The Dissertation Committee for Ioannis Rouselakis
certifies that this is the approved version of the following dissertation:

Attribute-Based Encryption: Robust and Efficient Constructions

Committee:

Attribute-Based Encryption: Robust and Efficient Constructions

by

Ioannis Rouselakis, B.E.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

August 2013

Acknowledgments

I wish to thank my advisor, Brent Waters, for the invaluable advice and continuous trust he bestowed upon me during my time at UT. I would also wish to thank Allison Lewko for our collaboration in various works, one of which is included in this thesis, and for her valuable feedback. In addition, I would like to thank my collaborators for the published works during my PhD: Omkant Pandey, Yevgeniy Dodis, and Sherman Chow.

I would like to show my appreciation to my numerous friends who showed me great hospitality and helped me enjoy life in Austin in its fullest: Christos Argyropoulos, Dimitris Prountzos, Theofilos Giagmouris, and especially, Thekla Boutsika, who was the best and most giving neighbor and friend. I wish to thank my parents, Litsa and Kostas, for everything they have done. If it wasn't for them, I would not be here today. Last but not least, I would like to express my gratitude to my girlfriend, Helen Costa, for supporting me unconditionally and sharing with me a happy life.

Attribute-Based Encryption: Robust and Efficient Constructions

Publication No. ____________________

Ioannis Rouselakis, Ph.D.
The University of Texas at Austin, 2013

Supervisor: Brent Waters

Attribute-based encryption is a promising cryptographic primitive that allows users to encrypt data according to specific policies on the credentials of the recipients. For example, a user might want to store data in a public server such that only subscribers with credentials of specific forms are allowed to access them. Encrypting the data once for each party is not only impractical but also raises important privacy issues. Therefore, it would be beneficial to be able to encrypt only once for all desired parties. This is achievable by attribute-based encryption schemes, which come into several types and are applicable to a wide range of settings.

Several attribute-based encryption schemes have been proposed and studied with a wide range of characteristics. For example, initial constructions proved to be significantly more challenging than constructing traditional public-key encryption systems and they imposed restrictions on the expressiveness of the Boolean formulas used during encryption. For several proposed schemes the total number of attributes was fixed during setup, while others allowed any string to be used as attribute ("large universe" constructions), but with considerable weaker security guarantees. Furthermore, these first constructions, although polynomial time, were impractical for wide deployment.

This thesis is motivated by two main goals for ABE schemes: robustness and efficiency. For robustness, we propose a novel construction that achieves strong security guarantees and at the same time augments the capabilities of previous schemes. More specifically, we adapt existing techniques to achieve leakage-resilient ABE schemes with augmented robustness features making no compromises on security. For the second direction, our goal is to create practical schemes with as many features as possible, such as "large universe" and multi-authority settings. We showcase these claims with working implementations, benchmarks, and comparisons to previous constructions. Finally, these constructions lead us to new directions that we propose and intend to investigate further.

Table of Contents

Acknowledgments **iv**

Abstract **v**

List of Tables **xi**

List of Figures **xii**

Abbreviations **xiii**

Notation **xiv**

Chapter 1. Introduction **1**

1.1 Brief History of Attribute-Based Encryption and Related Functionalities 3

1.2 Brief Overview of Side-Channel Attacks and Leakage Resilience 7

1.3 Summary of Our Results 9

1.4 Related Work 19

1.5 Organization 22

Chapter 2. Background **24**

2.1 Access Structures & Linear Secret Sharing Schemes 24

2.1.1 Multi-Authority Attributes 27

2.1.2 Converting Monotone Boolean Formulas to Share Generating Matrices 27

2.2 Leakage Models 30
2.3 Overview of Dual System Encryption 32

Chapter 3. Bilinear Groups 40
3.1 Abstract Properties of Bilinear Groups 41
3.1.1 Prime Order Bilinear Groups 41
3.1.2 Composite Order Bilinear Groups 42
3.2 Computational Assumptions 44
3.2.1 Three q-Type Assumptions on Prime Order Groups . . . 45
The q-DPBDH1 Assumption 46
The q-DPBDH2 Assumption 47
The q-DPBDH3 Assumption 48
3.2.2 Three Subgroup Decision Assumptions on Composite Order Groups 48
The Comp1 Assumption 50
The Comp2 Assumption 50
The Comp3 Assumption 50
3.3 Implementation of Bilinear Groups using Elliptic Curves . . . 51
3.3.1 Elliptic Curves 51
3.3.2 Tate-Lichtenbaum Pairing over Finite Fields 52
3.3.3 Supersingular Curves and Distortion Maps 54
3.3.4 Prime vs Composite Order Group Operations 54

Chapter 4. Identity and Attribute-Based Encryption Systems 57
4.1 IBE Definition 57
4.1.1 Dual System IBE 59
4.2 ABE Definitions 59
4.2.1 Ciphertext-Policy ABE 60
4.2.2 Key-Policy ABE 61
4.2.3 Multi-Authority ABE 63
4.2.4 Small Universe, Large Universe, and Unbounded Constructions 65
4.2.5 Dual System CP-ABE 67

4.3 Security Notions 68
4.3.1 CP-ABE Security Notions 68
4.3.2 KP-ABE Security Notions 70
4.3.3 Multi-Authority Static Security 71
4.3.4 IBE Leakage-Resilient Adaptive Security 73
4.3.5 CP-ABE Leakage-Resilient Adaptive Security 79
4.3.6 Three Properties for Leakage-Resilient Adaptive Security 82
4.4 Continual Leakage for IBE and ABE 89
4.4.1 Leakage from Updates 98

Chapter 5. Part I: Leakage-Resilient IBE and CP-ABE Systems 99
5.1 Leakage-Resilient IBE 100
5.1.1 Construction 100
5.1.2 Semi-Functionality 103
5.1.3 Continual Leakage 105
5.1.4 Security 105
5.2 Leakage-Resilient CP-ABE 121
5.2.1 Construction 121
5.2.2 Semi-Functionality 123
5.2.3 Security Proof 125
5.3 Leakage Fraction 141

Chapter 6. Part II: Three Practical Constructions 143
6.1 A Large-Universe KP-ABE System 143
6.1.1 Construction 144
6.1.2 Security Proof 146
6.2 A Large-Universe CP-ABE System 151
6.2.1 Construction 151
6.2.2 Security Proof 153
6.3 A Large-Universe Multi-Authority CP-ABE System 161
6.3.1 Construction 162
6.3.2 Security Proof 165

Chapter 7. Implementations and Benchmarks **176**
7.1 Charm Framework 176
7.2 Implementation Details 177
7.3 Benchmarks KP-ABE and CP-ABE 178
7.4 Group Operations (Asymptotic) for KP-ABE and CP-ABE . . 179
7.5 Benchmarks MA-CP-ABE 179

Chapter 8. Other Work and Future Directions **185**
8.1 Other Work 185
8.2 Future Directions 188

Appendices **189**

Appendix A. Generic Security of the Assumptions **190**
A.1 Two General Theorems 191
A.2 Proofs of Security in the Generic Group Model 195

Appendix B. Proofs of Various Lemmas Used **200**
B.1 A Useful Lemma for Leakage Analysis 200
B.2 Proof of claim 6.4 201

Appendix C. Source Code **206**
C.1 KP-ABE scheme of Sec. 6.1 206
C.2 CP-ABE scheme of Sec. 6.2 211
C.3 MA-CP-ABE scheme of Sec. 6.3 217
C.4 Implementation of Dual Vector Spaces 224

Index **229**

Bibliography **230**

Vita **244**

List of Tables

3.1 Two Types of Supersingular Elliptic Curves 55
3.2 Timing Results on Prime and Composite Order Groups 56

5.1 Leakage fractions of our IBE and CP-ABE schemes 142

7.1 Approximate security level of Charm ECC groups 177
7.2 Timing Results of our KP-ABE and CP-ABE Schemes 180
7.3 Asymptotic Complexity of our KP-ABE and CP-ABE Schemes 181
7.4 Timing Results of our MA-CP-ABE scheme and two single authority schemes 184

A.1 Exponent Table for the q-DPBDH1 Assumption 196
A.2 Exponent Table for the q-DPBDH2 Assumption 198

List of Figures

2.1 Tree representation for the formula F 29
2.2 Access Policy for the formula F 29

6.1 Unreconstructible Key Components by the Simulator 157
6.2 Transformation of the policy matrix A 167

B.1 Blockwise inversion formula 204

Abbreviations

The following abbreviations are used throughout the thesis.

PPT	Probabilistic Polynomial Time
IBE	Identity-Based Encryption
HIBE	Hierarchical Identity-Based Encryption
ABE	Attribute-Based Encryption
CP-ABE	Ciphertext-Policy Attribute-Based Encryption
KP-ABE	Key-Policy Attribute-Based Encryption
MA-CP-ABE	Multi - Authority Ciphertext - Policy Attribute - Based Encryption
LSSS	Linear Secret-Sharing Scheme

Notation

The following standard notation is used throughout the thesis. Other notation will be introduced close to the section being used.

$\mathbb{N}$, $\mathbb{Z}$	Natural numbers and integers, respectively.
$\mathbb{F}$, $\mathbb{F}_q$	General field and field of order q, respectively.
$\mathbb{G}$, $\mathbb{H}$, $\mathbb{G}_T$, ...	Various (multiplicative) groups.
$\mathbb{1}, \mathbb{1}_{\mathbb{G}}, \mathbb{1}_{\mathbb{H}}, \mathbb{1}_{\mathbb{G}_T}, \ldots$	The identity element of various groups.
$e(g, h)$	The bilinear mapping $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, where $g, h \in \mathbb{G}$.
$\mathbb{A}, \mathbb{A}^*, \mathbb{A}_i$	Access structures.
$\mathcal{U}, \mathcal{U}_\Theta$	Universe of attributes and universe of authorities, respectively.
$\lambda \in \mathbb{N}$	Security parameter of our schemes.
$\|p\|$	The number of bits of $p \in \mathbb{N}$, i.e. $\|p\| = \lfloor \log_2 p \rfloor + 1$ for $p \geq 1$.
$\vert\mathcal{S}\vert$	The number of elements of the set $\mathcal{S}$.
$\mathsf{poly}(\lambda)$	A polynomially bounded function in λ. Namely, there exists a constant $c > 0$ s.t. $\mathsf{poly}(\lambda) \leq \lambda^c$ for sufficiently large λ.

$\mathsf{negl}(\lambda)$	A negligible function in λ. Namely, for every constant $c > 0$ it is true that $\mathsf{negl}(\lambda) \leq \lambda^{-c}$ for sufficiently large λ.
$[n]$	The set $\{1, 2, \ldots, n\}$, where $n \in \mathbb{N}$.
$[n_1, n_2, \ldots n_k]$	The set $[n_1] \times [n_2] \times \ldots \times [n_k]$, where $n_1, n_2, \ldots, n_k \in \mathbb{N}$.
$\mathbb{Z}_p$	Integers modulo $p \in \mathbb{N}$ i.e. the set $\{0,\ 1,\ 2,\ \ldots,\ p-1\}$.
$\mathbb{Z}_p^{m \times n}$	Matrices of dimension $m \times n$ with elements in $\mathbb{Z}_p$. Special subsets are the set of row vectors of length n: $\mathbb{Z}_p^{1 \times n}$, and the set of column vectors of length n: $\mathbb{Z}_p^{n \times 1}$.
$\langle \vec{v}_1, \vec{v}_2 \rangle$	The inner product of the vectors $\vec{v}_1$ and $\vec{v}_2$ of same length. Each of them can be either a row or a column vector.
$\mathcal{A}$, $\mathcal{B}$, $\mathcal{C}$	PPT adversaries modeled as probabilistic interactive Turing machines. Typically, $\mathcal{A}$ denotes the attacker, $\mathcal{B}$ the simulator, and $\mathcal{C}$ the challenger.
Alg, Setup, $\ldots$	Polynomial - time algorithms (possibly probabilistic).
$s \coloneqq \mathsf{Alg}(\mathit{args})$	Setting the variable s to the result of the *deterministic* polynomial-time algorithm Alg run on inputs *args*.
$s \overset{R}{\leftarrow} \mathsf{Alg}(\mathit{args})$	Setting the variable s to the result of the *probabilistic* polynomial-time algorithm Alg run on inputs *args* and uniformly sampled random coins.
$s \overset{R}{\leftarrow} \mathcal{S}$	Setting the variable s to a uniformly random element of the set $\mathcal{S}$.
$s_1, s_2, \ldots, s_k \overset{R}{\leftarrow} \mathcal{S}$	Same as $s_1 \overset{R}{\leftarrow} \mathcal{S}$, $s_2 \overset{R}{\leftarrow} \mathcal{S}$, $\ldots$, $s_k \overset{R}{\leftarrow} \mathcal{S}$.
$\{X_i\}_{i \in [n]}$	The collection of terms $X_1, X_2, \ldots, X_n$.

$\left\{X_i \overset{R}{\leftarrow} \mathsf{Alg}(args)\right\}_{i\in[n]}$	The collection of terms $X_1, X_2, \ldots, X_n$, where each term is set to the result of the PPT algorithm Alg run on inputs $args$ and uniformly sampled random coins, independently sampled on each call of the algorithm.
$\left\{X_i \overset{R}{\leftarrow} \mathcal{S}\right\}_{i\in[n]}$	The collection of terms $X_1, X_2, \ldots, X_n$, where each term is set to a uniformly random element of the set $\mathcal{S}$, independently from the rest.
$\perp$	Dummy value denoting a failed operation or "no result".
$u^{\vec{a}}$	With $u \in \mathbb{G}$ and $\vec{a} \in \mathbb{Z}^n$, we define $u^{\vec{a}} = (u^{a_1}, u^{a_2}, \ldots, u^{a_n})$.
$\vec{u}^a$	With $\vec{u} = (u_1, u_2, \ldots, u_n) \in \mathbb{G}^n$ and $a \in \mathbb{Z}$, we define $\vec{u}^a := (u_1^a, u_2^a, \ldots, u_n^a)$.
$e_n(\vec{u}_1, \vec{u}_2)$	With $\vec{u}_1 = (u_{11}, u_{12}, \ldots, u_{1n}) \in \mathbb{G}^n$ and $\vec{u}_2 = (u_{21}, u_{22}, \ldots, u_{2n}) \in \mathbb{G}^n$, we define the multi - dimensional bilinear pairing as $e_n(\vec{u}_1, \vec{u}_2) = \prod_{i=1}^n e(u_{1i}, u_{2i}) \in \mathbb{G}_T$, where $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is the bilinear mapping of $\mathbb{G}$ and the product is the group operation of $\mathbb{G}_T$.
$\vec{u}_1 * \vec{u}_2$	With $\vec{u}_1 = (u_{11}, u_{12}, \ldots, u_{1n}) \in \mathbb{G}^n$ and $\vec{u}_2 = (u_{21}, u_{22}, \ldots, u_{2n}) \in \mathbb{G}^n$, we define the component-wise multiplication (group operation) of the two vectors. That is, $\vec{u}_3 = \vec{u}_1 * \vec{u}_2 \in \mathbb{G}^n$ if and only if for all $i \in [n]$: $u_{3i} = u_{1i} \cdot u_{2i}$ where $\cdot$ denotes the group operation.

CHAPTER 1

Introduction

Traditional public key cryptography has provided users with ways to achieve secure communication over a public channel. After the seminal work of Goldwasser and Micalli, the security notions needed for public key systems were formalized and opened the way for provably fully secure systems even against extremely powerful adversaries. Even if these adversaries could deploy chosen plaintext (or chosen ciphertext) attacks on a polynomial number of messages (or ciphertexts), the security proofs showed that no information can be extracted from the challenge plaintext.

However with the wide expansion of the world wide web and the need for secure communication in a wide network of users, each with complex credentials, various researchers suggested the use of extended functionalities, which would exceed the features of public key encryption in various ways. For example, if the sender of the message wants to post his ciphertext on a public bulletin board such that only the parties with the required credentials being able to decrypt, he would have to retrieve the public key of each party with the required credentials and create a separate ciphertext for each.

This approach suffers from three main drawbacks making it inapplicable to real world large scale networks. The first one is the obvious inefficient procedure of having to encrypt separately for each specific user. The number of users with the required credential might be prohibitively large. The second major drawback is the privacy issues raised when any user of the system is allowed to scan or acquire the credentials of any other user. It might be the case that these credentials are supposed to be confidential and not publicly available. Finally, in the common case that a new user is added with the required credentials, the encryptor has to re-encrypt his message under the public key of the new user. This places a large burden on the encrypting parties to store their plaintexts and continuously update their posted ciphertexts.

Attribute-based encryption (ABE) is a cryptographic primitive that addresses to the fullest the above issues and finds applications to a wide range of settings, from regular users over the world wide web to large multi structural corporations. It was a notion introduced by Sahai and Waters [99] that relates the cryptographic components with attribute sets, corresponding to available credentials for users, and access policies, corresponding to the possibly complex restrictions that the credentials have to satisfy. In the most common form of ABE, the ciphertext-policy ABE (CP-ABE), the secret key of each user is associated with a set of attributes/credentials and each ciphertext is associated with a policy on the universe of credentials. If the policy is satisfied by the set of credentials of the key, then the owner of this key can decrypt successfully the ciphertext. The second form of ABE is the key-policy ABE (KP-ABE), where each key is associated with a policy on the attribute sets and each ciphertext with a set of attributes. As in the former setting, if the policy of the key is satisfied by the set of credentials of the ciphertext, then decryption is possible.

The main objectives of this work are to expand the state-of-the-art ABE systems towards more efficient and more robust constructions. The first goal aims at "bringing ABE to practice" through constructions that work with the fastest possible cryptographic components, achieve advanced features, such as large-universe and multi-authority settings, and admit the fewest possible compromises in provable security. The second part of this thesis provides a fully secure ABE construction that achieves resilience against extensive side-channel attacks and showcases the use of the dual system encryption framework towards a novel direction. Namely, it shows how this framework is not only useful for providing full security, as presented in past works, but also for contributing to the orthogonal feature of leakage resilience.

1.1 Brief History of Attribute-Based Encryption and Related Functionalities

The first idea that was directly related to attribute-based encryption, and the more general notion of functional encryption, was the Identity-Based Encryption functionality (IBE), introduced by Shamir [100]. To encrypt using an identity-based encryption scheme a user only needs the public parameters of the scheme and the identity of the recipient, which can be for example his e-mail address or any arbitrary string. In some sense the “public key” of a user is his publicly known identity and no information has to be published by the recipient such as a regular public key, thus alleviating the need for a public key distribution system. The secret keys for each identity, which are needed for decrypting the ciphertexts, are generated by a trusted master authority and given to the corresponding users after authentication by the authority. The next generalization of IBE was the Hierarchical Identity-Based Encryption functionality (HIBE) in which a hierarchy is imposed on the set of users/identities of the system. In these systems, each user can serve as the trusted authority for a subset of other users by being able to generate secret keys for them using its own secret key. These users can in turn generate secret keys for a smaller subset of users using their secret keys, and so on. The master trusted authority lies at the top of the hierarchy tree and is able to generate secret keys for all identities.

Attribute-Based Encryption (ABE), introduced by Sahai and Waters [99], was another generalization of IBE. In its most natural form, called Ciphertext - Policy ABE (CP-ABE), each user possesses a set of attributes/credentials and a secret key corresponding to these attributes. The secret key is provided by a trusted master authority that authenticates and verifies the credentials of the users and provides the encryption functionality via a set of public parameters. The encrypting party can define any Boolean formula on the set of attributes and produce a ciphertext that is decryptable by only the users whose credentials satisfy the Boolean formula. This formula is referred to as the policy of the ciphertext. As a result, the encryptor creates only one ciphertext for all the target users and is oblivious of their exact identities. Also, the ciphertext creation is independent of the creation and distribution of secret keys. New users can acquire additional credentials and they might be able to legally decrypt the previously created ciphertext. IBE is a special form of ABE, since it can be seen as an attribute-based system where each user can acquire only one attribute, his identity, and each ciphertext is build using a policy that contains only one attribute, the required identity. Finally, we mention here that all these functionalities are special cases of Functional Encryption systems [24], where each secret key and ciphertext is associated with some information x and y, respectively, and the decryption of the ciphertext with the secret key outputs the value of a function $f(x, y)$. For example, in IBE each secret key is associated with an identity $\mathcal{ID} = x$ and the ciphertext with an identity / plaintext pair $(\mathcal{ID}', M) = y$. The function $f(x, y)$ outputs M if and only if $\mathcal{ID} = \mathcal{ID}'$.

Constructions The first construction of identity-based encryption was given by Boneh and Franklin [20] and presented one of the first uses of bilinear pairing groups in cryptography. The scheme was proved secure in the random oracle model under a new computational assumption on bilinear groups. The first construction in the standard model was given by Boneh and Boyen [21], but it was proved secure under a weaker security notion called selective security. The first works that provided fully secure IBE constructions were given by Boneh and Boyen [17] and Waters [110]. Both of these works used the common methodology of partitioning proofs. However the rich structure of the keys of HIBE and ABE systems imposed an exponential degradation to the security of these schemes using partitioning techniques. The first fully secure constructions were given later along with the development of the ABE notions.

Prior to our works, several attribute-based encryption schemes have been proposed in the literature with a wide range of characteristics. The refinement of the ABE notion was proposed shortly after its introduction in the work of Goyal *et al.* [57], where they considered two dual notions. One is the aforementioned notion of ciphertext-policy ABE, where the users possess a set of attributes and the ciphertext is tagged with a policy. On the other hand, in the Key-Policy ABE (KP-ABE), each secret key is tagged by a policy/Boolean formula while the ciphertext is related to a set of attributes. In that setting, each user holds a number of secret keys that are tagged with specific policies, and he can decrypt the ciphertext if and only if one of his keys has a policy that is satisfied by the ciphertext's attribute set. At the time of writing this thesis, most ABE schemes can work natively using any monotone Boolean formula as a policy. Non monotone Boolean formulas can be simulated by allowing negative attributes with linear overhead. One exception is the ABE scheme by Ostrofsky *et al.* [90] that supports natively non monotone Boolean formulas. Several selectively-secure ABE schemes were presented in these works and others [35, 56, 112].

The first fully secure constructions for HIBE under the standard model were presented by Waters and Lewko-Waters [72, 111] using the so-called dual encryption methodology. Using the same techniques Lewko *et al.* [68] and Okamoto *et al.* [86] presented the first fully secure ABE schemes. In this thesis, among other techniques, we will leverage the same methods to achieve the orthogonal direction of leakage resilience. Finally, we mention here the suggestion of the multi-authority setting by Sahai and Waters [99], which defines the problem of constructing an attribute-based scheme where several independent master authorities authorize users for different credentials. The first paper to attempt this was by Chase [33]. These constructions allow users to encrypt data according to policies that contain attributes from different control domains and maps better to the real world, where the credentials of each user are not all authorized by a central authority. For example, the authority "University of Texas at Austin" might issue secret keys for the attribute "PhD student", while the "U.S.A." authority might issue secret keys for the attribute "U.S. Citizen". These systems can be applied to numerous settings and constituted one of the main directions in "bringing ABE to practice".

1.2 Brief Overview of Side-Channel Attacks and Leakage Resilience

Defining and achieving the right security models is crucial to the value of provably secure cryptography. When security definitions fail to encompass *all* of the power of potential attackers, systems which are proven "secure" may actually be vulnerable in practice. It is often not realistic or desirable to address such problems solely at the implementation level. For example, public key cryptography alleviated the need for a hardware-secure communication channel by allowing users to transmit messages securely using only public information. Therefore, the ultimate goal of cryptography should be to provide efficient systems which are proven secure against the largest possible class of potential attackers. Additionally, these systems should provide the most advanced functionalities available.

In the recent years, a long line of research is motivated by a variety of side-channel attacks [12, 13, 18, 19, 49, 58, 65, 66, 81, 96], which allow attackers to learn partial information about secrets by observing physical properties of a cryptographic execution. The first techniques by Kocher and Boneh *et al.* [18, 65] used timing measurements on the machines implementing the cryptographic protocols to extract secret information about the public keys. Other works like Kocher *et al.* and Bihma *et al.* [13, 66] used differential power analysis to achieve the same goal. For example by examining the power trace of a device during elliptic curve operations one was able to extract the bits of the secret key one by one [14]. Works by Quisquater *et al.* [96] and Gandolfi *et al.* [49] used electromagnetic analysis. Finally, the cold-boot attack of Halderman *et al.* [58] allows an attacker to learn information about memory contents of a machine even after the machine is powered down. Following these works the emergence of *leakage-resilient cryptography* has led to constructions of many cryptographic primitives which can be proven secure even against adversaries who can obtain limited additional information about secret keys and other internal state.

Leakage-resilient cryptography in the strongest security notion (see Sec. 2.2) models a large class of side-channel attacks by allowing the attacker to specify an efficiently computable leakage function f and learn the output of f applied to the secret key and possibly other internal state at specified moments in the security game. Clearly, limits must be placed on f to prevent the attacker from obtaining the entire secret key and hence easily winning the game. One approach is to bound the total number of bits leaked over the lifetime of the system to be significantly less than the bit-length of the secret key [1, 80]. Another approach is to continually refresh the secret key and bound the leakage between each update (this is called "continual leakage") [28, 39]. Both of these approaches have been employed successfully in a variety of settings, yielding constructions of stream ciphers, signatures, symmetric key encryption, public key encryption, and identity-based encryption (IBE) which are leakage-resilient under various models of leakage [1, 4, 5, 27, 28, 30, 36, 38–40, 43–46, 63, 80, 94].

1.3 Summary of Our Results

Leakage Resilient Constructions The first half of the thesis is focused on resilience against side-channel attacks. Several security notions were presented to model the real world more closely and resulted in the so-called leakage resilient schemes (Akavia *et al.* [1]). A scheme which is leakage resilient is secure against all polynomial time attackers, who can not only gather public information about the system (such as public keys and transmitted ciphertexts), but they can specify an arbitrary leakage function that acts on the secret key of the system and receive its output. This function models the aforementioned side-channel attacks and has to satisfy several restrictions that map closer to the above attacks. For example, the timing measurements or power dissipation attacks lead to the model "only computation leaks information" by Micali *et al.* [77], where the leakage function is applied only on computations and can act only on the data (secret keys, random coins, etc.) that is "touched" during these computations. The cold-boot attacks of Halderman *et al.* [58] allow the attacker to gather some information from the memory contents of the machine. This and similar works suggested the need for the "bounded" and the "continual" leakage models, where the leakage function is more general and is allowed to act on any data stored in the challenger's machine regardless of whether they are used in computations or not. In the "bounded" model the function's output size is bounded by a specific threshold which is always less than the size of the secret key. If the attacker was able to acquire the entire secret key he would be able to trivially break the security of the scheme. Therefore, the main goal of these schemes is to allow as much leakage as possible (optimally $1 - \varepsilon$ of the secret key size, where ε is a very small positive constant), while at the same time remaining secure. In the "continual" leakage model there exists one (or more) update algorithms that act on the secret parameters and "re-randomize" them suitably. The leakage function is allowed to act only on the updated versions and the amount of leakage is bounded only between two updates. The total amount of leakage can be arbitrary. Several constructions have been presented in this setting with a diverse variety of features [28, 39].

Following these works we propose several methods to achieve leakage resilience for a variety of schemes. We present three identity-based encryption systems that are secure under bounded leakage attacks. Identity-based encryption is a simpler form of attribute-based encryption where the encryptor uses only the identity (possibly a string or an email address) of the recipient in order to encrypt. It is equivalent to the setting where a user owns only one "attribute": his name. We use a novel tagging technique to expand the secret key space such that leakage from the secret key does not give to the attacker any non-negligible advantage in breaking security of the scheme. This technique was applied to several non-leakage-resilient schemes to provide schemes that are provably leakage-resilient. The original systems used random secret keys with only one degree of freedom, which was explorable to the secret key holder. This means that the owner of the secret key could re-randomize his key arbitrarily without knowing the secret parameters of the IBE system (the master secret key). In this sense the information each key holds is deterministic. The new technique we applied was to add another randomness to the secret keys, called "tag", coupled with some master secret key terms. As a result, the secret-key holder cannot anymore re-randomize his key (in this degree of freedom). The added randomness allows the simulators of our security proofs to provide the attacker leaked information from a properly distributed secret key with a tag of our choice.

This idea of adding additional degrees of freedom to the secret keys of our systems is used in the main construction of the first part of the thesis. We utilize the dual system methodology introduced by Waters [111] in order to provide fully secure constructions of different IBE, HIBE, and ABE schemes. Our idea was to use this methodology to modify existing dual system encryption system to leakage-resilient ones. We show that the techniques of dual system encryption naturally lead to leakage resilience. We demonstrate this by providing leakage-resilient constructions of IBE, HIBE, and ABE systems which retain all of the desirable features of dual system constructions, like full security from static assumptions and close resemblance to previous selectively secure schemes. We present our combination of dual system encryption and leakage resilience as a convenient abstraction and reduce proving security to the establishment of three properties. Our approach not only combines the benefits of dual system encryption and leakage resilience, but also qualitatively improves upon the leakage tolerance of previous leakage-resilient schemes. In particular, our system can tolerate leakage on the master key, as well as leakage on several keys for each identity (this can be viewed as continual leakage, where secret keys are periodically updated and leakage is allowed only between updates, and not during updates). Previous schemes only allowed bounded leakage on one secret key per identity, and allow no leakage on the master key. Some works allowed bounded leakage on each of many keys per identity, but allowed no leakage on the master key. We develop a simple and versatile methodology for modifying a dual system encryption construction and proof to incorporate strong leakage resilience guarantees. The change to the constructions is minimal, and can be viewed as the adjoining of a separate piece which does not interfere with the intuitive and efficient structure of the original system. Essentially, we show that dual system encryption and leakage resilience are highly compatible, and their combination results in the strongest security guarantees available for cryptosystems with advanced functionalities.

Efficient Constructions While the dual system framework is highly useful for the proofs, the current constructions use bilinear groups of large composite order or a prime order framework, called *dual pairing vector spaces* (DPVS) [69, 84–86, 89], that "emulates" the characteristics of the composite order groups . Computations on group elements of composite order induce a significant overhead on the constructions, since they are inherently slower than their prime order counter-parts (see Sec. 3.3.4). On the other hand, DPVS schemes work roughly by substituting a small number of composite group elements with a vector of sufficiently high dimension of prime order group elements. The dimension of the vectors, and thus the number of group elements, should be high enough ($10 \sim 60$) to ensure security. As a result, the dual system schemes sustain a significant efficiency overhead in comparison to prime order ABE constructions.

Therefore, our second line of work focused on bringing attribute-based encryption closer to practical implementations and at the same time providing novel techniques to achieve augmented features and provable security. The trade-off we have to sustain is that relaxed security notions were used due to the fact that we did not use the dual system methodology and resorted to older “program and cancel” techniques. More specifically, we had to embed the terms of our complexity assumptions into the public parameters of ours systems in the security proof so that we achieve the necessary calculations. Our first work towards that direction focuses on the “large universe” feature of the schemes. A common classification property of ABE constructions is whether the attribute set is “small universe” or “large universe”. In “small universe” constructions the size of the attribute space is polynomially bounded in the security parameter and the attributes were fixed at setup. Moreover, the size of the public parameters grew linearly with the number of attributes. In “large universe” constructions, on the other hand, the size of the attribute universe can be exponentially large and is thus desirable to have.

Achieving the large universe property can be challenging on its own. Different works either imposed restrictions on the expressiveness of the policies or were proved secure in the random oracle model. For constructions that had no bounds on the expressiveness of policies and constant sized public parameters, the random oracle security model was used. The above restrictions place undesirable burdens on the deployment of ABE schemes. If the designer of the system desires the benefits of avoiding the random oracle heuristic, he has to pick a specific bound for the expressiveness of the system at the setup time; either the size of the attribute universe or the bound on the policies. If the bound is too small, the system might exhaust its functionality and will have to be completely rebuilt. For example, consider the design of a framework that allows attribute-based encryption in a huge multinational company and suppose that, as this company expands, a large number of new attributes have to be added to the system. If this number exceeds the bound set during the initial deployment of the system, then the company would have to re-deploy the (expanded) system and possibly re-encrypt all its data spending a huge amount of expenses. On the other hand, if the bound chosen is too big, the increased size of the public parameters will impose an unnecessary efficiency burden on all operations. The first large universe constructions in the standard model were presented in the work of Lewko *et al.* [74]. They presented the first large universe key-policy ABE construction, secure in the standard model using the dual system framework on composite order groups to prove security. The system was proved selectively secure under static assumptions.

We aim to get practical large universe ABE schemes by adapting and expanding the system from this work into the prime order setting. In proving security we go back to more traditional "program and cancel" techniques instead of the dual system framework. We present two practical large universe ABE constructions (one ciphertext-policy and one key-policy ABE) in prime order bilinear groups both selectively secure in the standard model under two different q-type assumptions. Our three main objectives in this work were large universe constructions, efficiency, and security in the standard model.

Both schemes support a “large universe” attribute space and their public parameters consist of a constant number of group elements. No bounds or other restrictions are imposed on the monotonic Boolean formulas or the attribute sets used by the algorithms of the schemes; thus eliminating the need for design decisions at setup. The efficiency objective refrained us from using composite order groups or dual pairing vector spaces, while to achieve security in the standard model we relied to non-static (q-type) assumptions and selective notions. These assumptions are non-static in the sense that a polynomial number of terms is given to the adversary and therefore they are intuitively stronger than the static ones. However, the polynomial number of terms gives the ability to the simulator of the proof to embed the additional entropy in the constant number of public parameters. We showcase different techniques for harnessing the power of these assumptions to achieve our large universe constructions. Finally, we demonstrate the efficiency of our constructions by implementing our schemes. We compare performance results to other ABE schemes in prime order groups.

Our second work towards more efficient ABE schemes focuses on the multi-authority setting of attribute-based encryption. The typical scenario presented for ABE is where a single authority issues all private keys. This works well in the setting where data is managed within one organization or trust domain. However, there are many scenarios when one will wish to describe a policy that spans multiple trust domains. For example, U.S. military and defense are several organizations that wish to manage the distribution of their own credentials. If we wished to write an access policy that referenced credentials from both of them using standard ABE, we would require one organization ceding control to another or a third party. To address this issue, multi-authority or decentralized ABE systems were constructed by Chase [33], where multiple parties could play the role of an authority. Initial attempts at such systems sacrificed a significant amount of expressiveness compared to analogs in the one authority setting. Lewko *et al.* in [73] provided a system that roughly matched the expressiveness. In their system a policy could be expressed as any monotonic Boolean formula over attributes that can be issued by any authority which publishes a public key. Their main construction technique is to use a hash over a global identifier. Upon decryption this extra component serves as a "blinding factor" that only disappears if the ciphertext is satisfied. While the expressiveness, of this distributed ABE system is relatively strong, there are three major aspects that impact its practical performance compared to single authority systems.

First, the construction is set in a group of composite order N where N is the product of three primes. This alone can make certain operations such as exponentiation over an order of magnitude slower (see Sec. 3.3.4). Second, each authority in the system can "natively" support only a single attribute. If in practice we would like one party to act as an authority for up to c attributes, the party would have to create a public key consisting of c native public keys (thus blowing up the size by a factor of c). Furthermore, this only works if the attributes managed by that party can be enumerated ahead of time. This means that the attribute universe is restricted to polynomial size. Finally, the system has the native property that each authority can be used only once in each formula. In practice, if we want to get around this and let it be used up to d times we can apply a simple encoding technique due to Lewko *et al.* [68].[1] This encoding however comes at the cost of blowing up both the parameters of the authority and the private key components issued by the authority by a factor of d. To make things concrete suppose that we wanted a system with an authority that managed 20 attributes each of which appeared at most 10 times in the any formula. Then the published parameters for just that one authority would need to blowup by a factor of 200 (compared to a contemporary single use CP-ABE system [25, 112]) just to deal with the encoding overhead.

We construct and implement a new decentralized ABE cryptosystem that aims to get performance close to existing single authority constructions. Our approach is to use the aforementioned construction as a substrate from which we make two significant changes to improve performance. First, we take the existing construction and pare it down to the prime order setting. This will make it inherently faster, but incompatible with the dual system proof techniques used before. Second, we add an additional piece to each ciphertext and private key component which allows us to use any string as an attribute — thus addressing the problem of an authority only supporting a single attribute and the small universe restriction. At the same time, the second change allows

[1]The one use restriction is needed to make the security proof of Lewko and Waters go through, if the one use restriction were violated there is neither a known attack nor a security proof.

the system to utilize each attribute as many times as needed in each policy. We create a proof of security in a static or selective model of security where both the challenge ciphertexts and key queries are issued before the parameters are published. In this setting we will extend the existing "program and cancel" techniques to adapt to the multi-authority setting and introduce two new ones. The trade-offs for our performance improvements are the use of the static model and an assumption whose size depends on the complexity of the challenge ciphertext policy. To demonstrate the abilities of our system we implemented our algorithms in Charm [2, 32], a framework developed for rapid cryptographic prototyping, and we provide timing results.

1.4 Related Work

Attribute-Based Encryption was introduced by Sahai and Waters [99]. In this work, the key-policy and ciphertext-policy notions were defined and many selectively secure constructions followed [11, 35, 56, 90, 95, 112]. Most of them work for non monotonic access structures with the exception of the schemes by Ostrovsky, Sahai, and Waters [90], who showed how to realize negation by incorporating specific revocation schemes into the GPSW construction. Fully secure constructions in the standard model were first provided by Okamoto and Takashima [86] and Lewko, Okamoto, Sahai, Takashima, and Waters [68]. The first large universe KP-ABE construction in the standard model was given by Lewko *et al.* [74] (composite order groups). Okamoto and Takashima initiated the dual pairing vector space framework in various works [84–86, 89], which lead to the first large universe KP-ABE construction in prime order group groups by Lewko [70]. Parameterized (non static) assumptions were introduced by Boneh *et al.* [21] and used in several subsequent works [53, 112]. The problem of an environment with multiple central authorities in ABE was considered in [33, 34, 73], while several authors have presented schemes that do not address the problem of collusion resistance [3, 7, 8, 26, 78, 105].

We note that several techniques in ABE schemes have roots in Identity-Based Encryption (IBE) and Hierarchical Identity-Based Encryption (HIBE) [21–23, 37, 52, 53, 101, 110]. Finally, we mention here the related concept of Predicate Encryption introduced by Katz, Sahai, and Waters [64] and further refined in [24, 68, 85, 86, 102, 103].

Leakage resilience has been studied in many previous works, under a variety of leakage models [1, 4, 5, 27–30, 38–46, 59, 61, 63, 77, 80, 93, 94, 107]. Exposure-resilient cryptography [29, 42, 61] addressed adversaries who could learn a subset of the bits representing the secret key or internal state. Subsequent works have considered more general leakage functions. Micali and Reyzin [77] introduced the assumption that "only computation leaks information." In other words, one assumes that leakage occurs every time the cryptographic device performs a computation, but that any parts of the memory not involved in the computation do not leak. Under this assumption, leakage-resilient stream ciphers and signatures have been constructed [45, 46, 94]. Additionally, Juma *et al.* and Goldawasser *et al.* [54, 60] have shown how to transform any cryptographic protocol into one that is secure with continual leakage, assuming that only computation leaks information and also relying on a simple, completely non-leaking hardware device.

Since attacks like the cold-boot attack [58] can reveal information about memory contents in the absence of computation, it is desirable to have leakage-resilient constructions that do not rely upon this assumption. Several works have accomplished this by bounding the total amount of leakage over the lifetime of the system, an approach introduced by Akavia *et al.* [1]. This has resulted in constructions of pseudorandom functions, signature schemes, public key encryption, and identity-based encryption [4, 5, 36, 40, 63, 80] which are secure in the presence of suitably bounded leakage. For IBE schemes in particular, this means that an attacker can leak a bounded amount of information from only *one secret key per user*. This does not allow a user to update/re-randomize his secret key during the lifetime of the system.

Recently, two works have achieved continual leakage resilience *without* assuming that only computation leaks information [28, 39]. Dodis, Haralambiev, Lopez-Alt, and Wichs [39] construct one-way relations, signatures, identification schemes, and authenticated key agreement protocols which are secure against attackers who can obtain leakage *between* updates of the secret key. It is assumed the leakage between consecutive updates is bounded in terms of a fraction of the secret key size, and also that there is no leakage during the update process. Brakerski, Kalai, Katz, and Vaikuntanathan [28] construct signatures, public key encryption schemes, and (selectively secure) identity-based encryption schemes which are secure against attackers who can obtain leakage between updates of the secret key, and also a very limited amount of leakage during updates and during the initial setup phase. The leakage between updates is bounded in terms of a fraction of the secret key size, while the leakage during updates and setup is logarithmically small as a function of the security parameter.

The dual system encryption methodology was introduced by Waters in [111]. It has been leveraged to obtain constructions of fully secure IBE and HIBE from simple assumptions [111], fully secure HIBE with short ciphertexts [72], fully secure ABE and Inner Product Encryption (IPE) [68], and fully secure functional encryption combining ABE and IPE [86].

Independently, Alwen and Ibraimi [6] have proposed a leakage resilient system for a special case of Attribute-Based Encryption, where the ciphertext policy is expressed as a DNF. Their work pursues a different technical direction to ours, and provides an interesting application of hash proof systems to the ABE setting. Security is proven from a "q-type" assumption.

1.5 Organization

In Chapter 2 we present some background information about access structures and attributes, about the various leakage-resilience models, and a brief overview of the ideas behind dual system encryption. In Chapter 3 we describe the cryptographic substrate, the building elements, used for our constructions. We present both the abstract properties of these groups, which are needed for the constructions and the security proofs, and some information about the concrete realizations of these objects in software. In Chapter 4 we present the formal definitions and security notions of all the functionalities presented in this thesis. The “robust” leakage-resilient ABE construction that works on composite order groups is shown in Chapter 5. Chapter 6 contains our “efficient” constructions along with their security proofs. These are the three ABE schemes (CP-ABE, KP-ABE, MA-CP-ABE) that work on prime order groups. In Chapter 7 we present implementation results and benchmarks that compare the efficiency of the schemes of Chapter 6 to other deployed ABE schemes. In Chapter 8 we discuss further work and future directions. In Appendices A and B we present the proofs of security of our assumptions in the generic group model and proofs of some lemmas not included in the main body. Finally, the source code of our implementations is included in in Appendix C.

CHAPTER 2

Background

2.1 Access Structures & Linear Secret Sharing Schemes

In this section, we present the formal definitions of access structures and linear secret-sharing schemes introduced by Amos Beimel [9], adapted to match our setting. Intuitively, the access structures describe the abstract notion of a policy on the attribute universe, while the linear secret sharing schemes (LSSS) describe a concrete way to implement the sharing of a secret according to a specific policy. All our constructions assume the use of an LSSS to express the policies, although they can easily be adapted to other secret sharing schemes, such as access trees [57], monotone span programs [62], and others.

Definition 2.1 (Access Structures [9])**.** Let $\mathcal{U}$ be the attribute universe. An *access structure on* $\mathcal{U}$ is a collection $\mathbb{A}$ of non-empty sets of attributes, i.e. $\mathbb{A} \subseteq 2^{\mathcal{U}} \setminus \{\}$. The sets in $\mathbb{A}$ are called the *authorized sets* and the sets not in

$\mathbb{A}$ are called the *unauthorized sets*.

Additionally, an access structure is called *monotone* if $\forall B, C \in \mathbb{A}$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$.

In our construction, we only consider monotone access structures, which means that as a user in the CP-ABE setting acquires more attributes, he will not lose his possible decryption privileges. In the KP-ABE setting, this means that as the message is encrypted with more attributes, the set of users that can decrypt it grows. General (not necessarily monotone) access structures in large universe ABE can be realized by splitting the attribute universe in half and treating the attributes of one half as the negated versions of the attributes in the other half [57].

Definition 2.2 (Linear Secret-Sharing Schemes (LSSS) [9])**.** Let p be a prime and $\mathcal{U}$ the attribute universe. A secret-sharing scheme Π with domain of secrets $\mathbb{Z}_p$ realizing access structures on $\mathcal{U}$ is *linear over* $\mathbb{Z}_p$ if

1. The shares of a secret $z \in \mathbb{Z}_p$ for each attribute form a vector over $\mathbb{Z}_p$.
2. For each access structure $\mathbb{A}$ on $\mathcal{U}$, there exists a matrix $A \in \mathbb{Z}_p^{\ell \times n}$, called the share-generating matrix, and a function δ, that labels the rows of A with attributes from $\mathcal{U}$, i.e. $\delta : [\ell] \rightarrow \mathcal{U}$, which satisfy the following:

   During the generation of the shares, we consider the column vector $\vec{v} = (z, r_2, \ldots, r_n)^{\perp}$, where $r_2, \ldots, r_n \xleftarrow{R} \mathbb{Z}_p$. Then the vector of ℓ shares of the secret z according to Π is equal to $\vec{\lambda} = A\vec{v} \in \mathbb{Z}_p^{\ell \times 1}$. The share λ_j with $j \in [\ell]$ "belongs" to attribute $\delta(j)$.

We will be referring to the pair (A, δ) as the policy of the access structure $\mathbb{A}$.

Each secret-sharing scheme (not only the linear ones) should satisfy the *reconstruction requirement*, i.e. each authorized set can reconstruct the secret, and the *security requirement*, i.e. any unauthorized set cannot reveal any partial information about the secret. More concretely, let S denote an authorized set of attributes and let I be the set of rows whose labels are in S. There exist constants $\{c_i\}_{i\in I}$ in $\mathbb{Z}_p$ such that for any valid shares $\{\lambda_i = (A\vec{v})_i\}_{i\in I}$ of a secret z according to Π, it is true that: $\sum_{i\in I} c_i\lambda_i = z$. Equivalently, $\sum_{i\in I} c_i\vec{A}_i = (1, 0, \ldots, 0)$, where $\vec{A}_i$ is the i-th row of A. Additionally, it has been proved in [9] that the constants $\{c_i\}_{i\in I}$ can be found in time polynomial in the size of the share-generating matrix A.

On the other hand, for unauthorized sets S' no such constants $\{c_i\}$ exist. In this case it is also true that if $I' = \{i | i \in [\ell] \wedge \rho(i) \in S'\}$, there exists a vector $\vec{d} \in \mathbb{Z}_p^{1\times n}$, such that its first component $d_1 = 1$ and $\left\langle \vec{A}_i, \vec{d} \right\rangle = 0$ for all $i \in I'$.

Finally, we note that if the access structure is encoded as a monotonic Boolean formula over attributes there is a generic algorithm that generates the corresponding access policy in polynomial time [9, 73] (See Sec. 2.1.2).

2.1.1 Multi-Authority Attributes

In the multi-authority setting, each attribute is controlled by a specific authority $\theta \in \mathcal{U}_\Theta$, where $\mathcal{U}_\Theta$ is the set (universe) of all authorities. We assume there is a publicly computable function $\mathsf{T} : \mathcal{U} \to \mathcal{U}_\Theta$ that maps each attribute to a unique authority. Using this mapping a second labeling of rows is defined in a policy (A, δ), which maps rows to attributes via the function $\rho(\cdot) \overset{\mathsf{def}}{=} \mathsf{T}(\delta(\cdot))$.

As an example of the mapping T, in our implementation of a multi-authority scheme, both the attribute id's and the authority id's consist of case-sensitive alphanumeric strings. The full attributes' names are of the form "[$\mathsf{attribute\text{–}id}$]@[$\mathsf{authority\text{–}id}$]" and the mapping T just extracts the part after the @ of the attribute string.

2.1.2 Converting Monotone Boolean Formulas to Share Generating Matrices

In this section we present a general transformation from any monotone access structure on the attribute universe, expressed as a Boolean formula on attributes, to a share generating matrix A that can be used to express this policy. Monotone access structures are expressed by monotone Boolean formulas, i.e. formulas that contain only AND ($\wedge$) and OR ($\vee$) gates and the atoms consist of attributes in the attribute universe. For example, consider the formula

$$F = ((V \wedge U) \vee (V \wedge W)) \wedge (U \vee X)$$

where $\mathcal{U} = \{V, U, W, X, Y\}$. This formula denotes an access structure $\mathbb{A}$ that satisfies (contains) the sets of attributes $\{V, U\}$, $\{V, W, X\}$, $\{V, U, Y\}$, etc., but does not satisfy the set $\{U, X\}$ or the set $\{W, Y\}$.

To create the share generating matrix A and the row labeling δ we treat the formula as a binary tree where the nodes are labeled with the gates and the leaves with the attributes (see Fig. 2.1). The algorithm will assign a row vector to each node and leaf, starting from the root of the tree. Also, the algorithm maintains a global counter c initially set to 1. The root is always labeled with the vector (1). Then the algorithm goes down the tree one gate at a time. If a node contains an OR ($\vee$) gate and is labeled with the vector $\vec{v}$, the algorithm assigns $\vec{v}$ to both of its children nodes, without changing the value of c. If a node contains an AND ($\wedge$) gate and is labeled with the vector $\vec{v}$, the algorithm first pads $\vec{v}$ with zeros until it gets length c. The algorithm assigns the vector $\vec{v}|1$ to the left child of the node and the vector $(0, 0, \ldots, 0)|-1$ to the right child, where $(0, 0, \ldots, 0)$ is of length c. Finally it updates the value of c to $c+1$ and moves on to the remaining nodes. The algorithm terminates when all leaves are assigned with vectors. The share generating matrix consists of all the vectors assigned to the leaves padded with zeros to length c. Each row is labeled in the δ mapping with the attribute of the leaf corresponding to it.

In the above example, the root node has the gate $\wedge$ and is labeled with the vector (1). Therefore the two children get $(1, 1)$ and $(0, -1)$ and c is updated to 2. Both of them have $\vee$ gates, hence the two nodes with $\wedge$ gates in the right subtree get $(1, 1)$ while the two leaves, U and X get labeled with

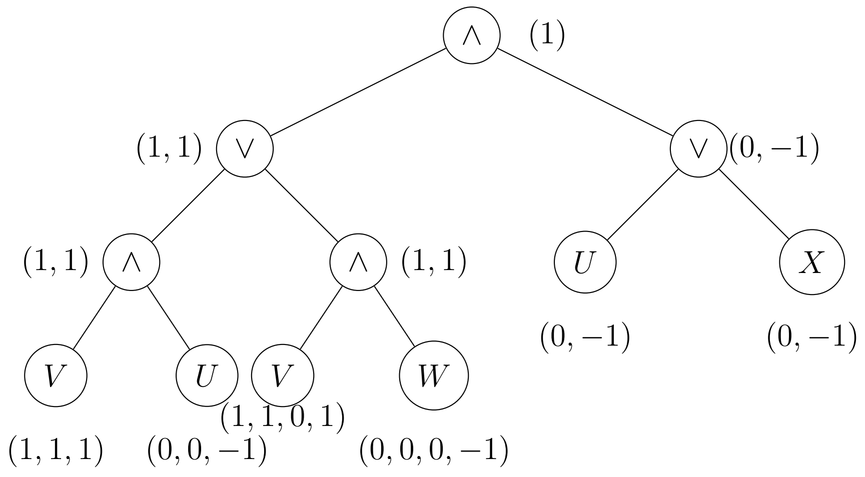

Figure 2.1: Binary tree and node vectors for the formula $F = ((V \wedge U) \vee (V \wedge W)) \wedge (U \vee X)$.

$(0, -1)$. Following the same procedure we reach to the policy (A, δ) shown in Fig. 2.2. Notice that the vector $(1, 0, \ldots, 0)$ is in the span of a set of rows if and only if the formula F is satisfied by the attributes labeling this set of rows.

$$A = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 \\ 0 & -1 & 0 & 0 \end{bmatrix} \qquad \delta = \begin{bmatrix} V \\ U \\ V \\ W \\ U \\ X \end{bmatrix}$$

Figure 2.2: Access policy for the formula $F = ((V \wedge U) \vee (V \wedge W)) \wedge (U \vee X)$.

2.2 Leakage Models

Leakage resilience has been studied in many previous works, under a variety of leakage models [1, 4, 5, 27–30, 38–46, 59, 61, 63, 77, 80, 93, 94, 107]. In this section we present the various models ordered from the weakest to the strongest security notion. Our robust ABE construction of Chap. 5 satisfies the last one.

Exposure - resilient cryptography The first works in the area of leakage-resilient cryptography [29, 42, 61] addressed adversaries who could learn a fixed or variable subset of the bits representing the secret key or internal state. Obviously the adversary was not allowed to retrieve all the bits of the secret key, because in that case security was trivially compromised. However many side-channel attacks leaked complicated functions of the secret key bits and therefore the interest of the cryptographic community turned to leakage models where the attacker had access to a polynomially computable function on the secret key or the internal state of his choice. The following leakage models consider this kind of leakage.

"Only Computation Leaks" Micali and Reyzin [77] introduced the assumption that "only computation leaks information". In other words, one assumes that leakage occurs every time the cryptographic device performs a computation, but that any parts of the memory not involved in the computation do not leak. Under this assumption, leakage-resilient stream ciphers and signatures have been constructed [45, 46, 94]. Additionally, [54, 60] have shown how to transform any cryptographic protocol into one that is secure with continual leakage, assuming that only computation leaks information and also relying on a simple, completely non-leaking hardware device. Continual leakage can occur indefinitely and the total amount of leaked bits can be more than the size of the secret key. This implies that the secret keys of the system have to be periodically updated and "reset" in some sense the leakage counter.

Bounded Leakage Model Since attacks like the cold-boot attack [58] can reveal information about memory contents in the absence of computation, it is desirable to have leakage-resilient constructions that do not rely upon the above assumption. Several works have accomplished this by bounding the total amount of leakage over the lifetime of the system, an approach introduced by Naor *et al.* [80] and Akavia *et al.* [1]. This has resulted in constructions of pseudorandom functions, signature schemes, public key encryption, and identity-based encryption [4, 5, 36, 40, 63, 80] which are secure in the presence of suitably bounded leakage. For IBE schemes in particular, this means that an attacker can leak a bounded amount of information from only *one secret key per user*. This does not allow a user to update/re-randomize his secret key during the lifetime of the system.

Continual Leakage Model Two subsequent works have achieved continual leakage resilience *without* assuming that only computation leaks information [28, 39]. Dodis, Haralambiev, Lopez-Alt, and Wichs [39] construct one-way relations, signatures, identification schemes, and authenticated key agreement protocols which are secure against attackers who can obtain leakage *between* updates of the secret key. It is assumed the leakage between consecutive updates is bounded in terms of a fraction of the secret key size, and also that there is no leakage during the update process. Brakerski, Kalai, Katz, and Vaikuntanathan [28] construct signatures, public key encryption schemes, and (selectively secure) identity-based encryption schemes which are secure against attackers who can obtain leakage between updates of the secret key, and also a very limited amount of leakage during updates and during the initial setup phase. The leakage between updates is bounded in terms of a fraction of the secret key size, while the leakage during updates and setup is logarithmically small as a function of the security parameter.

2.3 Overview of Dual System Encryption

Dual System Encryption is a proof methodology introduced in [111] and used to provide adaptively secure IBE, HIBE, and ABE systems. Our work in [71] was the first to leverage this methodology to achieve leakage resilience for all these functionalities and, at the same time, maintain full security for the proposed schemes. The ABE system of this work is described in Chap. 5. In this section we describe the framework of dual system encryption and describe the intuition behind the security of our robust construction.

Adaptive Security of ABE Systems The formal definition of adaptive (or full) security of ABE systems is given in Chap. 4. For the purpose of describing the dual system framework and the obstacles it tries to solve, we will briefly describe the security definition of CP-ABE schemes. The main ideas and challenges in the KP-ABE setting are similar. CP-ABE security is defined via a game between a challenger and an attacker. This game is a conservative model of the real world interactions between an eavesdropper and the users of the cryptographic scheme. The eavesdropper gets the public parameters of the scheme and can impersonate or compromise a set of users by acquiring their secret keys. We want to design the scheme in such a way that even in that case the eavesdropper can not get any information from a ciphertext of a different user. It is considered conservative since we give the attacker the possibility to choose adaptively the compromised users, the encrypted messages, and the challenge user.

More specifically the game works as follows: Initially, the challenger sets up the CP-ABE system and provides the public parameters to the attacker. Then the attacker can request a polynomial number of secret keys for various attribute sets. The attacker may choose these sets any way it wishes and it can do that in a *fully adaptive* fashion. That is, each of his choices may depend on the previous ones and the previously acquired secret keys. At some point, the attacker declares it wants to go into the challenge phase where it outputs a pair of messages of equal length and an access policy. As before it can choose both these *challenge* messages and the *challenge* access policy any way it wants with the only restriction that the access policy must not be satisfied by any of the attribute sets of the keys selected so far. Then the challenger flips a uniformly random bit and, depending on the value of the bit, it encrypts either the first or the second challenge message under the challenge access policy with the set public parameters and sends the *challenge* ciphertext to the attacker.

The attacker can then go into a second secret-key query phase, where it can request adaptively additional secret keys with the same restriction; that their attribute sets do not satisfy the access policy. Finally, it tries to guess the message encrypted in the challenge ciphertext and outputs a bit to signify its answer. We say that a scheme is *secure* when for all polynomial time attackers the probability of guessing correctly is negligibly close to 1/2. Security is proved formally via a security reduction, where we assume the existence of an attacker in the above game and define a *simulator* algorithm that simulates the challenger of the game and breaks a hard computational problem.

Notice that the restriction imposed on the queried keys ensures that the attacker can not decrypt the challenge ciphertext with any of these keys, since they correspond to non-authorized attribute sets. However, it can be the case that the *union* of attribute sets of two or more secret keys is an authorized set. This implies that our schemes should be resilient to the so-called *collusion attacks*, where several users combine somehow their secret keys in order to decrypt a ciphertext for which neither of them is individually authorized to decrypt. Therefore in a secure CP-ABE scheme the attributes should not be transferable among users. Security against collusion attacks was one of the first challenges of the (non-adaptive) ABE constructions and could be solved with careful personalization of the users' secret keys.

An additional obstacle however is present in the adaptive setting: The simulator of our reduction has to be ready to construct a secret key for *any* attribute set requested by the attacker and a ciphertext for *any* challenge access policy. Since the simulator also tries to leverage the attacker's success to solve the hard computational problem, it looks like it does not need the attacker's help. It can create a secret key that decrypts the challenge ciphertext and attack the scheme by itself. Since this appears to be impossible to achieve (unless the simulator knows the challenge access policy at the beginning of the game, as in the non-adaptive setting), dual system encryption aims in proving that the success probability of any PPT attacker in the above security game is negligibly close to the success probability of the same attacker in a game where the challenger outputs "non-working" keys; thus bypassing the above paradox. In the following paragraphs we describe how dual system encryption achieves that and how it allows to achieve leakage resilience in ABE schemes.

Normal and Semi-functional Components In a dual system encryption system the keys and the ciphertexts come in two flavors; either *normal* or *semi-functional*. The normal keys and ciphertexts are utilized by the users of the deployed system to decrypt and encrypt messages. The semi-functional components appear only in the security proofs of the schemes. An authorized normal key can decrypt a semifunctional ciphertext and a normal ciphertext can be decrypted by an authorized semi-functional key. By authorized we mean that the attribute set (of the key in CP-ABE or of the ciphertext in KP-ABE) is one of the authorized sets of the access policy (of the ciphertext in CP-ABE or of the key in KP-ABE). However when a semi-functional key is used to decrypt a semi-functional ciphertext decryption fails (with very high probability) even if the key is authorized.

The proof of security works by providing a sequence of games starting with the real security game where all keys and ciphertexts are normal and ending with the same game but with all keys and ciphertexts semi-functional. It is proved that the advantage of any PPT attacker in any of the games is negligibly close to its advantage of the next game. The second game in the sequence is one where the challenge ciphertext is semi-functional and all keys are normal. Then if the adversary makes at most Q secret key queries, there are Q subsequent games. In the i-th game the first i secret keys are semi-functional while the remaining keys are normal. In the final game all keys and the challenge ciphertext are semi-functional and therefore security can proved relatively easy, since the simulator outputs only "useless" keys.

Nominal Semi-functionality However, the aforementioned paradox that the simulator can construct a secret key that can decrypt the ciphertext seems now to have been moved to the transitions of the security games. Namely, the point where a simulator creates a secret key, let's say the i-th one, that is either normal or semi-functional in order to advance from the $(i-1)$-th game to the i-th. It is crucial that it does not know which form of secret key it created, because otherwise it wouldn't have to leverage the attacker. But since we are in the adaptive setting, the simulator can create a secret key, which is authorized for the semi-functional ciphertext and should be either semi-functional or normal, but unknown to it. The question is if the simulator tries to decrypt the challenge ciphertext with this key wouldn't it detect its nature?

The above issue is resolved by introducing the notion of nominal semi-functionality. A secret key is *nominally* semi-functional with respect to a semi-functional ciphertext if it is semi-functional, authorized for this ciphertext, and correlated to it in such a way that decryption succeeds. More specifically the random coins used for the creation of the secret key is correlated to the random coins used for the creation of the ciphertext and during decryption the semi-functional components of both cancel each other out. If the random coins were not correlated we would have a regular semi-functional key and then decryption would fail. In the case where our simulator tries to create the aforementioned secret key, the system and the reduction are set in such a way that it will either create a normal secret key or a nominal semi-functional key. Thus if it tries to decrypt the ciphertext and detect the nature of the key, decryption will succeed unconditionally providing no information about the nature of the key.

As it is implied by the previous paragraph, the randomness of some secret keys created by the simulator is correlated to the randomness of the challenge ciphertext. However the secret keys should be created using properly generated random coins. In the first dual system encryption schemes this obstacle was by-passed using the fact that the attacker can not request an authorized key. The fact that the keys could not decrypt the challenge ciphertext regardless of being normal or semi-functional was enough to hide information theoretically the aforementioned correlation by the attacker. However this was not true anymore for the leakage resilient setting.

Leakage Resilient Semi-functionality In the leakage resilient security game the attacker is given the ability to request "leakage" from secret keys keys of its own choice, even from authorized ones. Namely, the above game is modified such that the attacker is allowed to send several polynomially computable functions to its challenger before seeing the challenge ciphertext. Each time the challenger applies the function on a secret key of the attacker's choice and returns the result to the attacker. We require that the size of the output of the function is strictly smaller than the size of the entire key. Otherwise the attacker would be able to use the secret key to decrypt the challenge ciphertext and the security notion would be impossible to achieve. For our construction we utilized the strongest leakage-resilient security notion, i.e. the continual leakage model. In this model the attacker can request a bounded amount of leakage from each key, but can also request that a specific key is updated to a new one and essentially "reset" the leakage counter on it. We note that no leakage queries are allowed after the challenge phase, because in this case it is impossible to achieve security as the attacker can pick as the leakage function the decryption of the challenge ciphertext. Formally the definition of leakage resilient security is presented in Chap. 4.

Getting back to the dual system encryption framework, we see that we can not use past techniques to hide the nominality of the secret keys we give to the attacker. More specifically we can not hide with these techniques the nominality of the secret keys that are authorized to decrypt the ciphertext, but are leaked to the attacker. In our work [71], we presented the first technique that hid information theoretically the correlation of these secret keys with the challenge ciphertext and achieved a fully secure ABE scheme. Our technique used an information theoretic lemma introduced in [28] stating that “random subspaces are leakage resilient”. The authors of [28] showed how to create selectively secure leakage resilient IBE, HIBE, and ABEschemes (in the continual leakage model) by straight-forwardly utilizing this lemma.

The main idea of our work was to use dual system encryption, that was known to provide fully secure schemes, to achieve the orthogonal goal of leakage-resilience, as well. We build the keys and ciphertexts in such a way that the two semi-functional components were nominally semi-functional to each other if and only if specific vectors of their semi-functional space were orthogonal to each other. Therefore they were nominal if and only if a vector belonged to the orthogonal subspace of another one. According to the aforementioned lemma this correlation was hidden by the view of the attacker and the advantage it could acquire out of this correlation was only negligible.

In addition, we introduced secret key update algorithms and transformed the master secret key of our construction to have a similar form as the regular secret keys. As a result we achieved continual leakage and leakage resilience against master secret key attacks. The final scheme was fully (adaptively) secure and resilient against master or user secret key leakage attacks in the continual leakage model.

CHAPTER 3

Bilinear Groups

In this chapter we introduce the primitive components of our cryptographic constructions: the bilinear groups. In the first three sections we present the abstract mathematical properties of these groups and the assumptions we require that they satisfy, in order to prove the security of our schemes. In the fourth section we present explicit implementations of these groups using elliptic curves and we pinpoint the differences from the abstracted objects.

The main feature of bilinear groups is that they admit an efficiently computable mapping that takes two group elements and maps them to another group, called the target group. This mapping should be non-degenerate, i.e. it does not map everything to the identity element of the target group, and it should be linear on both of its arguments. Bilinear groups opened the way to numerous cryptographic primitives, such as identity-base encryption [20], three-party key agreement, hierarchical identity based encryption and numerous others.

In this thesis we will be concerned with bilinear groups of prime and composite order. The former are simpler and more efficient, but lack the necessary structure to facilitate proofs of adaptive security. The properties of the latter provide increased flexibility in proving the security of various schemes, but the actual implementations of them are significantly slower than the ones of prime order groups on the same security level. This reason has motivated several researchers to implement the functionality and the properties of composite order groups using prime order groups [47, 70]. Although these works provided constructions that simulate the structure of composite order groups and achieved a significant efficiency improvement with respect to them, they are still several factors slower than the prime order group implementations. These factors, as well as the exact simulation, depends on the application at hand.

3.1 Abstract Properties of Bilinear Groups

In this section we present the abstract properties of the groups used by our schemes and implementations. Throughout the thesis we use multiplicative notation for the group operation.

3.1.1 Prime Order Bilinear Groups

Let $\mathbb{G}$ and $\mathbb{G}_T$ denote cyclic groups of the same prime order p, where $\|p\| = \lambda \in \mathbb{N}$. We say that the tuple $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ is a bilinear group tuple if

g is a generator of $\mathbb{G}$ and there exists a mapping $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ that satisfies the following properties:

1. *Efficiently computable:* It can be computed in polynomial time in λ.
2. *Non-degenerate:* It is true that $e(g, g) \neq \mathbb{1}_{\mathbb{G}_T}$. As a result, $e(g, g)$ is a generator of $\mathbb{G}_T$.
3. *Bilinear:* For all all $a, b \in \mathbb{Z}_p$, it is true that $e(g^a, g^b) = e(g, g)^{ab}$.

For our assumptions and constructions we assume that there exists a probabilistic polynomial time group generator algorithm $\mathcal{G}(1^\lambda)$ that takes as input the security parameter λ encoded in unary and outputs (a description of) a bilinear group tuple $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ with $\|p\| = \lambda$.

Remark 3.1. The above definition encompasses the so called *symmetric* bilinear groups. These are the only groups that we consider in our constructions and our security proofs. In general the mapping can be asymmetric, i.e. $e : \mathbb{G} \times \mathbb{H} \to \mathbb{G}_T$, where $\mathbb{G}, \mathbb{H}, \mathbb{G}_T$ are all groups of prime order p. All our assumptions, constructions, and security proofs can be generically transformed to the asymmetric setting by substituting all terms of the form $g^a \in \mathbb{G}$ with terms $g^a \in \mathbb{G}$ and $h^a \in \mathbb{H}$, where g, h are generators of the groups $\mathbb{G}, \mathbb{H}$ respectively.

3.1.2 Composite Order Bilinear Groups

Another very useful tool for the construction of cryptographic primitives is the relaxation of the above definition such that the groups $\mathbb{G}$ and $\mathbb{G}_T$ are of composite order. In this case we will denote the order by $N = |\mathbb{G}| = |\mathbb{G}_T|$.

For our constructions N will be the product of three prime numbers p_1, p_2 and p_3. According to Lagrange's theorem this implies that $\mathbb{G}$ (and $\mathbb{G}_T$) contains three prime order subgroups. We will denote the subgroups of order p_1, p_2, and p_3 by $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$, respectively.

Suppose that $g \in \mathbb{G}$ is a generator of the full group $\mathbb{G}$, i.e. it has order $N = p_1 p_2 p_3$. Then a generator of the i-th subgroup is equal to

$$g_i = g^{\prod_{j \in [3]} p_j^{1-\delta_{i,j}}}$$

where $\delta_{i,j} = 0$ for $i \neq j$ and $\delta_{i,j} = 1$ for $i = j$; the Kronecker's delta. Thus, the generator of the i-th subgroup is obtained by raising the generator of the entire group $\mathbb{G}$ to the exponent N/p_i. For example, a generator of the $\mathbb{G}_1$ subgroup is $g_1 = g^{p_2 p_3}$.

An important property of the composite order bilinear groups, called *orthogonality*, is that pairing group elements from different subgroups always outputs the identity element of the target group. To see this, consider pairing an element $A \in \mathbb{G}_i$ to an element $B \in \mathbb{G}_{i'}$, with $i, i' \in [3]$ and $i \neq i'$. Since g_i and $g_{i'}$ are the respective generators, we have that $A = g_i^a$ and $B = g_{i'}^b$ for some $a \in \mathbb{Z}_{p_i}$ and $b \in \mathbb{Z}_{p_{i'}}$. Then according to the properties of the pairing operation we have:

$$\begin{aligned}
e(A,B) = e(g_i^a, g_{i'}^b) &= e(g^{a\prod_{j\in[3]} p_j^{1-\delta_{i,j}}}, g^{b\prod_{j'\in[3]} p_{j'}^{1-\delta_{i',j'}}}) \\
&= e(g,g)^{ab\cdot\prod_{j\in[3]} p_j^{2-\delta_{i,j}-\delta_{i',j}}} \\
&= e(g,g)^{p_1p_2p_3\cdot ab\cdot\prod_{j\in[3]} p_j^{1-\delta_{i,j}-\delta_{i',j}}} \\
&= (e(g,g)^{p_1p_2p_3})^{ab\cdot\prod_{j\in[3]} p_j^{1-\delta_{i,j}-\delta_{i',j}}} \\
&= \mathbb{1}_{\mathbb{G}_T}^{ab\cdot\prod_{j\in[3]} p_j^{1-\delta_{i,j}-\delta_{i',j}}} = \mathbb{1}_{\mathbb{G}_T}
\end{aligned}$$

As it was the case for the prime order groups, we assume that there exists a probabilistic polynomial time group generator algorithm $\mathcal{G}(1^\lambda)$ that takes as input the security parameter λ encoded in unary and outputs (a description of) a bilinear group tuple $(N, p_1, p_2, p_3, g, \mathbb{G}, \mathbb{G}_T, e)$ with $\|p_i\| = \lambda$ for all $i \in [3]$.

3.2 Computational Assumptions

In this section we present the computational assumptions which we utilize to prove the security of our schemes. First, we present three q-type assumptions on prime order groups needed for the security proofs of our large universe ABE schemes of Chap. 6. Afterwards we present three subgroup decision assumptions on composite order groups used in the security reductions of the leakage-resilient ABE construction of Chap. 5.

All assumptions are defined via a security game between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$, both modeled as probabilistic interactive Turing machines.

The challenger does not necessarily have to be efficient (polynomial time), although in our implementations and schemes this happens to be the case. On the other hand, we consider only polynomial time attackers for our security definitions.

In each security game the challenger $\mathcal{C}$ generates a set of given terms denoted by the D tuple, and two challenge terms denoted by T_0 and T_1. The distributions of D, T_0, T_1 are defined by the assumption at hand, which in this abstract example we denote by GenAss. Then $\mathcal{C}$ flips a uniformly random bit $b \xleftarrow{R} \{0,1\}$ and the attacker $\mathcal{A}$ is given (D, T_b). The goal of $\mathcal{A}$ is to correctly guess the bit b, i.e. it outputs a bit $b' \in \{0,1\}$. The success of the attacker is quantified by its advantage which is defined as

$$\begin{aligned}\mathsf{Adv}_{\mathcal{A}}^{\mathsf{GenAss}}(\lambda) &:= \left|2 \cdot \Pr\left[b' = b\right] - 1\right| \\ &= \left|\Pr\left[\mathcal{A}(D, T_1) = 1\right] - \Pr\left[\mathcal{A}(D, T_0) = 1\right]\right|\end{aligned}$$

3.2.1 Three q-Type Assumptions on Prime Order Groups

The three assumptions on prime order groups are similar to the q-Decisional Parallel Bilinear Diffie-Hellman Exponent assumption introduced in [112]. They are all q-type assumptions, meaning that they are parameterized by an integer q, polynomial in the security parameter λ. We will be referring to them as q-$\mathsf{DPBDH1}$, q-$\mathsf{DPBDH2}$, and q-$\mathsf{DPBDH3}$, and we will use them in the security proofs of our unbounded KP-ABE scheme, our unbounded CP-ABE scheme, and our unbounded multi-authority CP-ABE scheme, respectively. The original q-Decisional Parallel Bilinear Diffie-Hellman Exponent assumption, denoted by q-DPBDH, is almost the same as the q-DPBDH3 assumption and for completeness it is described as the delta of q-DPBDH3 (see Rem. 3.6). The generic security of all these computational assumptions is shown in appendix A.

We remind the reader that, for all assumptions, we require the existence of a group generator algorithm $\mathcal{G}$ that gets a security parameter 1^λ as input and produces a description of a prime order bilinear group (see Sec. 3.1.1). This algorithm outputs a tuple $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ and is called by the challenger at the beginning of the following security games. Also we remind that the notation $[q, q]$ or $[q, q, 2q]$ denotes the set of integer pairs $[q] \times [q]$ or $[q] \times [q] \times [2q]$, respectively.

The q-DPBDH1 Assumption

After the challenger $\mathcal{C}$ calls the group generation algorithm $\mathcal{G}\left(1^\lambda\right) \to (p, g, \mathbb{G}, \mathbb{G}_T, e)$, it picks $q+3$ random exponents $x, y, z, b_1, b_2, \ldots, b_q \xleftarrow{R} \mathbb{Z}_p$. The given tuple D consists of the group description $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ and all of the following terms:

$$\begin{array}{ll} g, g^x, g^y, g^z, g^{(xz)^2} & \\ g^{b_i}, g^{xzb_i}, g^{xz/b_i}, g^{x^2zb_i}, g^{y/b_i^2}, g^{y^2/b_i^2} & \forall i \in [q] \\ g^{xzb_i/b_j}, g^{yb_i/b_j^2}, g^{xyzb_i/b_j}, g^{(xz)^2b_i/b_j} & \forall (i,j) \in [q,q] \text{ with } i \neq j \end{array}$$

The challenge terms are

$$T_0 = e(g,g)^{xyz} \text{ and } T_1 = R \xleftarrow{R} \mathbb{G}_T$$

Definition 3.2. We say that the q-$\mathsf{DPBDH1}$ assumption holds if for all PPT attackers $\mathcal{A}$ it is true that $\mathsf{Adv}_{\mathcal{A}}^{q\text{-}\mathsf{DPBDH1}}(\lambda) \leq \mathsf{negl}(\lambda)$.

The q-$\mathsf{DPBDH2}$ Assumption

After the challenger $\mathcal{C}$ calls the group generation algorithm $\mathcal{G}\left(1^{\lambda}\right) \rightarrow (p, g, \mathbb{G}, \mathbb{G}_T, e)$, it picks $q+2$ random exponents $a, s, b_1, b_2, \ldots, b_q \xleftarrow{R} \mathbb{Z}_p$. The given tuple D consists of the group description $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ and all of the following terms:

$$\begin{array}{ll}
g, g^s & \\
g^{a^i}, g^{b_j}, g^{sb_j}, g^{a^i b_j}, g^{a^i/b_j^2} & \forall (i,j) \in [q,q] \\
g^{a^i/b_j} & \forall (i,j) \in [2q,q] \text{ with } i \neq q+1 \\
g^{a^i b_j / b_{j'}^2} & \forall (i,j,j') \in [2q,q,q] \text{ with } j \neq j' \\
g^{sa^i b_j / b_{j'}}, g^{sa^i b_j / b_{j'}^2} & \forall (i,j,j') \in [q,q,q] \text{ with } j \neq j'
\end{array}$$

The challenge terms are

$$T_0 = e(g,g)^{sa^{q+1}} \quad \text{and} \quad T_1 = R \xleftarrow{R} \mathbb{G}_T$$

Definition 3.3. We say that the q-$\mathsf{DPBDH2}$ assumption holds if for all PPT attackers $\mathcal{A}$ it is true that $\mathsf{Adv}_{\mathcal{A}}^{q\text{-}\mathsf{DPBDH2}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Remark 3.4. Notice the absence of the term g^{a^{q+1}/b_j} in the third line of the assumption. If this term were given to the attacker, then he could break the assumption trivially by pairing it with the corresponding g^{sb_j} term. On the other hand, the term $g^{a^{q+1}b_j/b_{j'}^2}$ is given, and this poses no problems in the generic group model since $j \neq j'$ and by possible pairing the adversary cannot get rid of the b_j's. See appendix A for further details.

The q-DPBDH3 Assumption

After the challenger $\mathcal{C}$ calls the group generation algorithm $\mathcal{G}\left(1^\lambda\right) \rightarrow (p, g, \mathbb{G}, \mathbb{G}_T, e)$, it picks $q+2$ random exponents $s, a, b_1, b_2, \ldots, b_q \xleftarrow{R} \mathbb{Z}_p$. The given tuple D consists of the group description $(p, g, \mathbb{G}, \mathbb{G}_T, e)$ and all of the following terms:

$$\begin{array}{ll} g, g^s & \\ g^{a^i}, g^{b_j a^i} & \forall (i,j) \in [2q, q] \text{ with } i \neq q+1 \\ g^{s/b_i} & \forall i \in [q] \\ g^{s a^i b_j / b_{j'}} & \forall (i, j, j') \in [q+1, q, q] \text{ with } j \neq j' \end{array}$$

The challenge terms are

$$T_0 = e(g,g)^{s a^{q+1}} \quad \text{and} \quad T_1 = R \xleftarrow{R} \mathbb{G}_T$$

Definition 3.5. We say that the q-DPBDH3 assumption holds if for all PPT attackers $\mathcal{A}$ it is true that $\mathsf{Adv}_{\mathcal{A}}^{q\text{-DPBDH3}}(\lambda) \leq \mathsf{negl}(\lambda)$.

Remark 3.6. The q-DPBDH assumption is the same as the q-DPBDH3 with the only difference that the $\{g^{s a^i b_j / b_{j'}}\}$ terms can not have $i = q+1$.

3.2.2 Three Subgroup Decision Assumptions on Composite Order Groups

To prove the security of our leakage-resilient construction of Chap. 5, we will use the following three assumptions in composite order groups, also used in [68, 72]. These are static assumptions, which hold in the generic group model if finding a nontrivial factor of the group order is hard. The proof of this can be found in [72].

The first two of our assumptions belong to the class of the *Source Group Subgroup Decision Assumptions* described in [10]. This class of assumptions is defined as follows: in a bilinear group of order $N = p_1 p_2 \dots p_n$, there is a subgroup of order $\prod_{i \in S} p_i$ for each subset $S \subseteq \{1, \dots, n\}$. We let S_0, S_1 denote two subsets. It is assumed to be hard to distinguish a random element from the subgroup associated with S_0 from a random element of the subgroup associated with S_1, even if one is given random elements from subgroups associated with several subsets Z_i which each satisfy either that $S_0 \cap Z_i = \emptyset = S_1 \cap Z_i$ or $S_0 \cap Z_i \neq \emptyset \neq S_1 \cap Z_i$. Note that when $S_0 \cap Z_i = \emptyset = S_1 \cap Z_i$, pairing the random element from Z_i with the unknown element under the bilinear map will always yield the identity element, while when $S_0 \cap Z_i \neq \emptyset \neq S_1 \cap Z_i$, pairing the random element from Z_i with the unknown element will not yield the identity element in either case. The third assumption is of similar flavor but the two target terms reside in the target group.

We remind the reader that, for all assumptions, we require the existence of a group generator algorithm $\mathcal{G}$ that gets a security parameter 1^λ as input and produces a description of a composite order bilinear group. That is, it outputs three primes p_1, p_2, p_3, two groups $\mathbb{G}$,$\mathbb{G}_T$ of order $N = p_1 p_2 p_3$, a map $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_T$ with the above properties, and three generators g_1, g_2, g_3 of subgroups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$, respectively. We also require that e is polynomial-time computable with respect to λ.

We will denote the three assumptions on the composite order groups by $\mathsf{Comp1}$, $\mathsf{Comp2}$, and $\mathsf{Comp2}$. They are the following:

The $\mathsf{Comp1}$ Assumption

Given $D = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_3)$ (notice that not all outputs of $\mathcal{G}$ are given), no PPT adversary has a non-negligible advantage in distinguishing

$$T_0 = g_1^z \text{ from } T_1 = g_1^z g_2^\nu,$$

where $z, \nu \overset{R}{\leftarrow} \mathbb{Z}_N$.

Definition 3.7. We say that the $\mathsf{Comp1}$ assumption holds if for all PPT attackers $\mathcal{A}$ it is true that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Comp1}}(\lambda) \leq \mathsf{negl}(\lambda)$.

The $\mathsf{Comp2}$ Assumption

Given $D = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_3, g_1^z g_2^\nu, g_2^\mu g_3^\rho)$, where $z, \nu, \mu, \rho \overset{R}{\leftarrow} \mathbb{Z}_N$, no PPT adversary has a non-negligible advantage in distinguishing

$$T_0 = g_1^w g_3^\sigma \text{ from } T_1 = g_1^w g_2^\kappa g_3^\sigma,$$

where $w, \kappa, \sigma \overset{R}{\leftarrow} \mathbb{Z}_N$.

Definition 3.8. We say that the $\mathsf{Comp2}$ assumption holds if for all PPT attackers $\mathcal{A}$ it is tru that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Comp2}}(\lambda) \leq \mathsf{negl}(\lambda)$.

The $\mathsf{Comp3}$ Assumption

Given $D = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_2, g_3, g_1^\alpha g_2^\nu, g_1^z g_2^\mu)$, where $\alpha, \nu, z, \mu \overset{R}{\leftarrow} \mathbb{Z}_N$, no PPT adversary has a non-negligible advantage in distinguishing

$$T_0 = e(g_1, g_1)^{\alpha z} \in \mathbb{G}_T \text{ from } T_1 \overset{R}{\leftarrow} \mathbb{G}_T.$$

Definition 3.9. We say that the $\mathsf{Comp3}$ assumption holds if for all PPT attackers $\mathcal{A}$ it is true that $\mathsf{Adv}_{\mathcal{A}}^{\mathsf{Comp2}}(\lambda) \leq \mathsf{negl}(\lambda)$.

3.3 Implementation of Bilinear Groups using Elliptic Curves

This section is a short introduction to the theory of elliptic curves and their use in implementing bilinear pairing groups. We give a high level overview of the elliptic curves over finite fields and the pairing operations on them. For more information the reader is referred to [14, 15, 108].

3.3.1 Elliptic Curves

An elliptic curve E *defined over a field* $\mathbb{F}$ contains the set of points of the equation

$$y^2 = x^3 + Ax + B$$

where A and B are constants in $\mathbb{F}$. The coordinates of the points can belong possibly to an extension field $\mathbb{L} \supseteq \mathbb{F}$. The set of points of the curve also includes a special point denoted by ∞. Formally, the elliptic curve group with points in the field $\mathbb{L}$ is defined as

$$E(\mathbb{L}) = \{\infty\} \cup \big\{(x, y) \in \mathbb{L} \times \mathbb{L} | y^2 = x^3 + Ax + B\big\}$$

A special "addition" operation is defined over the points of each elliptic curve. We will not go into the details or the definition of addition on elliptic curves, since we treat it as a black box operation with specific properties.

When the underlying field $\mathbb{L}$ is the field of the real numbers $\mathbb{R}$, the addition of two points $P_1, P_2 \in E(\mathbb{R})$ is defined as the symmetric point $P_3 = (x_3, y_3)$ with respect to the x-axis of the intersection point $-P_3 = (x_3, -y_3)$ of the straight line through P_1 and P_2 with the graph of the elliptic curve. The important property of any elliptic curve is that the points of $E(\mathbb{L})$ with the addition operation form an *abelian group* with ∞ as the identity element. For our constructions, we will denote this operation by the multiplicative sign $\cdot$ which is more common in the cryptographic community. For clarity with our multiplicative notation we denote the ∞ point as $\mathbb{1}, \mathbb{1}_{\mathbb{G}}$, etc. However in this section only we follow the additive notation for points of elliptic curves and the definition of pairings.

In cryptography we are interested only in elliptic curve groups with points in finite fields $\mathbb{F}_q$. These are fields of order $q = p^n$, where p is a prime number and n a positive integer. Since there is only a finite number of possible pairs $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$, we know that the number of points in $E(\mathbb{F}_q)$ is finite and therefore it is a finite abelian group. For finite elliptic curve groups there is no geometric intuition for the group operation but the formulas are the same as the case for $\mathbb{R}$.

3.3.2 Tate-Lichtenbaum Pairing over Finite Fields

In this section we introduce the modified Tate-Lichtenbaum pairing operation on elliptic curves of finite order. This is commonly used in applications of elliptic curves to cryptography, since it has several properties useful for simplifying the various implementations. However, the first pairing operation used for cryptographic application [21] was the Weil pairing.

Let $\mathbb{F}_q$ be a finite field of order q and E be an elliptic curve defined over $\mathbb{F}_q$. Let r be a positive integer coprime to q which divides the order of $E(\mathbb{F}_q)$. Let $\mu_r = \left\{x \in \overline{\mathbb{F}_q^*} | x^r = 1\right\}$, i.e. the set of the r-th roots of unity in the algebraic closure of $\mathbb{F}_q$. Then the field $\mathbb{K} = \mathbb{F}_q(\mu_r)$ is some finite extension $\mathbb{F}_{q^k}$. The number k is called the *embedding degree* of the elliptic curve and it is the smallest positive integer such that r divides $(q^k - 1)$.

Let $E(\mathbb{F}_{q^k})[r]$ be the set of points in $E(\mathbb{F}_{q^k})$ of order r. Formally,

$$E(\mathbb{F}_{q^k})[r] = \left\{P \in E(\mathbb{F}_{q^k}) | rP = \infty\right\}$$

The modified Tate-Lichtenbaum pairing is an efficiently computable binary operation

$$\tau_r : E(\mathbb{F}_{q^k})[r] \times E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k}) \to \mu_r$$

The first argument of the pairing is an element of the group $E(\mathbb{F}_{q^k})[r]$, while the second argument is an element of the group of the equivalence classes $E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$. Both groups have size r. The equivalence relation defining the classes of the second group acts on two points $P_1, P_2 \in E(\mathbb{F}_{q^k})$ and $P_1 \equiv P_2$ if and only if $P_1 - P_2 \in rE(\mathbb{F}_{q^k})$. The Tate-Lichtenbaum pairing satisfies the properties required in Sec. 3.1.1 adapted to the asymmetric group case, where $\mathbb{G}$, $\mathbb{H}$, $\mathbb{G}_T$ are cyclic groups and $\mathbb{G} \subseteq E(\mathbb{F}_{q^k})[r]$, $\mathbb{H} \subseteq E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$, and $\mathbb{G}_T \subseteq \mu_r$.

3.3.3 Supersingular Curves and Distortion Maps

An important class of elliptic curves with many applications in pairing based cryptography is the so called *supersingular curves.* An elliptic curve E over the field $\mathbb{F}_q$, where q is a power of a prime p, is called supersingular if the order of $E(\mathbb{F}_q)$ is congruent to 1 modulo p, i.e. $|E(\mathbb{F}_q)| \equiv 1 \pmod{p}$. For points on these curves there exists at least one efficiently computable endomorphism on E that maps a point of $E(\mathbb{F}_q)$ to a point in $E(\mathbb{F}_{q^k})$ called *distortion map.*

More specifically let P be a point in $E(\mathbb{F}_q)$ with prime order r. Therefore $P \in E(\mathbb{F}_{q^k})[r]$. Suppose the embedding degree k is greater than 1 and $E(\mathbb{F}_{q^k})$ has no points of order r^2. Let ϕ be an endomorphism of E. It can be proved that if $\phi(P) \notin E(\mathbb{F}_q)$, then $\tau_r(P, \phi(P)) \neq 1$.

The existence of the above distortion map ϕ allows as to consider symmetric bilinear groups where of order r. If we restrict the pairing to a single cyclic subgroup of the elliptic curve, it can be proved that $\tau_r(Q, \phi(P)) = \tau_r(P, \phi(Q))$ for any two points P, Q in the cyclic subgroup. As a result the pairing defined as $e(P,Q) := \tau_r(P, \phi(Q))$ is symmetric. In the Table 3.1 you can see the parameters and the distortion maps of the two types of supersingular elliptic curves used in this thesis.

3.3.4 Prime vs Composite Order Group Operations

In order to demonstrate the generic difference in the efficiency of prime order vs composite order implementations, we timed the group exponentiation

Curve	$\mathbb{F}_q$	k	$\lvert E(Fq) \rvert$	Distortion map
$y^2 = x^3 + a$	$q \equiv 2 \pmod 3$	2	$q+1$	$(x, y) \mapsto (\zeta_3 x, y)$
$y^2 = x^3 + x$	$q \equiv 3 \pmod 4$	2	$q+1$	$(x, y) \mapsto (-x, iy)$

Table 3.1: Two types of supersingular elliptic curves. These are curves over the field $\mathbb{F}_q$ where q satisfies the stated restrictions and k is the embedding degree. ζ_3 is a cubic root of unity in $\mathbb{F}_q$, while i is the imaginary element such that $i^2 = -1$ in $\mathbb{F}_q$.

(of a random group element with a random exponent) and pairing operations (on random group elements) in the MIRACL framework [31] for different security levels. The benchmarks were executed on a dual core Intel® Xeon® CPU W3503@2.40GHz with 2.0GB RAM running Ubuntu R10.04. The elliptic curve utilized for all benchmarks was the super-singular (symmetric) curve $y^2 = x^3 + 1 \bmod p$ with embedding degree 2 for suitable primes p.

In table 3.2 we can see the significant gap between the timings in prime and composite order groups for the same security levels. This is the main reason that we tried to utilize prime order groups for our construction.

Group exponentiation

Security Level (Bits)	Prime	Prod. of 2 primes	Prod. of 3 primes
80	3.5	66.9	201.6
112	14.8	448.1	1404.3
128	34.4	1402.5	4512.5
192	273.8	20097.0	66526.0

Pairing

Security Level (Bits)	Prime	Prod. of 2 primes	Prod. of 3 primes
80	13.9	245.3	762.3
112	65.7	1706.8	5485.2
128	176.6	5428.2	17494.4
192	1752.3	79046.8	263538.1

Table 3.2: Average timing results in milliseconds over 100 repeats of group exponentiations and pairings in MIRACL.

CHAPTER 4

Identity and Attribute-Based Encryption Systems

In this chapter we present the formal definitions for identity and attribute-based encryption systems. The former serve as an essential stepping stone toward the latter, as it was the case for our leakage-resilient ABE construction of Chap. 5.

4.1 IBE Definition

An Identity-Based Encryption (IBE) system (first introduced in [100]) is a public key cryptosystem which allows users to encrypt knowing only the recipient's identity and some public parameters of the systems (this means that individual public keys are not needed). Formally, an IBE scheme consists of four PPT algorithms. In order to allow leakage on many master keys, we extend the functionality of the usual key generation algorithm by allowing it to take the empty string, denoted by ϵ, as input.

$\mathsf{Setup}(1^\lambda) \to (\mathrm{PP}, \mathrm{MSK})$ The setup algorithm takes an integer security parameter, λ, as input and outputs the public parameters, PP, and the original master key, MSK. The remaining algorithms take implicitly the security parameter and the public parameters as inputs. The security parameter is encoded in unary, so that all algorithms run in polynomial time in λ.

$\mathsf{KeyGen}(\mathrm{MSK}', X) \to K$ The key generation algorithm takes in a master key, MSK', and either $X = \mathcal{ID}$, an identity, or $X = \epsilon$, the empty string[1]. In the former case, it outputs a secret key, $K = \mathrm{SK}$, for the identity $\mathcal{ID}$. In the latter case, it outputs another master key, $K = \mathrm{MSK}''$, such that $\|\mathrm{MSK}''\| = \|\mathrm{MSK}'\|$ [2]. This new master key can now be used instead of the original key in calls of **Keygen** (either with $\mathcal{ID}$ or with ϵ as input).

$\mathsf{Encrypt}(M, \mathcal{ID}) \to \mathrm{CT}$ The encryption algorithm takes in a message, M, and an identity, $\mathcal{ID}$, and outputs a ciphertext, CT.

$\mathsf{Decrypt}(\mathrm{CT}, \mathrm{SK}) \to M$ The decryption algorithm takes in a ciphertext, CT, and a secret key, SK. It outputs a message M.

The correctness requirement is that if the identity $\mathcal{ID}$ used during encryption is the same as the identity of the secret key used during decryption, then the output of Decrypt is the encrypted message M. That is, for all

[1]This is not the standard definition of KeyGen in IBE systems. We augmented it to accept the empty string in order to work as an update algorithm for the master key and eventually achieve security in the Continual Leakage Model (see Section).

[2]This restriction prevents expansion of the master key.

MSK, PP generated by a call to Setup for all master keys MSK' generated by applying the KeyGen algorithm with the empty string and a previously generated master key, and for all $M, \mathcal{ID}$

$$\mathsf{Decrypt}(\mathsf{Encrypt}(M, \mathcal{ID}), \mathsf{KeyGen}(\mathrm{MSK}', \mathcal{ID})) = M$$

4.1.1 Dual System IBE

As noted in Sec. 2.3 a dual system encryption system admits semi-functional secret keys and ciphertexts along with the normal components. Therefore it consists of two additional algorithms:

$\mathsf{KeyGenSf}(\mathrm{MSK}', X) \rightarrow \tilde{K}$ The semi-functional key generation algorithm takes in a *normal* master key, MSK', and either $X = \mathcal{ID}$, an identity, or $X = \epsilon$, the empty string. In the former case, it outputs a semi-functional secret key, $\tilde{K} = \tilde{\mathrm{SK}}$, for the identity $\mathcal{ID}$. In the latter case, it outputs a semi-functional master key, $\tilde{K} = \tilde{\mathrm{MSK}}''$. Notice that the master key used as input to the algorithm is always normal.

$\mathsf{EncryptSf}(M, \mathcal{ID}) \rightarrow \tilde{\mathrm{CT}}$ The encryption algorithm takes in a message, M, and an identity, $\mathcal{ID}$, and outputs a semi-functional ciphertext, CT.

4.2 ABE Definitions

In this section we present the formal definitions of the different notions of attribute-based encryption systems. For simplicity of notation, we do not include the security parameter λ as input to the algorithms, except Setup and

GlobalSetup. We assume that it is implicitly included in the (global) public parameters.

4.2.1 Ciphertext-Policy ABE

A ciphertext-policy attribute-based encryption scheme consists of the following four PPT algorithms:

$\mathsf{Setup}\left(1^\lambda, \mathcal{U}\right) \overset{R}{\to} (\mathrm{PP}, \mathrm{MSK})$**:** The Setup algorithm takes as inputs the security parameter $\lambda \in \mathbb{N}$ encoded in unary and a description of the attribute universe $\mathcal{U}$ [3]. It outputs the public parameters PP and the master secret key MSK.

$\mathsf{KeyGen}\,(\mathrm{PP}, \mathrm{MSK}, \mathcal{S}) \overset{R}{\to} \mathrm{SK}$**:** The key generation algorithm takes as inputs the public parameters PP, the master secret key MSK and a set of attributes $\mathcal{S} \subseteq \mathcal{U}$. The algorithm generates a secret key corresponding to $\mathcal{S}$.

$\mathsf{Encrypt}\,(\mathrm{PP}, M, \mathbb{A}) \overset{R}{\to} \mathrm{CT}$**:** The encryption algorithm takes as inputs the public parameters PP, a plaintext message M, and an access structure $\mathbb{A}$ on $\mathcal{U}$. It outputs the ciphertext CT.

$\mathsf{Decrypt}\,(\mathrm{PP}, \mathrm{SK}, \mathrm{CT}) \to \{M, \perp\}$**:** The *deterministic* decryption algorithm takes as inputs the public parameters PP, a secret key SK, and a ciphertext CT. It outputs either the plaintext M, when the collection of attributes satisfies the access structure of the ciphertext, or $\perp$ when decryption fails.

[3]See Sec. 4.2.4 for more information on the description of the attribute universe.

Correctness: We require that a CP-ABE scheme is correct, i.e the decryption algorithm correctly decrypts a ciphertext of an access structure $\mathbb{A}$ with a secret key on $\mathcal{S}$when $\mathcal{S}$ is an authorized set of $\mathbb{A}$. Formally:

Definition 4.1. A CP-ABE scheme is *correct* when for all messages M, and all attribute sets $\mathcal{S}$ and access structures $\mathbb{A}$ with $\mathcal{S} \in \mathbb{A}$ (i.e. for $\mathcal{S}$ authorized):

$$\Pr\left[\mathsf{Decrypt}\left(\mathrm{PP}, \mathrm{SK}, \mathrm{CT}\right) \neq M \;\middle|\; \begin{array}{c} (\mathrm{PP}, \mathrm{MSK}) \overset{R}{\leftarrow} \mathsf{Setup}\left(1^{\lambda}\right) \\ \mathrm{SK} \overset{R}{\leftarrow} \mathsf{KeyGen}\left(\mathrm{PP}, \mathrm{MSK}, \mathcal{S}\right) \\ \mathrm{CT} \overset{R}{\leftarrow} \mathsf{Encrypt}\left(\mathrm{PP}, M, \mathbb{A}\right) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over all random coins of all algorithms.

As noted in Sec. 2.3, in case the ABE scheme is of the dual system encryption type it also consists of the following two semi functional algorithms:

$\mathsf{KeyGenSf}\left(\mathrm{PP}, \mathrm{MSK}, \mathcal{S}\right) \overset{R}{\rightarrow} \mathrm{SK}$**:** The key generation algorithm takes as inputs the public parameters PP, a *normal* master secret key MSK and a set of attributes $\mathcal{S} \subseteq \mathcal{U}$. The algorithm generates a *semi-functional* secret key corresponding to $\mathcal{S}$.

$\mathsf{EncryptSf}\left(\mathrm{PP}, M, \mathbb{A}\right) \overset{R}{\rightarrow} \mathrm{CT}$**:** The encryption algorithm takes as inputs the public parameters PP, a plaintext message M, and an access structure $\mathbb{A}$ on $\mathcal{U}$. It outputs the *semi-functional* ciphertext CT.

4.2.2 Key-Policy ABE

The only difference between a key-policy and a ciphertext-policy ABE scheme is that the roles of the keys and the ciphertexts with respect to the policies and the attribute sets are reversed. In the KP-ABE case each key is tied to a policy instead of an attribute set. A key-policy attribute-based encryption scheme consists of the following four PPT algorithms:

$\mathsf{Setup}(1^\lambda, \mathcal{U}) \overset{R}{\rightarrow} (\mathrm{PP}, \mathrm{MSK})$**:** The Setup algorithm takes as inputs the security parameter $\lambda \in \mathbb{N}$ encoded in unary and a description of the attribute universe $\mathcal{U}$ [3]. It outputs the public parameters PP and the master secret key MSK.

$\mathsf{KeyGen}(\mathrm{PP}, \mathrm{MSK}, \mathbb{A}) \overset{R}{\rightarrow} \mathrm{SK}$**:** The key generation algorithm takes as inputs the public parameters PP, the master secret key MSK and an access structure $\mathbb{A}$ on $\mathcal{U}$. The algorithm generates a secret key corresponding to $\mathbb{A}$.

$\mathsf{Encrypt}(\mathrm{PP}, M, \mathcal{S}) \overset{R}{\rightarrow} \mathrm{CT}$**:** The encryption algorithm takes as inputs the public parameters PP, a plaintext message M, and a set of attributes $\mathcal{S} \subseteq \mathcal{U}$. It outputs the ciphertext CT.

$\mathsf{Decrypt}(\mathrm{PP}, \mathrm{SK}, \mathrm{CT}) \rightarrow \{M, \perp\}$**:** The *deterministic* decryption algorithm takes as inputs the public parameters PP, a secret key SK, and a ciphertext CT. It outputs either the plaintext M, when the collection of attributes satisfies the access structure of the ciphertext, or $\perp$ when decryption fails.

Correctness: We require that a KP-ABE scheme is correct, i.e the decryption algorithm correctly decrypts a ciphertext on $\mathcal{S}$ with a secret key of an access structure $\mathbb{A}$ when $\mathcal{S}$ is an authorized set of $\mathbb{A}$. Formally:

Definition 4.2. A KP-ABE scheme is *correct* when for all messages M, and all attribute sets $\mathcal{S}$ and access structures $\mathbb{A}$ with $\mathcal{S} \in \mathbb{A}$ (i.e. for $\mathcal{S}$ authorized):

$$\Pr\left[\mathsf{Decrypt}(\mathrm{PP}, \mathrm{SK}, \mathrm{CT}) \neq M \;\middle|\; \begin{array}{c} (\mathrm{PP}, \mathrm{MSK}) \overset{R}{\leftarrow} \mathsf{Setup}(1^\lambda) \\ \mathrm{SK} \overset{R}{\leftarrow} \mathsf{KeyGen}(\mathrm{PP}, \mathrm{MSK}, \mathbb{A}) \\ \mathrm{CT} \overset{R}{\leftarrow} \mathsf{Encrypt}(\mathrm{PP}, M, \mathcal{S}) \end{array}\right] \leq \mathsf{negl}\lambda$$

where the probability is taken over all random coins of all algorithms.

4.2.3 Multi-Authority ABE

A multi-authority ciphertext-policy attribute-based encryption system consists of the following five algorithms:

$\mathsf{GlobalSetup}(1^\lambda, \mathcal{U}, \mathcal{U}_\Theta) \overset{R}{\rightarrow} \mathrm{GP}$**:** The global setup algorithm takes as inputs the security parameter $\lambda \in \mathbb{N}$ encoded in unary, a description of the attribute universe $\mathcal{U}$, and a description on the authority universe $\mathcal{U}_\Theta$. It outputs the public global parameters for the system.

We assume that the description of a function $\mathsf{T} : \mathcal{U} \rightarrow \mathcal{U}_\Theta$ is included in the global parameters. This function maps each attribute to the *unique* authority it belongs to.

$\mathsf{AuthSetup}(\mathrm{GP}, \theta) \overset{R}{\rightarrow} (\mathrm{PK}_\theta, \mathrm{SK}_\theta)$**:** The authority $\theta \in \mathcal{U}_\Theta$ calls the authority setup algorithm during its initialization with the global parameters GP as input and receives its public / secret key pair $(\mathrm{PK}_\theta, \mathrm{SK}_\theta)$. Each authority is supposed to call this algorithm during its initialization, store the authority secret key SK_θ, and publish the public key PK_θ.

$\mathsf{KeyGen}(\mathrm{GP}, \mathcal{GID}, \mathrm{SK}_\theta, u) \overset{R}{\rightarrow} \mathrm{K}_{\mathcal{GID},u}$**:** The key generation algorithm takes in the global identifier $\mathcal{GID}$ of a user, the secret key of the authority

θ, an attribute u controlled by this authority, and the global parameters. It outputs a key for the identity - attribute pair $(\mathcal{GID}, u)$.

$\mathsf{Encrypt}(\mathrm{GP}, M, \mathbb{A}, \{\mathrm{PK}_\theta\}_{\theta \in \mathcal{C}_\Theta}) \xrightarrow{R} \mathrm{CT}$**:** The encryption takes in a message M, an access structure $\mathbb{A}$, a set of public keys $\{\mathrm{PK}_\theta\}_{\theta \in \mathcal{C}_\Theta}$, and the global parameters. The set $\mathcal{C}_\Theta$ is the set of the authorities relevant to the access structure $\mathbb{A}$(See Sec. 2.1.2 for a description of the relevant authorities). The algorithm outputs the ciphertext CT.

$\mathsf{Decrypt}(\mathrm{GP}, \{\mathrm{K}_{\mathcal{GID},u}\}_{u \in \mathcal{S}}, \mathrm{CT}) \to \{M, \perp\}$**:** The *deterministic* decryption algorithm takes in a ciphertext CT, the set of keys of a single user $\mathcal{GID}$ corresponding to different attributes u, and the global parameters. It outputs either the message M, when the collection of attributes satisfies the access structure of the ciphertext, or $\perp$ when decryption fails.

Correctness: We require that a MA-CP-ABE scheme is correct, i.e. the decryption algorithm correctly decrypts a ciphertext on $\mathbb{A}$ with a secret key of the attribute set $\mathcal{S}$, when $\mathcal{S}$ is an authorized set of $\mathbb{A}$. All the public keys on the relevant authorities of the access structure $\mathbb{A}$ and the secret keys of the attribute set $\mathcal{S}$ should have been created by the $\mathsf{AuthSetup}$ algorithm. For simplicity of the definition we include the entire universe of authorities. Formally:

Definition 4.3. A MA-CP-ABE scheme is *correct* when for any user $\mathcal{GID}$, for all messages M, and all attribute sets $\mathcal{S}$ and access structures $\mathbb{A}$ with $\mathcal{S} \in \mathbb{A}$

(i.e. for $\mathcal{S}$ authorized):

$$\Pr\left[\begin{array}{c|c} \begin{array}{c}\mathsf{Decrypt}(\mathrm{GP},\\ \{\mathrm{K}_{\mathcal{GID},u}\}_{u\in\mathcal{S}}, \mathrm{CT}) \neq M\end{array} & \begin{array}{c}\mathrm{GP} \xleftarrow{R} \mathsf{GlobalSetup}(1^\lambda)\\ \left\{(\mathrm{PK}_\theta, \mathrm{SK}_\theta) \xleftarrow{R} \mathsf{AuthSetup}(\mathrm{GP}, \theta)\right\}_{\theta\in\mathcal{U}_\Theta}\\ \left\{\mathrm{K}_{\mathcal{GID},u} \xleftarrow{R} \mathsf{KeyGen}\left(\mathrm{GP}, \mathcal{GID}, \mathrm{SK}_{\mathsf{T}(u)}, u\right)\right\}_{u\in\mathcal{S}}\\ \mathrm{CT} \xleftarrow{R} \mathsf{Encrypt}\left(\mathrm{GP}, M, \mathbb{A}, \{\mathrm{PK}_\theta\}_{\theta\in\mathcal{U}_\Theta}\right)\end{array}\end{array}\right] \leq \mathsf{negl}(\lambda)$$

where the probability is taken over all random coins of all algorithms.

4.2.4 Small Universe, Large Universe, and Unbounded Constructions

Orthogonal to the above definitions, we characterize the ABE schemes with respect to the attribute universe size with the following properties:

Small Universe Constructions In these systems, the size of the attribute universe $\mathcal{U}$ is polynomial in the security parameter λ. Typically for construction of this type, the number of group elements of the public parameters, secret keys, and/or ciphertexts depends polynomially on $|\mathcal{U}|$. As a result, the designer of these systems has to decide during setup on a trade off between the expressiveness of the different policies (the bigger the universe, the more expressive the policies) and the efficiency of the implementation (the larger the universe, the less efficient scheme).

Large Universe Constructions In these systems, the size of the attribute universe is exponential in the security parameter λ, while the size of the public parameters, secret keys, and ciphertexts is polynomial λ. This is obviously a desired feature, since in practice any possible attribute can be used in the policies of these systems. For example, the attribute universe can be the set of character strings of up to 128 characters.

An extra property that can be applied to both of the above notions is the following:

Unbounded Constructions In unbounded constructions the length of the public parameters, keys and ciphertexts depends only on λ and is independent of the size of the attribute universe $|\mathcal{U}|$.

According to the above definitions, a large universe ABE scheme is also unbounded, while a small universe ABE scheme can be either bounded or unbounded. Actually, in several works [74] the term unbounded implied a large universe construction. The only small universe and unbounded ABE is the construction of Okamoto and Takashima [88], where all the encryption components are of constant length, but the security proof requires the size of the attribute universe to be polynomial in the security parameter. In this case, the trade off is happening between expressiveness and proven security of the implementation.

A common approach to construct large universe schemes from small universe ones is to use a hash function that maps attributes from an exponential sized set to another exponential sized set. However, in this case the hash function has to be modeled as a random oracle in the security proof, which implies weaker security guarantees.

With respect to the inputs to the Setup algorithm of different schemes, the small universe constructions take as input (a description of) the entire universe $\mathcal{U}$, while for the large universe constructions the attribute universe depends typically on the security parameter only. In these cases, the Setup algorithm takes as input only 1^λ. Finally, for the multi-authority setting, we note that the universe of authorities is considered to be polynomially bounded (although this is not necessary for our security proof in Sec. 6.3), since we assume that at most a polynomial number of public authorities is going to be deployed at any given time.

Our constructions of Chap. 6 are large-universe constructions secure under various security notions, and more specifically the attribute universe is $\mathbb{Z}_p^*$ where p is a prime number with $\|p\| = \lambda$. Since practicality was one of our main goals in Chap. 6 we required our constructions to be large universe in order to achieve maximum expressiveness. On the other hand, the robust construction of Chap. 5 is a small-universe (bounded) construction. That is, the size of the public parameters depends linearly of the size of the attribute universe, set during the Setup algorithm.

4.2.5 Dual System CP-ABE

A dual system CP-ABE system consists of the additional two algorithms:

$\mathsf{KeyGenSf}\,(\mathrm{PP}, \mathrm{MSK}, \mathcal{S}) \xrightarrow{R} \widetilde{\mathrm{SK}}$**:** The key generation algorithm takes as inputs the public parameters PP, the master secret key MSK and a set of attributes $\mathcal{S} \subseteq \mathcal{U}$. The algorithm generates a semi-functional secret key corresponding to $\mathcal{S}$.

$\mathsf{EncryptSf}\,(\mathrm{PP}, M, \mathbb{A}) \overset{R}{\to} \tilde{\mathrm{CT}}$**:** The encryption algorithm takes as inputs the public parameters PP, a plaintext message M, and an access structure $\mathbb{A}$ on $\mathcal{U}$. It outputs the semi-functional ciphertext $\tilde{\mathrm{CT}}$.

4.3 Security Notions

In this section we present the various security notions under which our constructions are secure. All security notions are defined via security games between a challenger $\mathcal{C}$ and an attacker $\mathcal{A}$. In all games the attacker outputs a pair of messages M_0 and M_1 and receives an encryption of M_b, where $b \overset{R}{\leftarrow} \{0,1\}$ is a random bit picked by the challenger. The goal of the attacker is to guess which message was encrypted. Its guess/output is a bit $b' \in \{0,1\}$ and its success is measured by the advantage defined as

$$\mathsf{Adv}_{\mathcal{A}}(\lambda) := |2 \cdot \Pr[b' = b] - 1| = |\Pr[b' = 1 | b = 1] - \Pr[b' = 1 | b = 0]|$$

4.3.1 CP-ABE Security Notions

In this section we present the definition of adaptive (or full) security for CP-ABE schemes. This is described by a game between a challenger and an attacker and is parameterized by the security parameter $\lambda \in \mathbb{N}$. The phases of the *adaptive security game* are the following:

Setup: The challenger calls the $\mathsf{Setup}(1^\lambda)$ algorithm and sends the public parameters PP to the attacker.

Query Phase 1: In this phase the attacker can adaptively ask for secret keys for the sets of attributes $\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_{Q_1}$. For each $\mathcal{S}_i$ the challenger calls $\mathsf{KeyGen}(\mathrm{MSK}, \mathcal{S}_i) \to \mathrm{SK}_i$ and sends SK_i to the attacker.

Challenge: The attacker declares an access structure $\mathbb{A}^*$ and two equal-length plaintexts M_0 and M_1. The challenger receives $(\mathbb{A}^*, M_0, M_1)$ and flips a random coin $b \in \{0, 1\}$. After calling $\mathsf{Encrypt}(M_b, \mathbb{A}^*) \to \mathrm{CT}$, he sends CT to the attacker.

The restriction that has to be satisfied by the access structure $\mathbb{A}^*$ is that none of the queried sets in Phase 1 satisfies it, i.e. $\forall i \in [Q_1] : \mathcal{S}_i \notin \mathbb{A}^*$.

Query Phase 2: This the same as query phase 1. The attacker asks for the secret key for the sets $\mathcal{S}_{Q_1+1}, \mathcal{S}_{Q_1+2}, \ldots, \mathcal{S}_Q$, for which the same restriction holds: $\forall i \in [Q] : \mathcal{S}_i \notin \mathbb{A}^*$.

Guess: The attacker outputs his guess $b' \in \{0, 1\}$ for b.

Definition 4.4. A CP-ABE scheme is *adaptively secure* if all PPT attackers have at most a negligible advantage in λ in the adaptive security game.

A weaker notion of security for CP-ABE scheme is when the attacker declares the challenge access structure at the beginning of the game. That is, the *selective security game* is the same as the adaptive security game except that the Setup phase is preceded by the following phase:

Initialization: In this phase the attacker declares the challenge access structure $\mathbb{A}^*$, which he will try to attack, and sends it to the challenger.

As before all secret key queries should satisfy the same restriction: $\forall i \in [Q] : \mathcal{S}_i \notin \mathbb{A}^*$.

Definition 4.5. A CP-ABE scheme is *selectively secure* if all PPT attackers have at most a negligible advantage in λ in the selective security game.

4.3.2 KP-ABE Security Notions

Similarly, the phases of the *adaptive security game* for KP-ABE schemes are the following:

Setup: Here the challenger calls the $\mathsf{Setup}(1^\lambda)$ algorithm and sends the public parameters PP to the attacker.

Query Phase 1: In this phase the attacker can adaptively ask for secret keys for the access structures $\mathbb{A}_1, \mathbb{A}_2, \ldots, \mathbb{A}_{Q_1}$. For each $\mathbb{A}_i$ the challenger calls $\mathsf{KeyGen}(\mathrm{MSK}, \mathbb{A}_i) \rightarrow \mathrm{SK}_i$ and sends SK_i to the attacker.

Challenge: The attacker declares an attribute set $\mathcal{S}^* \subseteq \mathcal{U}$ and two equal-length plaintexts M_0 and M_1. The challenger receives $(\mathcal{S}^*, M_0, M_1)$ and flips a random coin $b \in \{0, 1\}$. After calling $\mathsf{Encrypt}(M_b, \mathcal{S}^*) \rightarrow \mathrm{CT}$, he sends CT to the attacker.

The restriction that has to be satisfied by the challenge set $\mathcal{S}^*$ is that none of the queried policies is satisfied by it, i.e. $\forall i \in [Q_1] : \mathcal{S}^* \notin \mathbb{A}_i$.

Query Phase 2: This the same as query phase 1. The attacker asks for the secret key for the access structures $\mathbb{A}_{Q_1+1}, \mathbb{A}_{Q_1+2}, \ldots, \mathbb{A}_Q$, for which the same restriction holds: $\forall i \in [Q] : \mathcal{S}^* \notin \mathbb{A}_i$.

Guess: The attacker outputs his guess $b' \in \{0,1\}$ for b.

Definition 4.6. A KP-ABE scheme is *adaptively secure* if all PPT attackers have at most a negligible advantage in λ in the adaptive security game, where the advantage of an attacker is defined as $\mathsf{Adv} = \Pr[b' = b] - 1/2$.

As before, we define the weaker notion of security via the *selective security game*. This is the same as the adaptive security game except that the Setup phase is preceded by the following phase:

Initialization: In this phase the attacker declares the challenge attribute set $\mathcal{S}^*$, which he will try to attack, and sends it to the challenger.

All secret key queries should satisfy the restriction: $\forall i \in [Q] : \mathcal{S}^* \notin \mathbb{A}_i^*$.

Definition 4.7. A KP-ABE scheme is *selectively secure* if all PPT attackers have at most a negligible advantage in λ in the selective security game.

4.3.3 Multi-Authority Static Security

In this section we will define the static (or non-adaptive) security game between a challenger and an attacker. The difference between this security game and the adaptive one is that all queries done by the attacker are sent to the challenger immediately after seeing the public parameters. As usual, we also allow the attacker to corrupt a certain set of authorities that he can control. These authorities are chosen by the attacker after seeing the global parameters and remain the same until the end of the game.

The *static security game* is parameterized by the security parameter λ and consists of the following phases:

Global Setup: The challenger calls $\mathsf{GlobalSetup}(1^\lambda) \to \mathrm{GP}$ and gives the global parameters GP to the attacker.

Attacker's Queries: Then the attacker responds with ($\mathcal{C}_\Theta$, $\mathcal{N}_\Theta$, $\mathcal{Q}$, (M_0, M_1), $\mathbb{A}$) where:

- A set $\mathcal{C}_\Theta \subseteq \mathcal{U}_\Theta$ of corrupt authorities and their respective public keys $\{\mathrm{PK}_\theta\}_{\theta \in \mathcal{C}_\Theta}$, which he might have created in a malicious way[4].
- A set $\mathcal{N}_\Theta \subseteq \mathcal{U}_\Theta$ of the non-corrupt authorities for which the adversary requests the public keys. Obviously, it should be disjoint from the set of corrupt authorities.
- A sequence $\mathcal{Q} = \{(\mathcal{GID}_i, \mathcal{S}_i)\}_{i=1}^m$ of the secret key queries, where the global identities $\mathcal{GID}_i$ are distinct and $\mathcal{S}_i \subseteq \mathcal{U}$ with $\mathsf{T}(S_i) \cap \mathcal{C}_\Theta = \emptyset$.

  A pair $(\mathcal{GID}_i, \mathcal{S}_i)$ in this sequence denotes that the attacker requests the secret keys for the user $\mathcal{GID}_i$ with attributes from the set S_i. That is, for every $u \in \mathcal{S}_i$ the attacker gets a key $\mathrm{K}_{\mathcal{GID}_i,u} \leftarrow \mathsf{KeyGen}(\mathcal{GID}_i$, $\mathrm{SK}_{\mathsf{T}(u)}$, u, GP). According to the restriction $\mathsf{T}(S_i) \cap \mathcal{C}_\Theta = \emptyset$, none of these keys come from a corrupt authority.
- Two messages M_0, M_1 of equal length, and a challenge access structure $\mathbb{A}$ encoded in a suitable form. We require that for every $i \in [m]$ the

[4]The only requirement is that they have the correct type.

set $\mathcal{S}_i \cup \bigcup_{\theta \in \mathcal{C}_\Theta} \mathsf{T}^{-1}(\theta)$ is a unauthorized set of the access structure $\mathbb{A}$, where $\bigcup_{\theta \in \mathcal{C}_\Theta} \mathsf{T}^{-1}(\theta)$ is the set of all the attributes belonging to corrupt authorities. This way, the attacker will not be able to trivially win the game by decrypting the challenge ciphertext with a secret key given to him augmented with the key components from the corrupt authorities.

Challenger's Replies: The challenger flips a random coin $b \xleftarrow{R} \{0, 1\}$ and replies with:

- The public keys $\mathrm{PK}_\theta \leftarrow \mathsf{AuthSetup}(\mathrm{GP}, \theta)$ for all $\theta \in \mathcal{N}_\Theta$.
- The secret keys $\mathrm{K}_{\mathcal{GID}_i,u} \leftarrow \mathsf{KeyGen}(\mathcal{GID}_i, \mathrm{SK}_{\mathsf{T}(u)}, u, \mathrm{GP})$ for all $i \in [m]$ and for all $u \in \mathcal{S}_i$.
- The challenge ciphertext $\mathrm{CT}^* \leftarrow \mathsf{Encrypt}(M_b,\ \mathbb{A},\ \{\mathrm{PK}_\theta\},\ \mathrm{GP})$ where $\{\mathrm{PK}_\theta\}$ is the set of all authority public keys (corrupt and non corrupt).

Guess: The attacker outputs a guess $b' \in \{0, 1\}$.

Definition 4.8. A MA-CP-ABE scheme is *statically secure* if all PPT attackers have at most a negligible advantage in λ in the static security game.

4.3.4 IBE Leakage-Resilient Adaptive Security

The security of our IBE system is based on a game, called MasterLeak. It is a modified version of the usual IbeCpa security game. In that game, the attacker can make a polynomial number of KeyGen queries for identities other than the challenge identity. Each of these queries returns a secret key of the requested identity. The main idea of our security game is to allow these queries and in addition allow leakage on the master key and secret keys of the challenge identity. The only restriction we impose is that it can not get leakage of more than ℓ_{MSK} bits per master key (remember we can have many master keys) and ℓ_{SK} bits per secret key, where $\ell_{\text{MSK}}, \ell_{\text{SK}}$ are parameters of the game.

The game starts with a setup phase, where the challenger runs the setup algorithm and gives the attacker the public parameters. It also gives the attacker a handle (i.e. reference) to the master key. We now allow the attacker to make three kinds of queries, called **Create**, **Leak** and **Reveal**. With a **Create** query, the attacker asks the challenger to create a key and store it. The attacker supplies a handle that refers to a master key to be used in the key generation algorithm. Each such query returns a unique handle-reference to the generated key, so that the attacker can refer to it later and either apply a leakage function to it and/or ask for the entire key. The original master key (the one created in the Setup algorithm) gets a handle of 0.

Using a handle, the attacker can make a leakage query **Leak** on any key of its choice. Since all queries are adaptive (the attacker has the ability to leak from each key a few bits at the time, instead of requiring the leakage to occur all at once) and the total amount of leakage allowed is bounded, the challenger has to keep track of all keys leaked via these queries and the number of leaked bits from each key so far. Thus, it creates a set $\mathcal{T}$ that holds tuples of handles, identities, keys, and the number of leaked bits. Each **Create** query adds a tuple to this set and each **Leak** query updates the number of bits leaked.

The **Reveal** queries allow the attacker to get access to an entire secret key. They get as input a handle to a key and the challenger returns this secret key to the attacker. The obvious restriction is that the attacker can not get a master key, since it would trivially break the system. For the same reason, no key for the challenge identity should be revealed and thus the challenger has to have another set to keep track of the revealed identities. We will denote this set by $\mathcal{R}$. We also note that the **Reveal** queries model the attacker's ability to "change its mind" in the middle of the game on the challenge identity. Maybe the attacker, after getting leakage from a secret key, decides that it is better to get the entire key via a **Reveal** query. Thus we achieve the maximum level of adaptiveness.

We now define our game formally. The security game is parameterized by a security parameter λ and two leakage bounds $\ell_{\text{MSK}} = \ell_{\text{MSK}}(\lambda)$, $\ell_{\text{SK}} = \ell_{\text{SK}}(\lambda)$. The master keys', secret keys' and identities' spaces are denoted by $\mathcal{MK}$, $\mathcal{SK}$, and $\mathcal{I}$, respectively. We assume that the handles' space is $\mathcal{H} = \mathbb{N}$. The game $\mathsf{MasterLeak}$ consists of the following phases:

Setup: The challenger makes a call to $\mathsf{Setup}(1^\lambda)$ and gets a master key MSK and the public parameters PP. It gives PP to the attacker. Also, it sets $\mathcal{R} = \emptyset$ and $\mathcal{T} = \{(0, \epsilon, \text{MSK}, 0)\}$. Remember that $\mathcal{R} \subseteq \mathcal{I}$ and $\mathcal{T} \subseteq \mathcal{H} \times \mathcal{I} \times (\mathcal{MK} \cup \mathcal{SK}) \times \mathbb{N}$ (handles - identities - keys - leaked bits). Thus initially the set $\mathcal{T}$ holds a record of the original master key (no identity for it and no leakage so far). Also a handle counter H is set to 0.

Phase 1: In this phase, the adversary can make the following queries to the challenger. All of them can be interleaved in any possible way and the input of a query can depend on the outputs of all previous queries (adaptive security).

- **Create**(h, X): h is a handle to a tuple of $\mathcal{T}$ that must refer to a master key and X can be either an identity $\mathcal{ID}$ or the empty string ϵ.

  The challenger initially scans $\mathcal{T}$ to find the tuple with handle h. If the identity part of the tuple is not ϵ, which means that the tuple holds a secret key of some identity, or if the handle does not exist, it responds with $\perp$.

  Otherwise, the tuple is of the form $(h, \epsilon, \mathrm{MSK}', L)$. Then the challenger makes a call to $\mathsf{KeyGen}(\mathrm{MSK}', X) \rightarrow K$ and adds the tuple $(H + 1, X, K, 0)$ to the set $\mathcal{T}$. K is either a secret key for identity $\mathcal{ID}$ or another master key. After that, it updates the handle counter to $H \leftarrow H + 1$.

- **Leak**(h, f): In this query, the adversary requests leakage from a key that has handle $h \in \mathbb{N}$ with a polynomial-time computable function f of constant output size[5] acting on the set of keys.

[5]We apply this restriction so that the adversary does not get any "extra" information about the input; only the output bits of the function. This restriction is also present in other works (e.g. in [28] they use circuits as leakage functions).

The challenger scans $\mathcal{T}$ to find the tuple with the specified handle. It is either of the form $(h, \mathcal{ID}, \mathrm{SK}, L)$ or $(h, \epsilon, \mathrm{MSK}', L)$ [6].

In the first case, it checks if $L + \|f(\mathrm{SK})\| \leq \ell_{\mathrm{SK}}$. If this is true, it responds with $f(\mathrm{SK})$ and updates the L in the tuple with $L + \|f(\mathrm{SK})\|$. If the checks fails, it returns $\perp$ to the adversary.

If the tuple holds a master key MSK', it checks if $L + \|f(\mathrm{MSK}')\| \leq \ell_{\mathrm{MSK}}$. If this is true, it responds with $f(\mathrm{MSK}')$ and updates the L with $L + \|f(\mathrm{MSK}')\|$. If the checks fails, it returns $\perp$ to the adversary.

- **Reveal**(h): Now the adversary requests the entire key with handle h. The challenger scans $\mathcal{T}$ to find the requested entry. If the handle refers to a master key tuple, then the challenger returns $\perp$. Otherwise, we denote the tuple by $(h, \mathcal{ID}, \mathrm{SK}, L)$. The challenger responds with SK and adds the identity $\mathcal{ID}$ to the set $\mathcal{R}$.

Challenge: The adversary submits a challenge identity $\mathcal{ID}^* \notin \mathcal{R}$ and two messages M_0, M_1 of equal size. The challenger flips a uniform coin $c \overset{R}{\leftarrow} \{0,1\}$ and encrypts M_c under $\mathcal{ID}^*$ with a call to $\mathsf{Encrypt}(M_c, \mathcal{ID}^*)$. It sends the resulting ciphertext CT^* to the adversary.

Phase 2: This is the same as **Phase 1** with the restriction that the only queries allowed are **Create** and **Reveal** queries that involve a (non-master) secret key with identity different than $\mathcal{ID}^*$. The reason for forbidding

[6]It can be the case that MSK' is the original master key.

Leak queries on a master key and on $\mathcal{ID}^*$ is that the adversary can encode the entire decryption algorithm of CT* as a function on a secret key, and thus win the game trivially if we allow these queries. For the same reason, the challenger can not give an entire secret key of $\mathcal{ID}^*$ to the adversary and hence no **Reveal** queries involving $\mathcal{ID}^*$ are allowed too. **Leak** queries on keys of identities other than $\mathcal{ID}^*$ are useless, since the adversary can get the entire secret keys.

Guess: The adversary outputs a bit $c' \in \{0,1\}$. We say it succeeds if $c' = c$.

The security definition we will use is the following:

Definition 4.9. An IBE encryption system Π is $(\ell_{\text{MSK}}, \ell_{\text{SK}})$-master-leakage secure if for all PPT adversaries $\mathcal{A}$ it is true that

$$\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi}(\lambda, \ell_{\text{MSK}}, \ell_{\text{SK}}) \leq \mathsf{negl}(\lambda)$$

where $\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi}(\lambda, \ell_{\text{MSK}}, \ell_{\text{SK}})$ is the advantage of $\mathcal{A}$ in game $\mathsf{MasterLeak}$ with security parameter λ and leakage parameters $\ell_{\text{MSK}} = \ell_{\text{MSK}}(\lambda), \ell_{\text{SK}} = \ell_{\text{SK}}(\lambda)$ and is formally defined as

$$\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi}(\lambda, \ell_{\text{MSK}}, \ell_{\text{SK}}) = \left| \Pr\left[\mathcal{A} \text{ succeeds}\right] - \frac{1}{2} \right|,$$

where the probability is over all random bits used by the challenger and the attacker.

4.3.5 CP-ABE Leakage-Resilient Adaptive Security

In this section we will define the MasterLeakAbe which models the adaptive (or full) security notion for the leakage resilient CP-ABE schemes. The main difference with the regular adaptive game for CP-ABE schemes is that in this case the attacker can ask for leakage on secret keys of his choice, even if these secret keys can decrypt the challenge ciphertext. He might even ask for leakage on the master secret key. Of course the attacker is not allowed to leak on the entire keys, because otherwise he would be able to trivially win the game. Hence, it is parameterized by two leakage bounds, $\ell_{\text{MSK}}, \ell_{\text{SK}} \in \mathbb{N}$, that bound the number of leaked bits that the attacker is allowed to acquire from each master secret key[7] and each secret key, respectively. As in the regular game, the attacker is also allowed to retrieve entire keys under the restriction that they can not decrypt the challenge ciphertext.

Since we are interested in adaptive security, we give to the attacker the ability to postpone his decision on which keys are leaked, which ones are revealed, and the challenge policy, as long as possible. For example, the attacker might request leakage on some secret key and, depending on the result, either get the entire key or stop leaking on it and use a challenge ciphertext that can be decrypted by it. To achieve this functionality, the challenger upon the creation of a key, assigns a handle to it (a unique integer) and provides to

[7]We might have many master secret keys because the scheme might support an update algorithm that can output a fresh master secret key, consistent with the original public parameters.

the attacker reference to it via this handle. Using this handle, the attacker can either make many leakage queries or ask for a complete reveal on the key. The challenger holds a record of the total number of leaked bits from each specific handle, so that the leakage does not exceed the leakage bounds. Also every time the attacker gets an entire key, the challenger stores the attribute set of this key in another record in order to be able to check the restriction that the challenge access structure is not satisfied by any of these keys

Formally the $\mathsf{MasterLeakAbe}$ game consists of the following phases:

Setup: The challenger makes a call to $\mathbf{Setup}(1^\lambda, \mathcal{U})$ and gets a master key MSK and the public parameters PP. It gives PP to the attacker. Also, it sets $\mathcal{R} = \emptyset$ (the revealed keys' list) and $\mathcal{T} = \{(0, \mathcal{U}, \mathrm{MSK}, 0)\}$ (the handles' list). Here, $\mathcal{R} \subseteq 2^{\mathcal{U}}$ and $\mathcal{T} \subseteq \mathcal{H} \times 2^{\mathcal{U}} \times (\mathcal{MK} \cup \mathcal{SK}) \times \mathbb{N}$ (handles - sets of attributes - keys - leaked bits so far), where $\mathcal{MK}$ and $\mathcal{SK}$ is the space of master keys and secret keys, respectively. Thus initially the set $\mathcal{T}$ holds a record of the master key (universal attribute set for it and no leakage so far). Also a handle counter H is set to 0.

Phase 1: In this phase, the adversary can make the following queries to the challenger. All of them can be interleaved in any possible way and therefore the input of a query can depend on the outputs of all previous queries (adaptive security).

- $\mathbf{Create}(h, \mathcal{S})$: h is a handle to a tuple of $\mathcal{T}$ that has to refer to a master key. The attribute set $\mathcal{S}$ can be any subset of the universe $\mathcal{U}$, including

$\mathcal{U}$ itself. If $\mathcal{S} = \mathcal{U}$, the attacker asks for the creation (via the update algorithm) of another master key.

The challenger initially scans $\mathcal{T}$ to find the tuple with handle h. If the attribute set field of the tuple is not $\mathcal{U}$, which means that the tuple holds a non-master key, or if the handle does not exist, it responds with $\perp$.

Otherwise, the tuple is of the form $(h, \mathcal{U}, \mathrm{MSK}', L)$. Then the challenger makes a call to $\mathbf{Keygen}(\mathrm{MSK}', \mathcal{S}) \rightarrow K$ and adds the tuple $(H+1, \mathcal{S}, K, 0)$ to the set $\mathcal{T}$. After that, it updates the handle counter to $H \leftarrow H+1$.

- **Leak**(h, f): In this query, the adversary requests leakage from a key that has handle $h \in \mathbb{N}$ with a polynomial-time computable function f of constant output size acting on the set of keys. The challenger scans $\mathcal{T}$ to find the tuple with the specified handle. It is either of the form $(h, \mathcal{S}, \mathrm{SK}, L)$ or $(h, \mathcal{U}, \mathrm{MSK}', L)$.

  In the first case, it checks first if $L + \|f(\mathrm{SK})\| \leq \ell_{\mathrm{SK}}$. If this is true, it responds with $f(\mathrm{SK})$ and updates the L in the tuple with $L + \|f(\mathrm{SK})\|$. If the check fails, it returns $\perp$ to the adversary.

  If the tuple holds a master key MSK', it checks if $L + \|f(\mathrm{MSK}')\| \leq \ell_{\mathrm{MSK}}$. If this is true, and responds with $f(\mathrm{MSK}')$ and updates the L with $L + \|f(\mathrm{MSK}')\|$. If the check fails, it returns $\perp$ to the adversary.

- **Reveal**(h): Now the adversary requests the entire key with handle h. The challenger scans $\mathcal{T}$ to find the requested entry. If the handle refers to a master key tuple, then the challenger returns $\perp$. Otherwise, let's say the tuple is $(h, \mathcal{S}, \mathrm{SK}, L)$. The challenger responds with SK and adds the subset $\mathcal{S}$ to the set $\mathcal{R}$.

Challenge: The adversary submits a challenge access structure $\mathbb{A}^*$ with the restriction that no subset in $\mathcal{R}$ satisfies it. It also submits two messages M_0, M_1 of equal size. The challenger flips a uniform coin $c \overset{R}{\leftarrow} \{0,1\}$ and encrypts M_c under $\mathbb{A}^*$ with a call to $\mathbf{Encrypt}(M_c, \mathbb{A}^*)$. It sends the resulting ciphertext CT^* to the adversary.

Phase 2: This is the same as **Phase 1**, except with the restriction that the only allowed queries are **Create** and **Reveal** queries for secret keys with attribute sets that do not satisfy $\mathbb{A}^*$.

Guess: The adversary outputs a bit $c' \in \{0,1\}$. We say it succeeds if $c' = c$.

Definition 4.10. A CP-ABE scheme is $(\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})$-*master leakage secure* if all PPT attackers have at most a negligible advantage in λ in the MasterLeakAbe game.

4.3.6 Three Properties for Leakage-Resilient Adaptive Security

To prove security of our leakage-resilient systems of Chapter 5, we will first define three properties. These are defined similarly in both the IBE and ABE cases. The only difference is that the keys correspond to identities in the IBE case and attribute sets in the ABE case. Here, we will give the definitions of *semi-functional ciphertext invariance*, the *one semi-functional key invariance*, and *semi-functional security* in the ABE setting.

The MasterLeakC game is exactly the same as the MasterLeakAbe game except that in the **Challenge** phase, the challenger uses **EncryptSF** instead of **Encrypt** to create a semi-functional ciphertext, and returns this to the adversary.

In the MasterLeakCK game the challenger again uses **EncryptSF** for the challenge phase. However, the set of tuples $\mathcal{T}$ has a different structure. Each tuple holds for each key (master or secret) a normal and a semi-functional version of it. In this game, all keys leaked or given to the attacker are semi-functional. As we have noted above, the semi-functional key generation algorithm takes as input a normal master key. Thus the challenger stores the normal versions, as well the semi-functional ones so that it can use the normal versions of master keys as input to KeyGen calls. More precisely, the challenger additionally stores a semi-functional master key in tuple 0 by calling **KeygenSF**(MSK, ϵ) after calling **Setup**. Thereafter, for all **Create**(h, X) queries, the challenger makes an additional call to **KeygenSF**(MSK', X), where MSK' is the *normal* version of the master key stored in tuple h. **Leak** and **Reveal** queries act always on the semi-functional versions of each key.

The $\mathsf{MasterLeak}_b$ game is similar to the MasterLeakCK game, with the main difference being that the attacker can choose on which version of each key to leak or reveal. In other words, on the first leakage or reveal query on a key of the augmented set $\mathcal{T}$, the attacker tells the challenger whether it wants the normal or the semi-functional version of the key. In order for the challenger to keep track of the attacker's choice on each key, we further augment each tuple of $\mathcal{T}$ with a lock-value denoted by $V \in \mathbb{N}$ that can take one of the three values:

- -1: That means that the attacker has not made a choice on this key yet and the key is "unlocked". This is the value the tuple gets, in a **Create** query.
- 0: The attacker chose to use the normal version of the key on the first leakage or reveal query on it. All subsequent **Leak** and **Reveal** queries act on the normal version.
- 1: The attacker chose the semi-functional version and the challenger works as above with the semi-functional version.

To summarize, each tuple is of the form $(h, X, K, \widetilde{K}, L, V)$ i.e. handle - attribute set - normal key - semi-functional key - leakage - lock. For example, the original master key is stored at the beginning of the game in the tuple $(0, U, \mathrm{MSK}, \mathbf{KeygenSF}(\mathrm{MSK}, U), 0, -1)$.

At some point, the attacker must decide on a *challenge key* which is "unlocked", $V = -1$, and tell this to the challenger. The challenger samples a uniformly random bit $b \xleftarrow{R} \{0, 1\}$ and sets $V = b$. Therefore, the attacker has access to either the normal (if $b = 0$) or the semi-functional (if $b = 1$) version of this key via **Leak** and **Reveal** queries. We note that if the attacker did not make a choice for the original master key in tuple 0, it can choose this master key as the challenge key.

The attacker is then allowed to resume queries addressed to either normal or semi-functional keys, with the usual restrictions (i.e. no leakage or reveal queries on keys capable of decrypting the challenge ciphertext after the attacker has seen the challenge ciphertext).

Semi-functional Ciphertext Invariance: We say that a dual system ABE scheme Π_D has *($\ell_{\text{MSK}}, \ell_{\text{SK}}$)-semi-functional ciphertext invariance* if for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in the game $\mathsf{MasterLeakAbe}$ is negligibly close to the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeakC}$ game. We denote this by:

$$\left|\mathsf{Adv}_{\mathcal{A},\Pi_D}^{\mathsf{MasterLeakAbe}}(\lambda, \ell_{\text{MSK}}, \ell_{\text{SK}}) - \mathsf{Adv}_{\mathcal{A},\Pi_D}^{\mathsf{MasterLeakC}}(\lambda, \ell_{\text{MSK}}, \ell_{\text{SK}})\right| \leq \mathsf{negl}(\lambda).$$

Semi-functional Key Invariance: We say that a dual system ABE scheme Π_D has *($\ell_{\text{MSK}}, \ell_{\text{SK}}$)-semi-functional key invariance* if for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeakC}$ game is negligibly close to the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeakCK}$ game. We denote this by:

$$\left|\mathsf{Adv}_{\mathcal{A},\Pi_D}^{\mathsf{MasterLeakC}}(\lambda, \ell_{\text{MSK}}, \ell_{\text{SK}}) - \mathsf{Adv}_{\mathcal{A},\Pi_D}^{\mathsf{MasterLeakCK}}(\lambda, \ell_{\text{MSK}}, \ell_{\text{SK}})\right| \leq \mathsf{negl}(\lambda).$$

One Semi-functional Key Invariance: We say that a dual system ABE scheme Π_D has ($\ell_{\text{MSK}}, \ell_{\text{SK}}$)-*one semi-functional key invariance* if, for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeak}_b$ game with $b = 0$ is negligibly close to the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeak}_b$ game with $b = 1$. We denote this by:

$$\left|\mathsf{Adv}^{\mathsf{MasterLeak}_0}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) - \mathsf{Adv}^{\mathsf{MasterLeak}_1}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})\right| \leq \mathsf{negl}(\lambda)$$

Semi-functional Security: We say that a dual system ABE scheme Π_D has *($\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}$)-semi-functional security* if for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeakCK}$ game is negligible. We denote this by:

$$\mathsf{Adv}^{\mathsf{MasterLeakCK}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) \leq \mathsf{negl}(\lambda).$$

The following theorems are proved below.

Theorem 4.11. *If a dual system ABE scheme Π_D = (**Setup**, **Keygen**, **Encrypt**, **Decrypt**, **KeygenSF**, **EncryptSF**) has (ℓ_{MSK}, ℓ_{SK}) - semi - functional ciphertext invariance, (ℓ_{MSK}, ℓ_{SK}) - semi - functional key invariance, and ($\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}$) - semi - functional security, then Π = (**Setup**, **Keygen**, **Encrypt**, **Decrypt**) is a ($\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}$) - master - leakage secure ABE scheme.*

Proof. The proof is straight-forward. We first observe that playing the $\mathsf{MasterLeak}$ game with system Π is exactly the same as playing the $\mathsf{MasterLeak}$ game with system Π_D. The methods called are exactly the same. Therefore we have that:

$$\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) = \mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})$$

By semi-functional ciphertext invariance, we have that:

$$\left|\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) - \mathsf{Adv}^{\mathsf{MasterLeakC}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})\right| \leq \mathsf{negl}(\lambda).$$

By semi-functional key invariance, we have that:

$$\left|\mathsf{Adv}^{\mathsf{MasterLeakC}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) - \mathsf{Adv}^{\mathsf{MasterLeakCK}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})\right| \leq \mathsf{negl}(\lambda).$$

Thus, by the triangle inequality (and the fact that the sum of two negligible functions is also a negligible function), we may conclude that

$$\left|\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) - \mathsf{Adv}^{\mathsf{MasterLeakCK}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})\right| \leq \mathsf{negl}(\lambda).$$

By semi-functional security, we know that

$$\mathsf{Adv}^{\mathsf{MasterLeakCK}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) \leq \mathsf{negl}(\lambda).$$

Hence,

$$\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) \leq \mathsf{negl}(\lambda),$$

which implies that

$$\mathsf{Adv}^{\mathsf{MasterLeak}}_{\mathcal{A},\Pi}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) \leq \mathsf{negl}(\lambda).$$

$\square$

Theorem 4.12. *If a dual system ABE scheme* Π_D *has* $(\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})$-one semi-functional key invariance, *then it also has* $(\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})$-semi-functional key invariance.

Proof. Suppose for contradiction that there is a PPT adversary $\mathcal{A}$ that breaks the semi-functional key invariance property of our system, but Π_D has one semi-functional key invariance. This means by definition that the difference

$$\left|\mathsf{Adv}^{\mathsf{MasterLeakC}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) - \mathsf{Adv}^{\mathsf{MasterLeakCK}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})\right| \tag{4.1}$$

is non-negligible. Then we will construct another PPT algorithm $\mathcal{B}$ that breaks the one semi-functional key invariance of Π_D, which is a contradiction.

We denote by $Q - 1$ the maximum number of **Create** queries that $\mathcal{A}$ makes. Thus, the total number of different secret keys is Q (since we also count the original master key). Since $\mathcal{A}$ is assumed to be polynomial-time, Q is a polynomial in λ.

For $q \in [0, Q]$ we define the game SF_q to be like the $\mathsf{MasterLeakC}$ game (with **EncryptSF** for the challenge phase), semi-functional versions for the first q different keys, and normal versions for the remaining keys. The order is defined by the first leakage or reveal query made on each key. As always, master keys input to **Keygen** calls are normal. The semi-functional versions are passed to $\mathcal{A}$ via leakage or reveal queries.

Notice that SF_0 is the $\mathsf{MasterLeakC}$ game and SF_Q is the $\mathsf{MasterLeakCK}$ game. Hence, since the difference in advantages of SF_0 and SF_Q is non-negligible in λ by (4.1) and Q is a polynomial in λ, there exists a $q^* \in [0, Q-1]$ such that the difference

$$\left|\mathsf{Adv}^{\mathsf{SF}_{q^*}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) - \mathsf{Adv}^{\mathsf{SF}_{q^*+1}}_{\mathcal{A},\Pi_D}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})\right|$$

is non-negligible. This means that the algorithm $\mathcal{A}$ has a non-negligible difference in the advantages when playing game SF_{q^*} and game SF_{q^*+1}.

So, to create an algorithm $\mathcal{B}$ that breaks the one semi-functional key invariance of Π_D, we use $\mathcal{A}$ in the $\mathsf{MasterLeak}_b$ game. When $\mathcal{A}$ makes a key request, $\mathcal{B}$ forwards this to the $\mathsf{MasterLeak}_b$ challenger as follows. $\mathcal{B}$ requests semi-functional keys for the first q^* keys, chooses the (q^*+1)-th key to be the challenge key, and requests normal keys for the remaining keys.

Notice that if the $\mathsf{MasterLeak}_b$ challenger picked $b = 0$, then $\mathcal{A}$ plays the SF_{q^*} game. Otherwise, it plays the SF_{q^*+1} game. This means that

$$\mathsf{Adv}_{\mathcal{B},\Pi_D}^{\mathsf{MasterLeak}_b}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) = \mathsf{Adv}_{\mathcal{A},\Pi_D}^{\mathsf{SF}_{q^*+b}}(\lambda, \ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}) \quad \text{for} \quad b \in \{0,1\}$$

Therefore, $\mathcal{B}$ breaks the one semi-functional key invariance of Π_D, which is a contradiction. $\square$

4.4 Continual Leakage for IBE and ABE

If an IBE or ABE scheme also comes equipped with an update algorithm which takes in a secret key and outputs a new, re-randomized key from the same distribution generated by a fresh call to KeyGen, then the above security definitions yields resilience to continual leakage "for free". Essentially, the many master keys and many keys per identity that our definition allows to leak can be interpreted as updated versions of keys. Hence, each time a key is updated, the attacker is allowed to obtain new leakage on the new version of the key.

We now formally compare our security definition to the Continual Leakage Model (CLM) for leakage resilience. Recent results in this model have appeared in [28] and [39]. This model allows for leakage on the randomness generated during the calls of different methods, as well as leakage on the keys of the system. It is essentially a combination of the types of leakage allowed in the "Only Computation Leaks" model and the memory leakage model. We will show that our definition of security implies a form of continual leakage if the scheme in question has a specific re-randomization property (our schemes in Sec. 5 have this property).

In the continual leakage model, there is only one master key and one secret key per user at any moment in time. Continual leakage on many master and secret keys is achieved with two new additional algorithms, called $\mathsf{UpdateMK}$ and $\mathsf{UpdateSK}$. These update master and secret keys, respectively, and as a result a brand new leakage "session" on the updated key is allowed. We will show that if an IBE or ABE scheme has an extra $\mathsf{UpdateSK}$ algorithm and a specific re-randomization property, then our definition of security implies security in the CLM. The $\mathsf{UpdateMK}$ algorithm is going to be implemented by our KeyGen algorithm with the empty string as input in the case of IBE and the entire attribute universe in the case of ABE. The additional algorithm for both is:

UpdateSK(SK) $\to$ SK$'$ This algorithm takes in a secret key, SK, and outputs a re-randomized key, SK$'$, such that $|\mathrm{SK}'| = |\mathrm{SK}|$.

Definition 4.13. An IBE (resp. ABE) scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt},$

$\mathsf{Decrypt}$, $\mathsf{UpdateSK}$) is called an *IBE (ABE) with re-randomization* if the following property holds:

For all MSK, PP generated by a call to Setup, for all master keys $\mathrm{MSK}', \mathrm{MSK}''$ generated by applying the KeyGen algorithm with the empty string (the universe set) and a previously generated master key, for all identities $\mathcal{ID}$ (all attribute sets $\mathcal{S}$), the distribution of a secret key SK' generated by the $\mathsf{UpdateSK}(\mathsf{KeyGen}(\mathrm{MSK}', I))$ method (by the $\mathsf{UpdateSK}(\mathsf{KeyGen}(\mathrm{MSK}', \mathcal{S}))$ method) is indistinguishable from the distribution of a secret key SK generated by $\mathsf{KeyGen}(\mathrm{MSK}'', I)$ (by $\mathsf{KeyGen}(\mathrm{MSK}'', \mathcal{S})$).

The security definition of IBE schemes in the Continual Leakage Model is defined via the following game, called ClmIbe. This is proposed (informally) in [28]. The CLM definition for ABE schemes is similar.

The game consists of three query phases, where in the first the attacker can make **Extract** queries on identities (similar to our **Reveal** queries) and leakage queries on the master key. Also, it can ask for an update on the master key. In the second phase, the attacker has decided on the challenge identity and can make leakage or update queries on its secret key, in addition to the previous queries. The third phase is like **Phase 2** of the $\mathsf{MasterLeak}$ game; no leakage queries are allowed.

The game is parameterized by the security parameter λ and five leakage parameters
$(\rho_G, \rho_{UM}, \rho_M, \rho_{US}, \rho_S)$. These are meant to be leakage on the generation algo-

rithm, on the update procedure of the master key, on the master key, on the update procedure of a secret key, and on the secret keys, respectively. As in the $\mathsf{MasterLeak}$ game, the challenger has to keep track of the total leakage on each master key and on every secret key. Since we have only one master key at a time, there is no need for the challenger to store master keys in $\mathcal{T}$. It has a master key leakage counter denoted L_{MSK}. The phases of the game are:

Setup - CLM: The challenger chooses "secret randomness" r and "public randomness" p and calls $\mathsf{Setup}(1^\lambda; r, p) \to (\mathrm{PP}, \mathrm{MSK})$. The adversary specifies a polynomial-time computable function f of constant output size such that $\|f(r,p)\| \leq \rho_G \cdot \|r\|$ for all r, p. The challenger sends to the adversary the tuple $(\mathrm{PP}, f(r,p), p)$. Also it sets the master leakage counter $L_{\mathrm{MSK}} = \|f(r,p)\|$ and a handle counter $H = 0$. It initializes $\mathcal{R} = \emptyset$.

Phase 1 - CLM: In this phase, the adversary can make one of the following queries to the challenger. All of them can be interleaved in any possible way and therefore the input of a query can depend on the outputs of all previous queries (adaptive security).

- **Keygen**$(\mathcal{ID})$: The challenger adds the identity $\mathcal{ID}$ to $\mathcal{R}$, since it should be considered "revealed" from now on and responds with the output of a call to $\mathsf{KeyGen}(\mathrm{MSK}, \mathcal{ID})$.
- **MasterLeak**(f): In this query, f is a polynomial-time computable function of constant output size such that $L_{\mathrm{MSK}} + \|f(\mathrm{MSK})\| \leq \rho_M \cdot \|\mathrm{MSK}\|$.

The adversary requests leakage from the master key here. The challenger responds with the value $f(\mathrm{MSK})$ and updates L_{MSK} to $L_{\mathrm{MSK}} + \|f(\mathrm{MSK})\|$. If $L_{\mathrm{MSK}} + \|f(\mathrm{MSK})\| > \rho_M \cdot \|\mathrm{MSK}\|$, it responds with the dummy value $\perp$.

- **UpdateMK**(f): Now the attacker requests an update (and leakage) on the master key with a polynomial-time computable function f of constant output size. It should be true that $\|f(\mathrm{MSK}, r, p)\| \leq \rho_{UM} \cdot (\|\mathrm{MSK}\| + \|r\|)$ for all MSK, r, p where r, p are the secret and public randomness, respectively, of the KeyGen method. The challenger chooses randomness r, p and generates a new master key, $\widehat{\mathrm{MSK}} \leftarrow \mathsf{KeyGen}$ $(\mathrm{MSK}, \epsilon; r, p)$. If $L_{\mathrm{MSK}} + \|f(\mathrm{MSK}, r, p)\| \leq \rho_M \cdot \|\mathrm{MSK}\|$, it gives to the attacker $f(\mathrm{MSK}, r, p)$. Finally, it sets $L_{\mathrm{MSK}} = \|f(\mathrm{MSK}, r, p)\|$ and $\mathrm{MSK} \leftarrow \widehat{\mathrm{MSK}}$, in that order.

Challenge Identity - CLM: In this phase, the attacker chooses the challenge identity $\mathcal{ID}^* \notin \mathcal{R}$ and the challenger creates a secret key for it: $\mathrm{SK}_{\mathcal{ID}^*} \leftarrow \mathsf{KeyGen}(MK, \mathcal{ID}^*)$. Also it sets a leakage counter $L_{\mathrm{SK}} = 0$.

Phase 2 - CLM: In this phase, we allow the following queries. The first three are same to the respective ones of **Phase 1 -CLM**.

- **Keygen**$(\mathcal{ID})$: Obviously $\mathcal{ID} \neq \mathcal{ID}^*$ is required.
- **MasterLeak**(f)
- **UpdateMK**(f)

- **Leak**(f): In this query, f is a polynomial-time computable function of constant output size such that $L_{\mathrm{SK}} + \|f(\mathrm{SK}_{\mathcal{ID}^*})\| \leq \rho_S \cdot \|\mathrm{SK}_{\mathcal{ID}^*}\|$. The adversary requests leakage from the secret key of $\mathcal{ID}^*$ here. The challenger responds with the value $f(\mathrm{SK}_{\mathcal{ID}^*})$ and updates L_{SK} to $L_{\mathrm{SK}} + \|f(\mathrm{SK}_{\mathcal{ID}^*})\|$. If $L_{\mathrm{SK}} + \|f(\mathrm{SK}_{\mathcal{ID}^*})\| > \rho_S \cdot \|\mathrm{SK}_{\mathcal{ID}^*}\|$, it responds with the dummy value $\perp$.

- **UpdateSK**(f): Now the attacker requests an update, and leakage, on the secret key of $\mathcal{ID}^*$ with a polynomial-time computable function f of constant output size. It should be true that $\|f(\mathrm{SK}_{\mathcal{ID}^*}, r, p)\| \leq \rho_{US} \cdot (\|\mathrm{SK}\| + \|r\|)$ for all SK, r, p where r, p are the secret and public randomness, respectively, of the **Keygen** method. The challenger chooses randomness r, p and generates a new secret key, $\widehat{\mathrm{SK}_{\mathcal{ID}^*}} \leftarrow \mathsf{KeyGen}$ $(\mathrm{MSK}, \mathcal{ID}^*; r, p)$. If $L_{\mathrm{SK}} + \|f(\mathrm{SK}_{\mathcal{ID}^*}, r, p)\| \leq \rho_S \cdot \|\mathrm{SK}_{\mathcal{ID}^*}\|$, it gives to the attacker $f(\mathrm{SK}_{\mathcal{ID}^*}, r, p)$. Finally, it sets $L_{\mathrm{SK}} = \|f(\mathrm{SK}_{\mathcal{ID}^*}, r, p)\|$ and $\mathrm{SK}_{\mathcal{ID}^*} \leftarrow \widehat{\mathrm{SK}_{\mathcal{ID}^*}}$, in that order.

Challenge: The adversary submits two messages M_0, M_1 of equal size. The challenger flips a uniform coin $c \overset{R}{\leftarrow} \{0,1\}$ and encrypts M_c under $\mathcal{ID}^*$ with a call to $\mathsf{Encrypt}(M, \mathcal{ID})$. It sends the resulting ciphertext CT^* to the adversary.

Phase 3 - CLM: Now only **Keygen**$(\mathcal{ID})$ queries with $\mathcal{ID} \neq \mathcal{ID}^*$ are allowed.

Guess: The adversary outputs a bit $c' \in \{0,1\}$. We say it succeeds if $c' = c$.

We say that a scheme $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{UpdateSK})$ is $(\rho_G, \rho_{UM}, \rho_M, \rho_{US}, \rho_S)$-secure in the CLM if any PPT adversary has at most a negligible advantage in winning the ClmIbe game.

We will prove the following theorem:

Theorem 4.14. *If an IBE system* $\Pi = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{UpdateSK})$ *with re-randomization is* $(\ell_{\text{MSK}}, \ell_{\text{SK}})$*-master-leakage secure, then it is also*

$$\left(0, 0, \frac{\ell_{\text{MSK}}}{\|\text{MSK}\|}, 0, \frac{\ell_{\text{SK}}}{\|\text{SK}\|}\right) \text{ - } \textit{secure}$$

in the Continual Leakage Model above.

Proof. To prove the theorem, we assume that we have a PPT attacker $\mathcal{A}$ that breaks our system in the continual leakage model with parameters $\Big(0,\ 0,\ \frac{\ell_{\text{MSK}}}{\|\text{MSK}\|},\ 0,\ \frac{\ell_{\text{SK}}}{\|\text{SK}\|}\Big)$. Notice that this attacker gets no leakage from the generation and update algorithms. We will construct a PPT algorithm $\mathcal{B}$ that uses $\mathcal{A}$ and breaks the $(\ell_{\text{MSK}}, \ell_{\text{SK}})$-master-leakage security of our system. $\mathcal{B}$ will play the role of $\mathcal{A}$'s challenger in the ClmIbe game.

Essentially, the main strategy of the new algorithm is to merge phases **1, Challenge Identity, 2** of the ClmIbe game into **Phase 1** of the $\mathsf{MasterLeak}$ game. It will use a handle h_{MSK} to denote the "current" master key and a handle $h_{\mathcal{ID}^*}$ to denote the current secret key of the challenge identity. Also, we state here that $\mathcal{B}$ chooses all randomness used to be private. Initially, it sets $h_{\text{MSK}} = 0$. $\mathcal{B}$ works as follows:

Setup: $\mathcal{B}$ executes the setup phase of $\mathsf{MasterLeak}$ game with its challenger and sends only the public parameters to $\mathcal{A}$. Since no leakage is allowed on the generation algorithm and our system does not use public randomness, this is exactly what $\mathcal{A}$ expects.

Phase 1: For every $\mathsf{KeyGen}(\mathcal{ID})$ query made by $\mathcal{A}$, $\mathcal{B}$ makes a **Create**$(h_{\text{MSK}}, \mathcal{ID}) \to h'$ query first and a **Reveal**$(h') \to \text{SK}'$ query afterwards. It gives to $\mathcal{A}$ the secret key SK'. It is obvious that this is exactly the output of $\mathsf{KeyGen}(\text{MSK}, \mathcal{ID})$, where MSK is the current master key.

For every **MasterLeak**(f) query made by $\mathcal{A}$, $\mathcal{B}$ makes a **Leak**(h_{MSK}, f) query. Since $L_{\text{MSK}} + \|f(\text{MSK})\| \leq \rho_M \cdot \|\text{MSK}\| \implies L_{\text{MSK}} + \|f(\text{MSK})\| \leq \ell_{\text{MSK}}$, the challenger of the $\mathsf{MasterLeak}$ game has to provide $\mathcal{B}$ with the requested leakage $f(\text{MSK})$. Notice that the update it makes to the L of the tuple is the same as update $\mathcal{A}$'s challenger should make on L_{MSK} - thus legitimate in the view of $\mathcal{A}$.

For every **UpdateMK**(f) query made by $\mathcal{A}$, $\mathcal{B}$ has only to update the master key. That is because $\rho_{UM} = 0$ and thus f outputs nothing. To simulate an update, it makes a **Create**$(h_{\text{MSK}}, \epsilon) \to h'$ query. It sets $h_{\text{MSK}} \leftarrow h'$, which changes the current master key to the new one. The method called is exactly the same, i.e. $\mathsf{KeyGen}(\text{MSK}, \epsilon)$; hence $\mathcal{A}$ sees no difference.

At some point, $\mathcal{A}$ reaches the challenge phase. After sending the challenge identity $\mathcal{ID}^*$, $\mathcal{B}$ makes a $\textbf{Create}(h_{\text{MSK}}, \mathcal{ID}^*) \to h_{\mathcal{ID}^*}$ query. The handle $h_{\mathcal{ID}^*}$ will point to the current secret key of the challenge identity. For the additional queries of **Phase 2 - CLM**, $\mathcal{B}$ works as follows:

For every $\textbf{Leak}(f)$ query, $\mathcal{B}$ makes a $\textbf{Leak}(h_{\mathcal{ID}^*}, f)$ query. Since $L_{\text{SK}} + \|f(\text{SK}_{\mathcal{ID}^*})\| \leq \rho_S \cdot \|\text{SK}_{\mathcal{ID}^*}\| \implies L_{\text{SK}} + \|f(\text{SK}_{\mathcal{ID}^*})\| \leq \ell_{\text{SK}}$, the challenger of the MasterLeak game has to provide $\mathcal{B}$ with the requested leakage $f(\text{SK}_{\mathcal{ID}^*})$. Notice that the update it makes to the L of the tuple is the same as update $\mathcal{A}$'s challenger should make on L_{SK} - thus legitimate in the view of $\mathcal{A}$.

For every $\textbf{UpdateSK}(f)$ query, $\mathcal{B}$ has only to update the secret key of $\mathcal{ID}^*$. That is because $\rho_{US} = 0$ and thus f outputs nothing. Instead of updating, it makes a $\textbf{Create}(h_{\text{MSK}}, \mathcal{ID}^*) \to h'$ query. It sets $h_{\mathcal{ID}^*} \leftarrow h'$, which changes the current secret key to the new one. However, now the method called is not what $\mathcal{A}$ expected. It expected the **UpdateSK** method, but $\mathcal{B}$ implicitly called the KeyGen method. Since *the output distributions of the two methods are indistinguishable by the property of re-randomization*, $\mathcal{A}$ cannot have a non-negligible change in its advantage. Thus, the advantage of $\mathcal{B}$ will still be non-negligible.

Challenge: Here, $\mathcal{B}$ simply forwards to its challenger the two messages and the challenge identity provided by $\mathcal{A}$. According to the MasterLeak game, the challenger encrypt the message under the challenge identity and returns the ciphertext to $\mathcal{B}$. It responds to $\mathcal{A}$ with this ciphertext. It is obvious that this is a correct simulation for $\mathcal{A}$.

Phase 2: In this phase $\mathcal{A}$ can make only KeyGen queries for $\mathcal{ID} \neq \mathcal{ID}^*$. For each such query, $\mathcal{B}$ makes a **Create**$(h_{\text{MSK}}, I) \to h'$ query first and a **Reveal**$(h') \to \text{SK}'$ query afterwards. It gives to $\mathcal{A}$ the secret key SK'. It is obvious that this is exactly the output of KeyGen (MSK, I), where MSK is the current master key.

Guess: $\mathcal{B}$ outputs $\mathcal{A}$'s guess bit.

The advantage of $\mathcal{B}$ in the $\mathsf{MasterLeak}$ game is exactly the same as the advantage of $\mathcal{A}$ in CImIbe. Thus, it breaks the $(\ell_{\text{MSK}}, \ell_{\text{SK}})$-master-leakage security. □

4.4.1 Leakage from Updates

We note here that using the same "guess and check" method of [28], we can tolerate a small amount leakage on the generation and update procedures in the Continual Leakage Model. More specifically, we can tolerate leakage which is logarithmic in the security parameter λ by guessing a value for the leakage and observing whether the attacker's advantage noticeably decreases. If not, we can use this value for the leakage during the key generation or update in question, and continue the simulation. By limiting the leakage size to logarithmic, we can efficiently check all possible leakage values and hence we will be able to find one that works in polynomial time. The details of this argument are given in [28].

CHAPTER 5

Part I: Leakage-Resilient IBE and CP-ABE Systems

Our IBE and ABE schemes [71] are suitable transformations of the Lewko-Waters HIBE constructions [72], designed to sustain master and secret key leakage from an arbitrary number of keys. To hide nominal semi-functionality in the attacker's view, we add vectors of dimension n to the front of the ciphertexts and secret keys of the LW system. Notice in the construction below that the last two elements of our secret keys and ciphertexts are very similar to the elements in the LW system. Nominal semi-functionality now corresponds to the vector of exponents of the semi-functional components of the key being orthogonal to the vector of exponents of the semi-functional components of the ciphertext. We can use the algebraic lemma of [28] to assert that this orthogonality is hidden from attackers with suitably bounded leakage. We note here that according to the lemma and our security game, the leakage is not applied to both vectors at once, since then the dot product would be trivial to leak. Finally, to allow leakage on the master key, we designed the master key to be similar in form to regular secret keys.

Like the original LW scheme, our systems use a bilinear group whose order is the product of three distinct primes. The role of the first prime order subgroup is to "carry" the necessary information of the plaintext message and the secret information of each user or the master authority. The second subgroup is used only in the proof to provide semi-functionality. The third subgroup is used to additionally randomize secret keys. Each of these components is orthogonal to the other two under the pairing operation.

5.1 Leakage-Resilient IBE

Our IBE system from [71] is the first IBE scheme resistant to continual leakage attacks on both the users' secret keys and master secret key. It serves as a stepping stone towards the more complicated CP-ABE construction of Sec. 5.2. It can also be extended to a continual-leakage-resilient HIBE construction using the same techniques.

5.1.1 Construction

Our dual system IBE scheme consists of the following algorithms:

$\mathsf{Setup}(1^\lambda)$ The setup algorithm generates a bilinear group $\mathbb{G}$ of composite order $N = p_1 p_2 p_3$, where p_1, p_2, p_3 are three different λ_1,λ_2,λ_3-bit prime numbers respectively[1]. Therefore, for every $i \in \{1,2,3\}$ we have that $2^{\lambda_i - 1} \leq p_i < 2^{\lambda_i}$. The subgroup of order p_i in $\mathbb{G}$ is denoted by $\mathbb{G}_i$. We assume that the identities of users in our system are elements of $\mathbb{Z}_N$.

We let n be a positive integer greater than or equal to 2. The value of n can be varied - higher values of n will lead to a better fraction of leakage being tolerated (see Sec. 5.3), while lower values of n will yield a system with fewer group elements in the keys and ciphertexts.

The algorithm picks 3 random generators $(g_1, u_1, h_1) \in \mathbb{G}_1 \times \mathbb{G}_1 \times \mathbb{G}_1$ and one generator $g_3 \in \mathbb{G}_3$. It also picks $n+1$ random exponents $(\alpha,\ x_1,\ x_2,\ \ldots,\ x_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$. It picks $(r,\ y_1,\ y_2,\ \ldots,\ y_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$, a random vector $\vec{\rho} = (\rho_1, \ldots, \rho_{n+2}) \xleftarrow{R} \mathbb{Z}_N^{n+2}$, and a random element $\rho_{n+3} \xleftarrow{R} \mathbb{Z}_N$. It outputs the following public parameters and master key:

$$\mathrm{PP} = (N, g_1, g_3, u_1, h_1, e(g_1, g_1)^{\alpha}, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$$
$$\mathrm{MSK} = \left(\vec{K^*}, K^*\right) = \left(\left(g_1^{y_1}, \ldots, g_1^{y_n}, g_1^{\alpha} h_1^{-r} \prod_{i=1}^{n} g_1^{-x_i y_i}, g_1^{r}\right) * g_3^{\vec{\rho}}, u_1^{r} g_3^{\rho_{n+3}}\right)$$

$\mathsf{KeyGen}(\mathrm{MSK}, \mathrm{PP}, X)$ We first consider when $X = \epsilon$, the empty string. Then this algorithm re-randomizes the master key by picking another $(r',\ y_1',\ y_2',\ \ldots,\ y_n') \xleftarrow{R} \mathbb{Z}_N^{n+1}$, a random vector $\vec{\rho'} = \left(\rho_1', \ldots, \rho_{n+2}'\right) \xleftarrow{R} \mathbb{Z}_N^{n+2}$, and a random element $\rho_{n+3}' \xleftarrow{R} \mathbb{Z}_N$. If $\mathrm{MSK} = \left(\vec{K^*}, K^*\right)$, it outputs the new (same-

[1]The three λ's depend on the security parameter and are chosen appropriately to get a better leakage fraction (see Sec. 5.3.

sized) master key:

$$\mathrm{MSK}' = \left(\vec{K'}, K'\right) = \left(K^* * \left(g_1^{y'_1}, \ldots, g_1^{y'_n}, h_1^{-r'} \prod_{i=1}^{n} g_1^{-x_i y'_i}, g_1^{r'}\right) * g_3^{\vec{\rho'}}, K^* u_1^{r'} g_3^{\rho'_{n+3}}\right)$$

If $X = \mathcal{ID} \in \mathbb{Z}_N$, an identity, the algorithm picks $n+1$ random exponents $(r', z_1, z_2, \ldots, z_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$. Also it picks $\vec{\rho'} \xleftarrow{R} \mathbb{Z}_N^{n+2}$ and outputs the secret key:

$$\mathrm{SK} = \vec{K_1} = \vec{K^*} * \left(g_1^{z_1}, g_1^{z_2}, \ldots, g_1^{z_n}, (K^*)^{-I}(u_1^{\mathcal{ID}} h_1)^{-r'} \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r'}\right) * g_3^{\vec{\rho'}}$$

The terms $g_1^{-x_i y'_i}$ and $g_1^{-x_i z_i}$ above are calculated by using the g^{x_i} terms of PP.

It is very important to notice that with knowledge of α alone, one can create properly distributed secret keys, because the random terms r, y_1, ..., y_n, ρ_{n+3}, $\vec{\rho}$ of the master key are all masked by the random terms $r', z_1, \ldots, z_n, \vec{\rho'}$ generated by the algorithm. However, instead of storing α, the master authority now stores $n+3$ elements of $\mathbb{G}$.

$\mathsf{Encrypt}(M, \mathcal{ID})$ The encryption algorithm picks $s \xleftarrow{R} \mathbb{Z}_N$ and outputs the ciphertext:

$$\begin{aligned} \mathrm{CT} &= \left(C_0, \vec{C_1}\right) = \\ &= \left(M \cdot (e(g_1, g_1)^{\alpha})^s, \left((g_1^{x_1})^s, \ldots, (g_1^{x_n})^s, g_1^s, (u_1^{\mathcal{ID}} h_1)^s\right)\right) \in \mathbb{G}_T \times \mathbb{G}^{n+2} \end{aligned}$$

$\mathsf{Decrypt}(\mathrm{CT}, \mathrm{SK})$ To calculate the blinding factor $e(g_1, g_1)^{\alpha s}$, one computes $e_{n+2}(\vec{K}_1,\ \vec{C}_1)$. If the encryption and decryption are correct, we get:

$$\begin{aligned}
e_{n+2}(\vec{K}_1, \vec{C}_1) &= e(g_1, g_1)^{\alpha s} e(g_1, u_1^{\mathcal{ID}} h_1)^{rs} e(g_1, u_1^{\mathcal{ID}} h_1)^{r's} \\
&\cdot e(h_1, g_1)^{-rs} e(u_1^{\mathcal{ID}}, g_1)^{-rs} e(u_1^{\mathcal{ID}} h_1, g_1)^{-r's} \\
&\cdot \prod_{i=1}^{n} e(g_1, g_1)^{-x_i y_i s} \prod_{i=1}^{n} e(g_1, g_1)^{-x_i z_i s} \\
&\cdot \prod_{i=1}^{n} e(g_1, g_1)^{x_i y_i s} \prod_{i=1}^{n} e(g_1, g_1)^{x_i z_i s} \\
&= e(g_1, g_1)^{\alpha s}
\end{aligned}$$

(Note that the $\mathbb{G}_3$ parts of the key do not contribute anything because they are orthogonal to the ciphertext under e.)

Hence, the message is computed as:

$$M = \frac{C_0}{e_{n+2}(\vec{K}_1, \vec{C}_1)}$$

5.1.2 Semi-Functionality

All the ciphertexts, master keys, and secret keys generated by the above algorithms are *normal*, where by normal we mean that they have no $\mathbb{G}_2$ parts. On the other hand, a *semi-functional* key or ciphertext has $\mathbb{G}_2$ parts. We let g_2 denote a generator of $\mathbb{G}_2$. The remaining algorithms of our dual system IBE are the following:

$\mathsf{KeyGenSf}(\text{MSK}, X) \rightarrow \widetilde{K}$ This algorithm calls first the normal key generation algorithm **Keygen**(MSK, X) to get a normal key MSK $= (\vec{K}^*,\ K^*)$ or SK $= \vec{K}_1$, depending on X.

In the former case, it picks $\vec{\theta} \xleftarrow{R} \mathbb{Z}_N^{n+2}$ and $\theta \xleftarrow{R} \mathbb{Z}_N$ and outputs

$$\widetilde{\mathrm{MSK}} = \left(\vec{K}^* * g_2^{\vec{\theta}}, K^* g_2^{\theta}\right)$$

In the latter case, it picks $\vec{\gamma} \xleftarrow{R} \mathbb{Z}_N^{n+2}$ and outputs

$$\widetilde{\mathrm{SK}} = \vec{K}_1 * g_2^{\vec{\gamma}}$$

$\mathsf{EncryptSf}(M, \mathcal{ID}) \to \widetilde{\mathrm{CT}}$ This algorithm calls first the normal encryption algorithm $\mathsf{Encrypt}(M, \mathcal{ID})$ to get the ciphertext $\mathrm{CT} = (C_0, \vec{C}_1)$. Then it picks $\vec{\delta} \xleftarrow{R} \mathbb{Z}_N^{n+2}$ and outputs

$$\widetilde{\mathrm{CT}} = \left(C_0, \vec{C}_1 * g_2^{\vec{\delta}}\right).$$

Notice that the above algorithms need a generator g_2 of the subgroup $\mathbb{G}_2$. We call the three terms $\left(\vec{\theta}, \theta\right), \vec{\gamma}, \vec{\delta}$ the *semi-functional parameters* of the master key, secret key, and ciphertext, respectively. Notice that a secret key that has been constructed using a semi-functional master key is considered *semi-functional*; not normal. For example, if someone uses the master key $\widetilde{\mathrm{MSK}}$, with parameters $\left(\vec{\theta}, \theta\right)$, to construct a secret key for identity $\mathcal{ID}$ with KeyGen, then this will be semi-functional with parameters $\vec{\gamma} = \vec{\theta} + (0, \ldots, 0, -\mathcal{ID}\theta, 0)$. Normal secret keys do not have a $\mathbb{G}_2$ part.

The semi-functional keys are partitioned in *nominal* semi - functional keys and in *truly* semi - functional keys, with respect to a specific semi - functional ciphertext. In short, a nominal secret key can correctly decrypt the ciphertext (by using Decrypt), while a nominal master key can generate a semi-functional secret key that correctly decrypts the ciphertext. A semi-functional secret key of identity $\mathcal{ID}_k$ with parameters $\vec{\gamma}$ is nominal with respect to a ciphertext for identity $\mathcal{ID}_c$ with parameters $\vec{\delta}$ if and only if

$$\vec{\gamma} \cdot \vec{\delta} = 0 \bmod p_2 \quad \text{and} \quad \mathcal{ID}_k = \mathcal{ID}_c$$

It is easy to see that only then the decryption is correct, because we get an extra term $e(g_2, g_2)^{\vec{\gamma} \cdot \vec{\delta}}$ by the pairing. A semi-functional master key with parameters $\vec{\theta}, \theta$ is nominal with respect to a ciphertext for identity $\mathcal{ID}$ with parameters $\vec{\delta}$ if and only if

$$\vec{\delta} \cdot \Big(\vec{\theta} + (0, \ldots, 0, -\mathcal{ID}\theta, 0)\Big) = 0 \bmod p_2.$$

5.1.3 Continual Leakage

For completeness, we give here the update algorithm for the secret keys. It is clear that it satisfies the re-randomization property.

$\mathsf{UpdateSK}(\mathrm{SK}) \to \mathrm{SK}'$ The update algorithm picks $n+1$ random exponents $(r', z_1, z_2, \ldots, z_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$ and $\vec{\rho'} \xleftarrow{R} \mathbb{Z}_N^{n+2}$. For $\mathrm{SK} = \vec{K_1}$, it outputs the new secret key:

$$\mathrm{SK}' = \vec{K_1'} = \vec{K_1} * \left(g_1^{z_1}, g_1^{z_2}, \ldots, g_1^{z_n}, (u_1^{\mathcal{ID}} h_1)^{-r'} \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r'} \right) * g_3^{\vec{\rho'}}.$$

5.1.4 Security

We prove the following theorem:

Theorem 5.1. *Under the assumptions* Comp1, Comp2, Comp3 *and for* ($\ell_{\text{MSK}} = (n-1-2c)\log(p_2)$, $\ell_{\text{SK}} = (n-1-2c)\log(p_2)$), *where $c > 0$ is a fixed positive constant, our dual system IBE scheme is* (ℓ_{MSK}, ℓ_{SK})*-master-leakage secure.*

In order to prove that our system is $(\ell_{\text{MSK}}, \ell_{\text{SK}})$-master-leakage secure, we have to prove that it has semi-functional ciphertext invariance, one semi-functional key invariance, and semi-functional security. Then according to Theorems 4.11 and 4.12, it is $(\ell_{\text{MSK}}, \ell_{\text{SK}})$-master-leakage secure. We will base each of properties on one of our three complexity assumptions of subsection 3.2.2.

Our values of ℓ_{MSK} and ℓ_{SK} are based on the following lemma, and will only become relevant in our proof of one semi-functional key invariance.

Semi-functional Ciphertext Invariance

Theorem 5.2. *If the assumption* Comp1 *holds, our system has* $(\ell_{\text{MSK}}, \ell_{\text{SK}})$*-semi-functional ciphertext invariance.*

Proof. We will build a PPT simulator $\mathcal{B}$ that breaks the assumption Comp1 with the help of a PPT attacker $\mathcal{A}$ that breaks the semi-functional ciphertext invariance of our system.

The simulator $\mathcal{B}$ initially receives input from the assumption's challenger, i.e. $D^1 = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_3)$ and a challenge term T, which is equal either to g_1^z or $g_1^z g_2^\nu$. Then it plays the MasterLeak or the MasterLeakC game with $\mathcal{A}$ in the following way:

Setup phase: $\mathcal{B}$ picks $(\alpha, x_1, x_2, \ldots, x_n, a, b) \xleftarrow{R} \mathbb{Z}_N^{n+3}$. It computes $u_1 = g_1^a, h_1 = g_1^b$, $e(g_1, g_1)^\alpha$, $g_1^{x_1}$, $g_1^{x_2}$, …, and $g_1^{x_n}$. It gives the public parameters

$$\mathrm{PP} = (N, g_1, g_3, u_1, h_1, e(g_1, g_1)^\alpha, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$$

to $\mathcal{A}$ where N, g_1 and g_3 are given by the challenger.

$\mathcal{B}$ also picks $(r, y_1, y_2, \ldots, y_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$, a random vector $\vec{\rho} = (\rho_1, \ldots, \rho_{n+2}) \xleftarrow{R} \mathbb{Z}_N^{n+2}$, and a random element $\rho_{n+3} \xleftarrow{R} \mathbb{Z}_N$. It stores in tuple 0 the normal master key:

$$\mathrm{MSK} = \left(\vec{K^*}, K^*\right) = \left(\left(g_1^{y_1}, \ldots, g_1^{y_n}, g_1^\alpha h_1^{-r} \prod_{i=1}^{n} g_1^{-x_i y_i}, g_1^r\right) * g_3^{\vec{\rho}}, u_1^r g_3^{\rho_{n+3}}\right).$$

Phase 1: The simulator $\mathcal{B}$ can answer all of $\mathcal{A}$'s queries, since it knows the master key of tuple 0. It works according to the definition of the game, by making the appropriate calls.

Challenge Phase: The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages M_0 and M_1 and the challenge identity $\mathcal{ID}^*$. The simulator $\mathcal{B}$ chooses $c \xleftarrow{R} \{0, 1\}$ and outputs the ciphertext:

$$CT = \left(C_0, \vec{C_1}\right) = \left(M_c \cdot e\left(T, g_1^\alpha\right), \left(T^{x_1}, T^{x_2}, \ldots, T^{x_n}, T, T^{a\mathcal{ID}^*+b}\right)\right),$$

where T is the challenge term from the assumption.

Phase 2: $\mathcal{B}$ works in the same way as **Phase 1**.

If $T = g_1^z g_2^\nu$, then the ciphertext is semi-functional, since

$$\begin{aligned}
&C_0 = M_c \cdot e\left(g_1^z g_2^\nu, g_1^\alpha\right) = M \cdot e(g_1, g_1)^{\alpha z} \\
&T^{aI^*+b} = (u_1^{I^*} h_1)^z g_2^{\nu(a\mathcal{ID}^*+b)} \\
&T = g_1^z g_2^\nu \\
&T^{x_i} = (g_1^{x_i})^z g_2^{\nu x_i} && \text{for } i \in \{1, 2, \ldots, n\}
\end{aligned}$$

This implicitly sets $s = z$ and $\vec{\delta} = (\nu x_1, \nu x_2, \ldots, \nu x_n, \nu, \nu(a\mathcal{ID}^* + b))$. Obviously, s is properly distributed since $z \xleftarrow{R} \mathbb{Z}_N$ according to the assumption. The vector $\vec{\delta}$ is properly distributed in the attacker's view because the multiplying factors $(aI^* + b), x_1, x_2, \ldots, x_n$ are only seen modulo p_1 in the public parameters and not modulo p_2. Thus, in $\mathcal{A}$'s view, they are random modulo p_2 by the Chinese Remainder Theorem. This means that $\mathcal{B}$ has properly simulated the $\mathsf{MasterLeakC}$ game.

If $T = g_1^z$, it is easy to see that the ciphertext is normal since it has no $\mathbb{G}_2$ part, and $\mathcal{B}$ has properly simulated the $\mathsf{MasterLeak}$ game.

Hence, if $\mathcal{A}$ has a non-negligible difference in the advantages of these two games, $\mathcal{B}$ can use it and break the assumption $\mathsf{Comp1}$ with non-negligible advantage. $\square$

One Semi-functional Key Invariance

Theorem 5.3. *If the assumption* $\mathsf{Comp2}$ *holds, our system has* $(\ell_{\text{MSK}}, \ell_{\text{SK}})$*-one semi - functional key invariance.*

Proof.

In order to prove this theorem we need the following two lemmas:

Lemma 5.4. *If the assumption* $\mathsf{Comp2}$ *holds, then for any PPT adversary* $\mathcal{A}$*,* $\mathcal{A}$*'s advantage in the* $\mathsf{MasterLeak}_b$ *game, where* $b = 0$ *or* $b = 1$*, changes only by a negligible amount if we restrict it to making queries only on the challenge identity and on identities that are not equal to the challenge identity modulo* p_2 .

Proof. If there exists an adversary whose advantage changes by a non-negligible amount under this restriction, we can find a non-trivial factor of N with non-negligible probability. This non-trivial factor can then be used to break the assumption $\mathsf{Comp2}$ (same proof as [72]).

The simulator plays the $\mathsf{MasterLeak}_b$ game, where $b = 0$ or $b = 1$, using the terms from the assumption $\mathsf{Comp2}$ to create the semi-functional keys and ciphertext. It works in a way similar to the simulator in the reduction shown after the proof of the following lemma. $\square$

Lemma 5.5. *We suppose that the leakage is at most* $(\ell_{\text{MSK}} = (n-1-2c)\log(p_2), \ell_{\text{SK}} = (n-1-2c)\log(p_2))$*, where* $c > 0$ *is any fixed positive constant. Then, for any PPT adversary* $\mathcal{A}$*,* $\mathcal{A}$*'s advantage in the* $\mathsf{MasterLeak}_1$ *game changes only by a negligible amount when the truly semi-functional challenge key is replaced by a nominal semi-functional challenge key whenever* $\mathcal{A}$ *declares the challenge key to be either a master key or a key for the same identity as the challenge ciphertext.*

Proof. We suppose there exists a PPT algorithm $\mathcal{A}$ whose advantage changes by a non-negligible amount ϵ when the $\mathsf{MasterLeak}_1$ game changes as described above. Using $\mathcal{A}$, we will create a PPT algorithm $\mathcal{B}$ which will distinguish between the distributions $(\vec{\delta}, f(\vec{\tau}))$ and $(\vec{\delta}, f(\vec{\tau'}))$ from Corollary B.1.1 with non-negligible advantage (where $m = n + 1$ and $p = p_2$). This will yield a contradiction, since these distributions have a negligible statistical distance.

$\mathcal{B}$ simulates the game $\mathsf{MasterLeak}_1$ with $\mathcal{A}$ as follows. It starts by running the Setup algorithm for itself, and giving $\mathcal{A}$ the public parameters. Since $\mathcal{B}$ knows the original master key and generators of all the subgroups, it can make normal as well as semi-functional keys. Hence, it can respond to $\mathcal{A}$'s non-challenge **Phase 1** queries by simply creating the queried keys.

With non-negligible probability, $\mathcal{A}$ must chose a challenge key in **Phase 1** which is either a master key or matches the identity of the challenge ciphertext. (If it only did this with negligible probability, then the difference in advantages whenever it declared the challenge key to be either a master key or a key for the same identity as the challenge ciphertext would be negligible.)

$\mathcal{B}$ will not create this challenge key, but instead will encode the leakage $\mathcal{A}$ asks for on this key in **Phase 1** as a single polynomial time computable function f with domain $\mathbb{Z}_{p_2}^{n+1}$ and with an image of size $2^{\ell_{\mathrm{SK}}}$. It can do this by fixing the values of all other keys and fixing all other variables involved in the challenge key (more details on this below). $\mathcal{B}$ then receives a sample $(\vec{\delta}, f(\vec{\Gamma}))$, where $\vec{\Gamma}$ is either distributed as $\vec{\tau}$ or as $\vec{\tau}'$, in the notation of the corollary. $\mathcal{B}$ will use $f(\vec{\Gamma})$ to answer all of $\mathcal{A}$'s leakage queries on the challenge key by implicitly defining the challenge key as follows.

If the challenge key is not a master key, $\mathcal{B}$ chooses two more random values $r_1, r_2 \in \mathbb{Z}_{p_2}$. If the challenge key is a master key, it chooses $r_1, r_2, \theta \in \mathbb{Z}_{p_2}$. We let g_2 denote a generator of $\mathbb{G}_2$. $\mathcal{B}$ implicitly sets the $\mathbb{G}_2$ components of the key to be $g_2^{\vec{\Gamma}'}$, where $\vec{\Gamma}'$ is defined to be $\left(\vec{\Gamma}, 0\right) + (0, \ldots, 0, r_1, r_2)$ in the case of a key which is not a master key, and is defined to be $\left(\vec{\Gamma}, 0, 0\right) +$

$(0, \ldots, 0, r_1, r_2, 0) + (0, \ldots, 0, \theta)$ in the case of a master key. (Recall that $\vec{\Gamma}$ is of length $n+1$.) $\mathcal{B}$ defines the non-$\mathbb{G}_2$ components of the key to fit their appropriate distribution.

At some point, $\mathcal{A}$ declares the identity for the challenge ciphertext. If the challenge key was not a master key and the challenge ciphertext identity does not match the challenge key's identity, then $\mathcal{B}$ aborts the simulation and guesses whether $\vec{\Gamma}$ is orthogonal to $\vec{\delta}$ randomly. However, the simulation continues with non-negligible probability.

$\mathcal{B}$ chooses a random element $t_2 \in \mathbb{Z}_{p_2}$ subject to one of two constraints: if the challenge key is a master key, it chooses t_2 so that $\delta_{n+1}(r_1 - \mathcal{ID}\theta) + t_2 r_2 \equiv 0 \mod p_2$, where $\mathcal{ID}$ is the challenge ciphertext identity. If the challenge key is for the identity $\mathcal{ID}$, it chooses t_2 so that $\delta_{n+1} r_1 + t_2 r_2 \equiv 0 \mod p_2$. It then constructs the challenge ciphertext, using $\left(\vec{\delta}, 0\right) + (0, \ldots, 0, 0, t_2)$ as the challenge vector (recall that $\vec{\delta}$ is of length $n+1$). Now, if $\vec{\Gamma}$ is orthogonal to $\vec{\delta}$, then the challenge key is nominally semi-functional (and well-distributed as such). If $\vec{\Gamma}$ is not orthogonal to $\vec{\delta}$, then the challenge key is truly semi-functional (and also well-distributed).

It is clear that $\mathcal{B}$ can easily handle **Phase 2** queries, since the challenge key cannot be queried on here when it is a master key or has the same identity as the challenge ciphertext. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to gain a non-negligible advantage in distinguishing the distributions $(\vec{\delta}, f(\vec{\tau}))$ and $(\vec{\delta}, f(\vec{\tau'}))$. This violates Corollary B.1.1, since these distributions have a negligible statistical distance for f with this output size. $\square$

To prove Theorem 5.3, we will build a PPT simulator $\mathcal{B}$ that breaks the assumption $\mathsf{Comp2}$ with the help of a PPT attacker $\mathcal{A}$ that breaks one semi-functional key invariance of our system. $\mathcal{B}$ will simulate the game $\mathsf{MasterLeak}_b$. Initially the simulator $\mathcal{B}$ receives input from the assumption's challenger, i.e. $D^2 = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_3, g_1^z g_2^\nu, g_2^\mu g_3^\rho)$ and a challenge term T, which is equal either to $g_1^w g_3^\sigma$ or $g_1^w g_2^\kappa g_3^\sigma$. Algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ picks $(\alpha, x_1, x_2, \ldots, x_n, a, b) \overset{R}{\leftarrow} \mathbb{Z}_N^{n+3}$. It computes $u_1 = g_1^a, h_1 = g_1^b$, $e(g_1, g_1)^\alpha$, $g_1^{x_1}$, $g_1^{x_2}$, ..., and $g_1^{x_n}$. It gives the public parameters

$$\mathrm{PP} = (N, g_1, g_3, u_1, h_1, e(g_1, g_1)^\alpha, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$$

to $\mathcal{A}$ where N, g_1 and g_3 are given by the challenger. No keys are stored in tuple 0.

Phase 1: We recall that in game $\mathsf{MasterLeak}_b$, the challenger has to store in each tuple both a normal and a semi-functional version of each key. However, since for the challenge key our goal is to allow leakage on an unknown version depending on the challenge, we postpone the creation of all keys until the point where the attacker $\mathcal{A}$ decides that they should be normal, semi-functional, or challenge. Therefore, each **Create** query returns a handle and stores an unlocked tuple, but with the two key fields empty. Since the attacker only gets the handle from each such query, it cannot tell the difference.

Also, our simulator *will not store both versions of each key in the tuple*, in contrast to the game rules. It will store only the version that the attacker chose to get leakage from (or reveal). But then one could ask how the simulator is going to handle the **Create**(h, X) queries, when the h refers to a tuple with a semi-functional master key. The answer is that for our system, *knowledge of* α alone allows the creation of any type of key. Since the simulator knows α, it always bypasses the normal **Keygen** algorithm and creates totally legitimate keys.

Thus, in this phase, as well as in **Phase 2**, the simulator $\mathcal{B}$ has to successfully store the appropriate key on six different types of first leakage or reveal queries:

- $\mathcal{A}$ *requested a normal master key:* In this case $\mathcal{B}$ creates a normal master key by picking $(r, y_1, y_2, \ldots, y_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$, a random vector $\vec{\rho} = (\rho_1, \ldots, \rho_{n+2}) \xleftarrow{R} \mathbb{Z}_N^{n+2}$, and a random element $\rho_{n+3} \xleftarrow{R} \mathbb{Z}_N$. It stores the following key in the tuple along with lock-value $V = 0$:

  $$\text{MSK} = \left(\vec{K^*}, K^*\right) = \left(\left(g_1^{y_1}, \ldots, g_1^{y_n}, g_1^{\alpha} h_1^{-r} \prod_{i=1}^{n} g_1^{-x_i y_i}, g_1^r\right) * g_3^{\vec{\rho}}, u_1^r g_3^{\rho_{n+3}}\right)$$

  Obviously, this is properly distributed, since it the same method used in the Setup algorithm and this is also the same distribution that occurs when a normal master key is created by a call to the KeyGen algorithm with the empty string and a previously created master key as input.

- $\mathcal{A}$ *requested a semi-functional master key:* As in the previous case, $\mathcal{B}$ chooses $(r,\ y_1,\ y_2,\ \ldots,\ y_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$, $\vec{\rho} = (\rho_1,\ \ldots,\ \rho_{n+2}) \xleftarrow{R} \mathbb{Z}_N^{n+2}$, and $\rho_{n+3} \xleftarrow{R} \mathbb{Z}_N$. Also it picks $\vec{\theta'} \xleftarrow{R} \mathbb{Z}_N^n$ and $\theta' \xleftarrow{R} \mathbb{Z}_N$ and generates the following key:

$$\widetilde{\mathrm{MSK}} = \left(\vec{K}^*, K^*\right)$$
$$= \left(\left(g_1^{y_1}, \ldots, g_1^{y_n}, g_1^{\alpha} h_1^{-r} \prod_{i=1}^{n} g_1^{-x_i y_i}, g_1^{r}\right) * (g_2^{\mu} g_3^{\rho})^{\vec{\theta'}} * g_3^{\vec{\rho}}, u_1^{r} (g_2^{\mu} g_3^{\rho})^{\theta'} g_3^{\rho_{n+3}}\right)$$

where $g_2^{\mu} g_3^{\rho}$ is given by the assumption's challenger. It is easy to see that the $\mathbb{G}_1, \mathbb{G}_3$ parts are properly distributed. For the $\mathbb{G}_2$ part, the semi-functional parameters are $\vec{\theta} = \mu\vec{\theta'}$ and $\theta = \mu\theta'$. Thus, this part is properly distributed as well.

- $\mathcal{A}$ *requested a normal secret key:* In this case, $\mathcal{B}$ picks $(r', z_1, z_2, \ldots, z_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$, and a random vector $\vec{\rho'} \xleftarrow{R} \mathbb{Z}_N^{n+2}$. It creates the following key:

$$\mathrm{SK} = \vec{K}_1 = \left(g_1^{z_1}, g_1^{z_2}, \ldots, g_1^{z_n}, g_1^{\alpha} (u_1^{\mathcal{ID}} h_1)^{-r'} \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r'}\right) * g_3^{\vec{\rho'}}$$

- $\mathcal{A}$ *requested a semi-functional secret key:* Now $\mathcal{B}$ picks $(r', z_1, z_2, \ldots, z_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$, a random vector $\vec{\rho'} \xleftarrow{R} \mathbb{Z}_N^{n+2}$, and a random vector $\vec{\gamma'} \xleftarrow{R} \mathbb{Z}_N^{n+2}$. It generates the following key:

$$\widetilde{\mathrm{SK}} = \vec{K}_1 = \left(g_1^{z_1}, \ldots, g_1^{z_n}, g_1^{\alpha} (u_1^{\mathcal{ID}} h_1)^{-r'} \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r'}\right) * (g_2^{\mu} g_3^{\rho})^{\vec{\gamma'}} * g_3^{\vec{\rho'}}$$

As before, it is easy to see that the $\mathbb{G}_1, \mathbb{G}_3$ parts are properly distributed and, for the $\mathbb{G}_2$ part, the semi-functional parameters are $\vec{\gamma} = \mu\vec{\gamma'}$. Thus, this part is properly distributed as well.

- $\mathcal{A}$ *requested to be challenged on a master key:* Remember that now $\mathcal{B}$ is supposed to flip a coin and store either a normal or a semi-functional master key. Instead of doing this, it will use the assumption's challenge term T to generate the master key. To do so, it picks $(y'_1, y'_2, \ldots, y'_n) \overset{R}{\leftarrow} \mathbb{Z}_N^n$, $\vec{\rho} = (\rho_1, \ldots, \rho_{n+2}) \overset{R}{\leftarrow} \mathbb{Z}_N^{n+2}$, and $\rho_{n+3} \overset{R}{\leftarrow} \mathbb{Z}_N$ and generates:

$$\text{MSK} = \left(\vec{K}^*, K^*\right) = \left(\left(T^{y'_1}, \ldots, T^{y'_n}, g_1^{\alpha} T^{-b} \prod_{i=1}^{n} T^{-x_i y'_i}, T\right) * g_3^{\vec{\rho}}, T^a g_3^{\rho_{n+3}}\right)$$

As before, it is easy to see that the $\mathbb{G}_3$ part is properly distributed. We will now argue that the $\mathbb{G}_1$ and $\mathbb{G}_2$ parts are also well-distributed.

If $T = g_1^w g_2^{\kappa} g_3^{\sigma}$, then for the $\mathbb{G}_1$ part, this sets (remember that $u = g_1^a$ and $h = g_1^b$):

$$r = w \quad \text{and} \quad y_i = s y'_i \quad \forall i \in [1, n].$$

Thus, all parameters are properly distributed. For the $\mathbb{G}_2$ part, the semi-functional parameters are:

$$\vec{\theta} = \kappa \left(y'_1, \ldots, y'_n, -b - \sum x_i y'_i, 1\right) \quad \text{and} \quad \theta = \kappa a$$

. Since all terms $y'_1, \ldots, y'_n, a, b$ are only seen modulo p_1 in the public parameters, they appear random modulo p_2 here. Therefore, in this case, $\mathcal{B}$ has formed a properly distributed semi-functional master key.

It is easy to see that if $T = g_1^w g_3^{\sigma}$, the $\mathbb{G}_2$ part above is omitted and $\mathcal{B}$ has formed a properly distributed normal master key.

- $\mathcal{A}$ *requested to be challenged on a secret key:* Now $\mathcal{B}$ picks $(z'_1, z'_2, \ldots, z'_n) \overset{R}{\leftarrow} \mathbb{Z}_N^n$, and a random vector $\vec{\rho'} \overset{R}{\leftarrow} \mathbb{Z}_N^{n+2}$. It stores the following key:

$$\text{SK} = \vec{K}_1 = \left(T^{z'_1}, T^{z'_2}, \ldots, T^{z'_n}, g_1^{\alpha} T^{-(a\mathcal{ID}+b)} \prod_{i=1}^{n} T^{-x_i z'_i}, T\right) * g_3^{\vec{\rho'}},$$

where $\mathcal{ID}$ is the identity of this key given by the adversary $\mathcal{A}$.

If $T = g_1^w g_2^\kappa g_3^\sigma$, then this key is semi-functional with

$$r' = w \quad \text{and} \quad z_i = s z_i' \quad \forall i \in [1, n]$$

$$\vec{\gamma} = \kappa \left(z_1', \ldots, z_n', -(aI + b) - \sum x_i z_i', 1 \right)$$

For the same reasons as before, all vectors seem random in $\mathcal{A}$'s view[2].

That concludes **Phase 1**. We mention here that $\mathcal{B}$ works the same way in **Phase 2**.

Challenge Phase: In this phase, $\mathcal{B}$ has to create a semi-functional ciphertext with $\mathsf{EncryptSf}$. It gets two messages M_0 and M_1 and the challenge identity $\mathcal{ID}^*$ from $\mathcal{A}$ and chooses $c \overset{R}{\leftarrow} \{0, 1\}$. Then it generates the following ciphertext:

$$\widetilde{CT} = \left(C_0, \vec{C_1} \right) =$$
$$= \left(M_c \cdot e\left((g_1^z g_2^\nu), g_1^\alpha \right), \left((g_1^z g_2^\nu)^{x_1}, \ldots, (g_1^z g_2^\nu)^{x_n}, (g_1^z g_2^\nu), (g_1^z g_2^\nu)^{a\mathcal{ID}^* + b} \right) \right)$$

where $g_1^z g_2^\nu$ is given by the assumption's challenger.

It is easy to see that the ciphertext's parameters are

$$s = z \quad \text{and} \quad \vec{\delta} = \nu \left(x_1, \ldots, x_n, 1, a\mathcal{ID}^* + b \right).$$

[2]We recall that the last two cases exclude each other. We cannot have both a master key and a secret key picked by $\mathcal{A}$ as the challenge key. Thus, for example the term κ is only seen once modulo p_2.

Although, the s is obviously properly distributed, the semi-functional parameters δ are not (if the challenge key is capable of decrypting the ciphertext). We can argue that the terms $x_1, \dots, x_n$ seem random modulo p_2 to the adversary (and ν) as before, but we can not do the same for $a\mathcal{ID}^* + b$. This happens, because it might be the case that a, b have been seen modulo p_2 (if the challenge key is the master key) or $a\mathcal{ID}^* + b$ is seen (if the identity of the challenge key $\mathcal{ID}$ is equal to $\mathcal{ID}^*$ modulo p_2 or the identity of the challenge key is the challenge identity). However, lemmas 5.4 and 5.5 assert that the change in any adversary's advantage is negligible.

Lemma 5.4 states that if the simulator $\mathcal{B}$ aborts and guesses a random value for the assumption in case it detects that $\mathcal{ID} = \mathcal{ID}^* \bmod p_2$ and $\mathcal{ID} \neq \mathcal{ID}^*$ (it can do that with N), the loss in advantage is only a negligible amount. Otherwise, the ciphertext is well-distributed when $\mathcal{ID} \neq \mathcal{ID}^*$, because the $a\mathcal{ID} + b$ in the secret key is uncorrelated to the $a\mathcal{ID}^* + b$ of the ciphertext.

On the other hand, notice that if $\mathcal{A}$ picks a master key as the challenge key, this will be nominally semi-functional with respect to the challenge ciphertext (i.e. the following holds modp_2):

$$\begin{aligned}&\vec{\delta} \cdot \left(\vec{\theta} + (0, \dots, 0, -\mathcal{ID}^*\theta, 0)\right)\\&= \nu\left(x_1, \dots, x_n, 1, a\mathcal{ID}^* + b\right) \cdot \kappa\left(y'_1, \dots, y'_n, -a\mathcal{ID}^* - b - \sum x_i y'_i, 1\right) = 0\end{aligned}$$

The same happens when the challenge key is a secret key for identity

$\mathcal{ID}^*$:

$$\begin{aligned}\vec{\delta}\cdot\vec{\gamma} &= \nu\left(x_1,\ldots,x_n,1,a\mathcal{ID}^*+b\right)\cdot\kappa\left(z'_1,\ldots,z'_n,-(a\mathcal{ID}^*+b)-\sum x_i z'_i,1\right)\\ &= 0 \bmod p_2\end{aligned}$$

Therefore, since $\vec{\delta}$ modulo p_2 has all terms random but one, it is distributed the same modulo p_2 as if it were chosen uniformly at random from the orthogonal complement of the key's semi-functional parameters modulo p_2. Remember that the above is true, only if $T = g_1^w g_2^\kappa g_3^\sigma$ and $\mathcal{B}$ simulates the $\mathsf{MasterLeak}_1$ game. Then, according to Lemma 5.5, no PPT adversary can distinguish this from a truly random vector. Thus, the ciphertext seems properly distributed to the attacker.

In summary, if $T = g_1^w g_3^\sigma$, algorithm $\mathcal{B}$ simulates a game in which $\mathcal{A}$'s advantage is only negligibly different from its advantage in the $\mathsf{MasterLeak}_0$ game, and if $T = g_1^w g_2^\kappa g_3^\sigma$, $\mathcal{B}$ simulates a game in which $\mathcal{A}$'s advantage is only negligibly different from its advantage in the $\mathsf{MasterLeak}_1$ game. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to break the assumption $\mathsf{Comp2}$ with non-negligible advantage. □

Semi-functional Security

Theorem 5.6. *If the assumption* $\mathsf{Comp3}$ *holds, our system has* $(\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})$*-semi-functional security.*

Proof. We will build a PPT simulator $\mathcal{B}$ that breaks the assumption $\mathsf{Comp3}$ with the help of a PPT attacker $\mathcal{A}$ that breaks the semi-functional security of our system.

The input from the assumption's challenger to $\mathcal{B}$ is $D^3 = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_2, g_3, g_1^{\alpha}g_2^{\nu}, g_1^{z}g_2^{\mu})$ and a challenge term T which is either $e(g_1, g_1)^{\alpha z}$ or a random term of $\mathbb{G}_T$. Algorithm $\mathcal{B}$ works as follows:

Setup phase: $\mathcal{B}$ picks $(x_1, x_2, \ldots, x_n, a, b) \xleftarrow{R} \mathbb{Z}_N^{n+2}$. It computes $u_1 = g_1^a, h_1 = g_1^b$, and $g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n}$. The term $e(g_1, g_1)^{\alpha}$ is computed as $e(g_1^{\alpha}g_2^{\nu}, g_1)$. (Notice that now α is unknown to $\mathcal{B}$.) It gives the public parameters PP $= (N, g_1, g_3, u_1, h_1, e(g_1, g_1)^{\alpha}, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$ to $\mathcal{A}$.

Phase 1: Although our simulator does not know α, it can still create properly distributed semi-functional keys, which are the only ones needed for this game. Now it bypasses the KeyGenSf algorithm using the challenge term $g_1^{\alpha}g_2^{\nu}$.

For **Create** queries on a master key (as well as the key of tuple 0), the simulator picks $(r, y_1, y_2, \ldots, y_n, \rho_{n+3}, \theta') \xleftarrow{R} \mathbb{Z}_N^{n+3}$ and two random vectors $\vec{\rho}, \vec{\theta'} \xleftarrow{R} \mathbb{Z}_N^n$ and constructs:

$$MK = \left(\vec{K}^*, K_u^*\right) = \left(\left(g_1^{y_1}, \ldots, g_1^{y_n}, (g_1^{\alpha}g_2^{\nu})h_1^{-r}\prod_{i=1}^{n} g_1^{-x_i y_i}, g_1^{r}\right) * g_2^{\vec{\theta'}} * g_3^{\vec{\rho}}, u_1^{r}g_2^{\theta'}g_3^{\rho}\right)$$

Remember that $g_1^{\alpha}g_2^{\nu}$ is given by the assumption's challenger. The above is a properly distributed semi-functional master key, with semi-functional parameters $\vec{\theta} = (0, \ldots, 0, \nu, 0) + \vec{\theta'}$ and $\theta = \theta'$.

For all secret keys requested by the adversary on identity $\mathcal{ID}$, the simulator creates and stores the following semi-functional keys:

$$SK = \vec{K}_1 = \left(g_1^{z_1}, \ldots, g_1^{z_n}, (g_1^{\alpha}g_2^{\nu})(u_1^{\mathcal{ID}}h_1)^{-r'}\prod_{i=1}^{n} g_1^{-z_i x_i}, g_1^{r'}\right) * g_2^{\vec{\gamma'}} * g_3^{\vec{\rho'}},$$

where the vectors $\vec{\rho}', \vec{\gamma}' \xleftarrow{R} \mathbb{Z}_N^n$ and $(r', z_1, \ldots, z_n) \xleftarrow{R} \mathbb{Z}_N^{n+1}$ are picked independently for each generated key. It is easy to see that the above is a properly distributed semi-functional key with semi-functional parameters $\vec{\gamma} = (0, \ldots, 0, \nu, 0) + \vec{\gamma}'$.

Challenge Phase: The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages M_0 and M_1 and the challenge identity $\mathcal{ID}^*$. The simulator $\mathcal{B}$ chooses $c \xleftarrow{R} \{0,1\}$ and outputs the following ciphertext:

$$\begin{aligned} CT &= \left(C_0, \vec{C}_1\right) = \\ &= \left(M_c \cdot T, \left((g_1^z g_2^\mu)^{x_1}, \ldots, (g_1^z g_2^\mu)^{x_n}, (g_1^z g_2^\mu), (g_1^z g_2^\mu)^{a\mathcal{ID}^*+b}\right)\right), \end{aligned}$$

where $g_1^z g_2^\mu$ is given by the assumption's challenger and T is the challenge term.

Phase 2: $\mathcal{B}$ works in the same way as **Phase 1**.

If $T = e(g_1, g_1)^{\alpha z}$, then we get a semi-functional ciphertext of M_c with parameters:

$$s = z \quad \text{and} \quad \vec{\delta} = (\mu x_1, \ldots, \mu x_n, \mu, \mu(a\mathcal{ID}^* + b))$$

As before, $\vec{\delta}$ is properly distributed since all terms $x_1, \ldots, x_n, a\mathcal{ID}^* + b$ are random modulo p_2. Therefore, $\mathcal{B}$ has properly distributed game $\mathsf{MasterLeakCK}$.

On the other hand, if $T \xleftarrow{R} \mathbb{G}_T$, the term C_0 is entirely random and we get a semi-functional ciphertext of a random message. Therefore, the value of c is information-theoretically hidden and the probability of success of any algorithm $\mathcal{A}$ in this game is exactly $1/2$, since $c \xleftarrow{R} \{0,1\}$. Thus, $\mathcal{B}$ can use the output of $\mathcal{A}$ to break the assumption Comp3 with non-negligible advantage. $\square$

This concludes the proof of Theorem 5.1.

5.2 Leakage-Resilient CP-ABE

5.2.1 Construction

The algorithms of our CP-ABE system are the following:

Setup$(1^\lambda, \mathcal{U}) \rightarrow (\mathrm{PP}, \mathrm{MSK})$**:** The setup algorithm calls the bilinear group generation algorithm for composite order groups $\mathcal{G}(1^\lambda) \rightarrow (N,\ p_1,\ p_2,\ p_3,\ g,\ \mathbb{G},\ \mathbb{G}_T,\ e)$.

It picks two random exponents $\alpha, a \overset{R}{\leftarrow} \mathbb{Z}_N$. We note that $\mathcal{U}$ denotes the universe of attributes. Therefore $|\mathcal{U}|$ is polynomial in the security parameter λ and the scheme is a small universe construction. For each attribute $i \in \mathcal{U}$, it chooses random $s_i \overset{R}{\leftarrow} \mathbb{Z}_N$. It also picks n random exponents $x_1, x_2, \ldots, x_n \overset{R}{\leftarrow} \mathbb{Z}_N$ to get the required vectors. For the master key, it picks $t^*, y_1, \ldots, y_n \in \mathbb{Z}_N$ and $\vec{\rho} \overset{R}{\leftarrow} \mathbb{Z}_N^{n+1}, \rho_{n+2} \overset{R}{\leftarrow} \mathbb{Z}_N, \forall i \in U \quad \rho'_i \overset{R}{\leftarrow} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part.

It outputs the following public parameters and master key:

$$\mathrm{PP} = \big(N, g_1, g_3, g_1^a, e(g_1, g_1)^\alpha, g_1^{x_1}, \ldots, g_1^{x_n}, \{T_i = g_1^{s_i}\}_{i \in \mathcal{U}}\big)$$

$$\mathrm{MSK} = \Big(\mathcal{U}, \vec{K_1^*}, L^*, \{K_i^*\}_{i \in \mathcal{U}}\Big) =$$

$$= \left(\mathcal{U}, \left(g_1^{y_1}, \ldots, g_1^{y_n}, g_1^\alpha g_1^{at^*} \prod_{i=1}^{n} g_1^{-x_i y_i}\right) * g_3^{\vec{\rho}}, g_1^{t^*} g_3^{\rho_{n+2}}, \left\{T_i^{t^*} g_3^{\rho'_i}\right\}_{i \in \mathcal{U}}\right)$$

Notice that $\vec{K}_1^*$ has $n+1$ elements.

$\mathsf{KeyGen}(\mathrm{MSK}, \mathcal{S}, \mathrm{PP}) \to \mathrm{SK}$**:** $\mathcal{S}$ denotes a set of attributes, $\mathcal{S} \subseteq \mathcal{U}$. The key generation algorithm chooses random values $t, z_1, \ldots, z_n \in \mathbb{Z}_N$ and random exponents $\vec{\rho} \xleftarrow{R} \mathbb{Z}_N^{n+1}, \rho_{n+2} \xleftarrow{R} \mathbb{Z}_N, \forall i \in \mathcal{S} \;\; \rho_i' \xleftarrow{R} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part. The secret key it generates is the following:

$$\begin{aligned}\mathrm{SK} &= \left(\mathcal{S}, \vec{K}_1, L, \{K_i\}_{i \in \mathcal{S}}\right) = \\ &= \left(\mathcal{S}, \vec{K}_1^* * \left(g_1^{z_1}, \ldots, g_1^{z_n}, g_1^{at} \prod_{i=1}^{n} g_1^{-x_i z_i}\right) * g_3^{\vec{\rho}}, L^* g_1^t g_3^{\rho_{n+2}}, \left\{K_i^* T_i^t g_3^{\rho_i'}\right\}_{i \in \mathcal{S}}\right)\end{aligned}$$

The update or re-randomization of a secret key is done using the secret key in question instead of the master secret key and the same attribute set $\mathcal{S}$. In case we want to re-randomize a master key, we use $\mathcal{S} = \mathcal{U}$.

$\mathsf{Encrypt}(M, (A, \delta)) \to \mathrm{CT}$**:** A is an $n_1 \times n_2$ LSSS matrix and δ is a mapping from each row A_x of A to an attribute $\delta(x) \in \mathcal{U}$. The algorithm picks a random vector $\vec{v} = (s, v_2, \ldots, v_{n_2}) \xleftarrow{R} \mathbb{Z}_N^{n_2}$. For each row A_x, it picks a random exponent $r_x \xleftarrow{R} \mathbb{Z}_N$. The ciphertext generated is the following:

$$\begin{aligned}\mathrm{CT} &= \left((A, \delta), C_0, \vec{C}_1, \{C_x, D_x\}_{x \in [n_1]}\right) = \\ &= \left((A, \delta), M \cdot (e(g_1, g_1)^{\alpha})^s, ((g_1^{x_1})^s, \ldots, (g_1^{x_n})^s, g_1^s)\right., \\ &\qquad \left.\left\{g_1^{a\langle A_x, \vec{v}\rangle} T_{\rho(x)}^{-r_x}, g_1^{r_x}\right\}_{x \in [n_1]}\right)\end{aligned}$$

$\mathsf{Decrypt}(\mathrm{CT}, \mathrm{SK}) \to M$**:** First the decryption algorithm computes constants $\omega_x \in \mathbb{Z}_N$ for every row of A (note that A is given in the ciphertext) such that $\sum_{\delta(x)\in S}\omega_x A_x = (1,0,\ldots,0) \in \mathbb{Z}_N^{n_2}$. To calculate the blinding factor, it computes:

$$\begin{aligned}
&\frac{e_{n+1}(\vec{C}_1,\vec{K}_1)}{\prod_{\delta(x)\in S}\left(e(C_x,L)e(D_x,K_{\delta(x)})\right)^{\omega_x}} = \\
&\quad = \frac{e(g_1,g_1)^{\alpha s}e(g_1,g_1)^{sat}\cdot\prod_{i=1}^{n}e(g_1,g_1)^{-sx_iz_i}\cdot\prod_{i=1}^{n}e(g_1,g_1)^{sx_iz_i}}{\prod_{\delta(x)\in S}\left(e(g_1,g_1)^{at\langle A_x,\vec{v}\rangle}e(g_1,g_1)^{-r_xs_{\rho(x)}t}e(g_1,g_1)^{r_xs_{\rho(x)}t}\right)^{\omega_x}} = \\
&\quad = \frac{e(g_1,g_1)^{\alpha s}e(g_1,g_1)^{sat}}{e(g_1,g_1)^{at\left\langle\sum_{\delta(x)\in S}\omega_xA_x,\vec{v}\right\rangle}} = \\
&\quad = e(g_1,g_1)^{\alpha s}
\end{aligned}$$

In the above calculation, the values t, z_i are meant denote the exponents of the secret key.

5.2.2 Semi-Functionality

In this section, we present the algorithms for creating semi-functional ciphertexts and secret keys for our CP-ABEsystem. In contrast to our previous systems, we now have two different types of semi-functional keys, called Type 1 and Type 2. Hence we have two different $\mathsf{KeyGenSf}$ algorithms. Another difference is that for every attribute $i \in \mathcal{U}$, random values $q_i \xleftarrow{R} \mathbb{Z}_N$ are chosen before the execution of any algorithm and are shared by the semi-functional ciphertexts and keys - they work similar to public parameters for the semi-functional algorithms. The algorithms are the following:

$\mathsf{KeyGenSf}_1(\text{MSK},\mathcal{S}) \to \widetilde{K}$ To create a semi-functional key of type 1, this algorithm first calls $\mathsf{KeyGen}(\text{MSK},\mathcal{S})$ and gets the key $K = (\mathcal{S},\vec{K}_1,L,$ $\{K_i\}_{i\in\mathcal{S}})$ (Notice that this can be a master key, if $\mathcal{S} = \mathcal{U}$). Then it picks

$\vec{\gamma} \xleftarrow{R} \mathbb{Z}_N^{n+1}$ and $\theta \xleftarrow{R} \mathbb{Z}_N$ and outputs

$$\widetilde{K} = \left(\mathcal{S}, \vec{K_1} * g_2^{\vec{\gamma}}, L g_2^{\theta}, \left\{K_i g_2^{\theta q_i}\right\}_{i \in \mathcal{S}}\right)$$

$\mathsf{KeyGenSf}_2(\mathrm{MSK}, \mathcal{S}) \to \widetilde{K}$ A semi-functional key of type 2 is generated the same way but without the terms g_2^{θ} and $g_2^{\theta q_i}$ (i.e. we now set $\theta = 0$). It outputs

$$\widetilde{K} = \left(\mathcal{S}, \vec{K_1} * g_2^{\vec{\gamma}}, L, \{K_i\}_{i \in \mathcal{S}}\right)$$

$\mathsf{EncryptSf}(M, (A, \delta)) \to \widetilde{\mathrm{CT}}$ This algorithm first calls the normal encryption algorithm $\mathsf{Encrypt}(M, (A, \delta))$ to get the ciphertext $\mathrm{CT} = ((A, \delta), C_0, \vec{C_1}, \{C_x, D_x\}_{x \in [n_1]})$. Then it picks $\vec{\delta} \xleftarrow{R} \mathbb{Z}_N^{n+1}$, a random vector $\vec{u} \xleftarrow{R} \mathbb{Z}_N^{n_2}$ (recall n_2 is the number of columns of A), and for every row A_x of A, it chooses $\delta'_x \xleftarrow{R} \mathbb{Z}_N$. It outputs

$$\widetilde{\mathrm{CT}} = \left((A, \delta), C_0, \vec{C_1} * g_2^{\vec{\delta}}, \left\{C_x g_2^{A_x \cdot \vec{u} + \delta'_x q_{\rho(x)}}, D_x g_2^{-\delta'_x}\right\}_{x \in [n_1]}\right)$$

Notice the use of $q_{\rho(x)}$, which are the same q's used by the $\mathsf{KeyGenSf}_1$ algorithm.

If we use the $\mathsf{Decrypt}$ algorithm to decrypt a semi-functional ciphertext with a semi-functional key, we get the extra term

$$e(g_2, g_2)^{\left\langle \vec{\gamma}, \vec{\delta} \right\rangle - \theta u_1},$$

where u_1 denotes the first coordinate of vector $\vec{u}$ picked during $\mathsf{EncryptSf}$. Hence we call a semi-functional key (of type 1 or type 2) nominally semi-functional with respect to a semi-functional ciphertext if $\left\langle \vec{\gamma}, \vec{\delta} \right\rangle - \theta u_1 = 0 \bmod p_2$.

5.2.3 Security Proof

Our ABE construction has two types of semi-functional key generation algorithms instead of one. We define here that *the semi-functional key generation algorithm used by game* $\mathsf{MasterLeak}_b$ *generates keys of type 2.* Essentially the main idea is to convert all keys to semi-functional keys of type 2. Type 1 keys serve as a "stepping stone" between the games $\mathsf{MasterLeak}_0$ and $\mathsf{MasterLeak}_1$. Remember that if these two games are indistinguishable, our scheme has one semi-functional key invariance. However, our assumptions do not allow us to go in one step from one game to the other. To achieve that, we add an intermediate game, called $\mathsf{MasterLeak}_{1/2}$, which is defined the exact same way as $\mathsf{MasterLeak}_b$, but with the difference that the challenger always uses a semi-functional key of *type 1* for the challenge key. All other semi-functional keys are of type 2.

We now give the proofs of semi-functional ciphertext invariance, one semi-functional key invariance (split in two parts), and semi-functional security for our system.

Theorem 5.7. *If the assumption* $\mathsf{Comp1}$ *holds, our system has* $(\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})$*-semi-functional ciphertext invariance.*[3]

Proof. We assume we have a PPT attacker $\mathcal{A}$ which breaks semi-functional ciphertext invariance of our system. We will create a PPT algorithm $\mathcal{B}$ which breaks the assumption $\mathsf{Comp1}$ with non-negligible advantage. The simulator

[3]For this theorem to be true the leakage bounds $\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}}$ can take any values in $\mathbb{N} \cup \{0\}$.

$\mathcal{B}$ plays the $\mathsf{MasterLeakAbe}$ or the $\mathsf{MasterLeakC}$ game with the attacker $\mathcal{A}$ in the following way:

Setup phase: $\mathcal{B}$ picks $\alpha, a \xleftarrow{R} \mathbb{Z}_N$ and for each attribute $i \in \mathcal{U}$, it chooses random $s_i \xleftarrow{R} \mathbb{Z}_N$. It also picks n random exponents $x_1, x_2, \ldots, x_n \xleftarrow{R} \mathbb{Z}_N$. It gives the public parameters

$$\mathrm{PP} = (N, g_1, g_3, g_1^a, e(g_1, g_1)^\alpha, g_1^{x_1}, \ldots, g_1^{x_n}, \forall i \in \mathcal{U} \;\; T_i = g_1^{s_i})$$

to $\mathcal{A}$, where N, g_1, and g_3 are given by the challenger.

Phase 1: Knowing α, the simulator can generate a normal master key as in the Setup algorithm and execute all secret key queries (create, leaked, keygen) with this master key.

Challenge Phase: The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages M_0 and M_1 and an access structure, encoded as an $n_1 \times n_2$ LSSS matrix: (A^*, δ^*). The simulator $\mathcal{B}$ chooses random values $v'_2, \ldots, v'_{n_2} \xleftarrow{R} \mathbb{Z}_N$ and for each row A^*_x of A^* one value $r_x \xleftarrow{R} \mathbb{Z}_N$. Using the v' values, it creates the vector $\vec{v'} = \left(1, v'_2, \ldots, v'_{n_2}\right)$. It flips a random coin $c \xleftarrow{R} \{0,1\}$ and outputs the ciphertext:

$$\begin{aligned} \mathrm{CT} = \Big((A^*, \delta^*), C_0, \vec{C_1}, \forall x \;\; C_x, \forall x \;\; D_x\Big) = \\ = \Bigg((A^*, \delta^*), M \cdot (e(T, g_1^\alpha))^s, (T^{x_1}, \ldots, T^{x_n}, T)\,, \\ \forall x \;\; T^{aA^*_x \cdot \vec{v'}} T^{-r'_x s_{\delta^*(x)}}, \forall x \;\; T^{r'_x}\Bigg), \end{aligned}$$

where T is the challenge term from the assumption.

Phase 2: $\mathcal{B}$ works in the same way as **Phase 1**.

If $T = g_1^z g_2^\nu$, then the ciphertext is semi-functional, since

$$\begin{array}{ll}
C_0 = M_c \cdot e\left(g_1^z g_2^\nu, g_1^\alpha\right) = M \cdot e(g_1, g_1)^{\alpha z} & \\
T^{x_i} = (g_1^{x_i})^z g_2^{\nu x_i} & \text{for } i \in \{1, 2, \ldots, n\} \\
T = g_1^z g_2^\nu & \\
T^{a A_x^* \cdot \vec{v}'} T^{-r_x' s_{\delta^*(x)}} = g_1^{a A_x^* \cdot z \vec{v}'} g_1^{-z r_x' s_{\delta^*(x)}} \cdot g_2^{A_x^* \cdot a \nu \vec{v}' - \nu r_x' s_{\delta^*(x)}} & \text{for every row } x \text{ of } A^* \\
T^{r_x'} = g_1^{z r_x'} \cdot g_2^{\nu r_x'} & \text{for every row } x \text{ of } A^*
\end{array}$$

For the $\mathbb{G}_1$ part, this implicitly sets $s = z$, $\vec{v} = z\vec{v'}$ and $r_x = z r_x'$. Thus all the $\mathbb{G}_1$ parts are properly distributed (remember that the first coordinate of $\vec{v}$ should be z).

For the $\mathbb{G}_2$ parts, this sets $\vec{\delta} = \nu\,(x_1, \ldots, x_n, 1)$, $\vec{u} = a\nu\vec{v'}$, $\delta_x' = -\nu r_x'$, and $q_{\rho*(x)} = s_{\rho*(x)}$. All the terms have been re-used only in the $\mathbb{G}_1$ part; hence they look random and uncorrelated modulo p_2 in the adversary's view. In other words, uniform randomness of the semi-functional parameters follows from uniform randomness modulo p_2 of the following terms: $x_1, x_2, \ldots, x_n, \nu, a, v_2', \ldots, v_{n_2}', r_x', s_{\delta^*(x)}$. So this is a properly distributed semi-functional ciphertext, and $\mathcal{B}$ has properly simulated the $\mathsf{MasterLeakC}$ game.

If on the other hand, if $T = g_1^z$, it is easy to see that the ciphertext is normal since it has no $\mathbb{G}_2$ parts and $\mathcal{B}$ has properly simulated the $\mathsf{MasterLeakAbe}$ game. $\square$

Theorem 5.8. *If the assumption* $\mathsf{Comp2}$ *holds, the difference between the advantages of any PPT attacker when playing the* $\mathsf{MasterLeak}_0$ *and* $\mathsf{MasterLeak}_{1/2}$ *games with leakage* $(\ell_{\text{MSK}}, \ell_{\text{SK}})$ *on our ABE system with the unique attribute restriction is negligible in* λ.

Proof. We suppose we have a PPT attacker $\mathcal{A}$ whose advantage changes non-negligibly between these two games. We will create a PPT algorithm $\mathcal{B}$ which breaks the assumption $\mathsf{Comp2}$ with non-negligible advantage. The simulator $\mathcal{B}$ will play either the $\mathsf{MasterLeak}_0$ or the $\mathsf{MasterLeak}_{1/2}$ game with the attacker $\mathcal{A}$. Recall that in the former game, all keys are either semi-functional of type 2 or normal (according to the attacker's choice) and the challenge key is normal. The latter game is the same, except the challenge key is semi-functional of type 1.

Setup phase: $\mathcal{B}$ picks $\alpha, a \xleftarrow{R} \mathbb{Z}_N$ and for each attribute $i \in \mathcal{U}$, it chooses random $s_i \xleftarrow{R} \mathbb{Z}_N$. It also picks n random exponents $x_1, x_2, \ldots, x_n \xleftarrow{R} \mathbb{Z}_N$. It gives the public parameters

$$\mathrm{PP} = (N, g_1, g_3, g_1^a, e(g_1, g_1)^\alpha, g_1^{x_1}, \ldots, g_1^{x_n}, \forall i \in \mathcal{U} \;\; T_i = g_1^{s_i})$$

to $\mathcal{A}$, where N, g_1, and g_3 are given by the challenger.

Phase 1: Knowing α, the simulator can generate a normal master key as in the Setup algorithm and answer all secret key queries for normal keys (remember that $\mathcal{A}$ queries for either a semi-functional or a normal key).

For semi-functional keys (of type 2), the simulator picks $t, z_1, \ldots, z_n \in \mathbb{Z}_N$ and random exponents $\vec{\rho} \xleftarrow{R} \mathbb{Z}_N^{n+1}$, $\rho_{n+2} \xleftarrow{R} \mathbb{Z}_N, \forall i \in \mathcal{S} \;\; \rho'_i \xleftarrow{R} \mathbb{Z}_N$ for the

$\mathbb{G}_3$ part. It uses the following secret key:

$$\begin{aligned}\text{SK} &= \left(\mathcal{S}, \vec{K}_1, L, \forall i \in \mathcal{S} \;\; K_i\right) = \\ &= \left(\mathcal{S}, \left(g_1^{z_1}, \ldots, g_1^{z_n}, g_1^{\alpha} g_1^{at} \prod_{i=1}^{n} g_1^{-x_i z_i}\right) * (g_2^{\mu} g_3^{\rho})^{\vec{\rho}}, g_1^t g_3^{\rho_{n+2}}, \forall i \in \mathcal{S} \;\; T_i^t g_3^{\rho_i'}\right),\end{aligned}$$

where $\mathcal{S}$ is the set of attributes given by $\mathcal{A}$ and $g_2^{\mu} g_3^{\rho}$ comes from the challenger. It is easy to see that this is a properly distributed semi-functional key of type 2.

For the challenge key, the simulator has to either give a normal key or a semi-functional key of type 1. To do this, it will use the assumption's challenge term T. It picks $z_1', \ldots, z_n' \in \mathbb{Z}_N$ and random exponents $\vec{\rho} \xleftarrow{R} \mathbb{Z}_N^{n+1}$, $\rho_{n+2} \xleftarrow{R} \mathbb{Z}_N, \forall i \in S \;\; \rho_i' \xleftarrow{R} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part. It uses the secret key:

$$\begin{aligned}\text{SK} &= \left(\mathcal{S}, \vec{K}_1, L, \forall i \in \mathcal{S} \;\; K_i\right) = \\ &= \left(\mathcal{S}, \left(T^{z_1'}, \ldots, T^{z_n'}, g_1^{\alpha} T^{a} \prod_{i=1}^{n} T^{-x_i z_i'}\right) * g_3^{\vec{\rho}}, T g_3^{\rho_{n+2}}, \forall i \in \mathcal{S} \;\; T^{s_i} g_3^{\rho_i'}\right)\end{aligned}$$

It is easy to see that the $\mathbb{G}_3$ parts are properly distributed. For the $\mathbb{G}_1$ parts, this always sets $t = w$ and $z_i = s z_i'$ for all $i \in [1, n]$ (remember that $T = g_1^w g_3^{\sigma}$ or $g_1^w g_2^{\kappa} g_3^{\sigma}$). Thus this part is always well-distributed.

If $T = g_1^w g_2^{\kappa} g_3^{\sigma}$, the key has $\mathbb{G}_2$ parts as well and we can see that it is a semi-functional key of type 1 with parameters:

$$\vec{\gamma} = \kappa \left(z_1', \ldots, z_n', a - \sum x_i z_i'\right) \quad , \quad \theta = \kappa \;\text{ and }\; q_i = s_i.$$

Since the terms $z_1', \ldots, z_n', \kappa, s_i$ are random modulo p_2, the key is properly distributed.

Challenge Phase: The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages M_0 and M_1 and an access structure, encoded as an $n_1 \times n_2$ LSSS matrix: (A^*, δ^*). The simulator $\mathcal{B}$ chooses random values $v'_2, \ldots, v'_{n_2} \overset{R}{\leftarrow} \mathbb{Z}_N$ and for each row A^*_x of A^* and one value $r'_x \overset{R}{\leftarrow} \mathbb{Z}_N$. Using the v' values, it creates the vector $\vec{v'} = \left(1, v'_2, \ldots, v'_{n_2}\right)$. It flips a random coin $c \overset{R}{\leftarrow} \{0,1\}$ and outputs the ciphertext:

$$
\begin{aligned}
\mathrm{CT} &= \Big((A^*, \delta^*), C_0, \vec{C_1}, \forall x \;\; C_x, \forall x \;\; D_x\Big) = \\
&= ((A^*, \delta^*), M \cdot (e((g_1^z g_2^\nu), g_1^\alpha)), ((g_1^z g_2^\nu)^{x_1}, \ldots, (g_1^z g_2^\nu)^{x_n}, (g_1^z g_2^\nu)), \\
&\quad \forall x \;\; (g_1^z g_2^\nu)^{aA^*_x \cdot \vec{v'}} (g_1^z g_2^\nu)^{-r'_x s_{\delta^*(x)}}, \forall x \;\; (g_1^z g_2^\nu)^{r'_x}\Big),
\end{aligned}
$$

where $g_1^z g_2^\nu$ is given from the assumption.

The ciphertext is semi-functional since

$$
\begin{aligned}
&C_0 = M_c \cdot e\left(g_1^z g_2^\nu, g_1^\alpha\right) = M \cdot e(g_1, g_1)^{\alpha z} \\
&(g_1^z g_2^\nu)^{x_i} = (g_1^{x_i})^z g_2^{\nu x_i} \text{ for } i \in \{1, 2, \ldots, n\} \\
&(g_1^z g_2^\nu) = g_1^z g_2^\nu
\end{aligned}
$$

$$
\begin{aligned}
&(g_1^z g_2^\nu)^{aA^*_x \cdot \vec{v'}} (g_1^z g_2^\nu)^{-r'_x s_{\delta^*(x)}} = g_1^{aA^*_x \cdot z\vec{v'}} T_{\delta^*(x)}^{-zr'_x} \cdot g_2^{A^*_x \cdot a\nu\vec{v'} - \nu r'_x s_{\delta^*(x)}} \\
&\quad \text{for every row } x \text{ of } A^* \\
&(g_1^z g_2^\nu)^{r'_x} = g_1^{zr'_x} \cdot g_2^{\nu r'_x} \text{ for every row } x \text{ of } A^*
\end{aligned}
$$

For the $\mathbb{G}_1$ parts, this implicitly sets $s = z$, $\vec{v} = z\vec{v'}$, and $r_x = zr'_x$. Thus all are properly distributed (remember that the first coordinate of $\vec{v}$ should be z).

For the $\mathbb{G}_2$ parts, this sets $\vec{\delta} = \nu\,(x_1, \ldots, x_n, 1)$, $\vec{u} = a\nu\vec{v'}$, $\delta'_x = -\nu r'_x$ and $q_{\rho*(x)} = s_{\rho*(x)}$.

First, notice that $q_{\delta*(x)} = s_{\delta*(x)}$ as in the challenge key if it happens to be of type 1. This is what we want since the q_i values used by $\mathsf{KeyGenSf}_1$ and $\mathsf{EncryptSf}$ should be the same. We recall here that type 2 keys do not have q_i terms.

The remaining semi-functional parameters of both the challenge key (if it is semi-functional of type 1) and the ciphertext are shown below:

$$\begin{gathered}\text{Secret key}\\ \vec{\gamma} = \kappa\,(z'_1, \ldots, z'_n, a - \textstyle\sum x_i z'_i) \quad \theta = \kappa\end{gathered}$$

$$\begin{gathered}\text{Ciphertext}\\ \vec{\delta} = \nu\,(x_1, \ldots, x_n, 1) \quad \vec{u} = a\nu\left(1, v'_2, \ldots, v'_{n_2}\right) \quad \delta_x = -\nu r'_x\end{gathered}$$

We note that the first term of vector $\vec{u}$ is always equal to $a\nu$. Both a and ν are "seen" by the adversary modulo p_2: in the last coordinate of $\vec{\gamma}$ and $\vec{\delta}$, respectively (for the last of $\vec{\gamma}$, we note that κ and all x_i, z'_i's are seen in other terms).

The first and easier case is when the attributes of the key satisfy the challenge access structure. Then this is nominal with respect to the ciphertext because:

$$\vec{\gamma}\cdot\vec{\delta} - \theta u_1 = \kappa\left(z'_1, \ldots, z'_n, a - \sum x_i z'_i\right)\cdot \nu\,(x_1, \ldots, x_n, 1) - \kappa a\nu = 0 \bmod p_2$$

According to the rules of the game, this key can not be revealed to the adversary, but only leakage is allowed on it.We can show that no PPT attacker can have more than negligible advantage in distinguishing these two keys. The required lemma is the following:

Lemma 5.9. *We suppose the leakage is at most* $(\ell_{\text{MSK}} = (n-1-2c)\log(p_2), \ell_{\text{SK}} = (n-1-2c)\log(p_2))$*, where* $c > 0$ *is a fixed positive constant. Then, for any PPT adversary* $\mathcal{A}$*,* $\mathcal{A}$*'s advantage in the* $\mathsf{MasterLeak}_1$ *game changes only by a negligible amount when the truly semi-functional challenge key is replaced by a nominal semi-functional challenge key whenever* $\mathcal{A}$ *declares the challenge key to have attributes which satisfy the challenge ciphertext's access structure.*

Proof. Throughout this proof we treat the master keys as a special form of secret keys, i.e. the ones that have as attributes the entire universe $\mathcal{U}$, which satisfies all monotone access structures.

We suppose there exists a PPT algorithm $\mathcal{A}$ whose advantage changes by a non-negligible amount ϵ when the $\mathsf{MasterLeak}_1$ game changes as described above. Using $\mathcal{A}$, we will create a PPT algorithm $\mathcal{B}$ which will distinguish between the distributions $(\vec{\delta}, f(\vec{\tau}))$ and $(\vec{\delta}, f(\vec{\tau'}))$ from Corollary B.1.1 with non-negligible advantage (where $m = n + 1$ and $p = p_2$). This will yield a contradiction, since these distributions have a negligible statistical distance.

$\mathcal{B}$ simulates the game $\mathsf{MasterLeak}_1$ with $\mathcal{A}$ as follows. It starts by running the Setupalgorithm for itself, and giving $\mathcal{A}$ the public parameters. Since $\mathcal{B}$ knows the original master key and generators of all the subgroups, it can make normal as well as semi-functional keys. Hence, it can respond to $\mathcal{A}$'s non-challenge **Phase 1** queries by simply creating the queried keys.

With non-negligible probability, $\mathcal{A}$ must chose a challenge key in **Phase 1** with attributes that satisfy the challenge ciphertext's access structure. (If it only did this with negligible probability, then the difference in advantages whenever the attributes satisfy the access structure would be negligible.)

$\mathcal{B}$ will not create this challenge key, but instead will encode the leakage $\mathcal{A}$ asks for on this key in **Phase 1** as a single polynomial time computable function f with domain $\mathbb{Z}_{p_2}^{n+1}$ and with an image of size $2^{\ell_{\text{SK}}}$ (or $2^{\ell_{\text{MSK}}}$). It can do this by fixing the values of all other keys and fixing all other variables involved in the challenge key (more details on this below). $\mathcal{B}$ then receives a sample $(\vec{\delta}, f(\vec{\Gamma}))$, where $\vec{\Gamma}$ is either distributed as $\vec{\tau}$ or as $\vec{\tau}'$, in the notation of the corollary. $\mathcal{B}$ will use $f(\vec{\Gamma})$ to answer all of $\mathcal{A}$'s leakage queries on the challenge key by implicitly defining the challenge key as follows.

$\mathcal{B}$ chooses $r_1, r_2 \in \mathbb{Z}_{p_2}$. We let g_2 denote a generator of $\mathbb{G}_2$. $\mathcal{B}$ implicitly sets the $\mathbb{G}_2$ components of the key to be $g_2^{\vec{\gamma}}$ and g_2^{θ}, where $\vec{\gamma}, \theta$ are defined to be

$$\vec{\gamma} = \vec{\Gamma} + \left(\overbrace{0, \ldots, 0}^{n}, r_1 \right) \quad \text{and} \quad \theta = r_2$$

Recall that $\vec{\Gamma}$ is of length $n+1$; thus r_1 is added to the last component of $\vec{\Gamma}$. $\mathcal{B}$ defines the non-$\mathbb{G}_2$ components of the key to fit their appropriate distribution.

At some point, $\mathcal{A}$ declares the access structure for the challenge ciphertext. If the challenge key had attributes that did not satisfy this access structure, then $\mathcal{B}$ aborts the simulation and guesses whether $\vec{\Gamma}$ is orthogonal to $\vec{\delta}$ randomly. However, the simulation continues with non-negligible probability.

$\mathcal{B}$ chooses a random element $t_2 \in \mathbb{Z}_{p_2}$ subject to the constraint $\delta_{n+1}r_1 - t_2r_2 = 0 \bmod p_2$. It then constructs the challenge ciphertext, using $\vec{\delta}$ and $u_1 = t_2$ as the challenge vector (recall that $\vec{\delta}$ is of length $n+1$). The remaining parameters (semi-functional or not) are chosen according to **EncryptSF** algorithm. Now, if $\vec{\Gamma}$ is orthogonal to $\vec{\delta}$, then the challenge key is nominally semi-functional (and well-distributed as such). If $\vec{\Gamma}$ is not orthogonal to $\vec{\delta}$, then the challenge key is truly semi-functional (and also well-distributed).

It is clear that $\mathcal{B}$ can easily handle **Phase 2** queries, since the challenge key cannot be queried on here when its attributes satisfy the challenge ciphertext's access structure. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to gain a non-negligible advantage in distinguishing the distributions $(\vec{\delta}, f(\vec{\tau}))$ and $(\vec{\delta}, f(\vec{\tau'}))$. This violates Corollary B.1.1, since these distributions have a negligible statistical distance for f with this output size. $\square$

However, if the attributes of the key do not satisfy the challenge access structure, the attacker can ask for the entire key to be revealed. In this scheme, we use the *unique attribute restriction* to argue that the value of $u_1 = a\nu$ is information-theoretically hidden modulo p_2.

Since the attributes of the key do not satisfy the challenge access structure, the rowspace $R \subseteq \mathbb{Z}_N^{n_2}$ formed by the rows of A^*, whose attributes are in S, does not include the vector $(1, 0, \ldots, 0)$. Otherwise, one could find ω_x's such that $\sum_{\delta^*(x) \in \mathcal{S}} \omega_x A_x^* = (1, 0, \ldots, 0)$ and decrypt. This means that there is a vector $\vec{w}$ that is in the orthogonal complement of R, but not orthogonal to $(1, 0, \ldots, 0)$. We can create a basis B of $\mathbb{Z}_N^{n_2}$ that includes $\vec{w}$. Then we can write $\vec{u} = f\vec{w} + \vec{u'}$ where $f \in \mathbb{Z}_N$ and $\vec{u'}$ is generated by all the vectors of B except $\vec{w}$. But then we have that

$$u_1 = \vec{u} \cdot (1, 0, \ldots, 0) = f\vec{w} \cdot (1, 0, \ldots, 0) + \vec{u'} \cdot (1, 0, \ldots, 0) \,.$$

Since $\vec{u'}$ reveals no information about f and $\vec{w}$ is not orthogonal to $(1, 0, \ldots, 0)$, the value of f is needed to determine the value of u_1.

The only places where $\vec{u}$ (and hence f) appears modulo p_2 are in the exponents of the form (see EncryptSf algorithm)

$$A_x^* \cdot \vec{u} + \delta'_x q_{\delta^*(x)} \quad \text{for every row} \quad x$$

However, not all of these are affected by the value of f. More specifically, we know that the rows for which the attribute $\delta^*(x)$ is in $\mathcal{S}$ (i.e. one of the key's attributes), hide the value of f since $\vec{w}$ is orthogonal to R.

For the remaining rows, we know that with certainty minus a negligible probability, all multiplicative factors δ'_x are not equal to 0 mod p_2 and thus the value of f is "masked" by the term $\delta'_x q_{\delta^*(x)}$. Here is where we use the restriction that each attribute in the access structure appears only once: Each of these $q_{\delta^*(x)}$ factors masks f entirely if it appears only once, since they are random modulo p_2. But this is true because they appear only in the access structure for which the unique attribute restriction holds and the only key that could have these terms is the challenge key (type 1). Therefore, the value of u_1 seems random to the adversary, as well. (This is proven the same way in [68].)

Phase 2: $\mathcal{B}$ works in the same way as in **Phase 1**.

The conclusion is that the attacker $\mathcal{A}$ plays either the $\mathsf{MasterLeak}_0$ or the $\mathsf{MasterLeak}_{1/2}$ game, depending on the assumption. Thus, if it has a non-negligible difference in the advantages, $\mathcal{B}$ can break the assumption $\mathsf{Comp2}$ with non-negligible advantage. $\square$

Theorem 5.10. *If the assumption* $\mathsf{Comp2}$ *holds, the difference between the advantages of any PPT attacker when playing the* $\mathsf{MasterLeak}_{1/2}$ *and* $\mathsf{MasterLeak}_1$ *games with leakage* $(\ell_{\text{MSK}}, \ell_{\text{SK}})$ *on our ABE system is negligible in* λ.[4]

Proof. We assume we have a PPT attacker $\mathcal{A}$ with a non-negligible difference in advantage between these two games. We will build a PPT algorithm $\mathcal{B}$ which breaks assumption 3.2.2 with non-negligible advantage. The simulator in this reduction works in the same way as in the previous one, with only one difference: it picks $\vec{h} \xleftarrow{R} \mathbb{Z}_N^{n+1}$ and generates the challenge key as:

$$\begin{aligned}\text{SK} &= \left(S, \vec{K}_1, L, \forall i \in S \;\; K_i\right) = \\ &= \left(S, \left(T^{z'_1}, \ldots, T^{z'_n}, g_1^{\alpha} T^{a} \prod_{i=1}^{n} T^{-x_i z'_i}\right) * (g_2^{\mu} g_3^{\rho})^{\vec{h}} * g_3^{\vec{\rho}}, T g_3^{\rho_{n+2}}, \right. \\ &\qquad \left. \forall i \in S \;\; T^{s_i} g_3^{\rho'_i}\right)\end{aligned}$$

[4]For this theorem to be true the leakage bounds $\ell_{\text{MSK}}, \ell_{\text{SK}}$ can take any values in $\mathbb{N} \cup \{0\}$.

The only difference from the previous simulator is the term $(g_2^{\mu} g_3^{\rho})^{\vec{h}}$, where $g_2^{\mu} g_3^{\rho}$ is given by the assumption.

If $T = g_1^w g_2^{\kappa} g_3^{\sigma}$, the semi-functional parameters of the challenge key and the ciphertext are:

$$\begin{array}{ll} \vec{\gamma} = \kappa\,(z'_1, \ldots, z'_n, a - \sum x_i z'_i) + \mu\vec{h} & \theta = \kappa \\ \vec{\delta} = \nu\,(x_1, \ldots, x_n, 1) & \vec{u} = a\nu\vec{v}' \end{array}$$

As before, $q_i = s_i$ for both the key and the ciphertext as they should be. Also the new term re-randomizes the $\mathbb{G}_2$ part of $\vec{K}_1$ so the key is no longer nominally semi-functional with respect to the ciphertext, i.e. $\vec{\gamma} \cdot \vec{\delta} - \theta u_1 = 0$ no longer holds. It is obvious that the extra vector $\mu\vec{h}$ makes all parameters random and uncorellated modulo p_2. So in this case, the challenge key is a well-distributed semi-functional key of type 1 and $\mathcal{A}$ plays game $\mathsf{MasterLeak}_{1/2}$ (all requested semi-functional keys of type 2, type 1 challenge key and the remaining keys normal).

If $T = g_1^w g_3^{\sigma}$, the key is semi-functional of type 2 with parameters $\vec{\gamma} = \mu\vec{h}$. Thus $\mathcal{A}$ plays game $\mathsf{MasterLeak}_1$ (all requested semi-functional keys and challenge key of type 2 and the remaining keys normal). $\square$

By the two previous theorems we get immediately one-semi-functional invariance:

Theorem 5.11. *If the assumption* $\mathsf{Comp2}$ *holds, our system has* $(\ell_{\mathrm{MSK}}, \ell_{\mathrm{SK}})$*-one semi-functional key invariance.*

Proof. By theorems 5.8 and 5.10, we have that for any PPT adversary $\mathcal{A}$

$$\left|\mathsf{Adv}^{\mathsf{MasterLeak}_0}_{\mathcal{A},\Pi_{ABE}}(\lambda,\ell_{\text{MSK}},\ell_{\text{SK}}) - \mathsf{Adv}^{\mathsf{MasterLeak}_{1/2}}_{\mathcal{A},\Pi_{ABE}}(\lambda,\ell_{\text{MSK}},\ell_{\text{SK}})\right| \leq \mathsf{negl}(\lambda)$$

$$\left|\mathsf{Adv}^{\mathsf{MasterLeak}_{1/2}}_{\mathcal{A},\Pi_{ABE}}(\lambda,\ell_{\text{MSK}},\ell_{\text{SK}}) - \mathsf{Adv}^{\mathsf{MasterLeak}_1}_{\mathcal{A},\Pi_{ABE}}(\lambda,\ell_{\text{MSK}},\ell_{\text{SK}})\right| \leq \mathsf{negl}(\lambda).$$

By the triangle inequality and the fact that the sum of two negligible functions is negligible, we get one semi-functional key invariance. □

Theorem 5.12. *If the assumption* Comp3 *holds, our system has* $(\ell_{\text{MSK}},\ell_{\text{SK}})$*-semi-functional security.*[5]

Proof. We assume we have a PPT attacker $\mathcal{A}$ which breaks semi-functional security with non-negligible advantage. We will construct a PPT algorithm $\mathcal{B}$ which breaks the assumption Comp3 with non-negligible advantage. As always, $\mathcal{B}$ that plays either the MasterLeakCK game or a final game with $\mathcal{A}$, where the advantage of any attacker is 0 in the final game because the ciphertext is an encryption of a random message, independent of the bit c.

Setup phase: $\mathcal{B}$ picks $a \xleftarrow{R} \mathbb{Z}_N$ and for each attribute $i \in U$, it chooses random $s_i \xleftarrow{R} \mathbb{Z}_N$. It will use α from the assumption's term $g_1^{\alpha} g_2^{\nu}$. It also picks n random exponents $x_1, x_2, \ldots, x_n \xleftarrow{R} \mathbb{Z}_N$. It gives the public parameters

$$\text{PP} = (N, g_1, g_3, g_1^a, e(g_1,g_1)^{\alpha} = e(g_1^{\alpha} g_2^{\nu}, g_1), g_1^{x_1}, \ldots, g_1^{x_n}, \forall i \in U \;\; T_i = g_1^{s_i})$$

[5]For this theorem to be true the leakage bounds $\ell_{\text{MSK}}, \ell_{\text{SK}}$ can take any values in $\mathbb{N} \cup \{0\}$.

to $\mathcal{A}$, where N, g_1, and g_3 are given by the challenger.

Phase 1: All keys generated by $\mathcal{B}$ should be semi-functional keys of type 2. For each one, the simulator picks $t, z_1, \ldots, z_n \in \mathbb{Z}_N$, a random vector $\vec{h} \overset{R}{\leftarrow} \mathbb{Z}_N^{n+1}$ for the $\mathbb{G}_2$ part and $\vec{\rho} \overset{R}{\leftarrow} \mathbb{Z}_N^{n+1}$, $\rho_{n+2} \overset{R}{\leftarrow} \mathbb{Z}_N, \forall i \in S \;\; \rho'_i \overset{R}{\leftarrow} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part. It creates the following secret key:

$$\begin{aligned} \mathrm{SK} &= \Big(\mathcal{S}, \vec{K_1}, L, \forall i \in \mathcal{S} \;\; K_i\Big) = \\ &= \left(\mathcal{S}, \left(g_1^{z_1}, \ldots, g_1^{z_n}, (g_1^{\alpha} g_2^{\nu}) g_1^{at} \prod_{i=1}^{n} g_1^{-x_i z_i}\right) * g_2^{\vec{h}} * g_3^{\vec{\rho}}, g_1^t g_3^{\rho_{n+2}},\right. \\ &\qquad \left. \forall i \in S \;\; T_i^t g_3^{\rho'_i}\right), \end{aligned}$$

where $\mathcal{S}$ is the set of attributes given by $\mathcal{A}$ and $g_1^{\alpha} g_2^{\nu}$ comes from the challenger. It is easy to see that this is a properly distributed semi-functional key of type 2 with semi-functional parameters $\vec{\gamma} = \vec{h} + (0, \ldots, 0, \nu)$.

Challenge Phase: The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages M_0 and M_1 and an access structure, encoded as a $n_1 \times n_2$ LSSS matrix: (A^*, δ^*). The simulator $\mathcal{B}$ chooses random values $v'_2, \ldots, v'_{n_2} \overset{R}{\leftarrow} \mathbb{Z}_N$ and for each row A^*_x of A^*, one value $r'_x \overset{R}{\leftarrow} \mathbb{Z}_N$. Using the v' values, it creates the vector $\vec{v'} = \big(1, v'_2, \ldots, v'_{n_2}\big)$. It flips a random coin $c \overset{R}{\leftarrow} \{0, 1\}$ and outputs the ciphertext:

$$\begin{aligned} \mathrm{CT} &= \Big((A^*, \delta^*), C_0, \vec{C_1}, \forall x \;\; C_x, \forall x \;\; D_x\Big) = \\ &= \left((A^*, \delta^*), M \cdot e(T, g_1^{\alpha}), \left((g_1^z g_2^{\mu})^{x_1}, \ldots, (g_1^z g_2^{\mu})^{x_n}, (g_1^z g_2^{\mu})\right),\right. \\ &\qquad \left. \forall x \;\; (g_1^z g_2^{\mu})^{a A^*_x \cdot \vec{v'}} (g_1^z g_2^{\mu})^{-r'_x s_{\delta^*(x)}}, \forall x \;\; (g_1^z g_2^{\mu})^{r'_x}\right), \end{aligned}$$

where $g_1^z g_2^{\mu}$ is given from the assumption and T is the challenge term.

The ciphertext is semi-functional since

$$(g_1^z g_2^\mu)^{x_i} = (g_1^{x_i})^z g_2^{\mu x_i} \text{ for } i \in \{1, 2, \dots, n\}$$
$$(g_1^z g_2^\mu)^{aA_x^* \cdot \vec{v'}} (g_1^z g_2^\mu)^{-r'_x s_{\delta^*(x)}} = g_1^{aA_x^* \cdot z\vec{v'}} T_{\delta^*(x)}^{-zr'_x} \cdot g_2^{A_x^* \cdot a\mu\vec{v'} - \mu r'_x s_{\delta^*(x)}}$$
$$\text{for every row } x \text{ of } A^*$$
$$(g_1^z g_2^\mu)^{r'_x} = g_1^{zr'_x} \cdot g_2^{\mu r'_x} \text{ for every row } x \text{ of } A^*$$

For the $\mathbb{G}_1$ parts, this implicitly sets $s = z$, $\vec{v} = z\vec{v'}$, and $r_x = zr'_x$. Thus all $\mathbb{G}_1$ parts are properly distributed (remember that the first coordinate of $\vec{v}$ should be z).

For the $\mathbb{G}_2$ parts, this sets $\vec{\delta} = \mu\,(x_1, \dots, x_n, 1)$, $\vec{u} = a\mu\vec{v'}$, $\delta'_x = -\mu r'_x$ and $q_{\delta*(x)} = s_{\delta*(x)}$. These are properly distributed modulo p_2 because the terms x_1, …, x_n, μ, a, v'_2, …, v'_{n_2}, r'_x, $s_{\delta^*(x)}$ are only seen modulo p_1 elsewhere.

Phase 2: Here $\mathcal{B}$ works in the same way as in **Phase 1**.

If $T = e(g_1, g_1)^{\alpha z}$, the above is a properly distributed semi-functional encryption of M_c. Otherwise, it is an encryption of a random message. Thus, the advantage of any adversary in this case is 0. $\square$

By theorems 5.7, 5.11, 5.12 we get that:

Theorem 5.13. *Under the assumptions* Comp1*,* Comp2*,* Comp3 *and for* $(\ell_{\text{MSK}} = (n-1-2c)\log(p_2)$, $\ell_{\text{SK}} = (n-1-2c)\log(p_2))$, *where* $c > 0$ *is a fixed positive constant, our dual system encryption ABE scheme is* $(\ell_{\text{MSK}}, \ell_{\text{SK}})$*-master-leakage secure.*

5.3 Leakage Fraction

Our systems allow the same absolute amount of leakage for both the master and the secret keys. That is, $\ell_{\text{MSK}} = \ell_{\text{SK}} = (n - 1 - 2c)\log p_2$ bits, where n is an arbitrary integer greater than or equal to 2 and c is a fixed positive constant. Notice that the leakage depends only on the size of the $\mathbb{G}_2$ subgroup, and not on the size of p_1 or p_3. Thus by varying the relative sizes of the $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_3$ subgroups, we can achieve variable key sizes and allow different fractions of the key size to be leaked. We use the term "leakage fraction" to mean the number of bits allowed to be leaked from a key divided by the number of bits required to represent that key.

Recall that p_1, p_2, p_3 are primes of $\lambda_1, \lambda_2, \lambda_3$ bits, respectively, and $N = p_1p_2p_3$ is the order of our group $\mathbb{G}$. We assume that each group element is represented by approximately $\lambda_1 + \lambda_2 + \lambda_3 = \Theta(\log N)$ bits. Then, by fixing $\lambda_1 = c_1\lambda$, $\lambda_2 = \lambda$, and $\lambda_3 = c_3\lambda$, where λ is the security parameter and c_1, c_3 are arbitrary positive constants, we get that the leakage fractions of our systems are the following:

The leakage fraction can be made arbitrarily close to 1 by modifying n, c_1 and c_3 (if we assume a fixed universe size for ABE). Higher values of n give a better leakage fraction, but larger public parameters, keys, and ciphertexts. Smaller values of c_1, c_3 give a better leakage fraction, but also give fewer bits of security in the $\mathbb{G}_1$ and $\mathbb{G}_3$ subspaces as a function of λ. We must choose λ so that $c_1\lambda$ and $c_3\lambda$ are sufficiently large.

Scheme	Master Key	Secret Key
IBE	$\frac{n-1-2c}{n+3} \cdot \frac{1}{1+c_1+c_3}$	$\frac{n-1-2c}{n+2} \cdot \frac{1}{1+c_1+c_3}$
ABE	$\frac{n-1-2c}{n+2+\lvert\mathcal{U}\rvert} \cdot \frac{1}{1+c_1+c_3}$	$\frac{n-1-2c}{n+2+\lvert\mathcal{S}\rvert} \cdot \frac{1}{1+c_1+c_3}$

Table 5.1: c, c_1, c_3 are arbitrary positive constants and n is an integer greater than 2. For the ABE scheme, $|\mathcal{U}|$ is the total number of attributes in the system, i.e. the size of the universe, and $|\mathcal{S}|$ is the number of attributes of the key in question. Notice that in the ABE scheme we ignored the size of the representations of $\mathcal{U}$ and $\mathcal{S}$. They are included in the keys, but they are considered public; thus not included in the leakage fraction.

CHAPTER 6

Part II: Three Practical Constructions

In this chapter we present three prime order group constructions with advanced features. Namely we present two large universe ABE constructions [98], both proved selectively secure in the standard model under suitable prime order q-type assumptions, and a multi-authority CP-ABE scheme [97], which is statically secure in the random oracle model under a third q-type assumption. All of the schemes were designed with three major goals: *practicality, augmented functionality, and sufficiently strong security guarantees.*

6.1 A Large-Universe KP-ABE System

In this section we present our large universe KP-ABE scheme from [98]. We mention here that it can be converted to an HIBE scheme using non repeating identities, "AND" policies and delegation capabilities (c.f. [74]). In this setting the public parameters consist of the five terms $(g, u, h, w, e(g,g)^\alpha)$, which intuitively are utilized in two separate "layers" to achieve secure large universe KP-ABE. In the "attribute layer", the u, h terms provide a Boneh-Boyen-style [21] hash function ($u^A h$), while in the "secret sharing layer" the g term holds the shares of the secret key α during key generations. The w term is used to "bind" this layer to the u, h "attribute layer".

6.1.1 Construction

Our scheme consists of the following four algorithms.

$\mathsf{Setup}(1^\lambda) \to (\mathrm{PP}, \mathrm{MSK})$**:** The setup algorithm calls the group generator algorithm $\mathcal{G}(1^\lambda)$ and gets the descriptions of the groups and the bilinear mapping $D = (p, \mathbb{G}, \mathbb{G}_T, e)$, where p is the prime order of the groups $\mathbb{G}$ and $\mathbb{G}_T$. The attribute universe is $\mathcal{U} = \mathbb{Z}_p$.

Then the algorithm picks the random terms $g, u, h, w \xleftarrow{R} \mathbb{G}$ and $\alpha \xleftarrow{R} \mathbb{Z}_p$. It outputs

$$\mathrm{PP} = (D, g, u, h, w, e(g,g)^\alpha) \qquad \mathrm{MSK} = (\alpha)$$

$\mathsf{KeyGen}(\mathrm{MSK}, (A, \delta)) \to \mathrm{SK}$**:** Initially the algorithm picks $\vec{y} = (\alpha, y_2, \ldots, y_n)^\top$ where $y_2, \ldots, y_n \xleftarrow{R} \mathbb{Z}_p$. In the terminology of section 2.1, the master secret key α is the secret to be shared among the shares. The vector of the shares is

$$\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)^\top = A\vec{y}$$

It then picks ℓ random exponents $t_1, t_2, \ldots, t_\ell \xleftarrow{R} \mathbb{Z}_p$ and for every $\tau \in [\ell]$

it computes

$$K_{\tau,0} = g^{\lambda_\tau} w^{t_\tau} \qquad K_{\tau,1} = \left(u^{\delta(\tau)} h\right)^{-t_\tau} \qquad K_{\tau,2} = g^{t_\tau}$$

The secret key is $\mathrm{SK} = ((A, \delta), \{K_{\tau,0}, K_{\tau,1}, K_{\tau,2}\}_{\tau \in [\ell]})$.

$\mathsf{Encrypt}(m, \mathcal{S} = \{A_1, A_2, \ldots, A_k\} \subseteq \mathbb{Z}_p) \to \mathrm{CT}$**:** Initially, the algorithm picks $k+1$ random exponents s, r_1, r_2, …, $r_k \overset{R}{\leftarrow} \mathbb{Z}_p$. It computes $C = m \cdot e(g, g)^{\alpha s}$, $C_0 = g^s$, and for every $\tau \in [k]$ it computes

$$C_{\tau,1} = g^{r_\tau} \qquad C_{\tau,2} = (u^{A_\tau} h)^{r_\tau} w^{-s}$$

The ciphertext is $\mathrm{CT} = (\mathcal{S}, C, C_0, \{C_{\tau,1}, C_{\tau,2}\}_{\tau \in [k]})$.

$\mathsf{Decrypt}\,\{\mathrm{SK}, \mathrm{CT}\} \to m$**:** The algorithm finds the set of rows in M that provide a share to attributes in $\mathcal{S}$, i.e. $I = \{i : \delta(i) \in \mathcal{S}\}$. Then it calculate constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that $\sum_{i \in I} \omega_i A_i = (1, 0, \ldots, 0)$, where A_i is the i-th row of the matrix A. According to the discussion in section 2.1, these constants exist if the set $\mathcal{S}$ is an authorized set of the policy.

Then it calculates

$$B = \prod_{i \in I} \{e(C_0, K_{i,0}) e(C_{\tau,1}, K_{i,1}) e(C_{\tau,2}, K_{i,2})\}^{\omega_i}$$

where τ is the index of the attribute $\delta(i)$ in $\mathcal{S}$ (it depends on i). The algorithm outputs $m = C/B$.

Correctness: If the attribute set $\mathcal{S}$ of the ciphertext is authorized, we have that $\sum_{i \in I} \omega_i \lambda_i = \alpha$. Therefore:

$$\begin{aligned} B &= \prod_{i \in I} e(g,g)^{s\omega_i\lambda_i} e(g,w)^{st_i\omega_i} e(g,u^{\delta(i)}h)^{-r_\tau t_i \omega_i} e(g,u^{\delta(i)}h)^{r_\tau t_i \omega_i} e(g,w)^{-st_i\omega_i} \\ &= e(g,g)^{s \sum_{i \in I} \omega_i \lambda_i} = e(g,g)^{\alpha s} \end{aligned}$$

6.1.2 Security Proof

We will prove the following theorem regarding the selective security of our KP-ABE scheme:

Theorem 6.1. *If the q-DPBDH1 assumption holds, then all PPT adversaries with a challenge attribute set of size k, where $k \leq q$, have a negligible advantage in selectively breaking our scheme.*

Proof. To prove the theorem we will assume that there exists a PPT attacker $\mathcal{A}$ with a challenge attribute set that satisfies the restriction, which has a non negligible advantage $\mathsf{Adv}_{\mathcal{A}}$ in selectively breaking our scheme. Using this attacker we will build a PPT simulator $\mathcal{B}$ that attacks the q-DPBDH1 assumption with a non negligible advantage.

Initialization: Initially, $\mathcal{B}$ receives the given terms from the assumption and an attribute set $\mathcal{S}^* = \{A_1^*, A_2^*, \ldots, A_k^*\} \subseteq \mathcal{U}$.

Setup: Now, the simulator $\mathcal{B}$ has to provide $\mathcal{A}$ the public parameters of the system. In order to do that it implicitly sets the master secret key of the scheme to be $\alpha = xy$, where x, y are set in the assumption. Notice that this way α is properly distributed. Then $\mathcal{B}$ picks the random exponents $\tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and gives to $\mathcal{A}$ the following terms:

$$\begin{array}{ll} g = g & w = g^x \\ u = g^{\tilde{u}} \cdot \prod_{i \in [k]} g^{y/b_i^2} & h = g^{\tilde{h}} \cdot \prod_{i \in [k]} g^{xz/b_i} \cdot \prod_{i \in [k]} \left(g^{y/b_i^2}\right)^{-A_i^*} \\ e(g,g)^\alpha = e(g^x, g^y) & \end{array}$$

Since x is information-theoretically hidden from $\mathcal{A}$, because it is multiplied by y in α, the term w is properly uniformly random in $\mathbb{G}$. The terms u, h are properly distributed due to $\tilde{u}, \tilde{h}$ respectively. Notice that all terms can be calculated by the simulator using suitable terms from the assumption and the challenge set $\mathcal{S}^*$ given by $\mathcal{A}$.

Query phases 1 and 2: The simulator has to produce secret keys for policies requested by $\mathcal{A}$, for which the set $\mathcal{S}^*$ is not authorized. In both phases the treatment is the same. We describe here the way $\mathcal{B}$ works in order to create a key for a policy (A, δ).

Since $\mathcal{S}^*$ is non authorized for (A, δ), there exists a vector $\vec{w} = (w_1, w_2, \ldots, w_n)^\top \in \mathbb{Z}_p{}^n$ such that $w_1 = 1$ and $\langle A_\tau, \vec{w} \rangle = 0$ for all $\tau \in [\ell]$ such that $\delta(\tau) \in \mathcal{S}^*$ (c.f. section 2.1). The simulator calculates $\vec{w}$ using linear algebra. The vector $\vec{y}$ that will be shared is implicitly

$$\vec{y} = xy\vec{w} + (0, \tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_n)^\top$$

where $\tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_n \xleftarrow{R} \mathbb{Z}_p$. This vector is properly distributed because its first component is $xy = \alpha$ and the remaining components are uniformly random in

$\mathbb{Z}_p$. Therefore for each row $\tau \in [\ell]$ the share is

$$\lambda_\tau = \langle A_\tau, \vec{y} \rangle = xy\langle A_\tau, \vec{w} \rangle + \langle A_\tau, (0, \tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_n)^\top \rangle = xy\langle A_\tau, \vec{w} \rangle + \tilde{\lambda}_\tau$$

As we mentioned above for each row τ for which $\delta(\tau) \in \mathcal{S}^*$ it is true that $\langle A_\tau, \vec{w} \rangle = 0$. Therefore in this case $\lambda_\tau = \tilde{\lambda}_\tau = \langle A_\tau, (0, \tilde{y}_2, \tilde{y}_3, \ldots, \tilde{y}_n)^\top \rangle$; hence its value is known to the simulator. In that case it picks $t_\tau \overset{R}{\leftarrow} \mathbb{Z}_p$ and outputs the terms $K_{\tau,0}, K_{\tau,1}, K_{\tau,2}$ as in the KeyGen algorithm.

On the other hand, for each row τ for which $\delta(\tau) \notin \mathcal{S}^*$ it picks $\tilde{t}_\tau \overset{R}{\leftarrow} \mathbb{Z}_p$ and sets implicitly

$$t_\tau = -y\langle A_\tau, \vec{w} \rangle + \sum_{i \in [k]} \frac{xzb_i \langle A_\tau, \vec{w} \rangle}{\delta(\tau) - A_i^*} + \tilde{t}_\tau$$

Since $\delta(\tau) \notin \mathcal{S}^*$ the above fractions are defined and t_τ is properly distributed due to $\tilde{t}_\tau$. The intuition behind this choice is that the y exponent "raises" the power of w to the secret $\alpha = xy$. However, this also results to xyz/b_i exponents from h. Thus, the cancellation is provided by the xzb_i exponents on the y/b_i^2 part. Now the simulator can compute the following terms using the assumption:

$$\begin{aligned} K_{\tau,0} &= g^{\lambda_\tau} w^{t_\tau} \\ &= g^{xy\langle A_\tau, \vec{w}\rangle + \tilde{\lambda}_\tau} \cdot g^{-xy\langle A_\tau, \vec{w}\rangle + \sum_{i\in[k]} \frac{x^2 zb_i \langle A_\tau, \vec{w}\rangle}{\delta(\tau) - A_i^*}} \cdot w^{\tilde{t}_\tau} \\ &= g^{\tilde{\lambda}_\tau} \cdot \prod_{i \in [n]} \left(g^{x^2 zb_i} \right)^{\langle A_\tau, \vec{w}\rangle / (\delta(\tau) - A_i^*)} \cdot w^{\tilde{t}_\tau} \end{aligned}$$

$$
\begin{aligned}
K_{\tau,1} &= (u^{\delta(\tau)}h)^{-t_\tau} = \\
&= \left(g^{\delta(\tau)\tilde{u}+\tilde{h}} \cdot \prod_{i\in[k]} g^{xz/b_i} \cdot \prod_{i\in[k]} g^{y\left(\delta(\tau)-A_i^*\right)/b_i^2} \right)^{y\langle A_\tau,\vec{w}\rangle - \sum_{i\in[k]} \frac{xzb_i\langle A_\tau,\vec{w}\rangle}{\delta(\tau)-A_i^*}} \\
&\quad \cdot (u^{\delta(\tau)}h)^{-\tilde{t}_\tau} \\
&= g^{y\langle A_\tau,\vec{w}\rangle(\delta(\tau)\tilde{u}+\tilde{h})} \prod_{i\in[k]} g^{-xzb_i(\delta(\tau)\tilde{u}+\tilde{h})\langle A_\tau,\vec{w}\rangle/(\delta(\tau)-A_i^*)} \\
&\quad \cdot \prod_{i\in[k]} g^{xyz\langle A_\tau,\vec{w}\rangle/b_i} \prod_{(i,j)\in[k,k]} g^{-(xz)^2 b_j\langle A_\tau,\vec{w}\rangle/b_i(\delta(\tau)-A_j^*)} \\
&\quad \cdot \prod_{i\in[k]} g^{y^2\langle A_\tau,\vec{w}\rangle\left(\delta(\tau)-A_i^*\right)/b_i^2} \prod_{(i,j)\in[k,k]} g^{-xyz\langle A_\tau,\vec{w}\rangle b_j\left(\delta(\tau)-A_i^*\right)/b_i^2(\delta(\tau)-A_j^*)} \\
&\quad \cdot (u^{\delta(\tau)}h)^{-\tilde{t}_\tau} \\
&= (g^y)^{\langle A_\tau,\vec{w}\rangle(\delta(\tau)\tilde{u}+\tilde{h})} \prod_{i\in[k]} \left(g^{xzb_i}\right)^{-(\delta(\tau)\tilde{u}+\tilde{h})\langle A_\tau,\vec{w}\rangle/(\delta(\tau)-A_i^*)} \\
&\quad \cdot \prod_{(i,j)\in[k,k]} \left(g^{(xz)^2 b_j/b_i}\right)^{-\langle A_\tau,\vec{w}\rangle/(\delta(\tau)-A_j^*)} \prod_{i\in[k]} \left(g^{y^2/b_i^2}\right)^{\langle A_\tau,\vec{w}\rangle\left(\delta(\tau)-A_i^*\right)} \\
&\quad \cdot \prod_{\substack{(i,j)\in[k,k]\\ i\neq j}} \left(g^{xyzb_j/b_i^2}\right)^{-\langle A_\tau,\vec{w}\rangle\left(\delta(\tau)-A_i^*\right)/(\delta(\tau)-A_j^*)} \cdot (u^{\delta(\tau)}h)^{-\tilde{t}_\tau}
\end{aligned}
$$

$$
\begin{aligned}
K_{\tau,2} &= g^{t_\tau} \\
&= (g^y)^{-\langle A_\tau,\vec{w}\rangle} \cdot \prod_{i\in[k]} \left(g^{xzb_i}\right)^{\langle A_\tau,\vec{w}\rangle/(\delta(\tau)-A_i^*)} \cdot g^{\tilde{t}_\tau}
\end{aligned}
$$

Therefore $\mathcal{B}$ can reply to $\mathcal{A}$'s query with the entire secret key $\mathrm{SK} = ((A,\delta),\ \{K_{\tau,0},\ K_{\tau,1},\ K_{\tau,2}\}_{\tau\in[\ell]})$.

Challenge: The attacker will output a pair of messages (M_0, M_1) of the same length. In this phase the simulator flips a random coin $b \overset{R}{\leftarrow} \{0,1\}$ and sets implicitly $s = z$ from the q-DPBDH1 assumption. Also, it sets

$r_\tau = b_\tau$ for every level $\tau \in [k]$. These parameters are properly distributed since $z, b_1, \dots, b_q$ are information-theoretically hidden from the attacker's view. Now the simulator can compute the following terms using the assumption:

$$
\begin{aligned}
C &= M_b \cdot T \qquad C_0 = g^s = g^z \\
C_{\tau,1} &= g^{r_\tau} = g^{b_\tau} \\
C_{\tau,2} &= (u^{A^*_\tau} h)^{r_\tau} \cdot w^{-s} \\
&= g^{b_\tau(\tilde{u}A^*_\tau + \tilde{h})} \cdot \prod_{i \in [k]} g^{xzb_\tau/b_i} \prod_{i \in [k]} g^{yb_\tau \left(A^*_k - A^*_i\right)/b_i^2} \cdot g^{-xz} \\
&= \left(g^{b_\tau}\right)^{\tilde{u}A^*_\tau + \tilde{h}} \cdot \prod_{\substack{i \in [k] \\ i \neq \tau}} g^{xzb_\tau/b_i} \prod_{\substack{i \in [k] \\ i \neq \tau}} \left(g^{yb_\tau/b_i^2}\right)^{A^*_\tau - A^*_i}
\end{aligned}
$$

As one can see, the choice of $r_\tau = b_\tau$ "raises" one of the xz/b_i components to xz and achieves the cancellation with w^{-s}. The simulator hands over the ciphertext $\mathrm{CT} = \left(\mathcal{S}^*, C, C_0, \{C_{\tau,1}, C_{\tau,2}\}_{\tau \in [k]}\right)$ to the attacker $\mathcal{A}$.

Guess: After the query phase 2, where the simulator creates the secret keys as described above, the attacker outputs a guess b' for the challenge bit. If $b' = b$ the simulator outputs 0, i.e. it claims that the challenge term is $T = e(g,g)^{xyz}$. Otherwise, it outputs 1.

If $T = e(g,g)^{xyz}$ then $\mathcal{A}$ played the proper security game, because $C = M_b \cdot T = M_b \cdot e(g,g)^{\alpha s}$. On the other hand, if T is a random term of $\mathbb{G}_T$ then all information about the message M_b is lost in the challenge ciphertext. Therefore the advantage of $\mathcal{A}$ is exactly 0. As a result if $\mathcal{A}$ breaks the proper security game with a non negligible advantage, then $\mathcal{B}$ has a non negligible advantage in breaking the q-DPBDH1 assumption. $\square$

6.2 A Large-Universe CP-ABE System

In this section we present our large universe CP-ABE construction from [98]. The public parameters consist of the six group elements (g, u, h, w, v, $e(g,g)^\alpha$), which intuitively are utilized in two separate "layers" to achieve secure large universe CP-ABE. In the "attribute layer", the u, h terms provide a Boneh-Boyen-style [21] hash function ($u^A h$), while in the "secret sharing layer" the w term holds the secret randomness r during key generation and the shares of the secret randomness s during encryption. The v term is used to "bind" the two layers together. The g and $e(g,g)^\alpha$ terms are used to introduce the master secret key functionality and allow correct decryption.

We see here that the "layered" construction is the same as the KP-ABE construction. However, we notice an extra term in binding the two main layers. This is because the master secret key α is no longer split in shares during key generation and is appearing as an exponent of the group generator g. Due to the extra binding term we need the extra functionality in the more complex assumption q-$\mathsf{DPBDH2}$ provided by the powers of a.

6.2.1 Construction

Our scheme consists of the following four algorithms:

$\mathsf{Setup}(1^\lambda) \rightarrow (\mathrm{PP}, \mathrm{MSK})$**:** The setup algorithm calls the group generator algorithm $\mathcal{G}(1^\lambda)$ and gets the descriptions of the groups and the bilinear mapping $D = (p, \mathbb{G}, \mathbb{G}_T, e)$, where p is the prime order of the groups $\mathbb{G}$ and $\mathbb{G}_T$. The attribute universe is $\mathcal{U} = \mathbb{Z}_p$.

Then the algorithm picks the random terms $g, u, h, w, v \overset{R}{\leftarrow} \mathbb{G}$ and $\alpha \overset{R}{\leftarrow} \mathbb{Z}_p$. It outputs

$$\text{PP} = (D, g, u, h, w, v, e(g,g)^{\alpha}) \qquad \text{MSK} = (\alpha)$$

$\mathsf{KeyGen}(\text{MSK}, \mathcal{S} = \{A_1, A_2, \ldots, A_k\} \subseteq \mathbb{Z}_p) \to \text{SK}$**:** Initially, the key generation algorithm picks $k+1$ random exponents $r, r_1, r_2, \ldots, r_k \overset{R}{\leftarrow} \mathbb{Z}_p$. Then it computes $K_0 = g^{\alpha} w^{r}$, $K_1 = g^{r}$, and for every $\tau \in [k]$

$$K_{\tau,2} = g^{r_\tau} \text{ and } K_{\tau,3} = (u^{A_\tau} h)^{r_\tau} v^{-r}$$

The secret key output is $\text{SK} = (\mathcal{S}, K_0, K_1, \{K_{\tau,2}, K_{\tau,3}\}_{\tau\in[k]})$.

$\mathsf{Encrypt}(M \in \mathbb{G}_T, (A, \delta)) \to \text{CT}$**:** The encryption algorithm takes the plaintext message M and picks $\vec{y} = (s,\, y_2,\, \ldots,\, y_n)^{\top} \overset{R}{\leftarrow} \mathbb{Z}_p^{n\times 1}$. In the terminology of section 2.1, s is the random secret to be shared among the shares. The vector of the shares is

$$\vec{\lambda} = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)^{\top} = M\vec{y}$$

It then picks ℓ random exponents $t_1, t_2, \ldots, t_\ell \overset{R}{\leftarrow} \mathbb{Z}_p$ and calculates $C = M \cdot e(g,g)^{\alpha s}$, $C_0 = g^{s}$, and for every $\tau \in [\ell]$

$$C_{\tau,1} = w^{\lambda_\tau} v^{t_\tau}, \quad C_{\tau,2} = (u^{\delta(\tau)} h)^{-t_\tau} \text{ and } C_{\tau,3} = g^{t_\tau}$$

The ciphertext output is $\text{CT} = ((A, \delta), C, C_0, \{C_{\tau,1}, C_{\tau,2}, C_{\tau,3}\}_{\tau\in[\ell]})$.

$\mathsf{Decrypt}\,\{\text{SK}, \text{CT}\} \to m$**:** Firstly, the decryption algorithm calculates the set of rows in A providing a share to attributes in $\mathcal{S}$, i.e. $I = \{i : \delta(i) \in \mathcal{S}\}$.

Then it computes the constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that $\sum_{i \in I} \omega_i A_i = (1, 0, \ldots, 0)$, where A_i is the i-th row of the matrix A. According to the discussion in section 2.1, these constants exist if the set $\mathcal{S}$ is an authorized set of the policy.

Then it calculates

$$B = \frac{e(C_0, K_0)}{\prod_{i \in I} \{e(C_{i,1}, K_1) e(C_{i,2}, K_{\tau,2}) e(C_{i,3}, K_{\tau,3})\}^{\omega_i}}$$

where τ is the index of the attribute $\delta(i)$ in $\mathcal{S}$ (it depends on i). The algorithm outputs $M = C/B$.

Correctness: If the attribute set $\mathcal{S}$ of the secret key is authorized, we have that $\sum_{i \in I} \omega_i \lambda_i = s$. Therefore:

$$\begin{aligned} B &= \frac{e(g,g)^{\alpha s} e(g,w)^{rs}}{\prod_{i \in I} e(g,w)^{r\omega_i\lambda_i} e(g,v)^{rt_i\omega_i} e(g,u^{\delta(i)}h)^{-r_\tau t_i \omega_i} e(g,u^{\delta(i)}h)^{r_\tau t_i \omega_i} e(g,v)^{-rt_i\omega_i}} \\ &= \frac{e(g,g)^{\alpha s} e(g,w)^{rs}}{e(g,w)^{r\sum_{i \in I} \omega_i \lambda_i}} = e(g,g)^{\alpha s} \end{aligned}$$

6.2.2 Security Proof

We will prove the following theorem regarding the selective security of our CP-ABE scheme:

Theorem 6.2. *If the q-DPBDH2 assumption holds then all PPT adversaries with a challenge matrix of size $\ell \times n$, where $\ell, n \leq q$, have a negligible advantage in selectively breaking our scheme.*

Proof. To prove the theorem we will assume that there exists a PPT attacker $\mathcal{A}$ with a challenge matrix that satisfies the restriction, which has a non negligible advantage $\mathsf{Adv}_{\mathcal{A}}$ in selectively breaking our scheme. Using this attacker we will build a PPT simulator $\mathcal{B}$ that attacks the q-$\mathsf{DPBDH2}$ assumption with a non negligible advantage.

Initialization: $\mathcal{B}$ receives the given terms from the assumption and a challenge policy (A^*, δ^*) from $\mathcal{A}$. We have that A^* is an $\ell \times n$ matrix, where $\ell, n \leq q$, and $\delta^* : [\ell] \to \mathbb{Z}_p$.

Setup: The simulator $\mathcal{B}$ has to provide $\mathcal{A}$ the public parameters of the system. In order to do that it implicitly sets the master secret key of the scheme to be $\alpha = a^{q+1} + \tilde{\alpha}$, where a, q are set in the assumption and $\tilde{\alpha} \xleftarrow{R} \mathbb{Z}_p$ is a known to $\mathcal{B}$ random exponent. Notice that this way α is correctly distributed and a is information-theoretically hidden from $\mathcal{A}$. Then $\mathcal{B}$ picks the random exponents $\tilde{v}, \tilde{u}, \tilde{h} \xleftarrow{R} \mathbb{Z}_p$ and using the assumption gives to $\mathcal{A}$ the following public parameters:

$$\begin{array}{ll}
g = g & w = g^a \\
v = g^{\tilde{v}} \cdot \prod_{(j,k)\in[\ell,n]} \left(g^{a^k/b_j}\right)^{A^*_{j,k}} & u = g^{\tilde{u}} \cdot \prod_{(j,k)\in[\ell,n]} \left(g^{a^k/b_j^2}\right)^{A^*_{j,k}} \\
h = g^{\tilde{h}} \cdot \prod_{(j,k)\in[\ell,n]} \left(g^{a^k/b_j^2}\right)^{-\delta^*(j)A^*_{j,k}} & e(g,g)^{\alpha} = e(g^a, a^{a^q}) \cdot e(g,g)^{\tilde{\alpha}}
\end{array}$$

Since a is information-theoretically hidden from $\mathcal{A}$, the term w is properly uniformly random in $\mathbb{G}$. The terms v, u, h are properly distributed due to $\tilde{v}, \tilde{u}, \tilde{h}$ respectively. Notice that all terms can be calculated by the simulator using suitable terms from the assumption and the challenge policy given by

$\mathcal{A}$.

As one can see, the "attribute layer", which consists of the terms u, h, is made up of terms whose exponents have b_i^2 in the denominator, the "binder term" v has b_i, and the "secret sharing layer" w has only one power of a. This scaling of the powers of b_i will allow our simulator to properly simulate all terms.

Query phases 1 and 2: Now the simulator has to produce secret keys for non authorized sets of attributes requested by $\mathcal{A}$. In both phases the treatment is the same. We describe here the way $\mathcal{B}$ works in order to create a key for an attribute set $\mathcal{S} = \left\{A_1, A_2, \ldots, A_{|\mathcal{S}|}\right\}$ received by $\mathcal{A}$.

Since $\mathcal{S}$ is non authorized for (A^*, δ^*), there exists a vector $\vec{w} = (w_1, w_2, \ldots, w_n)^\top \in {\mathbb{Z}_p}^n$ such that $w_1 = -1$ and $\langle A_i^*, \vec{w} \rangle = 0$ for all $i \in I = \{i | i \in [\ell] \wedge \delta^*(i) \in \mathcal{S}\}$ (c.f. section 2.1). The simulator calculates $\vec{w}$ using linear algebra. Then it picks $\tilde{r} \xleftarrow{R} \mathbb{Z}_p$ and implicitly sets

$$r = \tilde{r} + w_1 a^q + w_2 a^{q-1} + \ldots + w_n a^{q+1-n} = \tilde{r} + \sum_{i \in [n]} w_i a^{q+1-i}$$

This is properly distributed due to $\tilde{r}$. Then using the suitable terms from the assumption it calculates:

$$K_0 = g^\alpha w^r = g^{a^{q+1}} g^{\tilde{\alpha}} g^{a\tilde{r}} \prod_{i \in [n]} g^{w_i a^{q+2-i}} = g^{\tilde{\alpha}} \left(g^a\right)^{\tilde{r}} \prod_{i=2}^{n} \left(g^{a^{q+2-i}}\right)^{w_i}$$
$$K_1 = g^r = g^{\tilde{r}} \prod_{i \in [n]} \left(g^{a^{q+1-i}}\right)^{w_i}$$

Additionally, for all $\tau \in [|\mathcal{S}|]$ it has to compute the terms $K_{\tau,2} = g^{r_\tau}$ and $K_{\tau,3} = (u^{A_\tau} h)^{r_\tau} v^{-r}$. The common part v^{-r} for these terms is the following:

$$\begin{aligned}
v^{-r} &= v^{-\tilde{r}} \left(g^{\tilde{v}} \prod_{(j,k)\in[\ell,n]} g^{a^k A^*_{j,k}/b_j} \right)^{-\sum_{i\in[n]} w_i a^{q+1-i}} \\
&= v^{-\tilde{r}} \prod_{i\in[n]} \left(g^{a^{q+1-i}} \right)^{-\tilde{v} w_i} \cdot \prod_{(i,j,k)\in[n,\ell,n]} g^{-w_i A^*_{j,k} a^{q+1+k-i}/b_j} \\
&= \underbrace{v^{-\tilde{r}} \prod_{i\in[n]} \left(g^{a^{q+1-i}} \right)^{-\tilde{v} w_i} \cdot \prod_{\substack{(i,j,k)\in[n,\ell,n] \\ i\neq k}} \left(g^{a^{q+1+k-i}/b_j} \right)^{-w_i A^*_{j,k}}}_{\Phi} \\
&\qquad \cdot \prod_{(i,j)\in[n,\ell]} g^{-w_i A^*_{j,i} a^{q+1}/b_j} \\
&= \Phi \cdot \prod_{j\in[\ell]} g^{-\langle \vec{w}, A^*_j \rangle a^{q+1}/b_j} \\
&= \Phi \cdot \prod_{\substack{j\in[\ell] \\ \delta^*(j)\notin\mathcal{S}}} g^{-\langle \vec{w}, A^*_j \rangle a^{q+1}/b_j}
\end{aligned}$$

The Φ part can be calculated by the simulator using the assumption, while the second part has to be canceled by the $(u^{A_\tau} h)^{r_\tau}$ part. So for every attribute $A_\tau \in S$ the simulator sets implicitly

$$\begin{aligned}
r_\tau &= \tilde{r}_\tau + r \cdot \sum_{\substack{i'\in[\ell] \\ \delta^*(i')\notin\mathcal{S}}} \frac{b_{i'}}{A_\tau - \delta^*(i')} \\
&= \tilde{r}_\tau + \tilde{r} \cdot \sum_{\substack{i'\in[\ell] \\ \delta^*(i')\notin\mathcal{S}}} \frac{b_{i'}}{A_\tau - \delta^*(i')} + \sum_{\substack{(i,i')\in[n,\ell] \\ \delta^*(i')\notin\mathcal{S}}} \frac{w_i b_{i'} a^{q+1-i}}{A_\tau - \delta^*(i')}
\end{aligned}$$

Where $\tilde{r}_\tau \overset{R}{\leftarrow} \mathbb{Z}_p$ and therefore r_τ is properly distributed. The use of the b_i's in the numerators of the fractions is explained by the "layer" intuition presented before. Namely, these b_i will cancel with the b_i^2 denominators in the "attribute layer" and provide a cancellation for the unknown part of v^{-r}.

Also, notice that r_τ is well-defined only for attributes in the specific unauthorized set $\mathcal{S}$ or unrelated attributes (outside the policy), since the sum is over the i' such that $\delta^*(i') \notin \mathcal{S}$. Therefore, for all $A_\tau \in \mathcal{S}$ or $A_\tau \notin \delta^*([\ell])$, the denominators $A_\tau - \delta^*(i')$ are non zero. If the simulator tries to include more attributes of the policy in the key (and possibly make a key for an authorized set), he would have to divide by zero (c.f. Figure 6.1).

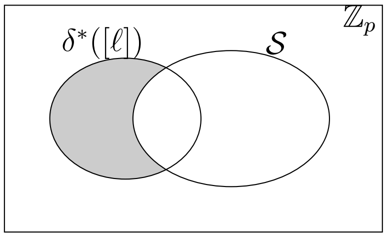

Figure 6.1: The simulator can not create the components for attributes in the gray area.

Therefore the first part of $K_{\tau,3} = (u^{A_\tau} h)^{r_\tau} v^{-r}$ is:

$$
\begin{aligned}
(u^{A_\tau} h)^{r_\tau} = {} & \left(g^{\tilde{u}A_\tau + \tilde{h}} \prod_{(j,k)\in[\ell,n]} g^{(A_\tau - \delta^*(j))A^*_{j,k} a^k / b_j^2} \right)^{\tilde{r} \cdot \sum_{i'\in[\ell],\delta^*(i')\notin\mathcal{S}} \frac{b_{i'}}{A_\tau - \delta^*(i')}} \\
& \cdot \left(g^{\tilde{u}A_\tau + \tilde{h}} \prod_{(j,k)\in[\ell,n]} g^{(A_\tau - \delta^*(j))A^*_{j,k} a^k / b_j^2} \right)^{\sum_{(i,i')\in[n,\ell],\delta^*(i')\notin\mathcal{S}} \frac{w_i b_{i'} a^{q+1-i}}{A_\tau - \delta^*(i')}} \cdot (u^{A_\tau} h)^{\tilde{r}_\tau} \\
= {} & (K_{\tau,2}/g^{\tilde{r}_\tau})^{\tilde{u}A_\tau + \tilde{h}} \cdot \prod_{\substack{(i',j,k)\in[\ell,\ell,n] \\ \delta^*(i')\notin\mathcal{S}}} g^{\tilde{r}(A_\tau - \delta^*(j))A^*_{j,k} b_{i'} a^k / (A_\tau - \delta^*(i')) b_j^2} \\
& \cdot \prod_{\substack{(i,i',j,k)\in[n,\ell,\ell,n] \\ \delta^*(i')\notin\mathcal{S}}} g^{(A_\tau - \delta^*(j)) w_i A^*_{j,k} b_{i'} a^{q+1+k-i} / (A_\tau - \delta^*(i')) b_j^2} \cdot (u^{A_\tau} h)^{\tilde{r}_\tau} \\
= {} & \Psi \cdot \prod_{\substack{(i,j)\in[n,\ell] \\ \delta^*(j)\notin\mathcal{S}}} g^{(A_\tau - \delta^*(j)) w_i A^*_{j,i} b_j a^{q+1+i-i} / (A_\tau - \delta^*(j)) b_j^2} \\
= {} & \Psi \cdot \prod_{\substack{j\in[\ell] \\ \delta^*(j)\notin\mathcal{S}}} g^{\langle \vec{w}, A^*_j \rangle a^{q+1} / b_j}
\end{aligned}
$$

Where $\Psi = (u^{A_\tau} h)^{\tilde{r}_\tau} \cdot (K_{\tau,2}/g^{\tilde{r}_\tau})^{\tilde{u}A_\tau + \tilde{h}}$

$$
\cdot \prod_{\substack{(i',j,k)\in[\ell,\ell,n] \\ \delta^*(i')\notin\mathcal{S}}} \left(g^{b_{i'} a^k / b_j^2} \right)^{\tilde{r}(A_\tau - \delta^*(j))A^*_{j,k} / (A_\tau - \delta^*(i'))}
$$

$$
\cdot \prod_{\substack{(i,i',j,k)\in[n,\ell,\ell,n] \\ \delta^*(i')\notin\mathcal{S},(j\neq i' \vee i\neq k)}} \left(g^{b_{i'} a^{q+1+k-i} / b_j^2} \right)^{(A_\tau - \delta^*(j)) w_i A^*_{j,k} / (A_\tau - \delta^*(i'))}
$$

and $K_{\tau,2} = g^{r_\tau} = g^{\tilde{r}_\tau} \cdot \prod_{\substack{i'\in[\ell] \\ \delta^*(i')\notin S}} \left(g^{b_{i'}} \right)^{\tilde{r}/(A_\tau - \delta^*(i'))} \cdot \prod_{\substack{(i,i')\in[n,\ell] \\ \delta^*(i')\notin S}} \left(g^{b_{i'} a^{q+1-i}} \right)^{w_i/(A_\tau - \delta^*(i'))}$

The Ψ and $K_{\tau,2}$ terms can be calculated using the suitable terms of our assumption[1]. The second part of $(u^{A_\tau}h)^{r_\tau}$ cancels exactly with the problematic part of v^{-r}. Therefore the simulator can calculate $K_{\tau,2}$ and $K_{\tau,3}$ for all $A_\tau \in \mathcal{S}$ and hand over the secret key $\mathrm{SK} = (\mathcal{S}, K_0, K_1, \{K_{\tau,2}, K_{\tau,3}\}_{\tau\in[|\mathcal{S}|]})$ to the attacker $\mathcal{A}$.

Challenge: The attacker will output a pair of messages (M_0, M_1) of the same length. In this phase the simulator flips a random coin $b \overset{R}{\leftarrow} \{0,1\}$ and constructs

$$C = M_b \cdot T \cdot e(g, g^s)^{\tilde{\alpha}} \qquad \text{and} \qquad C_0 = g^s$$

where T is the challenge term and g^s the corresponding term of the assumption.

The simulator sets implicitly $\vec{y} = (s,\, sa+\tilde{y}_2,\, sa^2+\tilde{y}_3,\, \ldots,\, sa^{n-1}+\tilde{y}_n)^\top$, where $\tilde{y}_2,\ \tilde{y}_3,\ \ldots,\ \tilde{y}_n \overset{R}{\leftarrow} \mathbb{Z}_p$. We see that the secret s and the vector $\vec{y}$ are properly distributed, since s was information theoretically hidden from $\mathcal{A}$ and the $\tilde{y}_i$'s are picked uniformly at random. As a result, since $\vec{\lambda} = A^*\vec{y}$ we have that

$$\lambda_\tau = \sum_{i\in[n]} A^*_{\tau,i} s a^{i-1} + \sum_{i=2}^{n} A^*_{\tau,i}\tilde{y}_i = \sum_{i\in[n]} A^*_{\tau,i} s a^{i-1} + \tilde{\lambda}_\tau$$

for each row $\tau \in [\ell]$. Notice that the terms $\tilde{\lambda}_\tau = \sum_{i=2}^{n} A^*_{\tau,i}\tilde{y}_i$ are known to the simulator. For each row the simulator $\mathcal{B}$ sets implicitly $t_\tau = -sb_\tau$. This

[1]Notice that for the products of Ψ we can have $j = i'$, but in that case the power of a is different than $q+1$. So the simulator can use the g^{a^i/b_j} terms.

is properly distributed as well, because the b_i's are information theoretically hidden from the attacker. Using the above, $\mathcal{B}$ calculates:

$$\begin{aligned}
C_{\tau,1} &= w^{\lambda_\tau} v^{t_\tau} = w^{\tilde{\lambda}_\tau} \cdot \prod_{i\in[n]} g^{A^*_{\tau,i} s a^i} \cdot \left(g^{sb_\tau}\right)^{-\tilde{v}} \cdot \prod_{(j,k)\in[\ell,n]} g^{-A^*_{j,k} a^k s b_\tau / b_j} = \\
&= w^{\tilde{\lambda}_\tau} \cdot \left(g^{sb_\tau}\right)^{-\tilde{v}} \cdot \prod_{i\in[n]} g^{A^*_{\tau,i} s a^i} \cdot \prod_{k\in[n]} g^{-A^*_{\tau,k} a^k s b_\tau / b_\tau} \cdot \prod_{\substack{(j,k)\in[\ell,n]\\ j\neq\tau}} g^{-A^*_{j,k} a^k s b_\tau / b_j} = \\
&= w^{\tilde{\lambda}_\tau} \cdot \left(g^{sb_\tau}\right)^{-\tilde{v}} \cdot \prod_{\substack{(j,k)\in[\ell,n]\\ j\neq\tau}} \left(g^{s a^k b_\tau / b_j}\right)^{-A^*_{j,k}} \\
C_{\tau,2} &= \left(u^{\delta^*(\tau)} h\right)^{t_\tau} = \left(g^{sb_\tau}\right)^{-(\tilde{u}\delta^*(\tau)+\tilde{h})} \cdot \left(\prod_{(j,k)\in[\ell,n]} g^{(\delta^*(\tau)-\delta^*(j)) A^*_{j,k} a^k / b_j^2} \right)^{-sb_\tau} \\
&= \left(g^{sb_\tau}\right)^{-(\tilde{u}\delta^*(\tau)+\tilde{h})} \cdot \prod_{\substack{(j,k)\in[\ell,n]\\ j\neq\tau}} \left(g^{s a^k b_\tau / b_j^2}\right)^{-(\delta^*(\tau)-\delta^*(j)) A^*_{j,k}} \\
C_{\tau,3} &= g^{t_\tau} = \left(g^{sb_\tau}\right)^{-1}
\end{aligned}$$

Notice that by using $t_\tau = -sb_\tau$ we "raised" the exponents of the "binder" term v so that they cancel with the unknown powers of w^{λ_τ}. Therefore, the simulator hands over the ciphertext $\mathrm{CT} = ((A^*, \delta^*), C, C_0, \{C_{\tau,1}, C_{\tau,2}, C_{\tau,3}\}_{\tau\in[\ell]})$ to the attacker $\mathcal{A}$.

Guess: After the query phase 2, where the simulator creates the secret keys as described above, the attacker outputs a guess b' for the challenge bit. If $b' = b$ the simulator outputs 0, i.e. it claims that the challenge term is $T = e(g,g)^{sa^{q+1}}$. Otherwise, it outputs 1.

If $T = e(g,g)^{sa^{q+1}}$ then $\mathcal{A}$ played the proper security game, because $C = M_b \cdot T \cdot e(g, g^s)^{\tilde{\alpha}} = M_b \cdot e(g,g)^{\alpha s}$. On the other hand, if T is a random term of $\mathbb{G}_T$ then all information about the message M_b is lost in the challenge ciphertext. Therefore the advantage of $\mathcal{A}$ is exactly 0. As a result if $\mathcal{A}$ breaks the proper security game with a non negligible advantage, then $\mathcal{B}$ has a non negligible advantage in breaking the q-DPBDH2 assumption. □

6.3 A Large-Universe Multi-Authority CP-ABE System

Our scheme in [97] constitutes an augmented version of the Lewko-Waters [73] CP-ABEconstruction and shares several of the existing techniques. Namely in order to allow for multiple authorities and prevent collusion between users' keys it utilizes a hash function H that maps global identities to group elements. This hash function is modeled as a Random Oracle in the security proof. We combined this technique with the technique from [112] that used a hash function F that hashes attributes to group elements; also modeled as a Random Oracle. This way we achieved a large universe construction and at the same time we overcome the restriction that each attribute is used only once. This is because the policies are not any more controlled by the authorities, but by the underlying attributes. And the Random Oracle usage naturally overcomes the "one-time" restriction. Finally in order to "bound" the different ciphertext terms together we use two secret sharing vectors: one that shares the secret z of the blinding factor and one that shares 0. In order to decrypt someone has to use them both; therefore collusion on decryption from different users is prevented.

6.3.1 Construction

Our proposed scheme consists of the following five algorithms:

$\mathsf{GlobalSetup}(1^\lambda) \rightarrow \mathrm{GP}$**:** The global setup algorithm takes as input the security parameter λ and chooses a bilinear group of prime order $p \in \Theta(2^\lambda)$. It also choses a function H mapping global identities $\mathcal{GID}$ to elements of $\mathbb{G}$ and another function F mapping strings, interpreted as attributes, to elements of $\mathbb{G}$. Both of these functions will be modeled as random oracles in the security proof.

We denote by $\mathcal{U}_\Theta$ the set of authorities and $\mathcal{U}$ the set of attributes. We assume that each attribute belongs to only one authority and it is easy to find the corresponding authority. $\mathcal{U}_\Theta$ is of polynomial size, while $\mathcal{U}$ may be of exponential size in the security parameter.

The algorithm outputs the global parameters $\mathrm{GP} = \{p, \mathbb{G}, H, F\}$.

$\mathsf{AuthSetup}(\mathrm{GP}) \rightarrow \{\mathrm{PK}, \mathrm{SK}\}$**:** The authority setup algorithm chooses two random exponents $\alpha, y \in \mathbb{Z}_p$ and publishes $\mathrm{PK} = \{e(g, g)^\alpha, g^y\}$ as its public key. It keeps $\mathrm{SK} = \{\alpha, y\}$ as its secret key.

$\mathsf{KeyGen}(\mathcal{GID}, \theta, u, \mathrm{SK}, \mathrm{GP}) \rightarrow \{\mathrm{K}_{\theta,u,\mathcal{GID}}, \mathrm{K}'_{\theta,u,\mathcal{GID}}\}$**:** The key generation algorithm takes as input the user's global identifier $\mathcal{GID}$, the identifier θ of the authority, the attribute u to create a key for as well as the authority's secret key and the global parameters.

The algorithm first chooses a fresh random $t \xleftarrow{R} \mathbb{Z}_p$ and it computes:

$$\mathrm{K}_{\theta,u,\mathcal{GID}} = g^{\alpha_\theta} H(\mathcal{GID})^{y_\theta} F(u)^t,\ \mathrm{K}'_{\theta,u,\mathcal{GID}} = g^t$$

$\mathsf{Encrypt}(M, (A, \rho, \delta), \mathrm{GP}, \{\mathrm{PK}\}) \rightarrow \mathrm{CT}$**:** The encryption algorithm takes in a message M, an $\ell \times n$ access matrix A with ρ mapping its rows to authorities and δ mapping rows to attributes for each authority. The algorithm also takes in global parameters, and the public keys of the relevant authorities. We use the notation $(e(g,g)^{\alpha_\theta}, g^{y_\theta})$ to refer to the public key of authority θ.

The algorithm first chooses a random $z \in \mathbb{Z}_p$ and a random vector $\vec{v} \in \mathbb{Z}_p^n$ with z as its first entry. We let λ_x denote the share $\left\langle \vec{A}_x, \vec{v} \right\rangle$, where $\vec{A}_x$ is row x of A. It also chooses a random vector $\vec{w} \in \mathbb{Z}_p^n$ with 0 as its first entry. We let ω_x denote the share $\left\langle \vec{A}_x, \vec{w} \right\rangle$.

For each row x of A, it chooses a random $t_x \in \mathbb{Z}_p$. The ciphertext is computed as:

$$\begin{aligned} C_0 &= Me(g,g)^s \\ \{C_{1,x} &= e(g,g)^{\lambda_x} e(g,g)^{\alpha_{\rho(x)} t_x}, C_{2,x} = g^{-t_x}, \\ C_{3,x} &= g^{y_{\rho(x)} t_x} g^{\omega_x}, C_{4,x} = F(\delta(x))^{t_x}\}_{x \in [\ell]} \end{aligned}$$

$\mathsf{Decrypt}(\mathrm{CT}, \{\mathrm{K}_{\theta,u,\mathcal{GID}}\}, \mathrm{GP}) \rightarrow M$**:** We assume the ciphertext is encrypted under an access matrix (A, ρ, δ). To decrypt, the decryptor first computes $H(\mathcal{GID})$. If the decryptor has the secret keys $\{\mathrm{K}_{\rho(x),\delta(x),\mathcal{GID}}\}$ for a subset of rows $\vec{A}_x$ of A such that $(1,0,\ldots,0)$ is in the span of these rows, then the decryptor proceeds as follows:

For each such x, the decryptor computes:

$$\begin{aligned}
&C_{1,x} \cdot e(\mathrm{K}_{\rho(x),\delta(x),\mathcal{GID}}, C_{2,x}) \cdot e(H(\mathcal{GID}), C_{3,x}) \cdot e(\mathrm{K}'_{\rho(x),\delta(x),\mathcal{GID}}, C_{4,x}) \\
&= e(g,g)^{\lambda_x} e(H(\mathcal{GID}), g)^{\omega_x}
\end{aligned}$$

The decryptor then chooses constants $c_x \in \mathbb{Z}_p$ such that $\sum_x c_x \vec{A}_x = (1,0,\ldots,0)$ and computes:

$$\prod_x \left(e(g,g)^{\lambda_x} e(H(\mathcal{GID}), g)^{\omega_x}\right)^{c_x} = e(g,g)^z$$

We recall that $\lambda_x = \left\langle \vec{A}_x, \vec{v} \right\rangle$ and $\omega_x = \left\langle \vec{A}_x, \vec{w} \right\rangle$, where $\langle (1,0,\ldots,0),\, \vec{v} \rangle = s$ and $\langle \vec{w},\, (1,0,\ldots,0) \rangle = 0$. The message can then be obtained as:

$$M = C_0 / e(g,g)^z$$

Remark 6.3. Notice that for the users' secret keys and the ciphertexts a re-randomizing technique is applicable using only the public parameters. Namely, if someone has a key $(\mathrm{K}_{\theta,u,\mathcal{GID}},\, \mathrm{K}'_{\theta,u,\mathcal{GID}})$, he can acquire a new key for $(\mathcal{GID},\, \theta,\, u)$ by picking $t' \xleftarrow{R} \mathbb{Z}_p$ and constructing $(\mathrm{K}_{\theta,u,\mathcal{GID}} F(u)^{t'},\, \mathrm{K}'_{\theta,u,\mathcal{GID}} g^{t'})$. For the ciphertext the re-randomization can be done by picking new random vectors $\vec{v}', \vec{w}'$ with the first element 0 and new $t'_x \xleftarrow{R} \mathbb{Z}_p$. Then the re-randomized terms for row x are

$$(C_{1,x}e(g,g)^{\langle \vec{A}_x, \vec{v}' \rangle} e(g,g)^{\alpha_{\rho(x)} t'_x}, C_{2,x} g^{-t'_x}, C_{3,x} g^{y_{\rho(x)} t'_x} g^{\langle \vec{A}_x, \vec{w}' \rangle}, C_{4,x} F(\delta(x))^{t_x})$$

We will use these re-randomizing techniques in our security reduction to provide properly distributed components.

6.3.2 Security Proof

In our security proof we combined several techniques, which we think might be of independent interest in the study of CP-ABE systems. The first technique allows the simulator of our reduction to isolate an unauthorized set of rows and essentially ignore it for the remaining of the security reduction. It can ignore the contributions of these rows even in the construction of the challenge ciphertext. In our case the simulator does that for the corrupt authorities, which are controlled by the adversary. The claim that makes this technique possible is shown below and the proof is in appendix B. The claim allows the simulator to "zero-out" a subset of columns for the unauthorized set.

Claim 6.4. *Let $A \in \mathbb{Z}_p^{\ell \times n}$ be the secret sharing matrix of a linear secret sharing scheme for an access policy $\mathbb{A}$ and let $\mathcal{C} \subseteq [\ell]$ be a non-authorized set of rows. Let $c \in \mathbb{N}$ be the dimension of the subspace spanned by the rows of $\mathcal{C}$.*

Then the distribution of the shares $\{\lambda_x\}_{x \in [\ell]}$ sharing the secret $z \in \mathbb{Z}_p$ generated with the matrix A is the same as the distribution of the shares $\{\lambda'_x\}_{x \in [\ell]}$ sharing the secret $z \in \mathbb{Z}_p$ generated with some matrix A', where $A'_{x,j} = 0$ for $x \in \mathcal{C}$ and $j \in [n-c]$ (see figure 6.2).

Moreover A' is computable from A in polynomial time.

Another technique utilized in the security proof is the "splitting" of the unknown parameters to two different vectors. Namely the secret sharing vector $\vec{v}$ will hold the secret sa^{q+1} on only the first position and the zero sharing vector $\vec{w}$ will hold the unknown terms $sa^q, sa^{q-1}, \ldots, sa^2$ on all positions but the first. During the generation of the secret keys these terms are "recombined" to give a full series of q terms that are canceled by the attribute term.

Our main theorem is the following:

Theorem 6.5. *If the q-DPBDHE assumption holds, then all PPT adversaries with a challenge matrix of size at most $q \times q$ have a negligible advantage in statically breaking our scheme in the Random Oracle Model.*

Proof. In order to prove the theorem we assume that there exists a PPT adversary Adv that breaks the scheme with more than negligible advantage and we show how to construct a PPT algorithm $\mathcal{B}$ that simulates the static security game with Adv and breaks the q-DPBDHE assumption. Our simulator works as follows:

Global Parameters: Initially, it gets (D, T) from its q-DPBDHE challenger and sends the public parameters $\mathrm{GP} = (\mathbb{G},\, p,\, g)$ to Adv. The two random oracles H, F will be programmed by the simulator.

Static security: According to the static security game, the attacker Adv outputs a set of corrupt authorities $\mathcal{C}_\Theta \subseteq \mathcal{U}_\Theta$. It also outputs the sequence

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} & \dots & A_{1,n} \\ A_{2,1} & A_{2,2} & \dots & A_{2,n} \\ A_{3,1} & A_{3,2} & \dots & A_{3,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{\ell,1} & A_{\ell,2} & \dots & A_{\ell,n} \end{bmatrix} \rightsquigarrow A' = \left[\begin{array}{ccc|ccc} 0 & \dots & 0 & A'_{1,n-c+1} & \dots & A'_{1,n} \\ A'_{2,1} & \dots & A'_{2,n-c} & A'_{2,n-c+1} & \dots & A'_{2,n} \\ 0 & \dots & 0 & A'_{3,n-c+1} & \dots & A'_{3,n} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ A'_{\ell,1} & \dots & A'_{1,n-c} & A'_{1,n-c+1} & \dots & A'_{\ell,n} \end{array}\right]$$

Figure 6.2: Transformation of the policy matrix A to be used by the simulator. Rows that belong to corrupted authorities are highlighted.

$\mathcal{Q} = \{(\mathcal{GID}_i, S_i)\}_{i=1}^{m}$ of the secret key queries, where $S_i \subseteq (\mathcal{U}_\Theta \setminus \mathcal{C}_\Theta) \times \mathcal{U}$, and sends two messages $(M_0, M_1) \in \mathbb{G}_T^2$ with a challenge policy (A, ρ, δ), where $A \in \mathbb{Z}_p^{\ell \times n}$, $\rho : [\ell] \to \mathcal{U}_\Theta$, and $\delta : [\ell] \to \mathcal{U}$.

Since we are in the Random Oracle Model, the attacker also outputs a sequence $\mathcal{L}_{\mathcal{ID}}$ of global identities for the H oracle queries and a sequence $\mathcal{L} \subseteq \mathcal{U}$ of attributes for the F oracle queries. W.l.o.g. we assume that all global ID's and all attributes present in $\mathcal{Q}$ are queried on their respective oracle.

In order to proceed, the simulator substitutes the secret sharing matrix A with the matrix A' from 6.4. After $\mathcal{B}$ calculates the matrix A' (shown in figure 6.2) where $\mathcal{C} = \mathcal{C}_\Theta$ it proceeds to compute all the inputs to Adv. According to the above claim, if $\mathcal{B}$ uses A' instead of A in the simulation the view of Adv in this game is information-theoretically the same as if it used the given matrix A. In the remaining of the proof, we use $n' = n - c$.

Authority Public Keys: The simulator has to provide the public keys of all non-corrupted authorities. To do that it considers two cases:

If the authority in question, θ, is not in the challenge policy, i.e. $\theta \notin \rho[\ell] \cup \mathcal{C}_\Theta$, the simulator $\mathcal{B}$ picks $\alpha_\theta, y_\theta \xleftarrow{R} \mathbb{Z}_p$ itself and outputs the public key

$$(e(g,g)^{\alpha_\theta}, g^{y_\theta})$$

For each authority $\theta \in \rho[\ell] \setminus \mathcal{C}_\Theta$, let $X = \{x | \rho(x) = \theta\} \subseteq [\ell]$. This is the set of rows in the challenge policy that belong to authority θ. Then the simulator $\mathcal{B}$ picks $\tilde{\alpha}_\theta, \tilde{y}_\theta \xleftarrow{R} \mathbb{Z}_p$ and sets implicitly

$$\alpha_\theta = \tilde{\alpha}_\theta + \sum_{x \in X} b_x a^{q+1} A'_{x,1} \quad \text{and} \quad y_\theta = \tilde{y}_\theta + \sum_{x \in X} \sum_{j=2}^{n'} b_x a^{q+2-j} A'_{x,j}$$

It outputs the public key

$$(e(g,g)^{\alpha_\theta}, g^{y_\theta}) = \left(e(g,g)^{\tilde{\alpha}_\theta} \prod_{x \in X} e(g^{b_x a}, g^{a^q})^{A'_{x,1}}, g^{\tilde{y}_\theta} \prod_{x \in X} \prod_{j=2}^{n'} \left(g^{b_x a^{q+2-j}} \right)^{A'_x,j} \right)$$

Since $n' = n - c \leq q$ and $\ell \leq q$, the simulator can compute all these terms using suitable terms of the assumption. Also due to $\tilde{\alpha}_\theta \tilde{y}_\theta$ these terms are properly distributed.

***H*-Oracle Queries:** If the queried global identity $\mathcal{GID}$ is in $\mathcal{L}_{\mathcal{ID}}$ but not in $\{\mathcal{GID}_i\}_{i \in [m]}$, then the simulator outputs a random element of $\mathbb{G}$ for $H(\mathcal{GID})$. These elements are not going to be used anywhere else.

If the queried global identity is equal to $\mathcal{GID}_i$ for some i and there is no row x such that $(\rho(x), \delta(x)) \in S_i$ (i.e. if this user is not entitled to any shares), then the simulator picks $\tilde{h}_i \xleftarrow{R} \mathbb{Z}_p$ and outputs

$$H(\mathcal{GID}_i) = g^{\tilde{h}_i} \cdot g^a \cdot g^{a^2} \cdot \cdots \cdot g^{a^{n'-1}} = g^{\tilde{h}_i} \prod_{k=2}^{n'} g^{a^{k-1}}$$

Otherwise, we should consider the case where for some rows $X' \subseteq [\ell]$ it is true that $(\rho(x), \delta(x)) \in S_i$. According to our restriction we know that the set of these rows together with the set of the rows that belong to corrupted authorities is non-authorized. This means that there exists a vector $\vec{d_i} \in \mathbb{Z}_p^n$ such that the first element is $d_{i,1} = 1$ and the inner product of it with any of the aforementioned rows is equal to zero.

Additionally, according to the construction of A' we know that the set of the corrupted rows spans the entire subspace of dimension c. That means the vector $\vec{d_i}$ is orthogonal to any of the vectors $(\overbrace{0, \ldots, 0}^{n'}, \overbrace{0, \ldots, 0, 1, 0, \ldots, 0}^{c}) \in \mathbb{Z}_p^n$. These are the vectors with exactly one "1" in one of the last c positions. This implies that $d_{i,j} = 0$ for $n - c + 1 \leq j \leq n$. Hence $\left\langle \vec{A'_x}, \vec{d_i} \right\rangle = 0$ even if we restrict the row $\vec{A'_x}$ and the vector $\vec{d_i}$ to the first $n' = n - c$ positions. We will denote this inner product by $\overline{\left\langle \vec{A'_x}, \vec{d_i} \right\rangle}$.

In this case the simulator picks $\tilde{h}_i \xleftarrow{R} \mathbb{Z}_p$ and outputs

$$H(\mathcal{GID}_i) = g^{\tilde{h}_i} \cdot (g^a)^{d_{i,2}} \cdot (g^{a^2})^{d_{i,3}} \cdot \cdots \cdot (g^{a^{n'-1}})^{d_{i,n'}} = g^{\tilde{h}_i} \prod_{k=2}^{n'} \left(g^{a^{k-1}} \right)^{d_{i,k}}$$

***F*-Oracle Queries:** Let θ be the authority of the queried attribute u. Then if $\theta \notin \rho[\ell]$ or $\theta \in \mathcal{C}_\Theta$, the simulator outputs a random element of $\mathbb{G}$ for $F(u)$ and stores the value so that he might reuse it in a secret key query.

If $\theta \in \rho[\ell]$, let $X = \{x | \rho(x) = \theta\} \subseteq [\ell]$. Then if $u \notin \delta[\ell]$ (i.e. the attribute is not used at the challenge policy), the simulator picks $\tilde{f}_u \xleftarrow{R} \mathbb{Z}_p$ and outputs

$$F(u) = g^{\tilde{f}_u} g^{\sum_{x\in X}\sum_{j\in[n']} b_x a^{q+1-j} A'_{x,j}} = \tilde{f}_u \prod_{x\in X}\prod_{j\in[n']}\left(g^{b_x a^{q+1-j}}\right)^{A'_{x,j}}$$

Otherwise, i.e. $u \in \delta[\ell]$, let $X'' = X\backslash\{x|\delta(x) = u\}$. Therefore, X'' is the set of rows that belong to authority θ but *do not have* u as the corresponding attribute. Then the simulator picks $\tilde{f}_u \xleftarrow{R} \mathbb{Z}_p$ and outputs

$$F(u) = g^{\tilde{f}_u} g^{\sum_{x\in X''}\sum_{j\in[n']} b_x a^{q+1-j} A'_{x,j}} = g^{\tilde{f}_u} \prod_{x\in X''}\prod_{j\in[n']}\left(g^{b_x a^{q+1-j}}\right)^{A'_{x,j}}$$

Secret Keys: Consider the query $(\mathcal{GID}_i, S_i)$ where $S_i \subseteq \mathcal{U}_\Theta \times \mathcal{U}$.

First, consider the case where there is no row x such that $(\rho(x), \delta(x)) \in S_i$. Then according to the above $H(\mathcal{GID}_i) = g^{\tilde{h}_i} g^{\sum_{k=2}^{n} a^{k-1}}$. We have to consider two cases for each element (θ, u) of S_i:

- $\theta \notin \rho[\ell]$: Here the simulator knows α_θ and y_θ. Therefore it picks $t \xleftarrow{R} \mathbb{Z}_p$ and outputs

$$K_{\theta,u,\mathcal{GID}_i} = g^{\alpha_\theta} H(\mathcal{GID}_i)^{y_\theta} F(u)^t \quad \text{and} \quad K'_{\theta,u,\mathcal{GID}_i} = g^t$$

- $\theta \in \rho[\ell]$: Here we know that there is no row (θ, u) in the policy. Therefore we have that $u \notin \delta[\ell]$ (remember that each attribute belongs to exactly one authority); hence $F(u) = g^{\tilde{f}_u} g^{\sum_{x\in X}\sum_{j\in[n']} b_x a^{q+1-j} A'_{x,j}}$. In this case the simulator sets implicitly $t = -\sum_{k\in[n']} a^k$ and computes the key

$$\begin{aligned}
K_{\theta,u,\mathcal{GID}_i} &= g^{\alpha_\theta} H(\mathcal{GID}_i)^{y_\theta} F(u)^t \\
&= g^{\sum_{x\in X} b_x a^{q+1} A'_{x,1}} g^{\sum_{x\in X}\sum_{j=2}^{n'}\sum_{k=2}^{n'} b_x a^{q+1+k-j} A'_{x,j}} \\
&\quad \cdot g^{-\sum_{x\in X}\sum_{j\in[n']}\sum_{k\in[n']} b_x a^{q+1+k-j} A'_{x,j}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
&= g^{-\sum_{x\in X}\sum_{j=2}^{n'} b_x a^{q+2-j} A'_{x,j}} g^{-\sum_{x\in X}\sum_{k=2}^{n'} b_x a^{q+k} A_{x,1}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
&= g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
&\quad \cdot \prod_{x\in X}\prod_{j=2}^{n'} \left(g^{b_x a^{q+2-j}}\right)^{-A'_{x,j}} \cdot \prod_{x\in X}\prod_{k=2}^{n'} \left(g^{b_x a^{q+k}}\right)^{-A'_{x,1}} \\
K'_{\theta,u,\mathcal{GID}_i} &= g^t = \prod_{k\in[n']} \left(g^{a_k}\right)^{-1}
\end{aligned}$$

Finally it re-randomizes this key on t using the public parameters and outputs the re-randomized key.

If there is a row x such that $(\rho(x), \delta(x)) \in S_i$, then $H(\mathcal{GID}_i) = g^{\tilde{h}_i} g^{\sum_{k=2}^{n} a^{k-1} d_{i,k}}$. We consider the following cases for each element (θ, u) of S_i:

- $\theta \notin \rho[\ell]$: Here the simulator knows α_θ and y_θ. Therefore it picks $t \xleftarrow{R} \mathbb{Z}_p$ and outputs

$$K_{\theta,u,\mathcal{GID}_i} = g^{\alpha_\theta} H(\mathcal{GID}_i)^{y_\theta} F(u)^t \quad \text{and} \quad K'_{\theta,u,\mathcal{GID}_i} = g^t$$

- $\theta \in \rho[\ell]$ and $u \notin \delta[\ell]$: As before $F(u) = g^{\tilde{f}_u} g^{\sum_{x\in X}\sum_{j\in[n']} b_x a^{q+1-j} A'_{x,j}}$. The simulator sets implicitly $t = -\sum_{k\in[n']} a^k d_{i,k}$ and outputs

$$\begin{aligned}
K_{\theta,u,\mathcal{GID}_i} &= g^{\alpha_\theta} H(\mathcal{GID}_i)^{y_\theta} F(u)^t \\
&= g^{\sum_{x\in X} b_x a^{q+1} A'_{x,1}} g^{\sum_{x\in X}\sum_{j=2}^{n'}\sum_{k=2}^{n'} b_x a^{q+1+k-j} A'_{x,j} d_{i,k}} \\
&\quad \cdot g^{-\sum_{x\in X}\sum_{j\in[n']}\sum_{k\in[n']} b_x a^{q+1+k-j} A'_{x,j} d_{i,k}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
&= g^{-\sum_{x\in X}\sum_{j=2}^{n'} b_x a^{q+2-j} A'_{x,j} d_{i,1}} g^{-\sum_{x\in X}\sum_{k=2}^{n'} b_x a^{q+k} A'_{x,1} d_{i,k}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
&= \prod_{x\in X}\prod_{j=2}^{n'} \left(g^{b_x a^{q+2-j}}\right)^{-A'_{x,j}} \cdot \prod_{x\in X}\prod_{k=2}^{n'} \left(g^{b_x a^{q+k}}\right)^{-A'_{x,1} d_{i,k}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
K'_{\theta,u,\mathcal{GID}_i} &= g^t = \prod_{k\in[n']} \left(g^{a_k}\right)^{-d_{i,k}}
\end{aligned}$$

As before it re-randomizes this key on t using the public parameters and outputs the re-randomized key.

- $\theta \in \rho[\ell]$ and $u \in \delta[\ell]$: In this case we have

$$F(u) = g^{\tilde{f}_u} g^{\sum_{x\in X''}\sum_{j\in[n']} b_x a^{q+1-j} A'_{x,j}} \text{ (with } X''\text{)}$$

The simulator sets implicitly $t = -\sum_{k\in[n']} a^k d_{i,k}$ and outputs

$$\begin{aligned}
K_{\theta,u,\mathcal{GID}_i} &= g^{\alpha_\theta} H(\mathcal{GID}_i)^{y_\theta} F(u)^t \\
&= g^{\sum_{x\in X} b_x a^{q+1} A'_{x,1}} g^{\sum_{x\in X}\sum_{j=2}^{n'}\sum_{k=2}^{n'} b_x a^{q+1+k-j} A'_{x,j} d_{i,k}} \\
&\quad \cdot g^{-\sum_{x\in X''}\sum_{j\in[n']}\sum_{k\in[n']} b_x a^{q+1+k-j} A'_{x,j} d_{i,k}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
&= g^{\sum_{x\in X\setminus X''} b_x a^{q+1} \overline{\left\langle \vec{A}'_x, \vec{d}_i \right\rangle}} g^{\sum_{x\in X\setminus X''} \sum_{\substack{j=2,k=2\\ j\neq k}}^{n',n'} b_x a^{q+1+k-j} A'_{x,j} d_{i,k}} \\
&\quad \cdot g^{-\sum_{x\in X}\sum_{j=2}^{n'} b_x a^{q+2-j} A'_{x,j} d_{i,1}} g^{-\sum_{x\in X}\sum_{k=2}^{n'} b_x a^{q+k} A'_{x,1} d_{i,k}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
&= \prod_{x\in X\setminus X''} \prod_{\substack{j=2,k=2\\ j\neq k}}^{n',n'} \left(g^{b_x a^{q+1+k-j}}\right)^{A'_{x,j} d_{i,k}} \cdot \prod_{x\in X} \prod_{j=2}^{n'} \left(g^{b_x a^{q+2-j}}\right)^{-A'_{x,j}} \\
&\quad \cdot \prod_{x\in X} \prod_{k=2}^{n'} \left(g^{b_x a^{q+k}}\right)^{-A'_{x,1} d_{i,k}} \\
&\quad \cdot g^{\tilde{\alpha}_\theta} H(\mathcal{GID}_i)^{\tilde{y}_\theta} \left(g^{y_\theta}\right)^{\tilde{h}_i} \left(g^t\right)^{\tilde{f}_u} \\
K'_{\theta,u,\mathcal{GID}_i} &= g^t = \prod_{k\in[n']} \left(g^{a_k}\right)^{-d_{i,k}}
\end{aligned}$$

As before it re-randomizes this key on t using the public parameters and outputs the re-randomized key. Notice that $X \setminus X''$ contains rows that map to (θ, u) in the challenge policy. Therefore, according to our discussion in the creation of the secret keys $\overline{\left\langle \vec{A}_x, \vec{d}_i \right\rangle} = 0$.

Challenge Ciphertext: The first part of the ciphertext is calculated as $C_0 = M_b \cdot T$, where $b \overset{R}{\leftarrow} \{0,1\}$ is a random coin and T is the challenge term. Thus the simulator $\mathcal{B}$ implicitly set $z = sa^{q+1}$.

The simulator also sets implicitly

$$\vec{v} = \left(sa^{q+1}, 0, \ldots, 0\right) \in \mathbb{Z}_p^n \quad \text{and} \quad \vec{w} = \left(\overbrace{0, sa^q, \ldots, sa^{q-n'+2}}^{n'}, 0, \ldots, 0\right) \in \mathbb{Z}_p^n$$

Therefore for a row $x^* \in [\ell]$ that belongs to a corrupted authority we have that $\lambda_{x^*} = 0$ and $\omega_{x^*} = 0$, due to the fact that these rows have all "0"s in the first n' columns. Thus for these rows the simulator picks $t_{x^*} \overset{R}{\leftarrow} \mathbb{Z}_p$ and using the public key $\{e(g,g)^{\alpha_\theta}, g^{y_\theta}\}$ of the corrupted authority it computes:

$$\begin{aligned}
C_{1,x^*} &= e(g,g)^{\lambda_{x^*}} e(g,g)^{\alpha_{\rho(x^*)} t_{x^*}} = \left(e(g,g)^{\alpha_{\rho(x^*)}}\right)^{t_{x^*}} \\
C_{2,x^*} &= g^{-t_{x^*}} \\
C_{3,x^*} &= g^{y_{\rho(x^*)} t_{x^*}} g^{\omega_{x^*}} = \left(g^{y_{\rho(x^*)}}\right)^{t_{x^*}} \\
C_{4,x^*} &= F(\delta(x^*))^{t_{x^*}}
\end{aligned}$$

On the other hand for a row x^* that does not belong to corrupted authorities, we have that $\lambda_{x^*} = sa^{q+1} \cdot A'_{x^*,1}$ and $\omega_{x^*} = \sum_{j=2}^{n'} sa^{q+2-j} A'_{x^*,j}$. For each one of these rows $\mathcal{B}$ sets implicitly $t_{x^*} = -s/b_{x^*}$ and computes:

$$\begin{aligned}
C_{1,x^*} &= e(g,g)^{\lambda_{x^*}} e(g,g)^{\alpha_{\rho(x^*)} t_{x^*}} = e(g,g)^{sa^{q+1} A'_{x^*,1}} e(g,g)^{-\sum_{x \in X} sb_x a^{q+1} A'_{x^*,1}/b_{x^*}} \\
&= \prod_{x \in X \setminus \{x^*\}} e(g, g^{sb_x a^{q+1}/b_{x^*}})^{-A'_{x^*,1}} \\
C_{2,x^*} &= g^{-t_{x^*}} = g^{s/b_{x^*}}
\end{aligned}$$

$$
\begin{aligned}
C_{3,x^*} &= g^{y_{\rho(x^*)} t_{x^*}} g^{\omega_{x^*}} = g^{-\sum_{x\in X}\sum_{j=2}^{n'} s b_x a^{q+2-j} A'_{x^*,j}/b_{x^*}} g^{\sum_{j=2}^{n'} s a^{q+2-j} A'_{x^*,j}} \\
&= \prod_{x\in X\setminus\{x^*\}} \prod_{j=2}^{n'} \left(g^{s b_x a^{q+2-j}/b_{x^*}}\right)^{-A'_{x^*,j}} \\
C_{4,x^*} &= F(\delta(x^*))^{t_{x^*}} = g^{-\sum_{x\in X''}\sum_{j\in[n']} s b_x a^{q+1-j} A'_{x^*,j}/b_{x^*}} \\
&= \prod_{x\in X''} \prod_{j\in[n']} \left(g^{s b_x a^{q+1-j}/b_{x^*}}\right)^{-A'_{x^*,j}}
\end{aligned}
$$

Notice that $x^* \notin X''$. Therefore the simulator can compute C_{4,x^*}. Finally the simulator re-randomizes the ciphertext on the vectors $\vec{v}$ and $\vec{w}$ and on the exponents t using the public parameters.

Guess: If the attacker Adv correctly guessed the bit b, then the simulator $\mathcal{B}$ outputs that the challenge term was $e(g,g)^{sa^{q+1}}$. That is because in this case it simulated the static security game perfectly. If the attacker did not guess the bit correctly, the simulator answer that T was a random group element. In this case the simulator produced an encryption of a random message. Therefore, if Adv is successful with more than negligible advantage, so is $\mathcal{B}$. $\square$

CHAPTER 7

Implementations and Benchmarks

7.1 Charm Framework

We implemented our scheme in Charm [2]; a framework developed to facilitate the rapid prototyping of cryptographic schemes and protocols. It is based on the Python language which allows the programmer to write code similar to the theoretical descriptions. However, the routines that implement the dominant group operations use the PBC library [76] (written natively in C) and the time overhead imposed by the use of Python is usually less than 1%. Charm also provides routines for applying and using LSSS schemes needed for Attribute-Based systems. For more information on Charm we refer the reader to [2, 32].

We tested several ABE constructions on all elliptic curve bilinear groups provided by Charm, i.e. three super-singular symmetric EC groups and two "MNT" [79] asymmetric EC groups. In Table 7.1 we present the approximate security level each group provides with respect to the discrete log problem. Although this does not necessarily translates to the security level of our assumption (or the various assumptions of the other ABE schemes), it provides an intuitive comparison between the security levels of the different instantiations. For more information on the security of discrete log and of q-type assumptions we refer the reader to [48, 67, 83, 91].

Curve	Security Level (Bits)
SS512	80
SS1024	112
MNT159	70
MNT201	90
MNT224	100

Table 7.1: Approximate security levels in bits of the ECC groups supported by the Charm framework. "SS" are super singular curves (symmetric bilinear groups), while "MNT" are the Miyaji, Nakabayashi, Takano curves (asymmetric bilinear groups). The number after the type of the curve denotes the size of the base field in bits.

7.2 Implementation Details

All Charm routines use formally asymmetric groups (although the underlining groups might be symmetric) and therefore we translated our schemes to the asymmetric setting. Namely, we have three groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$ and the pairing e is a function from $\mathbb{G}_1 \times \mathbb{G}_2$ to $\mathbb{G}_T$. We note here that we tried to implement our algorithms so that more operations are executed in the $\mathbb{G}_1$ group than in the $\mathbb{G}_2$ and that encryption consists mainly of operations in $\mathbb{G}_1$, compared to key generation. The reason is that the time taken to execute them in the $\mathbb{G}_1$ group is considerably smaller than $\mathbb{G}_2$ in specific asymmetric groups such as the "MNT" groups.

The source code of our implementations can be found in [106]. All our benchmarks were executed on a dual core Intel® Xeon® CPU W3503 @2.40GHz with 2.0GB RAM running Ubuntu R10.04 and Python3.2.3.

7.3 Benchmarks KP-ABE and CP-ABE

We compare our single-authority large universe constructions of Chap. 6 with the three known unbounded constructions on prime order groups. In Table 7.2 we present time benchmarks in different elliptic curve groups for some sample policies ($\approx$ size 4 attributes). Asymptotic results on the group exponentiations are shown in Sec. 7.4.

Regarding the comparison between our schemes and prior works, we notice the big gap between the timings of our constructions and prior ones. This is due to the fact that dual vector spaces of high dimension ($\approx$ 10 - 14) are utilized, which increase the number of group operations by big factors. We remind the reader that the OT schemes are fully secure, while the RW and LW selectively secure.

Regarding the practicality, in general, of both our schemes we notice that the KeyGen, Encrypt, and Decrypt times of our algorithms are relatively small. They are all under 100ms, with the exception of the super singular 1024-bit curve. Even for this curve the times for each algorithm are under the 700 msec mark. Although one would expect that as the policies and the attributes sets grow bigger these times will increase, the additional overhead will grow only linearly. Thus we believe that the two constructions constitute the most practical implementations of large universe ABE, secure in the standard model.

7.4 Group Operations (Asymptotic) for KP-ABE and CP-ABE

In Table 7.3, we demonstrate the asymptotic growth of the algorithms of the schemes implemented. For the KP-ABE setting the number of group operations during the key generation, encryption, and decryption calls depends linearly on the number of rows in the policy, on the size of the attribute set and the number of rows that are used during decryption, respectively. In the CP-ABE setting the key generation time grows linearly with the size of the attribute set and the encryption time with the number of rows in the policy. Some constant factors might not correspond exactly to the factors that can be derived from schemes in Sec. 6.2 and App. 6.1, because certain optimizations have been applied so that common parts are only computed once (see the source code in [106] for more details).

7.5 Benchmarks MA-CP-ABE

Regarding the comparisons of our MA-CP-ABE scheme to existing schemes, there are three other schemes that provided expressive policies in the multi-authority setting: the CP-ABE scheme of Lewko-Waters [73], the

Curve	Type	Scheme	Setup	KeyGen	Encrypt	Decrypt
“SS512”	KP-ABE	RW [6.1]	19.1	49.1	30.7	14.7
		LW [70]	447.2	642.3	483.4	44.7
		OT [87]	673.7	924.4	933.5	65.6
	CP-ABE	RW [6.2]	25.0	32.9	52.0	16.6
		OT [87]	678.0	922.9	938.5	66.0
“SS1024”	KP-ABE	RW [6.1]	71.5	626.3	396.8	325.3
		LW [70]	5553.3	9283.8	6978.3	1098.8
		OT [87]	7904.3	13389.3	13582.0	1735.7
	CP-ABE	RW [6.2]	110.8	431.0	669.3	374.4
		OT [87]	7898.9	13393.2	13598.7	1740.4
“MNT159”	KP-ABE	RW [6.1]	21.1	48.1	44.3	36.4
		LW [70]	692.2	1666.3	168.9	125.2
		OT [87]	930.7	2435.1	320.6	178.4
	CP-ABE	RW [6.2]	23.5	43.8	53.5	41.5
		OT [87]	929.9	2396.2	326.7	183.5
“MNT201”	KP-ABE	RW [6.1]	28.4	59.2	60.2	49.7
		LW [70]	929.8	2301.1	237.8	173.6
		OT [87]	1237.1	3338.3	453.3	251.5
	CP-ABE	RW [6.2]	31.3	58.7	71.9	57.4
		OT [87]	1235.1	3328.7	463.3	251.8
“MNT224”	KP-ABE	RW [6.1]	34.2	73.4	74.2	60.9
		LW [70]	1150.9	2896.0	302.1	215.6
		OT [87]	1514.9	4156.3	572.4	309.8
	CP-ABE	RW [6.2]	37.9	73.2	88.2	74.4
		OT [87]	1511.7	4140.0	584.5	310.7

Table 7.2: Typical running times in milliseconds of each scheme. KeyGen and Encrypt are called with attribute sets and policies of size 4, while Decrypt with common attribute sets of size 2. “MNT” are the Miyaji, Nakabayashi, Takano curves (asymmetric pairing groups), while “SS” are super singular curves (symmetric pairing groups). The number after the type of the curve denotes the size of the base field in bits.

Type	Scheme	Algorithm	$\mathbb{G}_1$	$\mathbb{G}_2$	$\mathbb{G}_T$	Pairings
KP-ABE	RW [6.1]	Setup	0	0	1	1
		KeyGen	$4k$	k	0	0
		Encrypt	$2m+1$	$m+1$	1	0
		Decrypt	0	0	n	$3n$
	LW [70]	Setup	60	80	2	1
		KeyGen	0	$60k$	0	0
		Encrypt	$40m+20$	0	2	0
		Decrypt	0	0	$10n$	$10n$
	OT [87]	Setup	100	99	1	1
		KeyGen	0	$84k+10$	0	0
		Encrypt	$84m+15$	0	1	0
		Decrypt	0	0	n	$14n+5$
CP-ABE	RW [6.2]	Setup	1	0	1	1
		KeyGen	$2m+2$	$m+1$	0	0
		Encrypt	$4k$	$k+1$	1	0
		Decrypt	0	0	n	$3n+1$
	OT [87]	Setup	100	99	1	1
		KeyGen	0	$84m+10$	0	0
		Encrypt	$84k+15$	0	1	0
		Decrypt	0	0	n	$14n+5$

Table 7.3: Asymptotic growth of the *exponentiations* in the three groups and the pairings. These are the dominant operations in each call. Notice the large constant factors in the schemes utilizing the dual vector spaces. These are due to the large dimensions of vectors. The k, m, n parameters denote the number of rows of the policy, the size of the attribute set, and the rows utilized during decryption, respectively.

prime order version of it [69], and the multi-authority signature scheme of Okamoto - Takashima [89]. However, we decided to defer implementation and benchmarking of them for several reasons. The first scheme utilizes composite order groups, which are several orders of magnitude slower than the prime order groups that provide the same security level. We expect our scheme to be significantly faster. More information on the comparison between prime and composite groups can be found in Sec. 3.3.4. In addition, Charm does not support composite order groups. The other two schemes utilize dual pairing vector spaces of high dimension and all their components (public parameters, secret keys) consist of a large number of group elements which degrades their efficiency. Secondly an one-use restriction per policy is imposed on each attribute on the first two systems. So even these schemes provide less flexibility than our construction. Finally, it is questionable the validity of the comparison between a prime order group and a composite order group, when the underlying elliptic curve is different and/or different optimizations have been applied to them.

Instead of this, we validate the claim that our system provides similar efficiency to existing single-authority ABE constructions, by providing implementation results of two single-authority ABE schemes. These are the Bethencourt-Sahai-Waters CP-ABE scheme [11] and the recent Waters CP-ABE [112]. Both of them were implemented by the Charm authors as typical examples. The former scheme is secure in the generic group model, while the implementation of the latter uses the random oracle version of it.

In this section we present the timing results of our multi-authority CP-ABE scheme. Since there are no other multi-authority large universe CP-ABE schemes on prime order groups, we compare it versus two know and established single-authority CP-ABE constructions and claim that our scheme achieves similar timings. Timing results in milliseconds are shown in Table 7.4. We see that our scheme achieves similar operation times to the two established single-authority schemes. In general, we attempted to keep execution times for encryption and decryption relatively low, while the times for setup and key generation can be significantly higher, since they are called only once.

Our CP-ABE [Sec. 6.3] (Multi-authority, random oracle model, statically secure)

Curve	GS	AS	KG(4)	KG(8)	KG(12)	EC(4)	EC(8)	EC(12)	DE(4)	DE(8)	DE(12)
SS512	8.4	4.1	91.5	182.9	274.6	75.0	150.4	226.4	34.5	59.2	82.3
SS1024	58.0	43.8	631.4	1263.5	1894.5	666.9	1331.2	1997.2	641.4	1275.4	1907.4
MNT159	14.4	3.7	295.9	502.7	799.7	155.9	299.4	450.1	99.3	159.8	237.5
MNT201	19.5	4.6	370.5	787.0	1205.8	191.6	401.2	592.1	133.8	237.9	321.5
MNT224	24.1	5.5	489.5	838.4	1335.2	244.1	473.0	695.9	157.0	273.2	390.3

BSW CP-ABE [11] (Single-authority, generic group model, adaptively secure)

Curve	GS	AS	KG(4)	KG(8)	KG(12)	EC(4)	EC(8)	EC(12)	DE(4)	DE(8)	DE(12)
SS512	20.1	N/A	52.9	100.1	146.9	51.0	98.5	147.6	22.5	40.3	55.3
SS1024	213.3	N/A	394.0	710.3	1026.5	360.1	681.6	997.1	482.2	909.0	1333.9
MNT159	31.2	N/A	152.8	265.2	399.4	107.5	268.2	376.9	56.4	104.7	149.1
MNT201	42.2	N/A	221.5	335.1	557.8	169.8	331.7	564.5	76.3	142.5	205.5
MNT224	52.3	N/A	192.8	447.5	566.1	209.1	329.3	595.2	94.7	175.0	253.6

Waters CP-ABE [112] (Single-authority, random oracle model, adaptively secure)

Curve	GS	AS	KG(4)	KG(8)	KG(12)	EC(4)	EC(8)	EC(12)	DE(4)	DE(8)	DE(12)
SS512	20.4	N/A	39.6	73.9	108.0	64.2	124.8	186.4	32.5	60.1	85.3
SS1024	216.3	N/A	237.7	397.5	558.6	516.4	992.9	1464.4	627.0	1200.5	1770.1
MNT159	32.7	N/A	18.3	21.8	25.7	43.4	84.5	125.2	56.3	104.5	148.8
MNT201	44.6	N/A	25.4	31.7	37.2	58.8	118.3	170.7	77.0	143.4	206.9
MNT224	55.1	N/A	31.4	38.4	45.2	71.3	137.3	205.7	95.2	177.9	258.4

Table 7.4: Average running times in milliseconds of our scheme and two single authority schemes. The algorithms are denoted as GS: Global setup, AS: Authority setup, KG: Key generation for a user, EC: Encrypt, DE: Decrypt. The numbers in parentheses refer to the number of attributes in key generation, the number of rows of the policy in encryption, and the number of rows utilized during decryption. We can see the linear dependence between these numbers and the corresponding times.

CHAPTER 8

Other Work and Future Directions

8.1 Other Work

Leakage-Resilient IBE In a joint work with Chow, Dodis, and Waters [36] we provide new constructions of leakage-resilient IBE in the standard model. We apply a hash proof technique in the existing IBE schemes of Boneh-Boyen, Waters, and Lewko-Waters. As a result, we achieve leakage-resilience under the respective static assumptions of the original systems in the standard model. The first two systems are secure under the simple Decisional Bilinear Diffie-Hellman assumption (DBDH). The first system is selectively secure and serves as a stepping stone to construct the fully secure system. This second system is the first leakage-resilient fully secure system under DBDH in the Standard model. Finally the third system achieves full security with shorter public parameters but is based on three non-standard static assumptions (similar to the composite order group assumptions of Sec. 3.2.2). The efficiency of our transformed systems is almost the same as the original ones.

Our main technique is different than the technique presented in Chap. 5. The original systems used random secret keys with only one degree of freedom, which was explorable to the secret key holder. This means that the owner of the secret key could re-randomize his key arbitrarily without knowing the secret parameters of the IBE system (the master secret key). In this sense the information each key holds is deterministic. The new technique we applied was to add another randomness to the secret keys, called "tag", coupled with some master secret key terms. As a result, the secret-key holder can not anymore re-randomize his key (in this degree of freedom). The added randomness allows the simulators of our security proofs to provide the attacker leaked information from a properly distributed secret key with a tag of our choice.

Obviously the ability of the simulator to create these secret keys allows him to decrypt the challenge ciphertext. One would ask then why is the attacker's response useful to the simulator. The answer is that we use a standard primitive in leakage-resilient constructions [5, 80] to "mask" the relationship between the leakage of the secret key and the ciphertext. This primitive, called extractor [82], makes it hard for any attacker to break the system given only a bounded amount of leakage from the secret key when the simulator "injects" the tag of the challenge identity to specific parts of the ciphertext.

Property-Preserving Encryption Processing on encrypted data is a subject of rich investigation. Several new and exotic encryption schemes, supporting a diverse set of features, have been developed for this purpose. In a joint work with Omkant Pandey [92] we consider encryption schemes that are suitable for applications such as data clustering on encrypted data. In such applications, the processing algorithm needs to learn certain properties about the encrypted data to make decisions. Often these decisions depend upon multiple data items, which might have been encrypted individually and independently. Current encryption schemes do not capture this setting where computation must be done on multiple ciphertexts to make a decision.

In this work, we seek encryption schemes which allow public computation of a pre-specified property P about the encrypted messages. That is, such schemes have an associated property P of fixed arity k, and a publicly computable algorithm Test, such that $\mathsf{Test}(ct_1, ct_2, \ldots, ct_k) = P(m_1, m_2, \ldots, m_k)$, where ct_i is an encryption of m_i for $i = 1, 2, \ldots, k$. Further, this requirement holds even if the ciphertexts $ct_1, ct_2, \ldots, ct_k$ were generated individually and independently. We call such schemes property preserving encryption schemes. Property preserving encryption (PPEnc) makes most sense in the symmetric setting due to the requirement that Test is publicly computable.

In this work, we present a thorough investigation of property preserving symmetric encryption. We start by formalizing several meaningful notions of security for PPEnc. Somewhat surprisingly, we show that there exists a hierarchy of security notions for PPEnc, indexed by integers $\eta \in \mathbb{N}$, which does not collapse. We also present a symmetric PPEnc scheme for encrypting vectors in $\mathbb{Z}_N$ of polynomial length. This construction supports the orthogonality property: for every two vectors $(\vec{x}, \vec{y})$ it is possible to publicly learn whether $\langle \vec{x}, \vec{y} \rangle = 0 \pmod{p}$. Our scheme is based on bilinear groups of composite order.

8.2 Future Directions

Leakage - Resilient Functional Encryption The CP-ABE construction that we presented in Chap. 5 supports any monotone Boolean formula as the policy of the ciphertext. Although these policies are sufficiently expressive for many practical scenarios, a very interesting direction is to investigate further the limits on the expressiveness of ABE systems, which in this case are referred to as Functional Encryption Schemes [24]. While the ultimate goal would be to construct a scheme where the policies are arbitrary Turing machines, the state-of-the-art systems provide functionality for regular languages [109] and circuits [50, 51, 55]. So far no leakage-resilient constructions have been presented for these schemes and it would be intriguing to investigate whether existing techniques for leakage resilience apply on these schemes.

Practical Implementations Another promising direction is the designing and testing of more practical ABE systems similar to the ones of Chap. 6. Adding more advanced features combined with fast operations might open the way to widespread deployment of ABE systems and / or functional encryption in general. As the work of Lewko - Waters [75] suggests, novel techniques and / or groups are needed to achieve stronger security guarantees such as full security,

Appendices

APPENDIX A

Generic Security of the Assumptions

In this section we consider the security of our assumptions in the generic group model introduced by Shoup [104]. In this model the attacker does not receive the actual representations of group elements in $\mathbb{G}$ or $\mathbb{G}_T$, but handles picked from a sufficiently large handle space. Whenever a new group element has to be given to the attacker, he receives a uniformly random handle from the handle space; not picked before. From now on this handle is "fixed" to this specific group element. The attacker is allowed to query for operations on the handles he has already received. Then the challenger executes the operation on the underlying group elements and returns either a freshly picked handle, if the result is new, or an existing handle. The only operations available on group elements to the attacker are multiplications in $\mathbb{G}$ or $\mathbb{G}_T$, pairings in $\mathbb{G}$ and the equality checking of two group elements in $\mathbb{G}$ or $\mathbb{G}_T$ by checking the equality of handles. The main goal of the generic group model proofs is to provide an indication of the absence of "security holes" which are independent from the specific group representations.

A.1 Two General Theorems

The following theorem will provide an easier way to argue about the security of our assumptions in the generic group model.

Definition A.1 ($\mathbb{G}_T$-monomial assumption)**.** A $\mathbb{G}_T$-monomial assumption is parameterized by a security parameter $\lambda \in \mathbb{N}$. It refers to a prime order bilinear group $D = (p, \mathbb{G}, \mathbb{G}_T, e)$ with $p = \Theta(2^\lambda)$, a matrix $A \in \mathbb{Z}^{L \times K}$ and two target vectors $\tilde{A}^0, \tilde{A}^1 \in \mathbb{Z}^{1 \times K}$.

We require that $\tilde{A}^0 \neq \tilde{A}^1$ and that the natural numbers K, L and the the absolute values of the integers in A, $\tilde{A}^0$ and $\tilde{A}^1$ are all polynomially bounded in λ.

The assumption is defined via a game between a challenger and an adversary. Initially, the challenger picks K *independent and uniformly random* variables $X_1, X_2, \ldots, X_K \overset{R}{\leftarrow} \mathbb{Z}_p$. Then it constructs the following monomials in these variables:

$$Y_i = \prod_{j \in [K]} X_j^{A_{i,j}} \text{ for all } i \in [L] \quad , \quad Z_0 = \prod_{j \in [K]} X_j^{\tilde{A}^0_j} \quad \text{and} \quad Z_1 = \prod_{j \in [K]} X_j^{\tilde{A}^1_j}$$

Finally, the challenger picks $g \overset{R}{\leftarrow} \mathbb{G}$ and $b \overset{R}{\leftarrow} \{0, 1\}$. He sends to the adversary the description of the group $D = (p, \mathbb{G}, \mathbb{G}_T, e)$, the matrix A, the vectors $\tilde{A}^0, \tilde{A}^1$, the terms $\left\{g^{Y_i}\right\}_{i \in [L]}$, and the challenge term $e(g, g)^{Z_b}$. The assumption claims that no PPT adversary has a non negligible advantage in guessing the bit b.

We will prove the following theorem that refers to the generic security of a $\mathbb{G}_T$-monomial assumption:

Theorem A.2. *The above assumption is secure in the generic group model if and only if for all $i, j \in [L]$ it is true that $\tilde{A}^0 \neq A^i + A^j$ and $\tilde{A}^1 \neq A^i + A^j$, where A^i is the i-th row of A.*

For the proof of the theorem A.2 we will use the following lemma:

Lemma A.3. *Consider any linear combination of the form*

$$T(X_1, X_2, \ldots, X_K) = \tilde{c}_0 Z_0 + \tilde{c}_1 Z_1 + \sum_{(i,j)\in[L,L]} c_{i,j} Y_i \cdot Y_j$$

with $\tilde{c}_0, \tilde{c}_1, c_{i,j}$ constants in $\mathbb{Z}_p$ and T is not identically zero as a rational function in variables X_1, X_2, …, X_K.

Then the probability that $T(X_1, X_2, \ldots, X_K) = 0 \pmod p$ is negligible in λ.

Proof. of lemma A.3 The proof is an immediate consequence of the Schwartz-Zippel lemma and the fact that the total degree of each monomial is polynomially bounded in λ. More specifically, consider the polynomial

$$T'(X_1, X_2, \ldots, X_K) = T(X_1, X_2, \ldots, X_K) \cdot C(X_1, X_2, \ldots, X_K)$$

where $C(X_1, X_2, \ldots, X_K) = \prod X_i^{d_i}$ with d_i being the absolute value of the minimum negative exponent of X_i in the monomials $Y_i Y_j$ for any i, j and the

Z_0, Z_1, or 0 if X_i has only positive exponents. As a result T' is a polynomial in variables X_1, X_2, …, X_K.

Since the absolute values of the elements of A, $\tilde{A}^0$, and $\tilde{A}^1$ are all polynomially bounded in λ, the total degree of the polynomial T' is also polynomially-bounded in λ. Since T, and therefore T', is not identically zero we can apply the Schwartz-Zippel lemma: the probability that T' becomes zero is at most $\mathcal{O}(\lambda^c)/p = \mathcal{O}(\lambda^c)/\Theta(2^\lambda) = \mathsf{negl}(\lambda)$. This is equal to the probability that T is zero or undefined (when some X_i with $d_i > 0$ is instantiated to zero). Therefore, the probability that T is zero is at most negligible in λ. $\square$

Proof. of theorem A.2 We will prove the forward direction first. Namely, suppose that there exist i, j and $b' \in \{0, 1\}$ such that $\tilde{A}^{b'} = A^i + A^j$. W.l.o.g. we assume $b' = 0$. Since according to the definition of the assumption $\tilde{A}^0 \neq \tilde{A}^1$, we conclude that $\tilde{A}^1 \neq A^i + A^j$.

The adversary can in polynomial time find these i, j, because L is polynomial, and request the handle of the term $e(g^{Y_i}, g^{Y_j})$. If this handle is equal to the handle of $e(g, g)^{Z_b}$ of the challenge term, it outputs 0. Otherwise it outputs 1.

Therefore, if b is indeed 0 the adversary is successful with certainty. If it is the case that $b = 1$, the adversary makes a wrong guess only when the handle of $e(g, g)^{Z_1}$ happens to be the same as the handle of $e(g^{Y_i}, g^{Y_j})$. According to the generic group model this is equivalent to $Z_1 = Y_i \cdot Y_j$ (after the instantiations). Since $\tilde{A}^1 \neq A^i + A^j$ we get that the expression $Z_1 - Y_i \cdot Y_j$ is not identically zero. Therefore, according to lemma A.3 we conclude that the error probability of the adversary when $b = 1$ is $\mathsf{negl}(\lambda)$. As a result the advantage of the adversary is non negligible and the assumption not secure in the generic group model.

For the backward direction we assume that there exists a PPT adversary that breaks the assumption in the generic group model game. First, we define a new security game where the random variables $X_1, X_2, \ldots, X_K$ are never instantiated and the handles returned to the adversary are the same only when the rational functions of the X_i's in the exponents of group elements are formally equal. This game differs from the real generic group model game only when two different linear combinations of monomials are instantiated to the same value. Since the number of queries by the adversary is polynomial and because of lemma A.3, the probability of this event is negligible. Therefore, the adversary has a non negligible advantage in the modified game.

Since the only decision query he can ask is to compare the handles of two terms, he can construct two terms $T_1, T_2 \in \mathbb{G}_T$ such that $T_1 = T_2$ (in terms of formal equality of the exponents) for one value of b but not for the other[1]. According to the allowable operations and the terms given to adversary, T_1 and T_2 should be of the form $e(g, g)^S$ where S is a linear combination of the set of monomials $\{Z_b\} \cup \{Y_i \cdot Y_j\}_{i,j \in [L]}$.

[1]A decision in $\mathbb{G}$ can be expressed in terms of $\mathbb{G}_T$ by pairing with the same element of $\mathbb{G}$

W.l.o.g. suppose that T_1 is equal to T_2 when $b = 0$ and different otherwise. Then if $T_1 = e(g,g)^{S_1}$ and $T_2 = e(g,g)^{S_2}$, we get that $T_1 = T_2 \implies S_1 = S_2 \implies Z_0 = S^*$, where S^* is a linear combination of only the monomials $\{Y_i \cdot Y_j\}_{i,j \in [L]}$. The coefficient of Z_0 has to be non-zero because otherwise the value of b would be information-theoretically hidden and the advantage of the adversary would be zero in this game. Since the $Z_0 = S^*$ is a formal equation and Z_0 is a monomial the only way this is possible is to have $Z_0 = Y_i \cdot Y_j$ for some i, j. Therefore, $\tilde{A}^0 = A^i + A^j$. □

The following corollary refers to a $\mathbb{G}_T$-monomial assumption where the second challenge term is uniformly random from $\mathbb{G}_T$. The proof from theorem A.2 is trivial and is omitted.

Corollary A.1.1. If $\tilde{A}^1 = (0, 0, \ldots, 0, 1) \in \mathbb{Z}^{1 \times K}$ and $\langle \tilde{A}^1, A^i \rangle = 0$ for all $i \in [L]$, the corresponding $\mathbb{G}_T$-monomial assumption is secure in the generic group model if and only if for all $i, j \in [L]$ it is true that $\tilde{A}^0 \neq A^i + A^j$.

A.2 Proofs of Security in the Generic Group Model

Using the above corollary we show that our assumptions are secure in the generic group model in lemmata A.5 and A.4.

Lemma A.4. *The "q-DPBDH1" assumption is secure in the generic group model.*

Proof. q-DPBDH1 is a $\mathbb{G}_T$-monomial assumption with random variables x, y, z, b_1, b_2, …, b_q instead of X_1, X_2, …, X_{K-1}. The matrix A and the target vector $\tilde{A}^0$ for this assumption are shown in table A.1.

Type	Given Terms	Conditions	x	y	z	b_1	b_2	$\ldots$	b_q
1	g		0	0	0	0	0	$\ldots$	0
2	g^x		1	0	0	0	0	$\ldots$	0
3	g^y		0	1	0	0	0	$\ldots$	0
4	g^z		0	0	1	0	0	$\ldots$	0
5	$g^{(xz)^2}$		2	0	2	0	0	$\ldots$	0
6	g^{b_i}	$\forall i \in [q]$	0	0	0	$[i:1]$			
7	g^{xzb_i}	$\forall i \in [q]$	1	0	1	$[i:1]$			
8	g^{xz/b_i}	$\forall i \in [q]$	1	0	1	$[i:(-1)]$			
9	$g^{x^2zb_i}$	$\forall i \in [q]$	2	0	1	$[i:1]$			
10	g^{y/b_i^2}	$\forall i \in [q]$	0	1	0	$[i:(-2)]$			
11	g^{y^2/b_i^2}	$\forall i \in [q]$	0	2	0	$[i:(-2)]$			
12	g^{xzb_i/b_j}	$\forall (i,j) \in [q,q]$ with $i \neq j$	1	0	1	$[i:1, j:(-1)]$			
13	g^{yb_i/b_j^2}	$\forall (i,j) \in [q,q]$ with $i \neq j$	0	1	0	$[i:1, j:(-2)]$			
14	g^{xyzb_i/b_j}	$\forall (i,j) \in [q,q]$ with $i \neq j$	1	1	1	$[i:1, j:(-1)]$			
15	$g^{(xz)^2b_i/b_j}$	$\forall (i,j) \in [q,q]$ with $i \neq j$	2	0	2	$[i:1, j:(-1)]$			

$\tilde{A}^0$	$e(g,g)^{xyz}$	1	1	1	0	0	$\ldots$	0

Table A.1: Compact form of matrix A and target vector $\tilde{A}^0$ for the q-DPBDH1 assumption.

In order to prove the lemma we have to show that by adding any two rows of matrix A we can not get the row vector $\tilde{A}^0 = (1,1,1,0,0,\ldots,0)$. We will mainly focus on the first three columns of matrix A, i.e. the x, y, z columns. First, we observe that the rows of types 5, 9, 11, and 15, can not be used since they have at least one 2 in the first three columns and all other rows are positive in these columns. The rows 2 and 4 have "100" and "001" in the first three columns, respectively. Since there are no rows with "011" or "110", they can not be used to give the required "111". Row 1 or a row of type 6 can only be combined to a row of type 14, and vice versa, because the former have "000" and the later "111". But since a row of type 14 has $[i : 1, j : (-1)]$ in the b_i columns, it can not be used to give all zeros in them. Finally, rows of type 7, 8, or 12, that have "101" in the first three columns, might be possibly combined to row 3 or rows of type 10, or 13, and vice versa. However, this still won't give the target vector, because none of the partial vectors $[i : 1]$, $[i : (-1)]$, and $[i : 1, j : (-1)]$ can be added to any of the vectors $(0, 0, \ldots, 0)$, $[i : (-2)]$, and $[i : 1, j : (-2)]$, and give the all zero vector in the b_i columns.

Therefore, according to corollary A.1.1 the q-DPBDH1 assumption is secure in the generic group model. $\square$

Lemma A.5. *The "q-DPBDH2" assumption is secure in the generic group model.*

Proof. First, notice that this is indeed a $\mathbb{G}_T$-monomial assumption with random variables a, s, b_1, b_2, $\ldots$, b_q instead of X_1, X_2, $\ldots$, X_{K-1}. X_K is the uniformly random exponent of the second challenge term; not present in any of the remaining terms. Thus corollary A.1.1 applies.

In table A.2 we denote by $[i : x]$ and $[i : x, i' : y]$ the row vectors in $\mathbb{Z}^{1\times q}$ with all components equal to 0, except the i-th component for the first vector and the i, i'-th components for the second. The non zero elements are x for the first vector and x, y for the i, i'-th positions, respectively, of the second vector. The table shows a compact form of the matrix A where rows of similar type are shown in one line.

Type	Given Terms	Conditions	a	s	b_1	b_2	$\dots$	b_q
1	g		0	0	0	0	$\dots$	0
2	g^s		0	1	0	0	$\dots$	0
3	g^{a^i}	$\forall i \in [q]$	i	0	0	0	$\dots$	0
4	g^{b_j}	$\forall j \in [q]$	0	0	$[j : 1]$			
5	g^{sb_j}	$\forall j \in [q]$	0	1	$[j : 1]$			
6	$g^{a^i b_j}$	$\forall (i, j) \in [q, q]$	i	0	$[j : 1]$			
7	g^{a^i / b_j^2}	$\forall (i, j) \in [q, q]$	i	0	$[j : (-2)]$			
8	g^{a^i / b_j}	$\forall (i, j) \in [2q, q]$ with $i \neq q + 1$	i	0	$[j : (-1)]$			
9	$g^{a^i b_j / b_{j'}^2}$	$\forall (i, j, j') \in [2q, q, q]$ with $j \neq j'$	i	0	$[j : 1, j' : (-2)]$			
10	$g^{s a^i b_j / b_{j'}}$	$\forall (i, j, j') \in [q, q, q]$ with $j \neq j'$	i	1	$[j : 1, j' : (-1)]$			
11	$g^{s a^i b_j / b_{j'}^2}$	$\forall (i, j, j') \in [q, q, q]$ with $j \neq j'$	i	1	$[j : 1, j' : (-2)]$			
$\tilde{A}^0$	$e(g, g)^{s a^{q+1}}$		$q + 1$	1	0	0	$\dots$	0

Table A.2: Compact form of matrix A and target vector $\tilde{A}^0$ for the q-DPBDH2 assumption.

In order to prove the lemma we have to show that by adding any two rows of matrix A we can not get the row vector $\tilde{A}^0 = (q+1, 1, 0, 0, \ldots, 0)$. By inspecting table A.2 we can easily see that we have to check only the rows of types 2, 5, 10, and 11, which have 1 in the s column.

The only rows that can be added to row 2 and give all zero's in the b_i columns are row 1 or rows of type 3. But in both of them we can not get the $q+1$ component in the a column. Rows of type 5 can be added only to rows of type 8 and give only zeros in the b_i columns. But the term with $i = q+1$ is excluded from rows of type 8; therefore the target vector can not be obtained. Finally, rows of type 10 or 11 can not be added to one of the rows 1, 3, 4, 6, 7, 8, and 9 without having at least one non zero element in the b_i columns. That is because none of these rows have vectors of the form $[j : (-1), j' : 1]$ or $[j : (-1), j' : 2]$, which are needed to cancel the b_i components.

Therefore, according to corollary A.1.1 the q-$\mathsf{DPBDH2}$ assumption is secure in the generic group model. $\square$

APPENDIX B

Proofs of Various Lemmas Used

B.1 A Useful Lemma for Leakage Analysis

Our analysis of the leakage resilience of our system will rely on the following lemma from [28], which is proven using the techniques from [16]. Below, we let $dist(X_1, X_2)$ denote the statistical distance of two random variables X_1 and X_2.

Lemma B.1. *Let $m, \ell, d \in \mathbb{N}$, $m \geq \ell \geq 2d$ and let p be a prime. Let $X \overset{R}{\leftarrow} \mathbb{Z}_p^{m \times \ell}$, let $Y \overset{R}{\leftarrow} \mathbb{Z}_p^{m \times d}$, and let $T \overset{R}{\leftarrow} Rk_d\left(\mathbb{Z}_p^{\ell \times d}\right)$, where $Rk_d\left(\mathbb{Z}_p^{\ell \times d}\right)$ denotes the set of $\ell \times d$ matrices of rank d with entries in $\mathbb{Z}_p$. Let $f : \mathbb{Z}_p^{m \times d} \to W$ be some function. Then:*

$$dist\left((X, f(X \cdot T)), (X, f(Y))\right) \leq \epsilon,$$

as long as

$$|W| \leq 4 \cdot \left(1 - \frac{1}{p}\right) \cdot p^{\ell - (2d-1)} \cdot \epsilon^2.$$

More precisely, we will use the following corollary:

Corollary B.1.1. Let $m \in \mathbb{N}$, $m \geq 3$, and let p be a prime. Let $\vec{\delta} \overset{R}{\leftarrow} \mathbb{Z}_p^m$, $\vec{\tau} \overset{R}{\leftarrow} \mathbb{Z}_p^m$, and let $\vec{\tau'}$ be chosen uniformly randomly from the set of vectors in $\mathbb{Z}_p^m$ which are orthogonal to $\vec{\delta}$ under the dot product modulo p. Let $f : \mathbb{Z}_p^m \to W$ be some function. Then:

$$dist\left((\vec{\delta}, f(\vec{\tau'})), (\vec{\delta}, f(\vec{\tau}))\right) \leq \epsilon,$$

as long as

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

Proof. We apply Lemma B.1 with $d = 1$ and $\ell = m - 1$. Y then corresponds to $\vec{\tau}$, while X corresponds to a basis of the orthogonal space of $\vec{\delta}$. We note that $\vec{\tau'}$ is then distributed as $X \cdot T$, where $T \overset{R}{\leftarrow} Rk_1\left(\mathbb{Z}_p^{m-1\times 1}\right)$. We note that X is determined by $\vec{\delta}$, and is distributed as $X \overset{R}{\leftarrow} \mathbb{Z}_p^{m\times m-1}$, since $\vec{\delta}$ is chosen uniformly randomly from $\mathbb{Z}_p^m$. It follows that:

$$dist\left((\vec{\delta}, f(\vec{\tau'})), (\vec{\delta}, f(\vec{\tau}))\right) = dist\left((X, f(X \cdot T)), (X, f(Y))\right) \leq \epsilon.$$

$\square$

This corollary allows us to set $\ell_{\text{MSK}} = \ell_{\text{SK}} = (n - 1 - 2c)\log(p_2)$ for our construction (we'll have $n + 1 = m$), where c is any fixed positive constant (so that $\epsilon := p_2^{-c}$ is negligible).

B.2 Proof of claim 6.4

In order to prove claim 6.4 we will use the following theorem:

Theorem B.2. *Let $A \in \mathbb{Z}_p^{\ell \times n}$ be the secret sharing matrix of a linear secret sharing scheme for an access policy $\mathbb{A}$ and $L \in \mathbb{Z}_p^{n \times n}$ be a matrix such that:*

- *The first row of L is $(1, 0, \ldots, 0) \in \mathbb{Z}_p^n$.*
- *The lower right matrix $L' \in \mathbb{Z}_p^{(n-1)\times(n-1)}$ of L has rank $n-1$.*

Then the distribution of the shares $\{\lambda_x\}_{x\in[\ell]}$ sharing the secret $z \in \mathbb{Z}_p$ generated with the matrix A is the same as the distribution of the shares $\{\lambda'_x\}_{x\in[\ell]}$ sharing the secret $z \in \mathbb{Z}_p$ generated with the matrix $A \cdot L$.

Proof. Consider the distribution of the shares $\{\lambda'_x\}_{x\in[\ell]}$. According to the construction of LSS schemes, it is true that $\lambda'_x = \left\langle \vec{AL}_x, \vec{v} \right\rangle$ where $\vec{AL}_x$ is the x-th row of the matrix AL and $\vec{v}$ is a random vector with its first element equal to z.

This implies that $\lambda'_x = \left\langle \vec{A}_x, \vec{Lv} \right\rangle$, where $\vec{Lv} \in \mathbb{Z}_p^n$ is the vector acquired by multiplying L with $\vec{v}$. Since L has the first row $(1, 0, \ldots, 0)$ we get that the first element of $\vec{Lv}$ is z. Moreover the remaining $n-1$ elements are uniformly random from $\mathbb{Z}_p$ because each one, say the i-th one, is equal to $z \cdot L_{i,1} + \left\langle \vec{L'_i}, \vec{v'} \right\rangle$ where $\vec{L'_i}$ is the i-th row of L' and $\vec{v'} \in \mathbb{Z}_p^{n-1}$ are the last $n-1$ elements of $\vec{v}$. Since these are uniformly random and L' is full rank, we get that $\left\langle \vec{L'_i}, \vec{v'} \right\rangle$ is uniformly random.

Therefore, $\vec{Lv}$ is distributed exactly the same as a secret sharing vector of z. Thus the shares $\{\lambda'_x\}$ have the same distribution as the shares $\{\lambda_x\}$. □

Proof of claim 6.4 To convert the matrix A to the target matrix A' we will apply theorem B.2. Let $\vec{W}_1, \vec{W}_2, \ldots, \vec{W}_c$ be the first c independent rows in $\mathcal{C}$. These rows form a basis of size c of the relevant subspace and they can be computed from $\mathcal{C}$ in polynomial time using linear algebra operations.

Next we are going to extend this basis to size n such that the final basis spans the entire space. The first step is to add the row $\vec{U} = (1, 0, \ldots, 0) \in \mathbb{Z}_p^n$ to the set. Since the set of rows in $\mathcal{C}$ is unauthorized, $\vec{U}$ is not in in the subspace spanned by them and therefore this is a valid choice.

We continue by picking $n-c-1$ rows, $\vec{V}_1, \vec{V}_2, \ldots, \vec{V}_{n-c-1}$, such that the set

$$\left\{\vec{U}, \vec{V}_1, \vec{V}_2, \ldots, \vec{V}_{n-c-1}, \vec{W}_1, \vec{W}_2, \ldots, \vec{W}_W\right\}$$

is a basis of $\mathbb{Z}_p^n$. Using linear algebra operations this can be done in polynomial time as well.

Finally construct the matrix

$$L = (L')^{-1} = \begin{bmatrix} \vec{U} \\ \vec{V}_1 \\ \cdots \\ \vec{V}_{n-c-1} \\ \vec{W}_1 \\ \cdots \\ \vec{W}_c \end{bmatrix}^{-1} \in \mathbb{Z}_p^{n \times n}$$

Using theorem B.2 we will argue that the matrix $A' = A \cdot L$ will give us same distribution for the $\{\lambda_x\}$ shares. We should argue first that the matrix L satisfies the requirements of the theorem. This can be done by trying to compute the inverse matrix, but one straightforward way is to use the blockwise inversion formula shown in figure B.1.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} + A^{-1}B(D - CA^{-1}B)^{-1}CA^{-1} & -A^{-1}B(D - CA^{-1}B)^{-1} \\ -(D - CA^{-1}B)^{-1}CA^{-1} & (D - CA^{-1}B)^{-1} \end{bmatrix}$$

Figure B.1: Blockwise inversion formula

In our case we have $A = [1]$, $B = (0, 0, \ldots, 0) \in \mathbb{Z}_p^{1\times(n-1)}$, $C = (0, 0, \ldots, 0)^\top \in \mathbb{Z}_p^{(n-1)\times 1}$, and D is the lower right submatrix of L' of size $(n-1) \times (n-1)$. Therefore we have that

$$L = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & & & \\ \vdots & & D^{-1} & \\ 0 & & & \end{bmatrix}$$

Since D is of full rank, we can see that L satisfies the requirements of theorem B.2. The only thing left to prove is that A' has the required form. That is, that for all rows $\vec{A'}_x$ with $x \in \mathcal{C}$, we have that the first $n - c$ elements are equal to 0. We know that for fixed $x \in \mathcal{C}$ the row $\vec{A}_x$ is a linear combination of the basis rows $\{\vec{W}_1, \vec{W}_2, \ldots, \vec{W}_c\}$. Therefore $\vec{A}_x = \sum_{i\in[c]} \gamma_i \vec{W}_i$ with γ_i constants in $\mathbb{Z}_p$.

Finally, notice that for all k such that $n - c + 1 \leq k \leq n$ we have that

$$(\overbrace{0, \ldots, 0}^{n-c \text{ terms}}, \overbrace{0, \ldots, 0, 1, 0, \ldots, 0}^{\substack{c \text{ terms} \\ 1 \text{ on the } k\text{-th position}}}) \cdot L' = \vec{W}_k$$

$$\implies \vec{W}_k \cdot (L')^{-1} = (\overbrace{0, \ldots, 0}^{n-c \text{ terms}}, \overbrace{0, \ldots, 0, 1, 0, \ldots, 0}^{\substack{c \text{ terms} \\ 1 \text{ on the } k\text{-th position}}})$$

Therefore the row $\vec{A'}_x = \vec{A}_x \cdot L = \sum_{i \in [c]} \gamma_i \vec{Z}_i$, where

$$\vec{Z}_i = (\overbrace{0, \ldots, 0}^{n-c \text{ terms}}, \overbrace{0, \ldots, 0, 1, 0, \ldots, 0}^{\substack{c \text{ terms} \\ 1 \text{ on the } i\text{-th position}}})$$

.

As a result the first $n - c$ elements of $\vec{A'}_x$ are all equal to 0.

$\square$

APPENDIX C

Source Code

C.1 KP-ABE scheme of Sec. 6.1

```
'''
Rouselakis - Waters Unbounded Key-Policy Attribute-
   Based Encryption

| From:
| Published in:
| Available from:
| Notes:

* type:           attribute-based encryption (public
   key)
* setting:        bilinear pairing group of prime
   order
* assumption:     complex q-type assumption

:Authors:   Yannis Rouselakis
:Date:         02/12
'''

from toolbox.pairinggroup import *
from charm.cryptobase import *
from toolbox.secretutil import SecretUtil
from toolbox.ABEnc import *
from BenchmarkFunctions import *

debug = False
class KPABE_RW12(ABEnc):
  def __init__(self, groupObj, verbose = False):
    ABEnc.__init__(self)
    global util, group
    group = groupObj
    util = SecretUtil(group, verbose)

  # Defining a function to pick explicit exponents in
      the group
  def exp(self,value):
    return group.init(ZR, value)

  def setup(self):
    # Due to assymmetry in the groups we prefer most
        of the terms to be in G1
    g = group.random(G2)
    g2, u, h, w = group.random(G1), group.random(G1),
        group.random(G1), group.random(G1)
    alpha = group.random( )
    egg = pair(g2,g)**alpha
    pp = {'g':g, 'g2':g2, 'u':u, 'h':h, 'w':w, 'egg':
        egg}
    mk = {'alpha':alpha }
    return (pp, mk)

  def keygen(self, pp, mk, policy_str):
    # the secret alpha will be shared according to
        the policy
    policy = util.createPolicy(policy_str)
    a_list = util.getAttributeList(policy)
    # print("\n\n THE A-LIST IS", a_list,"\n\n")
    shares = util.calculateSharesDict(mk['alpha'],
       policy) #These are correctly set to be
       exponents in Z_p; Here alpha is shared

    K0, K1, K2 = {}, {}, {}
    for i in a_list:
      inti = int(util.strip_index(i)) #NOTICE THE
         CONVERSION FROM STRING TO INT
      ri = group.random(ZR)
      K0[i] = pp['g2']**shares[i] * pp['w']**ri
      K1[i] = (pp['u']**self.exp(inti) * pp['h'])**ri
      K2[i] = pp['g']**ri

    return { 'Policy':policy_str, 'K0':K0, 'K1':K1, '
       K2':K2 }

  def encrypt(self, pp, message, S):
    # S is a list of attributes written as STRINGS i.
       e. {'1', '2', '3',...}
    s = group.random()

    C = message * (pp['egg']**s)
    C0 = pp['g']**s
    wS = pp['w']**s

    C1, C2 = {}, {}
    for i in S:
      ti = group.random()
      C1[i] = pp['g']**ti
      C2[i] = (pp['u']**self.exp(int(i)) * pp['h'])**
         ti * wS  #NOTICE THE CONVERSION FROM STRING
         TO INT
    S = [i for i in S] #Have to be an array for util.
       prune
    return { 'S':S, 'C':C, 'C0':C0, 'C1':C1, 'C2':C2
       }

  def decrypt(self, pp, sk, ct):
    policy = util.createPolicy(sk['Policy'])
    z = util.getCoefficients(policy)
    # print("\n\n THE COEFF-LIST IS", z,"\n\n")

    pruned_list = util.prune(policy, ct['S'])
    # print("\n\n THE PRUNED-LIST IS", pruned_list,"\
       n\n")

    if (pruned_list == False):
      return group.init(GT,1)


    B = group.init(GT,1) # the identity element of GT
    for i in range(0,len(pruned_list)):
      x = pruned_list[i].getAttribute( ) #without the
          underscore
      y = pruned_list[i].getAttributeAndIndex( ) #
         with the underscore
      B *= ( pair(sk['K0'][y], ct['C0']) * pair(sk['
         K1'][y], ct['C1'][x]) / pair(ct['C2'][x], sk
         ['K2'][y] ) )**z[y]

    return ct['C'] / B

  def randomMessage(self):
    return group.random(GT)


def main():
  curve = 'MNT224'

  groupObj = PairingGroup(curve)
  scheme = KPABE_RW12(groupObj)
  #print("Setup(",curve,")")

  ID = InitBenchmark()
```

```
    startAll(ID)
    (pp, mk) = scheme.setup()
    EndBenchmark(ID)

    #print("The Public Parameters are",pp)
    #print("And the Master Key is",mk)
    #print("Done!\n")
    box1 = getResAndClear(ID, "Setup("+curve+")", "Done
       !")

    #-----------------------------------------------

    policy = '(123␣or␣444)␣and␣(231␣or␣999)'
    #print("Keygen(", policy,")")

    ID = InitBenchmark()
    startAll(ID)
    sk = scheme.keygen(pp,mk,policy)
    EndBenchmark(ID)

    #print("The secret key is",sk)
    #print("Done!\n")
    box2 = getResAndClear(ID, "Keygen(" + policy + ")",
        "Done!")

    #-----------------------------------------------

    m = group.random(GT)
    #print("Encrypting the message",m)
    S = {'123', '842',  '231', '384'}
    #print("Encrypt(", str(S),")")

    ID = InitBenchmark()
    startAll(ID)
    ct = scheme.encrypt(pp,m,S)
    EndBenchmark(ID)

```

```
  #print("The ciphertext is",ct)
  #print("Done!\n")
  box3 = getResAndClear(ID, "Encrypt("+str(S)+")", "
     Done!")

  #----------------------------------------------

  #print("Decrypt")

  ID = InitBenchmark()
  startAll(ID)
  res = scheme.decrypt(pp, sk, ct)
  EndBenchmark(ID)

  #print("The resulting ciphertext is",res)
  if res == m:
    fin = "Successful␣Decryption␣:)"
  else:
    fin = "Failed␣Decryption␣:("
  box4 = getResAndClear(ID, "Decrypt", fin)

  print(formatNice(box1,box2,box3,box4))

if __name__ == '__main__':
  debug = True
  main()
```

C.2 CP-ABE scheme of Sec. 6.2

```
'''
Rouselakis - Waters Unbounded Ciphertext-Policy
   Attribute-Based Encryption

| From:
| Published in:
| Available from:
| Notes:

```

```
* type:          attribute-based encryption (public
   key)
* setting:       bilinear pairing group of prime
   order
* assumption:    complex q-type assumption

:Authors:   Yannis Rouselakis
:Date:        02/12
'''

from toolbox.pairinggroup import *
from charm.cryptobase import *
from toolbox.secretutil import SecretUtil
from toolbox.ABEnc import *
from BenchmarkFunctions import *

debug = False
class CPABE_RW12(ABEnc):
  def __init__(self, groupObj, verbose = False):
    ABEnc.__init__(self)
    global util, group
    group = groupObj
    util = SecretUtil(group, verbose)

  # Defining a function to pick explicit exponents in
      the group
  def exp(self,value):
    return group.init(ZR, value)

  def setup(self):
    # Due to assymmetry in the groups we prefer most
       of the terms to be in G1
    g = group.random(G2)
    g2, u, h, w, v = group.random(G1), group.random(
       G1), group.random(G1), group.random(G1), group
       .random(G1)
    alpha = group.random( )
    egg = pair(g2,g)**alpha
    pp = {'g':g, 'g2':g2, 'u':u, 'h':h, 'w':w, 'v':v,
        'egg':egg}
    mk = {'alpha':g2 ** alpha }
    return (pp, mk)

  def keygen(self, pp, mk, S):
    # S is a list of attributes written as STRINGS i.
       e. {'1', '2', '3',...}
    r = group.random( )
    K0 = mk['alpha'] * (pp['w']**r)
    K1 = pp['g']**r

    vR = pp['v']**r

    K2, K3 = {}, {}
    for i in S:
      ri = group.random( )
      K2[i] = pp['g']**ri
      K3[i] = (pp['u']**self.exp(int(i)) * pp['h'])**
         ri * vR #NOTICE THE CONVERSION FROM STRING
         TO INT
    S = [s for s in S] #Have to be an array for util.
       prune
    return { 'S':S, 'K0':K0, 'K1':K1, 'K2':K2, 'K3':
       K3 }

  def encrypt(self, pp, message, policy_str):
    s = group.random()

    policy = util.createPolicy(policy_str)
    a_list = util.getAttributeList(policy)
    #print("\n\n THE A-LIST IS", a_list,"\n\n")
    shares = util.calculateSharesDict(s, policy) #
       These are correctly set to be exponents in Z_p

    C = message * (pp['egg']**s)
    C0 = pp['g']**s

    C1, C2, C3 = {}, {}, {}
    for i in a_list:
      inti = int(util.strip_index(i)) #NOTICE THE CONVERSION FROM STRING TO INT
      #print('The exponent is ',inti)
      ti = group.random()
      C1[i] = pp['w']**shares[i] * pp['v']**ti
      C2[i] = (pp['u']**self.exp(inti) * pp['h'])**ti
      C3[i] = pp['g']**ti
    return { 'Policy':policy_str, 'C':C, 'C0':C0, 'C1':C1, 'C2':C2, 'C3':C3 }

  def decrypt(self, pp, sk, ct):
    policy = util.createPolicy(ct['Policy'])
    z = util.getCoefficients(policy)
    #print("\n\n THE COEFF-LIST IS", z,"\n\n")

    pruned_list = util.prune(policy, sk['S'])
    # print("\n\n THE PRUNED-LIST IS", pruned_list,"\n\n")

    if (pruned_list == False):
      return group.init(GT,1)

    B = group.init(GT,1)
    for i in range(len(pruned_list)):
      x = pruned_list[i].getAttribute( ) #without the underscore
      y = pruned_list[i].getAttributeAndIndex( ) # with the underscore
      #print(x,y)
      B *= ( pair( ct['C1'][y], sk['K1']) * pair( ct['C2'][y], sk['K2'][x]) / pair(sk['K3'][x], ct['C3'][y]) )**z[y]

    return ct['C'] * B / pair(sk['K0'] , ct['C0'])

  def randomMessage(self):
    return group.random(GT)


def main():
  curve = 'MNT224'

  groupObj = PairingGroup(curve)
  scheme = CPABE_RW12(groupObj)
  # print("Setup(",curve,")")

  ID = InitBenchmark()
  startAll(ID)
  (pp, mk) = scheme.setup()
  EndBenchmark(ID)

  #print("The Public Parameters are",pp)
  #print("And the Master Key is",mk)
  #print("Done!\n")
  box1 = getResAndClear(ID, "Setup("+curve+")", "Done
     !")

  #----------------------------------------------

  S = {'123', '842',  '231', '384'}
  #print("Keygen(", str(S),")")

  ID = InitBenchmark()
  startAll(ID)
  sk = scheme.keygen(pp,mk,S)
  EndBenchmark(ID)

  #print("The secret key is",sk)
  #print("Done!\n")
```

```
box2 = getResAndClear(ID, "Keygen(" + str(S) + ")", "Done!")

#-------------------------------------------

m = group.random(GT)
policy = '(123␣or␣444)␣and␣(231␣or␣999)'
#print("Encrypt(",policy,")")

ID = InitBenchmark()
startAll(ID)
ct = scheme.encrypt(pp,m,policy)
EndBenchmark(ID)

#print("The ciphertext is",ct)
#print("Done!\n")
box3 = getResAndClear(ID, "Encrypt("+policy+")", "Done!")

#-------------------------------------------

#print("Decrypt")

ID = InitBenchmark()
startAll(ID)
res = scheme.decrypt(pp, sk, ct)
EndBenchmark(ID)

#print("The resulting ciphertext is",res)
if res == m:
  fin = "Successful␣Decryption␣:)"
else:
  fin = "Failed␣Decryption␣:("
box4 = getResAndClear(ID, "Decrypt", fin)

print(formatNice(box1,box2,box3,box4))
```

```
if __name__ == '__main__':
  debug = True
  main()
```

C.3 MA-CP-ABE scheme of Sec. 6.3

```
'''
Rouselakis - Waters Unbounded Multi-Authority
   Ciphertext-Policy Attribute-Based Encryption

| From:
| Published in:
| Available from:
| Notes:

* type:          attribute-based encryption (public
   key)
* setting:       bilinear pairing group of prime
   order
* assumption:    complex q-type assumption

:Authors:   Yannis Rouselakis
:Date:        11/12
'''

from toolbox.pairinggroup import *
from charm.cryptobase import *
from toolbox.secretutil import SecretUtil
from toolbox.ABEnc import *
from BenchmarkFunctions import *

debug = False
class MAABE_RW12():


  def randomMessage(self):
    return group.random(GT)

```

```
# Defining a function to pick explicit exponents in
    the group
def exp(self,value):
  return group.init(ZR, value)

def getAuth(self,x):
  i = x.find("@")
  if (i==-1):
    print("Error:␣No␣@␣char␣in␣[auth@attr]␣name")
    return

  j = x.find("_")
  if (j==-1):
    return x[i+1:]
  else:
    return x[i+1:j]

def getAttr(self, attrWithUnderscore):
  i = attrWithUnderscore.rfind("_")
  if (i==-1):
    return attrWithUnderscore
  else:
    return attrWithUnderscore[:i]

def __init__(self, groupObj, verbose = False):

  global util, group
  group = groupObj
  util = SecretUtil(group, verbose)

def GlobalSetup(self):
  g1 = group.random(G1)
  g2 = group.random(G2)
  egg = pair(g1,g2)
  H = lambda x: group.hash(x, G2)
  F = lambda x: group.hash(x, G2)
  gp = {'g1':g1, 'g2':g2, 'egg':egg, 'H':H, 'F':F}
```

```
    return gp

  def AuthSetup(self, gp, name):
    alpha, y = group.random(), group.random()
    egga = gp['egg']**alpha
    gy = gp['g1']**y
    pk = {'name':name, 'egga':egga, 'gy':gy}
    sk = {'name':name, 'alpha':alpha, 'y':y}
    return (pk, sk)

  def KeyGenOne(self, gp, gid, sk, attr):   # the
     authority's name is included in the secret key

    # check here if gid name is legal

    # checking if attribute is legal
    if (sk['name'] != self.getAuth(attr)):
      print("Error:␣Attribute␣", attr, "␣does␣not␣
         belong␣to␣authority␣", sk['name'])
      return

    t = group.random()
    K = gp['g2']**sk['alpha'] * gp['H'](gid)**sk['y']
        * gp['F'](attr)**t
    #K = gp['g2']**sk['alpha'] * gp['F'](attr)**t
    KP = gp['g1']**t

    return { 'user':gid, 'auth':sk['name'], 'attr':
       attr, 'K':K, 'KP':KP }

  def KeyGen(self, gp, gid, authSkChain, attributes):
    #check here if gid name is legal

    sks = {}
    for attr in attributes:
      auth = self.getAuth(attr)
      sk = self.KeyGenOne(gp, gid, authSkChain[auth],
          attr)
      sks[attr] = sk

    return {'GID':gid, 'Attributes':attributes, '
       Chain': sks}

  def Encrypt(self, gp, pks, message, policy_str):
    s = group.random()    #secret to be shared
    w = group.init(ZR, 0) #0 to be shared

    policy = util.createPolicy(policy_str)
    a_list = util.getAttributeList(policy)
    #print("\n\n THE A-LIST IS", a_list,"\n\n")
    #for i in a_list:
    # print(self.getAuth(i))

    secretShares = util.calculateSharesDict(s, policy
       ) #These are correctly set to be exponents in
       Z_p
    zeroShares = util.calculateSharesDict(w, policy)

    C0 = message * (gp['egg']**s)

    C1, C2, C3, C4 = {}, {}, {}, {}
    for i in a_list:
      auth = self.getAuth(i)
      attr = self.getAttr(i) #take out the possible
         underscore
      tx = group.random()
      C1[i] = gp['egg']**secretShares[i] * pks[auth][
         'egga']**tx
      C2[i] = gp['g1']**(-tx)
      C3[i] = pks[auth]['gy']**tx * gp['g1']**
         zeroShares[i]
      C4[i] = gp['F'](attr)**tx

    return { 'Policy':policy_str, 'C0':C0, 'C1':C1, 'C2':C2, 'C3':C3, 'C4':C4 }

  def Decrypt(self, gp, sk_chain, ct):
    hgid = gp['H'](sk_chain['GID'])

    policy = util.createPolicy(ct['Policy'])
    z = util.getCoefficients(policy)
#   print("\n\n THE COEFF-LIST IS", z,"\n\n")

    pruned_list = util.prune(policy, sk_chain['Attributes'])
#   print("\n\n THE PRUNED-LIST IS", pruned_list,"\n\n")

    if (pruned_list == False):
      return group.init(GT,1)

    B = group.init(GT,1)
    for i in range(len(pruned_list)):
      x = pruned_list[i].getAttribute( ) #without the underscore
      y = pruned_list[i].getAttributeAndIndex( ) #with the underscore
      #print(x,y)
      #print(z[y])
      B *= ( ct['C1'][y] * pair(ct['C2'][y], sk_chain['Chain'][x]['K']) * pair(ct['C3'][y], hgid) * pair(sk_chain['Chain'][x]['KP'], ct['C4'][y] ) )**z[y]

    return ct['C0']/B

def prettyPrint(initStr, myDict, tab=""):
  typesEnum = ["ZP", "G1", "G2", "GT"]
  if (len(initStr)>0):
    print(initStr)
  for (k,v) in myDict.items():
    if (isinstance(v,dict)):
      print(tab, k, ":␣", type(v))
      prettyPrint("", v, tab + "␣␣␣␣")
    elif (isinstance(v,str)):
      print(tab, k, ":␣", v)
    elif (isinstance(v,set)):
      print(tab, k, ":␣", v)
    elif (isinstance(v,pairing)):
      print(tab, k, ":␣", typesEnum[v.type])
    else:
      print(tab, k, ":␣", type(v))
  if (tab==""):
    print("\n")

def main():
  curve = 'MNT224'

  groupObj = PairingGroup(curve)
  scheme = MAABE_RW12(groupObj)
  print("Curve␣=␣",curve)

  ID = InitBenchmark()
  startAll(ID)
  gp = scheme.GlobalSetup()
  EndBenchmark(ID)
  boxGS = getResAndClear(ID, "GSetup("+curve+")", "
     Done!")

  #prettyPrint("The global parameters are ", gp)

  pks, sks = {}, {}

  ID = InitBenchmark()
  startAll(ID)
  (pk,sk) = scheme.AuthSetup(gp,"UT")
  EndBenchmark(ID)
```

```
boxAS = getResAndClear(ID, "ASetup(" + "UT" + ")",
   "Done!")

pks[pk['name']] = pk
sks[sk['name']] = sk

(pk,sk) = scheme.AuthSetup(gp,"OU")
pks[pk['name']] = pk
sks[sk['name']] = sk

#prettyPrint("The authority public key chain is ",
   pks)
#prettyPrint("The authority secret key chain is ",
   sks)

ID = InitBenchmark()
startAll(ID)
key = scheme.KeyGen(gp, "YANNIS", sks, {"STUDENT@UT
   ", "PHD@UT"})
EndBenchmark(ID)
boxKG = getResAndClear(ID, "KeyGen", "Done!")

#prettyPrint("The secret key is ", key)

m = scheme.randomMessage()
policy = '(STUDENT@UT␣or␣PROFESSOR@OU)␣and␣(
   STUDENT@UT␣or␣MASTERS@OU)'

ID = InitBenchmark()
startAll(ID)
ct = scheme.Encrypt(gp, pks, m, policy)
EndBenchmark(ID)
boxEC = getResAndClear(ID, "Encrypt", "Done!")

#prettyPrint("The ciphertext is ", ct)

ID = InitBenchmark()
```

```
  startAll(ID)
  res = scheme.Decrypt(gp, key, ct)
  EndBenchmark(ID)

  if res == m:
    fin = "Successful␣Decryption␣:)"
  else:
    fin = "Failed␣Decryption␣:("

  boxDE = getResAndClear(ID, "Decrypt", fin)

  #print(fin)

  print(formatNice(boxGS,boxAS,boxKG,boxEC, boxDE))

if __name__ == '__main__':
  debug = True
  main()
```

C.4 Implementation of Dual Vector Spaces

```
'''
Class to create the Dual Pairing Vector Spaces

:Authors:   Yannis Rouselakis
:Date:    4/24/12
'''

from toolbox.pairinggroup import *
from charm.cryptobase import *

debug = False
class DualVectorSpace():

  def __init__(self, groupObj, dim, psiSet = 0):
    global group
    group = groupObj      #we will use Charm to do
       the modular arithmetic for us
    #global Base         # a 2 x dim x dim matrix
    #global Psi   # the common inner product of all d_i d*_i
    if (type(psiSet) == int):
      self.Psi = group.random(ZR)
    else:
      self.Psi = psiSet
    self.Base = [None]*2
    self.Base[0] = self.createRandomMatrix(dim)
    self.Base[1] = self.gaussElimin(self.Base[0])

  def getVector(self, b, i):
    return self.Base[b][i]

  def getPsi(self):
    return self.Psi

  def createRandomMatrix(self,dim):     #this function will return a random matrix in Z_p of dimension dim x dim
    return [ [group.random(ZR) for i in range(0,dim)] for j in range(0,dim)]

  def gaussElimin(self, mat):

    work = [ ([mat[i][j] for j in range(0,len(mat))] + [(self.Psi if (i==j) else group.init(ZR, 0)) for j in range(0,len(mat))] ) for i in range(0,len(mat[0]))]

    # self.printOneBasis(work)

    (h,w) = (len(work),len(work[0]))

    #making it upper triangular
    for i in range(0,h):
      for i2 in range(i+1,h):
        c = work[i2][i] / work[i][i]      #I should check here for singular matrices (no: negl prob)
        for j in range(i,w):
          work[i2][j] -= work[i][j] * c

    # print("")
    # self.printOneBasis(work)

    #backsubstitution
    for i in range(h-1, 0-1, -1):

      # Normalize row i
      c = work[i][i]
      for j in range(i, w):
        work[i][j] /= c

      for i2 in range(0,i):
        c = work[i2][i]
        for j in range(i,w):
          work[i2][j] -= c * work[i][j]

    # print("")
    # self.printOneBasis(work)

    # transposing + cropping
    result = [ [work[i][j] for i in range(0,h)] for j in range(int(w/2), w)]

    # print("")
    # self.printOneBasis(result)
    return result

  def printBases(self, full = False):
    for b in range(0,2):
      if b==0:
        print("Normal␣Basis:")
      else:
        print("Star␣Basis:")
      self.printOneBasis(self.Base[b],full)


  def printOneBasis(self, mat, full = False):
    if full:
      print(mat[i])
    else:
      for i in range(0,len(mat)):
        bigStr = "["
        for j in range(0,len(mat[i])):
          cut = (int(mat[i][j]) > 999)

          smallStr = str(int(mat[i][j]) % 1000)
          extraSp = 3 - len(smallStr)

          if cut:
            bigStr += ".."
            for k in range(0, extraSp):
              bigStr += "0"
          else:
            bigStr += "␣␣"
            for k in range(0, extraSp):
              bigStr += "␣"
          bigStr += smallStr

          if j!=(len(mat[i])-1):
            bigStr += ",␣"
          else:
            bigStr += "]"
        print(bigStr)

  def checkOrthogonality(self):
    dim = len(self.Base[0][0]) # getting the
       dimension

    res = [ [group.init(ZR, 0) for i in range(0,dim)] for j in range(0,dim)]

    # matrix multiplication (the naive way)
    for i in range(0,dim):
      for j in range(0,dim):
        curr = group.init(ZR, 0)
        for k in range(0,dim):
          curr += self.Base[0][i][k] * self.Base[1][j][k]
        res[i][j] = curr

    print("B␣times␣transpose␣B*")
    self.printOneBasis(res, False)


def main():
  groupObj = PairingGroup('MNT224')
  DV = DualVectorSpace(groupObj,10)

  DV.printBases()

  DV.checkOrthogonality() #visual check ;)

if __name__ == '__main__':
  debug = True
  main()
```

Index

ABE, 59–68
  Ciphertext-Policy, 60
  Key-Policy, 61
  Multi-Authority, 63
Abstract, v
Access Structures, 24
Acknowledgments, iv
Appendices, 189
Attribute-Based Encryption, *see* ABE

Bibliography, 243
Bilinear Groups, 40–55
  Composite Order, 42
  Prime Order, 41
Bounded Leakage Model, 31

Ciphertext-Policy ABE, *see* ABE
Composite Order Bilinear Groups, *see* Bilinear Groups
Continual Leakage Model, 31
CP-ABE, *see* ABE

Elliptic Curves, 51–55
Exposure - Resilient Cryptography, 30

IBE, 57–59
Idenity-Based Encryption, *see* IBE

Key-Policy ABE, *see* ABE
KP-ABE, *see* ABE

Large Universe ABE, 65
Linear Secret-Sharing Schemes, 25

MA-CP-ABE, *see* ABE
Multi-Authority ABE, *see* ABE
Multi-Authority Attributes, 27

Nominal Semi-Functionality, 36

"Only Computation Leaks", 30

Prime Order Bilinear Groups, *see* Bilinear Groups

Semi-Functionality, 35
Small Universe ABE, 65

Unbounded ABE, 66

Bibliography

[1] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[2] Joseph A. Akinyele, Matthew Green, and Avi Rubin. Charm: A framework for rapidly prototyping cryptosystems. Cryptology ePrint Archive, Report 2011/617, 2011. `http://eprint.iacr.org/`.

[3] Sattam S. Al-Riyami, John Malone-Lee, and Nigel P. Smart. Escrow-free encryption supporting cryptographic workflow. *Int. J. Inf. Sec.*, 5(4):217–229, 2006.

[4] J. Alwen, Y. Dodis, M. Naor, G. Segev, S. Walfish, and D. Wichs. Public - key encryption in the bounded - retrieval model. In *EUROCRYPT*, pages 113–134, 2010.

[5] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[6] J. Alwen and L. Ibraimi. Leakage resilient ciphertext-policy attribute-based encryption. manuscript, 2010.

[7] Walid Bagga, Refik Molva, and Stefano Crosta. Policy-based encryption schemes from bilinear pairings. In *ASIACCS*, page 368, 2006.

[8] Manuel Barbosa and Pooya Farshim. Secure cryptographic workflow in the standard model. In *INDOCRYPT*, pages 379–393, 2006.

[9] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution.* PhD thesis, Dept. of Computer Science, Technion, 1996.

[10] M. Bellare, B. Waters, and S. Yilek. Identity-based encryption secure under selective opening attack. In *TCC*, 2011.

[11] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[12] E. Biham, Y. Carmeli, and A. Shamir. Bug attacks. In *CRYPTO*, pages 221–240, 2008.

[13] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[14] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series).* Cambridge University Press, New York, NY, USA, 2005.

[15] Ian F. Blake, G. Seroussi, and N. P. Smart. *Elliptic curves in cryptography.* Cambridge University Press, New York, NY, USA, 1999.

[16] A. Boldyreva, S. Fehr, and A. O'Neill. On notions of security for deterministic encryption, and efficient constructions without random oracles. In *CRYPTO*, pages 335–359, 2008.

[17] D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

[18] D. Boneh and D. Brumley. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[19] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, pages 37–51, 1997.

[20] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[21] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[22] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003. extended abstract in Crypto 2001.

[23] Dan Boneh, Craig Gentry, and Michael Hamburg. Space-efficient identity based encryption without pairings. In *FOCS*, pages 647–657, 2007.

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

[25] Dan Boneh and Brent Waters. Conjunctive, subset, and range queries on encrypted data. In *TCC*, pages 535–554, 2007.

[26] Robert W. Bradshaw, Jason E. Holt, and Kent E. Seamons. Concealing complex policies with hidden credentials. In *ACM Conference on Computer and Communications Security*, pages 146–157, 2004.

[27] Z. Brakerski and S. Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability (or: Quadratic residuosity strikes back). In *CRYPTO*, pages 1–20, 2010.

[28] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[29] R. Canetti, Y. Dodis, S. Halevi, E. Kushilevitz, and A. Sahai. Exposure-resilient functions and all-or-nothing transforms. In *EUROCRYPT*, pages 453–469, 2000.

[30] D. Cash, Y. Z. Ding, Y. Dodis, W. Lee, R. J. Lipton, and S. Walfish. Intrusion-resilient key exchange in the bounded retrieval model. In *TCC*, pages 479–498, 2007.

[31] Certivox. Miracl crypto sdk. `https://certivox.com/solutions/miracl-crypto-sdk/`.

[32] Charm. `http://www.charm-crypto.com`.

[33] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, pages 515–534, 2007.

[34] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *ACM Conference on Computer and Communications Security*, pages 121–130, 2009.

[35] Ling Cheung and Calvin C. Newport. Provably secure ciphertext policy abe. In *ACM Conference on Computer and Communications Security*, pages 456–465, 2007.

[36] S. Chow, Y. Dodis, Y. Rouselakis, and B. Waters. Practical leakage-resilient identity-based encryption from simple assumptions. In *ACM Conference on Computer and Communications Security*, pages 152–161, 2010.

[37] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[38] D. Di Crescenzo, R. J. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *TCC*, pages 225–244, 2006.

[39] Y. Dodis, K. Haralambiev, A. Lopez-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *FOCS*, pages 511–520, 2010.

[40] Y. Dodis, Y. Kalai, and S. Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[41] Y. Dodis and K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, pages 21–40, 2010.

[42] Y. Dodis, A. Sahai, and A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In *EUROCRYPT*, pages 301–324, 2001.

[43] S. Dziembowski. Intrusion-resilience via the bounded-storage model. In *TCC*, pages 207–224, 2006.

[44] S. Dziembowski and K. Pietrzak. Intrusion-resilient secret sharing. In *FOCS*, pages 227–237, 2007.

[45] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[46] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[47] David Mandell Freeman. Converting pairing-based cryptosystems from composite-order groups to prime-order groups. In *EUROCRYPT*, pages 44–61, 2010.

[48] Steven D. Galbraith, Kenneth G. Paterson, Nigel P. Smart, and Nigel P. Smart. Pairings for cryptographers. In *Discrete Applied Mathematics*, 2008.

[49] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[50] Sanjam Garg, Craig Gentry, Shai Halevi, Amit Sahai, and Brent Waters. Attribute-based encryption for circuits from multilinear maps. In *CRYPTO*, 2013.

[51] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476, 2013.

[52] C. Gentry and S. Halevi. Hierarchical identity based encryption with polynomially many levels. In *TCC*, pages 437–456, 2009.

[53] Craig Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

[54] S. Goldwasser and G. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[55] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits. In *STOC*, 2013.

[56] Vipul Goyal, Abhishek Jain, Omkant Pandey, and Amit Sahai. Bounded ciphertext policy attribute based encryption. In *ICALP (2)*, pages 579–591, 2008.

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

[58] A. Halderman, S. Schoen, N. Heninger, W. Clarkson, W. Paul, J. Calandrino, A. Feldman, J. Applebaum, and E. Felten. Lest we remember: Cold boot attacks on encryption keys. In *USENIX Security Symposium*, pages 45–60, 2008.

[59] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[60] A. Juma and Y. Vahlis. On protecting cryptographic keys against side-channel attacks. In *CRYPTO*, pages 41–58, 2010.

[61] J. Kamp and D. Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. In *FOCS*, pages 92–101, 2003.

[62] Mauricio Karchmer, Avi Wigderson, and Avi Wigderson. On span programs. In *Structure in Complexity Theory Conference*, pages 102–111, 1993.

[63] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

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

[65] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[66] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[67] Arjen K. Lenstra, Eric R. Verheul, and Eric R. Verheul. Selecting cryptographic key sizes. In *Public Key Cryptography*, pages 446–465, 2000.

[68] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, pages 62–91, 2010.

[69] Allison Lewko. *Functional Encryption: New Proof Techniques and Advancing Capabilities.* PhD thesis, The University of Texas at Austin, 2012.

[70] Allison B. Lewko. Tools for simulating features of composite order bilinear groups in the prime order setting. In *EUROCRYPT*, pages 318–335, 2012.

[71] Allison B. Lewko, Yannis Rouselakis, and Brent Waters. Achieving leakage resilience through dual system encryption. In *TCC*, pages 70–88, 2011.

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

[73] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.

[74] Allison B. Lewko and Brent Waters. Unbounded hibe and attribute-based encryption. In *EUROCRYPT*, pages 547–567, 2011.

[75] Allison B. Lewko and Brent Waters. Why proving hibe systems secure is difficult. *IACR Cryptology ePrint Archive*, 2013:68, 2013.

[76] Ben Lynn. The stanford pairing based crypto library. `http://crypto.stanford.edu/pbc`.

[77] S. Micali and L. Reyzin. Physically observable cryptography. In *TCC*, pages 278–296, 2004.

[78] Gerome Miklau and Dan Suciu. Controlling access to published data using cryptography. In *VLDB*, pages 898–909, 2003.

[79] Atsuko Miyaji, Masaki Nakabayashi, and Shunzo Takano. Characterization of elliptic curve traces under fr-reduction. In *ICISC*, pages 90–108, 2000.

[80] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[81] P. Q. Nguyen and I. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptology*, 15(3):151–176, 2002.

[82] Noam Nisan. Extracting randomness: How and why a survey. In *IEEE Conference on Computational Complexity*, pages 44–58, 1996.

[83] National Institute of Standards and Technology. Digital signature standard (dss), June 2009. `http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf`.

[84] Tatsuaki Okamoto and Katsuyuki Takashima. Homomorphic encryption and signatures from vector decomposition. In *Pairing*, pages 57–74, 2008.

[85] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical predicate encryption for inner-products. In *ASIACRYPT*, pages 214–231, 2009.

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

[87] Tatsuaki Okamoto and Katsuyuki Takashima. Adaptively attribute-hiding (hierarchical) inner product encryption. In *EUROCRYPT*, pages 591–608, 2012.

[88] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure unbounded inner-product and attribute-based encryption. In *ASIACRYPT*, pages 349–366, 2012.

[89] Tatsuaki Okamoto and Katsuyuki Takashima. Decentralized attribute-based signatures. In *Public Key Cryptography*, pages 125–142, 2013.

[90] Rafail Ostrovsky, Amit Sahai, and Brent Waters. Attribute-based encryption with non-monotonic access structures. In *ACM Conference on Computer and Communications Security*, pages 195–203, 2007.

[91] Dan Page, Nigel P. Smart, Frederik Vercauteren, and Frederik Vercauteren. A comparison of mnt curves and supersingular curves. In *IACR Cryptology ePrint Archive*, page 165, 2004.

[92] Omkant Pandey and Yannis Rouselakis. Property preserving symmetric encryption. In *EUROCRYPT*, pages 375–391, 2012.

[93] C. Petit, F.X. Standaert, O. Pereira, T. Malkin, and M. Yung. A block cipher based pseudo random number generator secure against side-channel key recovery. In *ASIACCS*, pages 56–65, 2008.

[94] K. Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[95] Matthew Pirretti, Patrick Traynor, Patrick McDaniel, and Brent Waters. Secure attribute-based systems. In *ACM Conference on Computer and Communications Security*, pages 99–112, 2006.

[96] J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[97] Yannis Rouselakis and Brent Waters. Efficient large universe multi - authority attribute - based encryption. Manuscript, 2012.

[98] Yannis Rouselakis and Brent Waters. Practical constructions and new proof methods for large universe attribute-based encryption. In *ACM Conference on Computer and Communications Security*, 2013.

[99] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[100] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[101] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[102] Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In *TCC*, pages 457–473, 2009.

[103] Elaine Shi and Brent Waters. Delegating capabilities in predicate encryption systems. In *ICALP (2)*, pages 560–578, 2008.

[104] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[105] Nigel P. Smart. Access control using pairing based cryptography. In *CT-RSA*, pages 111–121, 2003.

[106] Source code of our constructions. `www.cs.utexas.edu/~jrous`.

[107] F.X. Standaert, T. Malkin, and M. Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, pages 443–461, 2009.

[108] Lawrence C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition.* Chapman & Hall/CRC, 2 edition, 2008.

[109] B. Waters. Functional encryption for regular languages. In *CRYPTO*, pages 218–235, 2012.

[110] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

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

[112] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *Public Key Cryptography*, pages 53–70, 2011.

Vita

Ioannis "Yannis" Rouselakis was born in Athens, Greece on September 24, 1982, the son of Zoumpoulia I. Kriara and Konstantinos I. Rouselakis. Originally from Ierapetra, Crete, Yannis spent most of his childhood in Athens. He received his B.E. degree in Electrical and Computer Engineering from the National Technical University of Athens in 2007 with grade 9.74/10. He began his doctoral studies at the University of Texas at Austin in the spring of 2008.

Email address: `johnysrouss@gmail.com`

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.