{\mathbf{c}(t_1\mathbf{s},\mathbf{h})},$$

where $t_1\mathbf{s} (\text{mod } p_1)$ can be viewed as a random variable in $Z_{p_1}^{w_2+1}$. It is easy to see that this challenge ciphertext is distributed identically as a normal ciphertext.

- If $T \xleftarrow{\$} G_{p_1 p_2}$, $\mathcal{B}$ properly simulates $\mathbf{Game}_0$. In this situation, $\mathcal{B}$ generates the challenge ciphertext is a properly distributed semi-functional ciphertext. Assume $T = g_1^{t_1} g_2^{t_2}$, where $t_1 \xleftarrow{\$} Z_{p_1}$ and $t_2 \xleftarrow{\$} Z_{p_2}$. Then we observe that the challenge ciphertext is

$$C_0 = e(g_1, g_1)^{t_1 \alpha s} \times M_b \qquad C_1 = g_1^{t_1 \mathbf{c}(\mathbf{s},\mathbf{h})} \cdot g_2^{t_2 \mathbf{c}(\mathbf{s},\mathbf{h})},$$

where we implicitly set $\hat{\mathbf{s}} = t_2 \mathbf{s} \bmod p_2$ and the semi-functional parameters $\hat{\mathbf{h}} = \mathbf{h} \bmod p_2$. By Chinese Reminder Theorem, $\hat{\mathbf{h}}$ is properly distributed as they are independent from $\mathbf{h}$ mod p_1 respectively. In this way, $\mathcal{B}$ perfectly generates a semi-functional ciphertext for $\mathcal{A}$.

□

Lemma A.1.3. $|\boldsymbol{G}_{k-1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_k\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq 2\boldsymbol{Adv}^{\mathtt{CPA}}_{\mathcal{B}_1}(1^\lambda) + 4\boldsymbol{Adv}^{SD2}_{\mathcal{B}_2}(1^\lambda) + \boldsymbol{Adv}^{CMH}_{\mathcal{B}_3}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. In order to prove **Lemma** A.1.3, we consider the following two conditions.

- **Condition 1**: The k-th key query to $\mathcal{O}_K^0$ in **Phase 1** is a re-encryption query about $rk_{\Pi_0.X \to \Pi_i.Y}$ such that $R(\Pi_0.X, \Pi_0.Y^*) = 1$, where $\Pi_0.Y^*$ is the ciphertext attribute in the challenge ciphertext.
- **Condition 2**: Otherwise.

The proof structure is shown in Fig. A.1. We will use two **Corollaries** (**Corollary** A.1.1 and **Corollary** A.1.6) to show that in both **Condition**s, the difference between the advantage of adversary $\mathcal{A}$ in $\mathbf{Game}_{k-1}$ and $\mathbf{Game}_k$ is negligible.

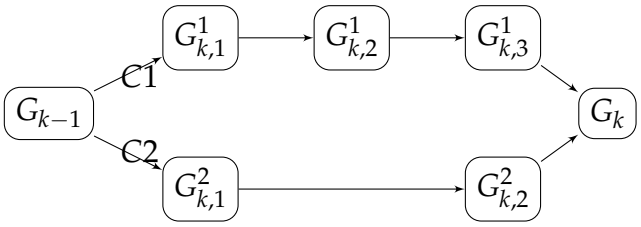


Figure A.1: C1 and C2 are the abbreviations of **Condition 1** and **Condition 2** respectively.

In **Condition 1**:

Corollary A.1.1. $|\boldsymbol{G}_{k-1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_k\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq 2q\boldsymbol{Adv}^{\mathtt{CPA}}_{\mathcal{B}_1}(1^\lambda) + 2\boldsymbol{Adv}^{SD2}_{\mathcal{B}_2}(1^\lambda)$, $2 \leq k \leq q_1$.

Proof. In order to prove **Corollary** A.1.1, we define a sequence of games between $\mathbf{Game}_{k-1}$ and $\mathbf{Game}_k$. Notably, in this phase of proof, we assume that the challenge ciphertext has been changed to be semi-functional. Here we only focus on how to change all queried keys to be semi-functional type-3.

$\mathbf{Game}^1_{k,1}$: $\mathcal{C}$ answers the k-th key query $rk_{\Pi_0.X \to \Pi_i.Y}$ to $\mathcal{O}^0_K$ in **Phase 1** as follows.

$$(\vec{rk}, \Pi_i.\vec{C}) = ((\Pi_0.SK_X \cdot g_1^{\vec{k}(0,\vec{r}',\vec{h})})^{H(\delta)}, \Pi_i.\mathtt{Enc}(\Pi_i.\mathsf{PK}, \Pi_i.Y, U))$$

where $\delta, U \xleftarrow{\$} \mathcal{M}_{\Pi_i}$.

$\mathbf{Game}^1_{k,2}$: $\mathcal{C}$ answers the k-th key query $rk_{\Pi_0.X \to \Pi_i.Y}$ as follows.

$$(\vec{rk}, \Pi_i.\vec{C}) = ((\Pi_0.SK^{type2}_X \cdot g_1^{\vec{k}(0,\vec{r}',\vec{h})})^{H(\delta)}, \Pi_i.\mathtt{Enc}(\Pi_i.\mathsf{PK}, \Pi_i.Y, U))$$

where $\delta, U \xleftarrow{\$} \mathcal{M}_{\Pi_i}$.

$\mathbf{Game}^1_{k,3}$: $\mathcal{C}$ answers the k-th key query $rk_{\Pi_0.X \to \Pi_i.Y}$ as follows.

$$(\vec{rk}, \Pi_i.\vec{C}) = ((\Pi_0.SK^{type3}_X \cdot g_1^{\vec{k}(0,\vec{r}',\vec{h})})^{H(\delta)}, \Pi_i.\mathtt{Enc}(\Pi_i.\mathsf{PK}, \Pi_i.Y, U))$$

where $\delta, U \xleftarrow{\$} \mathcal{M}_{\Pi_i}$.

Claim A.1.2. $|\boldsymbol{G}_{k-1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}^1_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq q\boldsymbol{Adv}^{\mathtt{CPA}}_{\mathcal{B}_1}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. If there is an adversary $\mathcal{A}$ whose advantage is not negligible in these two games, then we can construct an algorithm $\mathcal{B}$ who will break the IND-CPA security of one of the underlying target ABE scheme used for encrypting the hiding factor δ in $\vec{rk}$. $\mathcal{B}$ is given PK^* at the beginning of game and $\mathcal{B}$ randomly guesses $d \in [q]$, and returns PK^* to $\mathcal{A}$'s at its d-th setup queries to $\mathcal{O}_{\mathtt{Setup}}$. Then $\mathcal{B}$ runs $(\mathbb{G}, \mathbb{G}_T, N, p_1, p_2, p_3) \leftarrow \mathcal{G}(1^\lambda)$, $n \leftarrow \mathtt{Param}(\mathsf{k})$, picks $\alpha \xleftarrow{\$} Z_N, \vec{h} \xleftarrow{\$} Z^n_N$, computes $e(g_1, g_1)^\alpha$ and $g_1^{\vec{h}}$, gives the public key $\Pi_0.\mathsf{PK}$ of Π_0 to $\mathcal{A}$, where $\Pi_0.\mathsf{PK} = (pk, g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\vec{h}})$.

Phase 1: For all queries to $\mathcal{O}_{\mathtt{Setup}}(\Pi_i)(i \neq d)$, $\mathcal{B}$ runs algorithm $(\Pi_i.\mathsf{PK}, \Pi_i.\mathsf{MSK}) \leftarrow \Pi_i.\mathtt{Setup}\ (1^\lambda)$ $(i \in [q])$ to setup this scheme. Observe that $\mathcal{B}$ can also answer those queries for oracles $\mathcal{O}_K$ and $\mathcal{O}_{cor}$ for scheme $\Pi_i (i \neq d)$. For those secret key queries to $\mathcal{O}_K(\Pi_d, \Pi_d.X)$, $\mathcal{B}$ forward these queries to its own challenger and returns the output for $\mathcal{A}$. If $\mathcal{A}$ makes query about $\mathcal{O}_{cor}(\Pi_d)$, $\mathcal{B}$ aborts this simulation. When $\mathcal{A}$

makes the j-th key query to $\mathcal{O}_K^0$, which is either a secret key query about $\Pi_0.X_j$ or a re-encryption key query from $\Pi_0.X_j$ to $\Pi_i.Y$, $\mathcal{B}$ does as follows:

- Case 1: $j < k$. $\mathcal{B}$ generates semi-functional type-3 secret keys for all $\Pi_0.X_j$. To do so, $\mathcal{B}$ runs $(\vec{k}; m_2) \leftarrow \texttt{Enc1}(\Pi_0.X_j, N)$, picks $\hat{\alpha} \xleftarrow{\$} Z_N$, computes

$$\Pi_0.SK_{X_j} = g_1^{\vec{k}(\alpha,\vec{r},\vec{h})} \cdot g_2^{\vec{k}(\hat{\alpha},\vec{0},\vec{0})} \cdot g_3^{R_3}$$

  If $\mathcal{A}'s$ j-th key query to $\mathcal{O}_K^0$ is a secret key query on $\Pi_0.X_j$, returns $\Pi_0.SK_{X_j}$ to $\mathcal{A}$ directly. Else, runs $rk_{\Pi_0.X_j \to \Pi_i.Y} \leftarrow \texttt{SFRE-KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_{X_j}, \Pi_i.Y, \Pi_i.\mathsf{PK})$, and returns $rk_{\Pi_0.X_j \to \Pi_i.Y}$ to $\mathcal{A}$.

- Case 2: $j = k$. Conditioned on **Condition 1** happens, $\mathcal{A}$ will make a re-encryption key query about $\Pi_0.X_k$ to $\Pi_i.Y$ such that $R(\Pi_0.X_k, \Pi_0.Y^*) = 1$. If $d \neq i$, $\mathcal{B}$ output $\perp$. Otherwise, $\mathcal{B}$ runs $(\vec{k}; m_2) \leftarrow \texttt{Enc1}(\Pi_0.X_k, N)$, picks $\vec{r}, \vec{r}' \xleftarrow{\$} Z_N^n, \delta_0, \delta_1 \xleftarrow{\$} \mathcal{M}_{\Pi_i}$, a hash function H: $\{0,1\}^* \to Z_N$, and computes

$$\Pi_0.SK_{X_k} = g_1^{\vec{k}(\alpha,\vec{r},\vec{h})} \cdot g_3^{R_3}, \qquad \vec{rk} = (\Pi_0.SK_{X_k} \cdot g_1^{\vec{k}(0,\vec{r}',\vec{h})})^{H(\delta_0)}.$$

  Then $\mathcal{B}$ makes a challenge ciphertext query about $(\delta_0, \delta_1, \Pi_i.Y)$ to its challenger, gets the challenge ciphertext $\Pi_i.\vec{C}$, returns $rk_{\Pi_0.X_k \to \Pi_i.Y}$ = $(\vec{rk}, \Pi_i.\vec{C})$ to $\mathcal{A}$.

- Case 3: $j > k$. For all j-th ($j > k$) queries made by $\mathcal{A}$, $\mathcal{B}$ generates normal type secret keys or the corresponding normal re-encryption keys.

Challenge Phase: $\mathcal{A}$ outputs $M_0, M_1, \Pi_0.Y^*$, $\mathcal{B}$ computes

$$C_0 = M_b \times e(g_1, g_1)^{\alpha s} \qquad \vec{C}_1 = g_1^{\vec{c}(\vec{s},\vec{h})} \cdot g_2^{\vec{c}(\hat{\vec{s}},\vec{h})},$$

where $b \xleftarrow{\$} \{0,1\}$,$(\vec{c}; w_2) \leftarrow \texttt{Enc2}(\Pi_0.Y^*, N)$, $\vec{s}$ = (s,s_1, …, s_{w_2}), $\hat{\vec{s}} \xleftarrow{\$} Z_N^{w_2+1}$.

Phase 2: $\mathcal{B}$ does the same as in **Phase 1**, except that for any query to $\mathcal{O}_K^0$, $\mathcal{B}$ generates the corresponding normal secret keys. If the key query is a re-encryption key query, $\mathcal{B}$ then runs the algorithm `RE-KeyGen`, and returns the result to $\mathcal{A}$.

Guess phase: $\mathcal{A}$ outputs a guess b', and $\mathcal{A}$ wins if $b' = b$.

Then we analyze that $\mathcal{B}$ simulates $\mathbf{Game}_{k-1}$ or $\mathbf{Game}^1_{k,1}$ for $\mathcal{A}$.

- If $\Pi_i.\vec{C} \leftarrow \Pi_i.\mathtt{Enc}(\Pi_i.\mathsf{PK}^*, \delta_0, \Pi_i.Y)$, then $\mathcal{B}$ simulates $\mathbf{Game}_{k-1}$ for $\mathcal{A}$.
- If $\Pi_i.\vec{C} \leftarrow \Pi_i.\mathtt{Enc}(\Pi_i.\mathsf{PK}^*, \delta_1, \Pi_i.Y)$, as δ_0 and δ_1 are independent and uniformly distributed in $\mathcal{M}_{\Pi_i}$. Hence, given $\vec{rk}_j$, $\Pi_i.\vec{C}$ can be viewed as the encryption of a randomness. In this case, $\mathcal{B}$ simulates $\mathbf{Game}^1_{k,1}$ for $\mathcal{A}$.

□

Claim A.1.3. $|\boldsymbol{G}^1_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}^1_{k,2}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}^{SD2}_{\mathcal{B}_2}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. If there is an adversary $\mathcal{A}$ can distinguish $\mathbf{Game}^1_{k,1}$ and $\mathbf{Game}^1_{k,2}$, we will build an algorithm $\mathcal{B}$ who can break Assumption **SD2**. On input a problem instance ($\mathbb{G}$, $\mathbb{G}_T$, N, e, g_1, Z_1Z_2, g_3, W_2W_3, T), $\mathcal{B}$ works as follows:

Setup Phase: $\mathcal{B}$ runs $n \leftarrow \mathtt{Param}(\mathsf{k})$, computes $\Pi_0.\mathsf{PK}$ to $\mathcal{A}$, where $\Pi_0.\mathsf{PK} = (g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\vec{h}})$.

Phase 1: For all $\mathcal{O}_{\mathsf{Setup}}(\Pi_i)$ made by $\mathcal{A}$ in this phase, $\mathcal{B}$ generates the corresponding target scheme Π_i via $(\Pi_i.\mathsf{PK}, \Pi_i.\mathsf{MSK}) \leftarrow \Pi_i.\mathtt{Setup}(1^\lambda)$ $(i \in [q])$. Obviously, $\mathcal{B}$ can also answer those queries for oracles $\mathcal{O}_K$, and $\mathcal{O}_{cor}$ for scheme Π_i. When $\mathcal{A}$ makes the j-th key query to $\mathcal{O}^0_K$, $\mathcal{B}$ does as follows:

- Case 1: $j < i$. $\mathcal{B}$ generates a semi-functional type-3 key for $\Pi_0.X_j$. $\mathcal{B}$ runs $(\vec{k}; m_2) \leftarrow \mathtt{Enc1}(\Pi_0.X_j, N)$, picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\vec{r} \xleftarrow{\$} Z_N^{m_2}$, $\vec{R} \xleftarrow{\$} Z_N^{m_1}$ and computes:

$$\Pi_0.SK_{X_j} = g_1^{\vec{k}(\alpha,\vec{r},\vec{h})} \cdot (W_2W_3)^{\vec{k}(\hat{\alpha},\vec{0},\vec{0})} \cdot g_3^{\vec{R}}.$$

  If $\mathcal{A}'s$ j-th key query is a re-encryption key query, runs $rk_{\Pi_0.X_j \to \Pi_i.Y} \leftarrow \mathtt{SFRE\text{-}KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_{X_j}, \Pi_i.Y, \Pi_i.\mathsf{PK})$, and returns result to $\mathcal{A}$.

- Case 2: $j = k$. In **Condition 1**, the k-th query is a re-encryption key query. Hence, $\mathcal{B}$ runs $(\vec{k}; m_2) \leftarrow \mathtt{Enc1}(\Pi_0.X_k, N)$, picks $\vec{r} \xleftarrow{\$} Z_N^{m_2}$, $U \xleftarrow{\$} \mathcal{M}_{\Pi_i}$, and computes

$$\vec{rk} = T^{\vec{k}(\alpha,\vec{h},\vec{r})} \qquad \Pi_i.\vec{C} \leftarrow \Pi_i.\mathtt{Enc}(\Pi_i.\mathsf{PK}, \Pi_i.Y, U).$$

- Case 3: $j > k$. In this case, $\mathcal{B}$ generates normal secret key or normal re-encryption key to $\mathcal{A}$.

Challenge phase: $\mathcal{A}$ outputs $M_0, M_1 \in \mathbb{G}_T$ and $\Pi_0.Y^*$. $\mathcal{B}$ runs $(\vec{c}; w_2) \leftarrow \texttt{Enc2}(\Pi_0.Y^*, N)$, picks $\vec{s} = (s, s_1, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$, $b \xleftarrow{\$} Z_N$, computes challenge ciphertext $\Pi_0.CT = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = e(Z_1Z_2, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = (Z_1Z_2)^{\mathbf{c}(\mathbf{s},\mathbf{h})}$$

Phase 2: $\mathcal{B}$ does the same as in **Phase 1**, except that for any key query to $\mathcal{O}_K^0$, $\mathcal{B}$ generates a normal secret key $\Pi_0.SK_{X_j}$ or the corresponding re-encryption key $rk_{\Pi_0.X_j \to \Pi_i.Y} \leftarrow \texttt{RE-KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_{X_j}, \Pi_i.Y, \Pi_i.\mathsf{PK})$.

Guess phase: $\mathcal{A}$ outputs a guess b', and $\mathcal{A}$ wins if $b' = b$.

As $T \xleftarrow{\$} \mathbb{G}_{p_1p_3}$ or $\mathbb{G}_{p_1p_2p_3}$, we denote it as $T = g_1^{t_1}g_3^{t_3}$ or $T = g_1^{t_1}g_2^{t_2}g_3^{t_3}$. Observe that

- If $T \in G_{p_1p_3}$, then
$$\vec{rk} = g_1^{t_1\vec{k}(\alpha,\vec{h},\vec{r})} \cdot g_3^{\vec{R}}.$$
Here we implicitly set $H(\delta) = t_1$. $\mathcal{B}$ simulates $\mathbf{Game}_{k,1}^1$ for $\mathcal{A}$.

- If $T \in G_{p_1p_2p_3}$, then
$$\vec{rk} = g_1^{t_1\vec{k}(\alpha,\vec{h},\vec{r})} \cdot g_2^{t_2\vec{k}(\alpha,\vec{h},\vec{r})} \cdot g_3^{\vec{R}}.$$
Here we implicitly set $H(\delta) \bmod p_1 = t_1$, $H(\delta) \bmod p_2 = t_2$. By the Chinese Remainder Theorem, $\vec{h} \bmod p_1$ (rsep. $\alpha \bmod p_1$ and $\vec{r} \bmod p_1$) and $\vec{h} \bmod p_2$ (rsep. $\alpha \bmod p_2$ and $\vec{r} \bmod p_2$) are independent and uniformly distributed. Hence, the re-encryption key $rk_{\Pi_0.X_j \to \Pi_i.Y_j}$ generated by $\mathcal{B}$ is semi-functional type-2, and $\mathcal{B}$ simulates $\mathbf{Game}_{k,2}^1$ for $\mathcal{A}$.

□

Claim A.1.4. $|\boldsymbol{G}_{k,2}^1\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{k,3}^1\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}_1}^{SD2}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. This proof is nearly the same as the proof of CLAIM A.1.3 except the way that $\mathcal{B}$ used to answer the k-th key query in **Phase 1**. While in this proof, $\mathcal{B}$ works as follows:

Phase 1: Case 2: $j = k$. $\mathcal{B}$ runs $(\vec{k}; m_2) \leftarrow \mathtt{Enc1}(\Pi_0.X, N)$, picks $\vec{r}, \vec{r}' \xleftarrow{\$} Z_N^{m_2}$, $U \xleftarrow{\$} \mathcal{M}_{\Pi_i}$, $\tau_1, \tau_2 \xleftarrow{\$} Z_N$, $\vec{R} \xleftarrow{\$} Z_N^{m_1}$ and computes

$$\vec{rk} = g_1^{\tau_1 \vec{k}(\alpha, \vec{h}, \vec{r})} \cdot (W_2 W_3)^{\tau_2 \vec{k}(\hat{\alpha}, \vec{h}, \vec{r})} \cdot T^{\vec{k}(0, \vec{h}, \vec{r}')} \cdot g_3^{\vec{R}}$$
$$\Pi_i.\vec{C} \leftarrow \Pi_i.\mathtt{Enc}(\Pi_i.\mathsf{PK}, \Pi_i.Y, U).$$

Then $\mathcal{B}$ properly simulated $\mathbf{Game}_{k,2}^1$ if $T = g_1^{t_1} g_2^{t_2} g_3^{t_3}$ or $\mathbf{Game}_{k,3}^1$ if $T = g_1^{t_1} g_3^{t_3}$.

□

Claim A.1.5. $|\boldsymbol{G}_{k,3}^1 \boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_k \boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq q \boldsymbol{Adv}_{\mathcal{B}_1}^{\mathtt{CPA}}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. This proof is nearly the same as the security proof for the indistinguishability between $\mathbf{Game}_{k-1}$ and $\mathbf{Game}_{k,1}^1$, except that for the k-th key query to $\mathcal{O}_K^0$, $\mathcal{B}$ generates a semi-functional type-3 re-encryption key for $\mathcal{A}$, which works as follows.

Run $(\vec{k}; m_2) \leftarrow \mathtt{Enc1}(\Pi_0.X_k, N)$, pick $\vec{r}, \vec{r}' \xleftarrow{\$} Z_N^n, \delta_0, \delta_1 \xleftarrow{\$} \mathcal{M}_{\mathsf{PKE}}$, computes

$$\Pi_0.SK_{X_k} = g_1^{\vec{k}(\alpha, \vec{r}, \vec{h})} \cdot g_2^{\vec{k}(\hat{\alpha}, \vec{0}, \vec{0})} \cdot g_3^{R_3}, \quad \vec{rk} = (\Pi_0.SK_{X_k} \cdot g_1^{\vec{k}(0, \vec{r}', \vec{h})})^{H(\delta_0)}.$$

Then $\mathcal{B}$ makes a challenge ciphertext query about $(\delta_0, \delta_1, \Pi_i.Y)$ to its challenger, gets the challenge ciphertext $\Pi_i.\vec{C}$, returns $rk_{\Pi_0.X_k \to \Pi_i.Y} = (\vec{rk}, \Pi_i.\vec{C})$ to $\mathcal{A}$. □

□

In **Condition 2**:

Corollary A.1.6. $|\boldsymbol{G}_{k-1} \boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_k \boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}_3}^{CMH}(1^\lambda) + 2\boldsymbol{Adv}_{\mathcal{B}_2}^{SD2}(1^\lambda)$, $1 \leq k \leq q_1$.

Game$_{k,1}^2$: For all key queries and re-encryption key queries to $\mathcal{O}_K^0$ about $\Pi_0.X_j$ made by $\mathcal{A}$, challenger $\mathcal{C}$ answers the j-th query by generating the secret key as follows.

$\hat{\alpha}_j \leftarrow Z_N$,

$$\Pi_0.SK_{X_j} \leftarrow \begin{cases} SFKeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}, g_2, 3, \hat{\alpha}_j, \vec{0}) & j < k \\ SFKeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}, g_2, 1, 0, \vec{\hat{h}}) & j = k \\ KeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}) & j > k \end{cases}$$

Game$^2_{k,2}$: For all key queries and re-encryption key queries to $\mathcal{O}^0_K$ about $\Pi_0.X_j$ made by $\mathcal{A}$, $\mathcal{C}$ answers the j-th query by generating the secret key as follows.

$\hat{\alpha}_j \leftarrow Z_N$,

$$\Pi_0.SK_{X_j} \leftarrow \begin{cases} SFKeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}, g_2, 3, \hat{\alpha}_j, \vec{0}) & j < k \\ SFKeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}, g_2, 2, \hat{\alpha}_j, \hat{\vec{h}}) & j = k \\ KeyGen(\Pi_0.X_j, \Pi_0.\mathsf{MSK}, \Pi_0.\mathsf{PK}) & j > k \end{cases}$$

Claim A.1.7. $|\boldsymbol{G}_{k-1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}^2_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}^{SD2}_{\mathcal{B}_2}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. This proof is nearly the same as the proof of CLAIM A.1.3 except the way that $\mathcal{B}$ used to answer the k-th key query in **Phase 1**. In this proof, $\mathcal{B}$ works as follows:

Phase 1: Case 2: $j = k$. $\mathcal{B}$ runs $(\vec{k}; m_2) \leftarrow \mathtt{Enc1}(\Pi_0.X_k, N)$, picks $\vec{r}, \hat{\vec{r}} \xleftarrow{\$} Z_N^{m_2}$, $\vec{R} \xleftarrow{\$} Z_N^{m_1}$ and computes

$$\Pi_0.SK_{X_k} = g_1^{\vec{k}(\alpha,\vec{h},\vec{r})} \cdot T^{\vec{k}(0,\vec{h},\hat{\vec{r}})} \cdot g_3^{\vec{R}}.$$

If $\mathcal{A}'s$ k-th query to $\mathcal{O}^0_K$ is about a re-encryption key, $\mathcal{B}$ runs $\mathtt{SFRE\text{-}KeyGen}$ ($\Pi_0.\mathsf{PK}$, $\Pi_0.SK_{X_k}$, $\Pi_i.Y$, $\Pi_i.\mathsf{PK}$), and returns result to $\mathcal{A}$. Notably, in **Condition 2**, $\Pi_0.SK_{X_k}$ cannot decrypt the challenge ciphertext. Hence, $\mathcal{B}$ can always generate $\Pi_0.SK_{X_k}$.

As $T \xleftarrow{\$} \mathbb{G}_{p_1p_3}$ or $\mathbb{G}_{p_1p_2p_3}$, we denote it as $T = g_1^{t_1}g_3^{t_3}$ or $T = g_1^{t_1}g_2^{t_2}g_3^{t_3}$. Then if $T = g_1^{t_1}g_3^{t_3}$, $\mathcal{B}$ has properly simulated $\mathbf{Game}_{k-1}$; otherwise, $\mathcal{B}$ has properly simulated $\mathbf{Game}^2_{k,1}$. □

Claim A.1.8. $|\boldsymbol{G}^2_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}^2_{k,2}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}^{CMH}_{\mathcal{B}_1}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. Let $\mathcal{B}$ be an adversary for the co-selective security of the underlying encoding P. $\mathcal{B}$ is given (g_1, g_2, g_3) as in co-selective game and works as follows:

Setup phase: $\mathcal{B}$ picks $\vec{h} \xleftarrow{\$} Z_N^n$, $\alpha \xleftarrow{\$} Z_N$, computes $\Pi_0.\mathsf{PK} = (g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\vec{h}})$, and sends it to $\mathcal{A}$.

Phase 1: For all $\mathcal{O}_{\mathsf{Setup}}(\Pi_i)$ made by $\mathcal{A}$ in this phase, $\mathcal{B}$ generates the corresponding target scheme Π_i via algorithm $\Pi_i.\mathtt{Setup}(1^\lambda) \rightarrow (\Pi_i.\mathsf{PK}, \Pi_i.\mathsf{MSK})$ $(i \in [q])$. Furthermore, $\mathcal{B}$ can also answer those queries for oracles $\mathcal{O}_K$, and $\mathcal{O}_{cor}$ for scheme Π_i.

When $\mathcal{A}$ makes the j-th query to $\mathcal{O}^0_K$, $\mathcal{B}$ works as follows.

- Case 1: $j < k$. $\mathcal{B}$ generates a semi-functional type-3 key for $\Pi_0.X_j$ in the same way as mentioned in the proof of last *Claim*.
- Case 2: $j = k$. $\mathcal{A}$ makes a secret key query $\Pi_0.X$ or re-encryption key query $\Pi_0.X \rightarrow \Pi_i.Y$. In this case, $\mathcal{B}$ makes a secret query $\Pi_0.X$ to its challenger and receives $T = g_2^{\vec{k}(\eta,\hat{\vec{h}},\hat{\vec{r}})}$, where $\eta \xleftarrow{\$} Z_{p_2}$ or $\eta = 0$. Then $\mathcal{B}$ creates the k-th key for $\mathcal{A}$ as follows

$$\Pi_0.SK_{X_k} = g_1^{\vec{k}(\alpha,\vec{h},\vec{r})} \cdot T \cdot g_3^{\vec{R}}.$$

  If $\mathcal{A}'s$ k-th key query to $\mathcal{O}_K^0$ is a re-encryption key query, then runs $\texttt{SFRE-KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_{X_k}, \Pi_i.Y, \Pi_i.\mathsf{PK})$, and returns result to $\mathcal{A}$.
- Case 3: $j > k$. $\mathcal{B}$ responds with a normal key for $\Pi_0.X_j$ or $rk_{\Pi_0.X_j \rightarrow \Pi_i.Y_j} \leftarrow \texttt{RE-KeyGen}(\Pi_0.\mathsf{PK}, \Pi_0.SK_{X_j}, \Pi_i.Y, \Pi_i.\mathsf{PK})$.

Challenge phase: $\mathcal{A}$ outputs messages $M_0, M_1 \in \mathbb{G}_T$ along with a challenge attributes $\Pi_0.Y^*$. $\mathcal{B}$ then makes a query for $\Pi_0.Y^*$ to its challenger and receives back $\vec{T}_c = g_2^{\vec{c}(\hat{s},\hat{\vec{h}})}$. $\mathcal{B}$ runs $(\vec{c}; w_2) \leftarrow \texttt{Enc2}(\Pi_0.Y^*, N)$, picks $b \xleftarrow{\$} \{0,1\}, \vec{s} = (s, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$, and computes the challenge ciphertext $\Pi_0.CT^* = (C_0, \vec{C}_1)$ as follows:

$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = g_1^{\vec{c}(\vec{s},\vec{h})} \cdot \vec{T}_c.$$

Phase 2: $\mathcal{B}$ does the same as in **Phase 1**, except that for all query to $\mathcal{O}_K^0$, responds with normal secret keys or normal re-encryption keys.

Guess phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ wins.

□

Claim A.1.9. $|\boldsymbol{G}_{k,2}^2 \boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)$ - $\boldsymbol{G}_k \boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}_2}^{SD2}(1^\lambda)$, $1 \leq k \leq q_1$.

Proof. This proof is nearly the same as the proof of the indistinguishability between $\mathbf{Game}_{k-1}$ and $\mathbf{Game}_{k,1}^2$, except that for the k-th query of $\mathcal{A}$ to $\mathcal{O}_K^0$, $\mathcal{B}$ generates a semi-functional type-3 key. To do this , $\mathcal{B}$ runs $(\vec{k}; m_2) \leftarrow \texttt{Enc1}(\Pi_0.X_k, N)$, picks $\vec{r}, \hat{\vec{r}} \xleftarrow{\$} Z_N^{m_2}, \vec{R} \xleftarrow{\$} Z_N^{m_1}$ and computes

$$\Pi_0.SK_{X_k} = g_1^{\vec{k}(\alpha,\vec{h},\vec{r})} \cdot (W_2 W_3)^{\vec{k}(\hat{\alpha},\vec{0},\vec{0})} \cdot T^{\vec{k}(0,\vec{h},\hat{\vec{r}})} \cdot g_3^{\vec{R}}.$$

If $\mathcal{A}'s$ k-th key query to $\mathcal{O}_K^0$ is a secret query about $\Pi_0.X_k$, then returns $\Pi_0.SK_{X_k}$ to $\mathcal{A}$ directly. Otherwise, runs $rk_{\Pi_0.X_k \rightarrow \Pi_i.Y} \leftarrow \texttt{SFRE-KeyGen}(\Pi_0.\mathsf{PK},$

$\Pi_0.SK_{X_k}$, $\Pi_i.Y$, $\Pi_i.\mathsf{PK}$), and returns the re-encryption key $rk_{\Pi_0.X_j \to \Pi_i.Y}$ to $\mathcal{A}$. If $T = g_1^{t_1} g_3^{t_3}$, $\mathcal{B}$ simulated $\mathbf{G}_k$ and otherwise $\mathcal{B}$ simulated $\mathbf{G}_{k,2}^2$. □

□

Lemma A.1.4. $|\boldsymbol{G}_{k-1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_k\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq 2\boldsymbol{Adv}_{\mathcal{B}_1}^{\mathtt{CPA}}(1^\lambda) + 4\boldsymbol{Adv}_{\mathcal{B}_2}^{SD2}(1^\lambda) + \boldsymbol{Adv}_{\mathcal{B}_3}^{SMH}(1^\lambda)$, $q_1 + 1 \leq k \leq q_1 + q_2$.

Proof. This proof is very similar to Lemma A.1.3, so we omit it here. □

Lemma A.1.5. $|\boldsymbol{G}_{q_1+q_2}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{final}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{SD3}(1^\lambda)$.

Proof. Algorithm $\mathcal{B}$ takes input a problem instance(g_1, g_2,g_3,$g_1^s W_2$, $g_1^\alpha Y_2$,T) of Assumption **SD3**. It needs to decide whether $T = e(g_1, g_1)^{\alpha s}$ or $T \xleftarrow{\$} \mathbb{G}_T$. Algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ runs $\mathtt{Param}(\mathsf{k}) \to n$, picks $\vec{h} \xleftarrow{\$} Z_N^n$, computes $\Pi_0.\mathsf{PK}$ = (g_1, g_3, $e(g_1, g_1^\alpha Y_2)$, $g_1^{\vec{h}}$), and sents $\Pi_0.\mathsf{PK}$ to $\mathcal{A}$.

Phase 1: In this phase, $\mathcal{B}$ generates a semi-functional type-3 key for any kinds of key queries to $\mathcal{O}_K^0$ in the following way:
runs $Enc1(\Pi_0.X, N) \to (\mathbf{k}; m_2)$, picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\vec{r} \xleftarrow{\$} Z_N^{m_2}$, $\vec{R} \xleftarrow{\$} Z_N^{m_1}$ and computes:

$$\Pi_0.SK_X = (g_1^\alpha Y_2)^{\mathbf{k}(1,\mathbf{0},\mathbf{0})} \cdot g_1^{\vec{k}(0,\vec{r},\vec{h})} \cdot g_2^{\vec{k}(\hat{\alpha},\vec{0},\vec{0})} \cdot g_3^{\vec{R}}$$

Challenge phase: $\mathcal{A}$ outpus two challenge message $M_0, M_1 \in \mathbb{G}_T$ and a challenge attribute $\Pi_0.Y^*$, $\mathcal{B}$ runs $Enc2(\Pi_0.Y^*, N) \to (\vec{c}; w_2)$, picks $\vec{s} = (1, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$,, $b \xleftarrow{\$} \{0,1\}$, computes challenge ciphertext $\Pi_0.CT = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = T \cdot M_b \qquad \mathbf{C}_1 = (g_1^s W_2)^{\vec{c}(\vec{s},\vec{h})}$$

Phase 2: $\mathcal{B}$ generates a semi-functional type-3 key for key queries in the same way as **Phase 1**.

Guess Phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

□

A.2 Proof for Theorem 4.3.1

In this section, we clarify the security proof of **Theorem 1**. We use the following lemmas to prove the indistinguishability between the consecutive games. We use $\mathbf{G}_j\mathbf{Adv}_{\mathcal{A}}(1^\lambda)$ to denote the advantage of $\mathcal{A}$ in game $\mathbf{Game}_j$. Let $\mathbf{Adv}_{\mathcal{B}}^{HP}$ be the advantage of adversary $\mathcal{B}$ for hard problem HP. By reducing the indistinguishability between games to the hardness of the challenge problems for simulator, we complete the proof.

In order to simulate properly for $\mathcal{A}$, simulator $\mathcal{B}$ maintains a set $\mathcal{T} = \{(h, \mathcal{X}, Key, L)\}$. Each tuple is made up of handles, attributes, keys(normal or semi-functional keys) and the corresponding leakage bits of keys. Handles index $h \in Z_{q_1+q_2}$ (q_1 and q_2 are the numbers of **Create** queries made in **Phase 1** and **Phase 2** respectively) and sets the handle counter H to 0. Notably, once a key in a tuple is created by $\mathcal{B}$, all subsequent queries (**Leakge** or **Reveal**) about this key act on the created version.

Proof.

Lemma A.2.1.

$$|\boldsymbol{G}_{real}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) \; - \; \boldsymbol{G}_{res}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \quad \leq \quad \boldsymbol{Adv}_{\mathcal{B}}^{SD1}(1^\lambda) \; + \; \boldsymbol{Adv}_{\mathcal{B}}^{SD2}(1^\lambda).$$

Proof. Assume there is an adversary $\mathcal{A}$ that can submit a query X such that $R(X, Y^*) = 1(\text{mod } p_2)$ but $R(X, Y^*) = 0(\text{mod } N)$. Then we can construct an adversaty $\mathcal{B}$ that will break Assumption **SD1** or Assumption **SD2**. Given $(g_1 \xleftarrow{\$} G_{p_1}, Z_3 \xleftarrow{\$} G_{p_1}, N, \mathbb{G}, \mathbb{G}_{\mathbb{T}}, e)$ $\mathcal{B}$ simulates $Game_{real}$ for $\mathcal{A}$. Once $\mathcal{A}$ submits a query X such that $R(X, Y^*) = 1(\text{mod } p_2)$ but $R(X, Y^*) = 0$ (mod N) with non-negligible probability, $\mathcal{B}$ use the corresponding algorithm $\mathcal{F}$ to find a factor a, such that $p_2|a$ and $a|N$. Let $b = \frac{N}{a}$. Then we consider two cases:

- Case 1: $p_1|b$. $\mathcal{B}$ will break Assumption **SD1** by verifying g_1^b is the identity and then check whether T^b is identity. If yes, then $T \xleftarrow{\$} G_{p_1}$, else $T \xleftarrow{\$} G_{p_1p_2}$.
- Case 2: $a = p_1p_2, b = p_3$. $\mathcal{B}$ will break Assumption **SD2**. As $\mathcal{B}$ is additionally given $(Z_1Z_2 \xleftarrow{\$} G_{p_1p_2}, Z_3 \xleftarrow{\$} G_{p_3}, W_2W_3 \xleftarrow{\$} G_{p_2p_3})$, it will check $(Z_1Z_2)^a$ and Z_3^b are both indetity. Then it will verify whether $e(W_2W_3, T^b)$ is identity, if yes, $T \xleftarrow{\$} G_{p_1p_3}$; else $T \xleftarrow{\$} G_{p_1p_2p_3}$.

□

Lemma A.2.2. $|\boldsymbol{G}_{res}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_0\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}^{SD1}_{\mathcal{B}}(1^\lambda)$.

Proof. If the advantage of $\mathcal{A}$ between $\mathbf{G}_{res}$ and $\mathbf{G}_0$ is non-negligible, then we can build an algorithm $\mathcal{B}$ that can break Assumption **SD1**. On input a problem instance $(\mathbb{G}, \mathbb{G}_\mathbb{T}, N, e, g_1, g_3, T)$ for Assumption **SD1**, $\mathcal{B}$ needs to decide whether $T \xleftarrow{\$} G_{p_1}$ or $T \xleftarrow{\$} G_{p_1p_2}$.
$\mathcal{B}$ works as follows:

1. **Setup Phase.** $\mathcal{B}$ runs $Param(\lambda) \rightarrow (n_1, n_2)$ and picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}$, $\mathbf{x} \xleftarrow{\$} Z_N^{n_2}$ and $\alpha \xleftarrow{\$} Z_N$. $\mathcal{B}$ runs $Enc1(\varepsilon, N) \rightarrow (\mathbf{k}_1, \mathbf{k}_2; m_1, m_2, m_3)$. and computes PK = $(g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$. $\mathcal{B}$ gives PK to $\mathcal{A}$ and adds $(0, \varepsilon, \varepsilon, 0)$ to set $\mathcal{T}$.

2. **Phase 1.** For all $Create(h, X_j)(j \in [1, q_1])$ queries requested by $\mathcal{A}$ in this phase, $\mathcal{B}$ first scans $\mathcal{T}$ to find whether tuple h refers to a master-key tuple. If not, return $\perp$. Otherwise, $\mathcal{B}$ generates a normal key for X_j in the following way:
runs $Enc1(X_j, N) \rightarrow (\mathbf{k}_{X_j,1}, \mathbf{k}_{X_j,2}; m_1, m_2, m_3)$, picks $\mathbf{r}_j \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{X_j,1} \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_{X_j,2} \xleftarrow{\$} Z_N^{m_2}$ and computes:

$$K_j = (g_1^{\mathbf{k}_{X_j,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}_j)} * g_3^{\mathbf{R}_{X_j,1}}, g_1^{\mathbf{k}_{X_j,2}(\mathbf{h},\mathbf{r}_j)} * g_3^{\mathbf{R}_{X_j,2}}).$$

After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_j, 0)$ to $\mathcal{T}$ and updates handle counter to $H+1 \leftarrow H$. Notably, if X_j is empty, this means K_j is a master key.

3. **Challenge Phase** In this phase, $\mathcal{A}$ outputs two messages $M_0, M_1 \in \mathbb{G}_T$ and the challenge attribute Y^*. $\mathcal{B}$ picks $b \xleftarrow{\$} \{0, 1\}$, runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}, w_2)$, picks $\mathbf{s} = (s, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{w_2+1}$, and computes the challenge ciphertext $CT = (C_0, \mathbf{C}_1)$.

$$C_0 = M_b \cdot e(T, g_1)^{\alpha s}, \qquad \mathbf{C}_1 = T^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}.$$

4. **Phase 2.** $\mathcal{B}$ answers the create queries made by $\mathcal{A}$ as the same way in **Phase 1**.

5. **Guess Phase.** $\mathcal{A}$ outputs a guess b' . If $b' = b$, then $\mathcal{A}$ successes.

We claim that as T chosen from different subgroups, $\mathcal{B}$ properly simulates $\mathbf{Game}_{res}$ or $\mathbf{Game}_0$ for $\mathcal{A}$.

- If $T \xleftarrow{\$} G_{p_1}$, $\mathcal{B}$ properly simulates $\mathbf{Game}_{res}$ as the challenge ciphertext generated by $\mathcal{B}$ is distributed identically as normal ciphertexts. Without loss of

generatity, we assume that $T = g_1^{t_1}$, where $t_1 \xleftarrow{\$} Z_{p_1}$. Then we observe that the challenge ciphertext is

$$C_0 = e(g_1, g_1)^{t_1 \alpha s} M_b \qquad C_1 = g_1^{t_1 \mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})} = g_1^{\mathbf{c}(t_1\mathbf{s},\mathbf{h},\mathbf{x})},$$

where $t_1\mathbf{s} (\text{mod } p_1)$ can be viewed as a randomly variable in $Z_{p_1}^{w_2+1}$. It is easy to see that this challenge ciphertext is distributed identically as a normal ciphertext.

- If $T \xleftarrow{\$} G_{p_1 p_2}$, $\mathcal{B}$ properly simulates $\mathbf{Game}_0$. In this situation, $\mathcal{B}$ generates the challenge ciphertext distributed identically as semi-functional ciphertext. Assume $T = g_1^{t_1} g_2^{t_2}$, where $t_1 \xleftarrow{\$} Z_{p_1}$ and $t_2 \xleftarrow{\$} Z_{p_2}$. Then we observe that the challenge ciphertext is

$$C_0 = e(g_1, g_1)^{t_1 \alpha s} M_b \qquad C_1 = g_1^{t_1 \mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})} * g_2^{t_2 \mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}.$$

where we implicitly set $\hat{\mathbf{s}} = t_2\mathbf{s} \bmod p_2$ and the semi-functional parameters $\hat{\mathbf{h}} = \mathbf{h} \bmod p_2$ and $\hat{\mathbf{x}} = \mathbf{x} \bmod p_2$. By Chinese Reminder Theorem, $\hat{\mathbf{h}}$ and $\hat{\mathbf{x}}$ are properly distributed as they are independent from $\mathbf{h} \bmod p_1$ and $\mathbf{x} \bmod p_1$ respectively. In this way, $\mathcal{B}$ perfectly generates a semi-functional ciphertext for $\mathcal{A}$.

□

Lemma A.2.3. $|\boldsymbol{G}_{k-1,3}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{SD2}(1^\lambda), k \in [1, q_1]$.

Proof. If there is an adversary $\mathcal{A}$ can distinguish $\mathbf{G}_{k-1,3}$ and $\mathbf{G}_{k,1}$, we will build an algorithm $\mathcal{B}$ who can break Assumption **SD2**. On input a problem instance $(\mathbb{G}, \mathbb{G}_T, N, e, g_1, Z_1 Z_2, g_3, W_2 W_3, T)$, $\mathcal{B}$ needs to decide whether $T \in G_{p_1 p_2 p_3}$ or $G_{p_1 p_3}$.
Algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ runs $Param(k) \rightarrow (n_1, n_2)$, picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}, \mathbf{r} \xleftarrow{\$} Z_N^{m_3}$ and $\alpha \leftarrow Z_N$, computes $PK = (g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$. $\mathcal{B}$ sends PK to $\mathcal{A}$ and adds tuple 0 of $\mathcal{T}$ to be the first tuple $(0, \varepsilon, \varepsilon, 0)$.

Phase 1: In this phase, when $\mathcal{A}$ makes the j-th $\mathbf{Create}(h, X_j)$ queries for $X_j \in \mathcal{X}$ or $X_j = \varepsilon$, $\mathcal{B}$ first scans $\mathcal{T}$ to find tuple h. If h does not refer to a master key tuple or this tuple does not exist, then responds with $\perp$.
Otherwise, $\mathcal{B}$ does as follows:

- **Case 1:** $[j < k]$. $\mathcal{B}$ generates a semi-functional type-3 key for X_j by working as follows:
  runs $Enc1(X_j, N) \rightarrow (\mathbf{k}_{X_j,1}, \mathbf{k}_{X_j,2}; m_1, m_2, m_3)$,picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\mathbf{r} \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_1 \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_2 \xleftarrow{\$} Z_N^{m_2}$ and computes:

$$K_{j,SF3} = (g_1^{\mathbf{k}_{X_j,1}(\alpha,\mathbf{h},\boldsymbol{x},\mathbf{r})} * (W_2W_3)^{\mathbf{k}_{X_j,1}(\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})} * g_3^{R_1}, \\ g_1^{\mathbf{k}_{X_j,2}(\mathbf{h},\mathbf{r})} * (W_2W_3)^{\mathbf{k}_{X_j,2}(\mathbf{0},\mathbf{0})} * g_3^{R_2})$$

  After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_{j,SF3}, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H+1$.

- **Case 2:**$[j = k]$. $\mathcal{B}$ runs $Enc1(X_k, N) \rightarrow (\mathbf{k}_{X_k,1}, \mathbf{k}_{X_k,2}; m_1, m_2, m_3)$, picks $\mathbf{r}_k, \hat{\mathbf{r}}_k \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{k,1} \xleftarrow{\$} Z_N^{m_1}$ and $\mathbf{R}_{k,2} \xleftarrow{\$} Z_N^{m_2}$ and computes

$$K^* = (g_1^{\mathbf{k}_{X_k,1}(\alpha,\mathbf{h},\boldsymbol{x},\mathbf{r}_k)} * T^{\mathbf{k}_{X_k,1}(0,\mathbf{h},\boldsymbol{x},\hat{\mathbf{r}}_k)} * g_3^{\mathbf{R}_{k,1}}, g_1^{\mathbf{k}_{X_k,2}(\mathbf{h},\mathbf{r}_k)} * T^{\mathbf{k}_{X_k,2}(\mathbf{h},\hat{\mathbf{r}}_k)} * g_3^{\mathbf{R}_{k,2}})$$

  After that, $\mathcal{B}$ adds tuple $(H+1, X_k, K^*, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H+1$.

- **Case 3:**$[j > k]$. $\mathcal{B}$ generates a normal key for X_j.

Challenge phase: $\mathcal{A}$ outpus two challenge message $M_0, M_1 \in \mathbb{G}_T$ and a challenge attribute Y^*. $\mathcal{B}$ first checks whether $Y^* \in R$, if yes, return $\perp$. Otherwise, $\mathcal{B}$ runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}; \omega_2)$, picks $\mathbf{s} = (s, s_1, s_2, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$, $b \xleftarrow{\$} Z_N$, computes challenge ciphertext $CT = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = (Z_1Z_2)^{\mathbf{c}(\mathbf{s},\mathbf{h},\boldsymbol{x})}$$

Phase 2: In this phase, $\mathcal{B}$ answers all **Create** queries by generating normal keys.

Guess phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

We claim that the master key or user secret key generated by $\mathcal{B}$ in **Phase 1** is properly distributed as them in $\mathbf{G}_{k-1,3}$ or $\mathbf{G}_{k,1}$.

- It is obvious that keys generated by $\mathcal{B}$ in **Case 1** (resp. **Case 3**) are properly distributed as semi-functional type-3 keys(resp. normal keys).

- In **Case 2:** As $T \xleftarrow{\$} G_{p_1p_3}$ or $G_{p_1p_2p_3}$, we denote it as $T = g_1^{t_1}g_3^{t_3}$ or $T = g_1^{t_1}g_2^{t_2}g_3^{t_3}$. The $\mathbb{G}_{p_1}$ component in the k-th key is :

$$(g_1^{\mathbf{k}_{X_k,1}(\alpha,\mathbf{h},\mathbf{x},(\mathbf{r}+t_1\hat{\mathbf{r}}_j))}, g_1^{\mathbf{k}_{X_k,2}(\mathbf{h},(\mathbf{r}+t_1\hat{\mathbf{r}}_j))}).$$

  It is easy to see that this component is properly distributed as in normal key or semi-functional type-1 key if we take the radomness as $\mathbf{r} + t_1\hat{\mathbf{r}}_j$.
  If $T \in G_{p_1p_3}$, then K^* has no $\mathbb{G}_{p_2}$ parts, K^* is a normal key, then $\mathcal{B}$ has properly simulated $\mathbf{G}_{k-1,3}$.
  If $T \in G_{p_1p_2p_3}$, the $\mathbb{G}_{p_2}$ parts in K^* is as follows:

$$(g_2^{\mathbf{k}_{X_k,1}(0,\mathbf{h},\mathbf{x},t_2\hat{\mathbf{r}}_j)}, g_2^{\mathbf{k}_{X_k,2}(\mathbf{h},t_2\hat{\mathbf{r}}_j))}).$$

  By the Chinese Remainder Theorem, $\mathbf{h}$ mod p_2 and $\mathbf{x}$ mod p_2 are independent with $\mathbf{h}$ mod p_1 and $\mathbf{x}$ mod p_1. They can be viewed as random variables, so K^* is properly distributed as semi-functional type-1 master keys. $\mathcal{B}$ has properly simulated $\mathbf{G}_{k,1}$.

□

Lemma A.2.4.

$$|\boldsymbol{G}_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{k,2}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{CMH} + \boldsymbol{Adv}_{\mathcal{B}}^{LR}(1^\lambda), k \in [1, q_1].$$

Proof. If there is an advansary $\mathcal{A}$ whose advantage in distinguishing $\mathbf{G}_{k,1}$ and $\mathbf{G}_{k,2}$ is non-negligible, then we can build an algorithm $\mathcal{B}$ who can break the co-selective security or leakage-resilient security of the underlying pair encoding scheme P.

Let K^* denotes the key generated by simulator $\mathcal{B}$ for the k-th **Create**(h, X_k) query of $\mathcal{A}$ in **Phase 1**. Let us define two events as follows:

- F be the event that: for all subsequent queries, $\mathcal{A}$ will not make reveal queries for K^*.
- $\neg F$ denotes the event that for all subsequent queries, $\mathcal{A}$ will make reveal queries for K^*.

It is easy to see that F and $\neg F$ are complementary. Let $Pr[F]$ denotes the probability that event F occurs, and $Pr[\neg F]$ denotes the probability that event $\neg F$ occurs. Indeed, $Pr[F] + Pr[\neg F] = 1$.

In order to complete the proof of the **Lemma** A.2.4, it suffices to prove the following claims instead:

Claim A.2.1. *In the case that event F occurs, we will prove that if $\mathcal{A}$ can distinguish these two games, then $\mathcal{B}$ will break the leakage-resilient security of the underlying pair encoding scheme P. Indeed, $|\boldsymbol{G}_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)$ - $\boldsymbol{G}_{k,2}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}^{LR}_{\mathcal{B}}(1^\lambda)$, conditioned on F occurs.*

Proof. Let $\mathcal{B}$ be an adversary for leakage-resilient security of P. Given (g_1, g_2, g_3), $\mathcal{B}$ works as follows:

Setup: Algorithm $\mathcal{B}$ runs $Param(\lambda) \rightarrow (n_1, n_2)$, picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}$, $\mathbf{x} \xleftarrow{\$} Z_N^{n_2}$ and $\alpha \xleftarrow{\$} Z_N$, computes $PK = (g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$, and gives PK to $\mathcal{A}$.

Phase 1: When $\mathcal{A}$ makes the j-th **Create**(h, X_j)query, $\mathcal{B}$ first checks whether h refers to a master key tuple. If not, return $\perp$. Otherwise, $\mathcal{B}$ works as follows:

- **Case 1:** $j < k$. $\mathcal{B}$ generates a semi-functional type-3 key for X_j by working as follows:
  runs $Enc1(X_j, N) \rightarrow (\mathbf{k}_{X_j,1}, \mathbf{k}_{X_j,2}; m_1, m_2, m_3)$, picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\mathbf{r}_j \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{j,1} \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_{j,2} \xleftarrow{\$} Z_N^{m_2}$ and computes:

  $$K_{j,SF3} = (g_1^{\mathbf{k}_{X_j,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}_j)} * g_2^{\mathbf{k}_{X_j,1}(\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})} * g_3^{R_{j,1}}, g_1^{\mathbf{k}_{X_j,2}(\mathbf{h},\mathbf{r}_j)} * g_2^{\mathbf{k}_{X_j,2}(\mathbf{0},\mathbf{0})} * g_3^{R_{j,2}})$$

  After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_{j,SF3}, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H + 1$.

- **Case 2:** $j = k$. In this case, $\mathcal{B}$ adds a tuple $(H+1, X_k, \varepsilon, 0)$ to set $\mathcal{T}$ and updates the handle counter to $H \leftarrow H + 1$, where the key filed is set to be empty.
  As this reduction conditioned on event F occurs, this means $\mathcal{A}$ can only make leakage query to this tuple in the subsequent queries. Once $\mathcal{A}$ makes a leakage query for this key by inputting a leakage function f, $\mathcal{B}$ encodes f into its own leakage queries for T (T =($g_2^{\mathbf{k}_{X_k,1}(\beta,\hat{\mathbf{h}},\hat{\mathbf{x}},\hat{\mathbf{r}})}$, $g_2^{\mathbf{k}_{X_k,2}(\hat{\mathbf{h}},\hat{\mathbf{r}})}$), $\beta = 0$ or $\beta \xleftarrow{\$} Z_{p_2}$) by implicitly setting the challenge key to be:

  $$K^* = (g_1^{\mathbf{k}_{X_k,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r})} * g_3^{\mathbf{R}_1}, g_1^{\mathbf{k}_{X_k,2}(\mathbf{h},\mathbf{r})} * g_3^{\mathbf{R}_2}) * T,$$

  where $Enc1(X_k, N) \rightarrow (\mathbf{k}_{X_k,1}, \mathbf{k}_{X_k,2}; m_1, m_2, m_3)$, $\mathbf{r} \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_1 \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_2 \xleftarrow{\$} Z_N^{m_2}$. $\mathcal{B}$ makes leakage queries $f'(T) = f(K^*)$ to its own challenger and forwards the result to $\mathcal{A}$.

- **Case 3:** $j > k$. In this case, $\mathcal{B}$ generates a normal key for X_j.

Challenge phase: $\mathcal{A}$ outputs messages $M_0, M_1 \in \mathbb{G}_T$ along with a challenge attributes Y^* such that for all $X \in \mathcal{R}$, $R(X, Y^*) = 0 \bmod p_2$. $\mathcal{B}$ then makes a query for Y^* to its challenger and receives back $\mathbf{T}_c = g_2^{\mathbf{c}_{Y^*}(\hat{s},\hat{\mathbf{h}},\hat{\mathbf{x}})}$. Then runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}_{Y^*}; w_2)$, picks $b \xleftarrow{\$} \{0,1\}$, $\mathbf{s} = (s, s_1, s_2, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$,and computes the properly distributed semi-functional challenge ciphertext $CT^* = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = g_1^{\mathbf{c}_{Y^*}(\mathbf{s},\mathbf{h},\mathbf{x})} * \mathbf{T}_c.$$

Phase 2: In this phase, $\mathcal{B}$ answers all **Create** queries by generating normal keys as in **Phase 1: Case 3**.

Guess phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

□

Claim A.2.2. *In the case that event F does not occur, we will prove that if $\mathcal{A}$ can distinguish these two games, then $\mathcal{B}$ will break the co-selective security of the underlying pair encoding scheme P. Indeed, $|\boldsymbol{G}_{k,1}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)$ - $\boldsymbol{G}_{k,2}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{CMH}(1^\lambda)$, conditioned on F does not occur.*

Proof. Let $\mathcal{B}$ be an adversary for the co-selective security of P. $\mathcal{B}$ is given (g_1, g_2, g_3) as in co-selective game and works as follows:

Setup: Algorithm $\mathcal{B}$ runs $Param(\lambda) \rightarrow (n_1, n_2)$, picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}$ and $\mathbf{x} \xleftarrow{\$} Z_N^{n_2}$, computes $PK = (g_1, g_3, e(g_1, g_1)^\alpha, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$, and gives PK to $\mathcal{A}$.

Phase 1: When $\mathcal{A}$ makes the j-th **Create**(h, X_j) query, $\mathcal{B}$ first checks whether h refers to a master key tuple. If not, return $\perp$. Otherwise, $\mathcal{B}$ works as follows:

- **Case 1:** $j < k$. $\mathcal{B}$ generates a semi-functional type-3 key for X_j in the same way as mentioned in the proof of **Claim A5**.
- **Case 2:** $j = k$.
  In this case, $\mathcal{B}$ makes a query X_j to its challenger and receives T =($g_2^{\mathbf{k}_{X_j,1}(\beta,\hat{\mathbf{h}},\hat{\mathbf{x}},\hat{\mathbf{r}})}$, $g_2^{\mathbf{k}_{X_j,2}(\hat{\mathbf{h}},\hat{\mathbf{r}})}$). $\mathcal{B}$ need to guess if β is randomly chosen from Z_{p_2} or is 0. Then $\mathcal{B}$ creates the k-th key for $\mathcal{A}$ as follows

$$K^* = (g_1^{\mathbf{k}_{X_k,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}_k)} * g_3^{\mathbf{R}_{k,1}}, g_1^{\mathbf{k}_{X_k,2}(\mathbf{h},\mathbf{r}_k)} * g_3^{\mathbf{R}_{k,2}}) * T,$$

where $Enc1(X_k, N) \rightarrow (\mathbf{k}_{X_k,1}, \mathbf{k}_{X_k,2}; m_1, m_2, m_3)$, $\mathbf{r}_k \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{k,1} \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_{k,2} \xleftarrow{\$} Z_N^{m_2}$.

After that, $\mathcal{B}$ adds tuple $(H+1, X_k, K^*, 0)$ to the set $\mathcal{T}$ and updates the handle counter to $H \leftarrow H+1$.

- **Case 3:** $j > k$. $\mathcal{B}$ generates normal key for X_j.

Challenge phase: $\mathcal{A}$ outputs messages $M_0, M_1 \in \mathbb{G}_T$ along with a challenge attributes Y^* such that for all $X \in \mathcal{R}$, $R(X, Y^*) = 0 \bmod p_2$. $\mathcal{B}$ then makes a query for Y^* to its challenger and receives back $\mathbf{T}_c = g_2^{\mathbf{c}_{Y^*}(\hat{s}, \hat{\mathbf{h}}, \hat{\mathbf{x}})}$. $\mathcal{B}$ runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}_{Y^*}; w_2)$, picks $b \xleftarrow{\$} \{0,1\}$, $\mathbf{s} = (s, s_1, s_2, \ldots, s_{w_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$,, and computes the challenge ciphertext $CT^* = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = g_1^{\mathbf{c}_{Y^*}(\mathbf{s}, \mathbf{h}, \mathbf{x})} * \mathbf{T}_c.$$

Note that this reduction based on the fact that $\neg F$ occurs, which means that $\mathcal{A}$ must make reveal query for K^*, namely X_k must be added to the reveal attribute set $\mathcal{R}$. Thus, it is obvious that $Y^* \neq X_k$. As a result, $\mathcal{B}$ can always receive back $\mathbf{T}_c$ for Y^* from its challenger.

Phase 2: In this phase, $\mathcal{B}$ answers all **Create** queries by generating normal keys.

Guess phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

□

□

Lemma A.2.5. *For all* $k \in [1, q_1]$, $|\boldsymbol{G}_{k,2}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{k,3}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{SD2}(1^\lambda)$.

Proof. If there is an adversary $\mathcal{A}$ who can distinguish $\mathbf{G}_{k,2}$ and $\mathbf{G}_{k,3}$, then we will build an algorithm $\mathcal{B}$ who can break Assumption **SD2**. On input a problem instance $(\mathbb{G}, \mathbb{G}_T, N, e, g_1, Z_1Z_2, Z_3, W_2W_3, T)$, $\mathcal{B}$ needs to decide whether $T \in G_{p_1p_2p_3}$ or $G_{p_1p_3}$. We denote it as $T = g_1^{t_1}g_3^{t_3}$ or $T = g_1^{t_1}g_2^{t_2}g_3^{t_3}$. Algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ runs $Param(k) \rightarrow (n_1, n_2)$, picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}$, and $\alpha \leftarrow Z_N$, computes $PK = (g_1, Z_3, e(g_1, g_1)^\alpha, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$. $\mathcal{B}$ sends PK to $\mathcal{A}$ and adds tuple 0 of $\mathcal{T}$ to be the first tuple $(0, \varepsilon, \varepsilon, 0)$.

Phase 1: In this phase, when $\mathcal{A}$ makes the j-th **Create**(h, X_j) queries for $X_j \in \mathcal{X}$ or $X_j = \varepsilon$, $\mathcal{B}$ first scans $\mathcal{T}$ to find whether tuple h refers to a master key

tuple. If not, then responds with $\perp$.
Otherwise $\mathcal{B}$ does as follows:

- **Case 1:** $[j < k]$. $\mathcal{B}$ generates a semi-functional type-3 key for X_j in the following way:
  runs $Enc1(X_j, N) \rightarrow (\mathbf{k}_{X_j,1}, \mathbf{k}_{X_j,2}; m_1, m_2, m_3)$,picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\mathbf{r}_j \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{j,1} \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_{j,2} \xleftarrow{\$} Z_N^{m_2}$ and computes:

$$K_{j,SF3} = (g_1^{\mathbf{k}_{X_j,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}_j)} * (W_2W_3)^{\mathbf{k}_{X_j,1}(\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})} * g_3^{R_{j,1}},$$
$$g_1^{\mathbf{k}_{X_j,2}(\mathbf{h},\mathbf{r}_j)} * (W_2W_3)^{\mathbf{k}_{X_j,2}(\mathbf{0},\mathbf{0})} * g_3^{R_{j,2}}).$$

  After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_{j,SF3}, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H+1$.

- **Case 2:**$[j = k]$. $\mathcal{B}$ runs $Enc1(X_k, N) \rightarrow (\mathbf{k}_{X_k,1}, \mathbf{k}_{X_k,2;m_1,m_2,m_3})$, picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\mathbf{r}_k, \hat{\mathbf{r}}_k \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{k,1} \xleftarrow{\$} Z_N^{m_1}$ and $\mathbf{R}_{k,2} \xleftarrow{\$} Z_N^{m_2}$, and computes

$$K^* = (g_1^{\mathbf{k}_{X_k,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}_k)} * (W_2W_3)^{\mathbf{k}_{X_k,1}(\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})} * (T)^{\mathbf{k}_{X_k,1}(0,\mathbf{h},\mathbf{x},\hat{\mathbf{r}}_k)} \cdot Z_3^{\mathbf{R}_{k,1}},$$
$$g_1^{\mathbf{k}_{X_k,2}(\mathbf{h},\mathbf{r}_k)} * (W_2W_3)^{\mathbf{k}_{X_k,2}(\mathbf{0},\mathbf{0})} \cdot (T)^{\mathbf{k}_{X_k,2}(\mathbf{h},\hat{\mathbf{r}}_k)} * Z_3^{\mathbf{R}_{k,2}}).$$

  After that, $\mathcal{B}$ adds tuple $(H+1, X, K^*, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H+1$.

  We claim that K^* is the properly distributed semi-functional type-2 or semi-functional type-3 key. Firstly, we observe that the $\mathbb{G}_{p_1}$ component in K^* is :

$$(g_1^{\mathbf{k}_{X_k,1}(\alpha,\mathbf{h},\mathbf{x},(\mathbf{r}_k+t_1\hat{\mathbf{r}}_k))}, g_1^{\mathbf{k}_{X_k,2}(\mathbf{h},(\mathbf{r}+t_1\hat{\mathbf{r}}_k))}).$$

  It is easy to see this component is properly distributed as in semi-functional type-2 or semi-functional type-3 if we take the radomness as $\mathbf{r} + t_1\hat{\mathbf{r}}_j$.
  Let W_2W_3 to be $g_2^{\hat{W}_2} g_3^{\hat{W}_3}$.

  - if $T = g_1^{t_1} g_3^{t_3}$, then K^* is a properly distributed semi-functional type-3 key. The $\mathbb{G}_{p_2}$ component in K^* is distributed as follows

$$(g_2^{\mathbf{k}_1(\hat{W}_2\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})}, g_2^{\mathbf{k}_2(\mathbf{0},\mathbf{0}))}).$$

– if $T = g_1^{t_1} g_2^{t_2} g_3^{t_3}$, then K^* is a properly distributed semi-functional type-2 key. The $\mathbb{G}_{p_2}$ component in K^* is distributed as follows

$$(g_2^{\mathbf{k}_1(\hat{W}_2\hat{\alpha},\mathbf{h},\mathbf{x},t_2\hat{\mathbf{r}}_j)}, g_2^{\mathbf{k}_2(\mathbf{h},t_2\hat{\mathbf{r}}_j))}).$$

- **Case 3:**$[j > k]$. $\mathcal{B}$ generates a normal key for X_j.

Challenge phase: $\mathcal{A}$ outpus two challenge message $M_0, M_1 \in \mathbb{G}_T$ and a challenge attribute Y^*. $\mathcal{B}$ first checks whether $Y^* \in R$, if yes, return $\perp$. Otherwise, $\mathcal{B}$ runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}; w_2)$, picks $\mathbf{s} = (s, s_1, s_2, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$, $b \xleftarrow{\$} \{0,1\}$, computes challenge ciphertext $CT = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = (Z_1 Z_2)^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}$$

Phase 2: In this phase, $\mathcal{B}$ answers all **Create** queries by generating normal keys .

Guess phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

□

Lemma A.2.6. $|\boldsymbol{G}_{q_1,3}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{21}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{SD2}(1^\lambda)$.

Proof. If there is an adversary $\mathcal{A}$ who can distinguish $\mathbf{G}_{q_1,3}$ and $\mathbf{G}_{2,1}$, we will build an algorithm $\mathcal{B}$ who can break Assumption **SD2**. On input a problem instance $(\mathbb{G}, \mathbb{G}_T, N, e, g_1, Z_1Z_2, Z_3, W_2W_3, T)$, $\mathcal{B}$ needs to decide whether $T \in G_{p_1p_2p_3}$ or $G_{p_1p_3}$. We denote it as $T = g_1^{t_1} g_3^{t_3}$ or $g_1^{t_1} g_2^{t_2} g_3^{t_3}$.
Algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ runs $Param(k) \rightarrow (n_1, n_2)$, picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}$, and $\alpha \leftarrow Z_N$, computes $PK = (g_1, Z_3, e(g_1, g_1)^\alpha, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$. $\mathcal{B}$ sends PK to $\mathcal{A}$ and adds tuple 0 of $\mathcal{T}$ to be the first tuple $(0, \varepsilon, \varepsilon, 0)$.

Phase 1: In this phase, when $\mathcal{A}$ makes the j-th **Create**(h, X_j) queries for $X_j \in \mathcal{X}$ or $X_j = \varepsilon$, $\mathcal{B}$ first scans $\mathcal{T}$ to find whether tuple h refers to a master key tuple. If not, then responds with $\perp$. Otherwise, $\mathcal{B}$ generates a semi-functional type-3 key for X_j in the following way:
runs $Enc1(X_j, N) \rightarrow (\mathbf{k}_{X_j,1}, \mathbf{k}_{X_j,2}; m_1, m_2, m_3)$,picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\mathbf{r}_j \xleftarrow{\$} Z_N^{m_3}$,

$\mathbf{R}_{j,1} \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_{j,2} \xleftarrow{\$} Z_N^{m_2}$ and computes:

$$K_{j,SF3} = (g_1^{\mathbf{k}_{X_j,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}_j)} * (W_2W_3)^{\mathbf{k}_{X_j,1}(\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})} * g_3^{R_{j,1}}, \\ g_1^{\mathbf{k}_{X_j,2}(\mathbf{h},\mathbf{r}_j)} * (W_2W_3)^{\mathbf{k}_{X_j,2}(\mathbf{0},\mathbf{0})} * g_3^{R_{j,2}}).$$

After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_{j,SF3}, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H+1$.

Challenge phase: $\mathcal{A}$ outpus two challenge message $M_0, M_1 \in \mathbb{G}_T$ and a challenge attribute Y^*. $\mathcal{B}$ first checks whether $Y^* \in R$, if yes, return $\perp$. Otherwise, $\mathcal{B}$ runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}; w_2)$, picks $\mathbf{s} = (s, s_1, s_2, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$,, $b \xleftarrow{\$} \{0,1\}$, computes challenge ciphertext $CT = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = (Z_1Z_2)^{\mathbf{c}(\mathbf{s},\mathbf{h},\mathbf{x})}$$

Phase 2: For all **Create**(h, X_j) queries in this phase, $\mathcal{B}$ runs $Enc1(X_j, N) \rightarrow (\mathbf{k}_{X_j,1}, \mathbf{k}_{X_j,2;m_1,m_2,m_3})$, picks $\mathbf{r}_j, \hat{\mathbf{r}}_j \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{j,1} \xleftarrow{\$} Z_N^{m_1}$ and $\mathbf{R}_{j,2} \xleftarrow{\$} Z_N^{m_2}$, and computes

$$K_j = (g_1^{\mathbf{k}_{X_j,1}(\alpha,\mathbf{h},\mathbf{x},\mathbf{r}_j)} * (T)^{\mathbf{k}_{X_j,1}(0,\mathbf{h},\mathbf{x},\hat{\mathbf{r}}_{j,})} * Z_3^{\mathbf{R}_{j,1}}, g_1^{\mathbf{k}_{X_j,2}(\mathbf{h},\mathbf{r}_j)} * (T)^{\mathbf{k}_{X_j,2}(\mathbf{h},\hat{\mathbf{r}}_j)} * Z_3^{\mathbf{R}_{j,2}}).$$

After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_j, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H+1$.

We claim that for $j \in [1, q_2]$, K_j is the properly distributed as normal key or semi-functional type-1 key. If $T = g_1^{t_1} g_3^{t_3}$, K_j is properly distributed as normal key. If $T = g_1^{t_1} g_2^{t_2} g_3^{t_3}$, K_j is properly distributed as semi-functional type-1 key.

Guess phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

□

Lemma A.2.7. $|\boldsymbol{G_{21}Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G_{22}Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{SMH}(1^\lambda)$.

Proof. If there is an adversary $\mathcal{A}$ who can distinguish $\mathbf{G}_{21}$ and $\mathbf{G}_{22}$, we will build an algorithm $\mathcal{B}$ who can break the selective security of the underlying pair encoding scheme P. Given (g_1, g_2, g_3) as in selective game, algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ runs $Param(k) \rightarrow (n_1, n_2)$, picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}$, and $\alpha \leftarrow Z_N$, computes $PK = (g_1, g_3, e(g_1, g_1)^{\alpha}, g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$. $\mathcal{B}$ sends PK to $\mathcal{A}$ and adds tuple 0 of $\mathcal{T}$ to be the first tuple $(0, \varepsilon, -, 0)$.

Phase 1: In this phase, when $\mathcal{A}$ makes the j-th **Create**(h, X_j) queries, $\mathcal{B}$ first scans $\mathcal{T}$ to find whether tuple h refers to a master key tuple. If not, then responds with $\perp$. Otherwise, $\mathcal{B}$ generates a semi-functional type-3 key for X_j.

Challenge phase: $\mathcal{A}$ outputs messages $M_0, M_1 \in \mathbb{G}_T$ along with a challenge attributes Y^* such that for all $X \in \mathcal{R}$, $R(X, Y^*) = 0 \bmod p_2$. $\mathcal{B}$ then makes a query for Y^* to its challenger and receives back $\mathbf{T}_c = g_2^{\mathbf{c}_{Y^*}(\hat{s}, \hat{\mathbf{h}}, \hat{\mathbf{x}})}$. $\mathcal{B}$ runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}_{Y^*}; w_2)$, picks $b \xleftarrow{\$} \{0, 1\}$, $\mathbf{s} = (s, s_1, s_2, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$, and computes the challenge ciphertext $CT^* = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = e(g_1, g_1)^{\alpha s} \cdot M_b \qquad \mathbf{C}_1 = g_1^{\mathbf{c}_{Y^*}(\mathbf{s}, \mathbf{h}, \mathbf{x})} * \mathbf{T}_c.$$

Phase 2: For all **Create**(h, X_j) queries requested by $\mathcal{A}$, $\mathcal{B}$ makes a query X_j to its challenger and receives $T_j = (g_2^{\mathbf{k}_{X_j,1}(\beta, \hat{\mathbf{h}}, \hat{\mathbf{x}}, \hat{\mathbf{r}})}, g_2^{\mathbf{k}_{X_j,2}(\hat{\mathbf{h}}, \hat{\mathbf{r}})})$. $\mathcal{B}$ need to guess if β is randomly chosen from Z_{p_2} or is 0. Then $\mathcal{B}$ creates the j-th key for $\mathcal{A}$ as follows

$$K_j = (g_1^{\mathbf{k}_{X_k,1}(\alpha, \mathbf{h}, \mathbf{x}, \mathbf{r}_j)} * g_3^{\mathbf{R}_{j,1}}, g_1^{\mathbf{k}_{X_k,2}(\mathbf{h}, \mathbf{r}_j)} * g_3^{\mathbf{R}_{j,2}}) * T_j,$$

where $Enc1(X_k, N) \rightarrow (\mathbf{k}_{X_k,1}, \mathbf{k}_{X_k,2}; m_1, m_2, m_3)$, $\mathbf{r}_j \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{j,1} \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_{j,2} \xleftarrow{\$} Z_N^{m_2}$.

After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_j, 0)$ to the set $\mathcal{T}$ and updates the handle counter to $H \leftarrow H + 1$.

Guess Phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

If $\beta = 0$, $\mathcal{B}$ perfectly simulates $\mathbf{G}_{21}$ for $\mathcal{A}$; Otherwise, $\mathcal{B}$ perfectly simulates $\mathbf{G}_{22}$. □

Lemma A.2.8. $|\boldsymbol{G}_{22}\boldsymbol{Adv}_{\mathcal{A}}(1^{\lambda}) - \boldsymbol{G}_{23}\boldsymbol{Adv}_{\mathcal{A}}(1^{\lambda})| \leq \boldsymbol{Adv}_{\mathcal{B}}^{SD2}(1^{\lambda})$.

Proof Outline. This proof is very similar to the proof of **Lemma** A.2.5, so we just give a proof outline here. In this proof, simulator $\mathcal{B}$ encodes its own challenge string into those keys generated in Phase 2, and simulates the proper environment for $\mathcal{A}$. Hence, the indistinguishability between these two games can be reduced to the hardness of Assumptin **SD2**. □

Lemma A.2.9. $|\boldsymbol{G_{23}Adv}_{\mathcal{A}}(1^\lambda) - \boldsymbol{G}_{final}\boldsymbol{Adv}_{\mathcal{A}}(1^\lambda)| \leq \boldsymbol{Adv}_{\mathcal{B}}^{SD3}(1^\lambda)$.

Proof. Algorithm $\mathcal{B}$ takes input a problem instance$(g_1, g_2, g_3, g_1^s W_2, g_1^\alpha Y_2, T)$ of Assumption **SD3**. It needs to decide whether $T = e(g_1, g_1)^{\alpha s}$ or $T \xleftarrow{\$} \mathbb{G}_T$. Algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ runs $Param(k) \rightarrow (n_1, n_2)$, picks $\mathbf{h} \xleftarrow{\$} Z_N^{n_1}, \mathbf{x} \xleftarrow{\$} Z_N^{n_2}$, and computes $PK = (g_1, g_3, e(g_1, g_1^\alpha Y_2), g_1^{\mathbf{h}}, g_1^{\mathbf{x}})$, sends PK to $\mathcal{A}$ and adds tuple 0 of $\mathcal{T}$ to be the first tuple $(0, \varepsilon, \varepsilon, 0)$, where the key filed is empty.

Phase 1: In this phase, when $\mathcal{A}$ makes the j-th **Create**(h, X_j) queries, $\mathcal{B}$ first scans $\mathcal{T}$ to find whether tuple h refers to a master key tuple. If not, then responds with $\perp$.
Otherwise, $\mathcal{B}$ generates a semi-functional type-3 key for X_j in the following way:
runs $Enc1(X_j, N) \rightarrow (\mathbf{k}_{X_j,1}, \mathbf{k}_{X_j,2}; m_1, m_2, m_3)$,picks $\hat{\alpha} \xleftarrow{\$} Z_N$, $\mathbf{r}_j \xleftarrow{\$} Z_N^{m_3}$, $\mathbf{R}_{j,1} \xleftarrow{\$} Z_N^{m_1}$, $\mathbf{R}_{j,2} \xleftarrow{\$} Z_N^{m_2}$ and computes:

$$K_{j,SF3} = ((g_1^\alpha Y_2)^{\mathbf{k}_{X_j,1}(1,\mathbf{0},\mathbf{0},\mathbf{0})} * g_1^{\mathbf{k}_{X_j,1}(0,\mathbf{h},\mathbf{x},\mathbf{r}_j)} * g_2^{\mathbf{k}_{X_j,1}(\hat{\alpha},\mathbf{0},\mathbf{0},\mathbf{0})} * g_3^{R_{j,1}}, \\ g_1^{\mathbf{k}_{X_j,2}(\mathbf{h},\mathbf{r}_j)} * g_2^{\mathbf{k}_{X_j,2}(\mathbf{0},\mathbf{0})} * g_3^{R_{j,2}}).$$

After that, $\mathcal{B}$ adds tuple $(H+1, X_j, K_{j,SF3}, 0)$ to set $\mathcal{T}$ and updates handle counter to $H \leftarrow H+1$.

Challenge phase: $\mathcal{A}$ outpus two challenge message $M_0, M_1 \in \mathbb{G}_T$ and a challenge attribute Y^*. $\mathcal{B}$ first checks whether $Y^* \in R$, if yes, return $\perp$.
Otherwise, $\mathcal{B}$ runs $Enc2(Y^*, N) \rightarrow (\mathbf{c}; w_2)$, picks $\mathbf{s}' = (1, s_1, s_2, \ldots, s_{\omega_2}) \xleftarrow{\$} Z_N^{\omega_2+1}$, $b \xleftarrow{\$} \{0,1\}$, computes challenge ciphertext $CT = (C_0, \mathbf{C}_1)$ as follows:

$$C_0 = T \cdot M_b \qquad \mathbf{C}_1 = (g_1^s W_2)^{\mathbf{c}(\mathbf{s}',\mathbf{h},\mathbf{x})}$$

Phase 2: $\mathcal{B}$ generates a semi-functional type-3 key for X_j in the same way as **Phase 1**.

Guess Phase: $\mathcal{A}$ outputs a guess b'. If $b' = b$, then $\mathcal{A}$ successes.

□

Combining all the lemmas mentioned above, we can prove the indistinguishability between those games defined in our proof, hence we complete this proof. □

Bibliography

[AFGH06] Giuseppe Ateniese, Kevin Fu, Matthew Green, and Susan Hohenberger. Improved proxy re-encryption schemes with applications to secure distributed storage. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):1–30, 2006.

[AL10] Nuttapong Attrapadung and Benoît Libert. Functional encryption for inner product: Achieving constant-size ciphertexts with adaptive security or support for negation. In *Public Key Cryptography–PKC 2010*, pages 384–402. Springer, 2010.

[ALDP11] Nuttapong Attrapadung, Benoît Libert, and Elie De Panafieu. Expressive key-policy attribute-based encryption with constant-size ciphertexts. In *Public Key Cryptography–PKC 2011*, pages 90–108. Springer, 2011.

[AN11] Tolga Acar and Lan Nguyen. Revocation for delegatable anonymous credentials. In *International Workshop on Public Key Cryptography*, pages 423–440. Springer, 2011.

[ATSM09] Man Ho Au, Patrick P Tsang, Willy Susilo, and Yi Mu. Dynamic universal accumulators for ddh groups and their application to attribute-based anonymous credential systems. In *Cryptographers? Track at the RSA Conference*, pages 295–308. Springer, 2009.

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In *Advances in Cryptology–EUROCRYPT 2014*, pages 557–577. Springer, 2014.

[Att15] Nuttapong Attrapadung. Dual system encryption framework in prime-order groups. *IACR Cryptology ePrint Archive*, 2015:390, 2015.

[BB04a] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In

Advances in Cryptology-EUROCRYPT 2004, pages 223–238. Springer, 2004.

[BB04b] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In *Advances in Cryptology–Crypto 2004*, pages 443–459. Springer, 2004.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *Advances in Cryptology–EUROCRYPT 2005*, pages 440–456. Springer, 2005.

[BBS98] Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *Advances in Cryptology-EUROCRYPT'98*, pages 127–144. Springer, 1998.

[BCC+16] Jonathan Bootle, Andrea Cerulli, Pyrros Chaidos, Essam Ghadafi, and Jens Groth. Foundations of fully dynamic group signatures. In *International Conference on Applied Cryptography and Network Security*, pages 117–136. Springer, 2016.

[BDM93] Josh Benaloh and Michael De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 274–285. Springer, 1993.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Advances in Cryptology—CRYPTO 2001*, pages 213–229. Springer, 2001.

[BGJK12] Elette Boyle, Shafi Goldwasser, Abhishek Jain, and Yael Tauman Kalai. Multiparty computation secure against continual memory leakage. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 1235–1254. ACM, 2012.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-dnf formulas on ciphertexts. In *Theory of cryptography*, pages 325–341. Springer, 2005.

[BGT15] Michael Backes, Martin Gagné, and Sri Aravinda Krishnan Thyagarajan. Fully secure inner-product proxy re-encryption

with constant size ciphertext. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pages 31–40. ACM, 2015.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 501–510. IEEE, 2010.

[BLL00] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In *Proceedings of the 7th ACM conference on Computer and communications security*, pages 9–17. ACM, 2000.

[BLL02] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Eliminating counterevidence with applications to accountable certificate management 1. *Journal of Computer Security*, 10(3):273–296, 2002.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 480–494. Springer, 1997.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.

[BW06] Xavier Boyen and Brent Waters. Anonymous hierarchical identity-based encryption (without random oracles). In *Advances in Cryptology-CRYPTO 2006*, pages 290–307. Springer, 2006.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *Journal of the ACM (JACM)*, 51(4):557–594, 2004.

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system abe in prime-order groups via predicate encodings.

In *Advances in Cryptology-EUROCRYPT 2015*, pages 595–624. Springer, 2015.

[CH07] Ran Canetti and Susan Hohenberger. Chosen-ciphertext secure proxy re-encryption. In *Proceedings of the 14th ACM conference on Computer and communications security*, pages 185–194. ACM, 2007.

[CHKO08] Philippe Camacho, Alejandro Hevia, Marcos Kiwi, and Roberto Opazo. Strong accumulators from collision-resistant hashing. In *International Conference on Information Security*, pages 471–486. Springer, 2008.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 523–552. Springer, 2010.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In *International Workshop on Public Key Cryptography*, pages 481–500. Springer, 2009.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Annual International Cryptology Conference*, pages 61–76. Springer, 2002.

[coi] Cryptocurrencies market capacity. URL:`https://coinmarketcap.com/all/views/all/`, Online accessed 16 April 2018.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[CT07] Cheng-Kang Chu and Wen-Guey Tzeng. Identity-based proxy re-encryption without random oracles. In *International Conference on Information Security*, pages 189–202. Springer, 2007.

[CW13] Jie Chen and Hoeteck Wee. Fully,(almost) tightly secure ibe and dual system groups. In *Advances in Cryptology–CRYPTO 2013*, pages 435–460. Springer, 2013.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[DHLAW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *Foundations of Computer Science (FOCS), 2010 51st Annual IEEE Symposium on*, pages 511–520. IEEE, 2010.

[DHS15] David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *CT-RSA*, pages 127–144, 2015.

[DKNS04] Yevgeniy Dodis, Aggelos Kiayias, Antonio Nicolosi, and Victor Shoup. Anonymous identification in ad hoc groups. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 609–626. Springer, 2004.

[DLWW11] Yevgeniy Dodis, Allison Lewko, Brent Waters, and Daniel Wichs. Storing secrets on continually leaky devices. In *Foundations of Computer Science (FOCS), 2011 IEEE 52nd Annual Symposium on*, pages 688–697. IEEE, 2011.

[DT08] Ivan Damgård and Nikos Triandopoulos. Supporting non-membership proofs with bilinear-map accumulators. *IACR Cryptology ePrint Archive*, 2008:538, 2008.

[DWQ+15] Hua Deng, Qianhong Wu, Bo Qin, Willy Susilo, Joseph K. Liu, and Wenchang Shi. Asymmetric cross-cryptosystem re-encryption applicable to efficient and secure mobile access to outsourced data. In Feng Bao, Steven Miller, Jianying Zhou, and Gail-Joon Ahn, editors, *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '15, Singapore, April 14-17, 2015*, pages 393–404. ACM, 2015.

[ElG84] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Advances in cryptology*, pages 10–18. Springer, 1984.

[EMO10] Keita Emura, Atsuko Miyaji, and Kazumasa Omote. An identity-based proxy re-encryption scheme with source hiding property, and its application to a mailing-list system. In *European Public Key Infrastructure Workshop*, pages 77–92. Springer, 2010.

[Fey82] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6):467–488, 1982.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the Theory and Application of Cryptographic Techniques*, pages 186–194. Springer, 1986.

[GA07] Matthew Green and Giuseppe Ateniese. Identity-based proxy re-encryption. In *Applied Cryptography and Network Security*, pages 288–306. Springer, 2007.

[GH09] Craig Gentry and Shai Halevi. Hierarchical identity based encryption with polynomially many levels. In *TCC*, volume 5444, pages 437–456. Springer, 2009.

[GHW+11] Matthew Green, Susan Hohenberger, Brent Waters, et al. Outsourcing the decryption of abe ciphertexts. In *USENIX Security Symposium*, volume 2011, 2011.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[Goe15] Michel Goemans. Lecture notes in chernoff bounds, and some applications, February 2015. URL:`http://math.mit.edu/~goemans/18310S15/chernoff-notes.pdf`.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm, 2006.

[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*, pages 197–206. ACM, 2008.

[GTH02] Michael T Goodrich, Roberto Tamassia, and Jasminka Hasić. An efficient dynamic and distributed cryptographic accumulator. In *International Conference on Information Security*, pages 372–388. Springer, 2002.

[Ham11] Mike Hamburg. *SPATIAL ENCRYPTION*. PhD thesis, STANFORD UNIVERSITY, 2011.

[HSH+09] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[ID03] Anca-Andreea Ivan and Yevgeniy Dodis. Proxy cryptography revisited. In *NDSS*, 2003.

[JMB11] Sonia Jahid, Prateek Mittal, and Nikita Borisov. Easier: Encryption-based access control in social networks with efficient revocation. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 411–415. ACM, 2011.

[KFTS17] Amrit Kumar, Clément Fischer, Shruti Tople, and Prateek Saxena. A traceability analysis of monero's blockchain. In *European Symposium on Research in Computer Security*, pages 153–173. Springer, 2017.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99*, pages 388–397. Springer, 1999.

[Koc96] Paul C Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology—CRYPTO'96*, pages 104–113. Springer, 1996.

[KTX08] Akinori Kawachi, Keisuke Tanaka, and Keita Xagawa. Concurrently secure identification schemes based on the worst-case hardness of lattice problems. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 372–389. Springer, 2008.

[LAL+14] Kaitai Liang, Man Ho Au, Joseph K Liu, Willy Susilo, Duncan S Wong, Guomin Yang, Tran Viet Xuan Phuong, and Qi Xie. A dfa-based functional proxy re-encryption scheme for secure public cloud data sharing. *IEEE Transactions on Information Forensics and Security*, 9(10):1667–1680, 2014.

[LAL+15] Kaitai Liang, Man Ho Au, Joseph K Liu, Willy Susilo, Duncan S Wong, Guomin Yang, Yong Yu, and Anjia Yang. A secure and efficient ciphertext-policy attribute-based proxy re-encryption for cloud data sharing. *Future Generation Computer Systems*, 52:95–108, 2015.

[LCLS09] Xiaohui Liang, Zhenfu Cao, Huang Lin, and Jun Shao. Attribute based proxy re-encryption with delegating capabilities. In *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, pages 276–286. ACM, 2009.

[Lew12] Allison Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In *Advances in Cryptology–EUROCRYPT 2012*, pages 318–335. Springer, 2012.

[LH10] Zi Lin and Nicholas Hopper. Jack: Scalable accumulator-based nymble system. In *Proceedings of the 9th annual ACM workshop on Privacy in the electronic society*, pages 53–62. ACM, 2010.

[LLM+16] Benoît Libert, San Ling, Fabrice Mouhartem, Khoa Nguyen, and Huaxiong Wang. Signature schemes with efficient protocols and dynamic group signatures from lattice assumptions. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II 22*, pages 373–403. Springer, 2016.

[LLNW16] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based accumulators: Logarithmic-size ring signatures and group signatures without trapdoors. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 1–31. Springer, 2016.

[LLNW17] Benoît Libert, San Ling, Khoa Nguyen, and Huaxiong Wang. Zero-knowledge arguments for lattice-based prfs and applications to e-cash. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 304–335. Springer, 2017.

[LLX07] Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *Applied Cryptography and Network Security*, pages 253–269. Springer, 2007.

[LNSW13] San Ling, Khoa Nguyen, Damien Stehlé, and Huaxiong Wang. Improved zero-knowledge proofs of knowledge for the isis problem, and applications. In *Public-Key Cryptography–PKC 2013*, pages 107–124. Springer, 2013.

[LNWX17] San Ling, Khoa Nguyen, Huaxiong Wang, and Yanhong Xu. Lattice-based group signatures: Achieving full dynamicity with ease. In *International Conference on Applied Cryptography and Network Security*, pages 293–312. Springer, 2017.

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

[LRW11] Allison Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *Theory of Cryptography*, pages 70–88. Springer, 2011.

[LSW06] Joseph K Liu, Willy Susilo, and Duncan S Wong. Ring signature with designated linkability. In *International Workshop on Security*, pages 104–119. Springer, 2006.

[LV08] Benoît Libert and Damien Vergnaud. Unidirectional chosen-ciphertext secure proxy re-encryption. In *Public Key Cryptography–PKC 2008*, pages 360–379. Springer, 2008.

[LW10] Allison Lewko and Brent Waters. New techniques for dual system encryption and fully secure hibe with short ciphertexts. In *Theory of Cryptography*, pages 455–479. Springer, 2010.

[LW11] Allison Lewko and Brent Waters. Unbounded hibe and attribute-based encryption. In *Advances in Cryptology–EUROCRYPT 2011*, pages 547–567. Springer, 2011.

[LWW14] Qin Liu, Guojun Wang, and Jie Wu. Time-based proxy re-encryption scheme for secure data sharing in a cloud environment. *Information Sciences*, 258:355–370, 2014.

[Mat07] Toshihiko Matsuo. Proxy re-encryption systems for identity-based encryption. In *Pairing-Based Cryptography–Pairing 2007*, pages 247–267. Springer, 2007.

[MD09] Takeo Mizuno and Hiroshi Doi. Hybrid proxy re-encryption scheme for attribute-based encryption. In *Information security and cryptology*, pages 288–302. Springer, 2009.

[MMLN17] Andrew Miller, Malte Möser, Kevin Lee, and Arvind Narayanan. An empirical analysis of linkability in the monero blockchain. *arXiv preprint arXiv:1704.04299*, 2017.

[MNT10] Toshihide Matsuda, Ryo Nishimaki, and Keisuke Tanaka. Cca proxy re-encryption without bilinear maps in the standard model. In *International Workshop on Public Key Cryptography*, pages 261–278. Springer, 2010.

[mon] Monero source code. URL:`https://github.com/monero-project/monero`,Online accessed 30 March 2018.

[MPJ+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M Voelker, and Stefan Savage. A fistful of bitcoins: characterizing payments among men with no names. In *Proceedings of the 2013 conference on Internet measurement conference*, pages 127–140. ACM, 2013.

[MSH+18] Malte Möser, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, et al. An empirical analysis of traceability in the monero blockchain. *Proceedings on Privacy Enhancing Technologies*, 3:143–163, 2018.

[MTVY11] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In *Theory of Cryptography*, pages 89–106. Springer, 2011.

[Nak08] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

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

[OT12] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *Advances in Cryptology–EUROCRYPT 2012*, pages 591–608. Springer, 2012.

[P+16] Chris Peikert et al. A decade of lattice cryptography. *Foundations and Trends® in Theoretical Computer Science*, 10(4):283–424, 2016.

[RH11] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on*, pages 1318–1326. IEEE, 2011.

[RS13] Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.

[RSA78] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[RST01] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to leak a secret. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 552–565. Springer, 2001.

[RW13] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 463–474. ACM, 2013.

[SC09] Jun Shao and Zhenfu Cao. Cca-secure proxy re-encryption without pairings. In *International Workshop on Public Key Cryptography*, pages 357–376. Springer, 2009.

[SC12] Jun Shao and Zhenfu Cao. Multi-use unidirectional identity-based proxy re-encryption from hierarchical identity-based encryption. *Information Sciences*, 206:83–95, 2012.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Advances in cryptology*, pages 47–53. Springer, 1984.

[Sho94] Peter W Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. Ieee, 1994.

[SLP+12] Ron Steinfeld, San Ling, Josef Pieprzyk, Christophe Tartary, and Huaxiong Wang. Ntrucca: How to strengthen ntruencrypt to chosen-ciphertext security in the standard model. In *Public Key Cryptography–PKC 2012*, pages 353–371. Springer, 2012.

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

[THJ08] Qiang Tang, Pieter Hartel, and Willem Jonker. Inter-domain identity-based proxy re-encryption. In *International Conference on Information Security and Cryptology*, pages 332–347. Springer, 2008.

[TX03] Gene Tsudik and Shouhuai Xu. Accumulating composites and improved group signing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 269–286. Springer, 2003.

[VS13] Nicolas Van Saberhagen. Cryptonote v 2. 0, 2013.

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *Advances in Cryptology–EUROCRYPT 2005*, pages 114–127. Springer, 2005.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *Advances in Cryptology-CRYPTO 2009*, pages 619–636. Springer, 2009.

[Wat12] Brent Waters. Functional encryption for regular languages. In *Advances in Cryptology–CRYPTO 2012*, pages 218–235. Springer, 2012.

[WCW10] Hongbing Wang, Zhenfu Cao, and Licheng Wang. Multi-use and unidirectional identity-based proxy re-encryption schemes. *Information Sciences*, 180(20):4042–4059, 2010.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In *Theory of Cryptography*, pages 616–637. Springer, 2014.

[WWMO10] Lihua Wang, Licheng Wang, Masahiro Mambo, and Eiji Okamoto. New identity-based proxy re-encryption schemes to prevent collusion attacks. In *International Conference on Pairing-Based Cryptography*, pages 327–346. Springer, 2010.

[YCZY12] Tsz Hon Yuen, Sherman SM Chow, Ye Zhang, and Siu Ming Yiu. Identity-based encryption resilient to continual auxiliary leakage. In *Advances in Cryptology–EUROCRYPT 2012*, pages 117–134. Springer, 2012.

[YWRL10] Shucheng Yu, Cong Wang, Kui Ren, and Wenjing Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 261–270. ACM, 2010.

[YXZ+15] Rupeng Yang, Qiuliang Xu, Yongbin Zhou, Rui Zhang, Chengyu Hu, and Zuoxia Yu. Updatable hash proof system and its applications. In *Computer Security–ESORICS 2015*, pages 266–285. Springer, 2015.