Achieving Leakage Resilience Through Dual System Encryption

Allison Lewko [*]
University of Texas at Austin
alewko@cs.utexas.edu

Yannis Rouselakis
University of Texas at Austin
jrous@cs.utexas.edu

Brent Waters [†]
University of Texas at Austin
bwaters@cs.utexas.edu

Abstract

In this work, we show that strong leakage resilience for cryptosystems with advanced functionalities can be obtained quite naturally within the methodology of dual system encryption, recently introduced by Waters. We demonstrate this concretely by providing fully secure IBE, HIBE, and ABE systems which are resilient to bounded leakage from each of many secret keys per user, as well as many master keys. This can be realized as resilience against continual leakage if we assume keys are periodically updated and no (or logarithmic) leakage is allowed during the update process. Our systems are obtained by applying a simple modification to previous dual system encryption constructions: essentially this provides a generic tool for making dual system encryption schemes leakage-resilient.

1 Introduction

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. Instead, 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.

Recently, much progress has been made in obtaining increasingly complex systems with stronger security guarantees. 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. This line of research is motivated by a variety of side-channel attacks [49, 14, 7, 13, 55, 8, 50, 60, 35, 43], which allow attackers to learn partial information about secrets by observing physical properties of a cryptographic execution such as timing, power usage, etc. The cold-boot attack [43] allows an attacker to learn information about memory contents of a machine even after the machine is powered down.

Leakage-resilient cryptography 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

[*]Supported by a National Defense Science and Engineering Graduate Fellowship.

[†]Supported by NSF CNS-0716199, CNS-0915361, and CNS-0952692, Air Force Office of Scientific Research (AFO SR) under the MURI award for "Collaborative policies and assured information sharing" (Project PRESIDIO), Department of Homeland Security Grant 2006-CS-001-000001-02 (sub-award 641), and the Alfred P. Sloan Foundation.

than the bit-length of the secret key. Another approach is to continually refresh the secret key and bound the leakage between each update (this is called "continual leakage"). 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 [54, 48, 33, 59, 28, 1, 2, 34, 31, 26, 21, 32, 3, 17, 23, 18, 27].

Concurrently, the methodology of dual system encryption has emerged as a useful tool for improving the security guarantees for efficient cryptosystems with advanced functionalities like identity-based encryption (IBE), hierarchical identity-based encryption (HIBE), attribute-based encryption (ABE) [66, 52, 51]. These works provide efficient systems with short parameters which are proven fully secure in the standard model under static assumptions. Previous constructions of IBE and HIBE either used random oracles, had large parameters, were only proven selectively secure (a weaker model of security where the attacker must declare its target immediately instead of choosing it adaptively in the course of the security game), or relied on "q-based" assumptions (where the size of the assumption depends on the number of the attacker's queries) [15, 24, 39, 20, 10, 11, 64, 12, 36, 38, 37]. All previous constructions of ABE were only proven selectively secure [61, 42, 22, 6, 57, 41, 65]. Like leakage resilience, moving from selectively secure systems to fully secure systems is important because it results in security against a more powerful class of attackers.

Our Contribution In this work, 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 IBE schemes [18, 2, 23]. In particular, our IBE 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).[1] The IBE schemes of [2, 23] only allow bounded leakage on one secret key per identity, and allow no leakage on the master key. The IBE scheme of [18] allows bounded leakage on each of many keys per identity, but allows 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, with no sacrifice of efficiency.

Our Techniques In a dual system encryption scheme, keys and ciphertexts can each take on two forms: normal and semi-functional. Normal keys can decrypt both forms of ciphertexts, while semi-functional keys can only decrypt normal ciphertexts. In the real security game, the ciphertext and all keys are normal. Security is proven by a hybrid argument, where first the ciphertext is changed to semi-functional, and then the keys are changed to semi-functional one by one. We must prove that the attacker cannot detect these changes. Finally, we arrive at a game where the simulator need only produce semi-functional objects, which cannot correctly decrypt. This greatly reduces the burden on the simulator and allows us to now prove security directly.

There is an important challenge inherent in this technique: when we argue the indistinguishability of games where a certain key is changing from normal to semi-functional, it is crucial that the simulator cannot determine the nature of this key for itself by test decrypting a semi-functional ciphertext. However, the simulator should also be prepared to make a semi-functional ciphertext for *any* identity and to use *any*

[1]For simplicity, we present our system as allowing no leakage during key updates, but our system can tolerate leakage which is logarithmic in terms of the security parameter using the same methods employed in [18].

identity for this particular key. This challenge is overcome by allowing the simulator to make *nominal* semi-functional keys: these are keys that are distributed like ordinary semi-functional keys in the attacker's view, but in the simulator's view they are correlated with the challenge ciphertext, so that if the simulator tries to decrypt a semi-functional ciphertext, decryption will always succeed, and hence will not reveal whether the key is normal or nominally semi-functional.

To keep nominal semi-functionality hidden from the attacker's view, previous dual system encryption constructions relied crucially on the fact that the attacker cannot ask for a key capable of decrypting the challenge ciphertext. When we add leakage to this framework, the attacker is now able to ask for leakage on keys which are capable of decrypting the challenge ciphertext: hence we need a new mechanism to hide nominal semi-functionality from attackers who can leak on these keys.

We accomplish this by expanding the semi-functional space to form m dimensional vectors, where $m \geq 3$ is a parameter determining the leakage tolerance. Nominality now corresponds to the vector in the semi-functional space of the key being orthogonal to the vector in the semi-functional space of the ciphertext. Because the leakage function on the key must be determined *before* the challenge ciphertext is revealed, an attacker whose leakage is suitably bounded cannot distinguish orthogonal vectors from uniformly random vectors in this context (this is a corollary of the result from [18], which shows that "random subspaces are leakage-resilient"). Hence, the attacker cannot distinguish leakage on a nominally semi-functional key from leakage on an ordinary semi-functional key. This allows us to obtain leakage resilience within the dual system encryption framework.

Comparison to Previous Techniques One of the leakage-resilient IBE constructions of [23] also applied the dual system encryption methodology, but ultimately relied on the technique of hash proof systems [25, 54, 2] to obtain leakage resilience, instead of deriving leakage resilience from the dual system encryption methodology itself, as we do in this work. More precisely, they used the dual system encryption framework to allow the simulator to produce keys incapable of decrypting the challenge ciphertext, but did not apply dual system encryption to handle leakage on keys which are capable of decrypting the challenge ciphertext. Instead, they relied on a hash proof mechanism for this part of the proof. This leads them to impose the restriction that the attacker can only leak from one key for the challenge identity, and no leakage on the master key is allowed. Essentially, their application of dual system encryption is "orthogonal" to their techniques for achieving leakage resilience. In contrast, our techniques allow us to handle all key generation and leakage queries *within the dual system encryption framework*, eliminating the need for a separate technique to achieve leakage resilience. This enables us to allow leakage from multiple keys which can decrypt the challenge ciphertext, as well as leakage from the master key.

The leakage-resilient IBE construction of [18] in the continual leakage model relies on selective security to allow the simulator to produce the keys incapable of decrypting challenge ciphertext. This is accomplished with a partitioning technique. Their technique for handling leakage on secret keys for the challenge identity is more similar to ours: they produce these keys and ciphertext in such a way that each is independently well-distributed, but the keys for the challenge identity exhibit degenerate behavior relative to the challenge ciphertext. This correlation, however, is information-theoretically hidden from the adversary because the leakage per key is suitably bounded. We employ a similar information-theoretic argument to hide nominal semi-functionality of leaked keys from the attacker's view. However, their technique does not quite fit our dual system encryption framework, and only achieves selective security in their implementation, with no leakage allowed from the master key.

1.1 Related Work

Leakage resilience has been studied in many previous works, under a variety of leakage models [63, 58, 48, 3, 21, 26, 32, 30, 47, 31, 54, 1, 2, 19, 28, 33, 45, 53, 59, 34, 17, 29, 18, 27]. Exposure-resilient cryptography [19, 30, 47] 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 [53] 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 [33, 59, 34]. Additionally, [46, 40] 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 [43] 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 [1]. This has resulted in constructions of pseudorandom functions, signature schemes, public key encryption, and identity-based encryption [28, 54, 3, 48, 2, 23] 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 [18, 27]. Dodis, Haralambiev, Lopez-Alt, and Wichs [27] 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 [18] 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 [66]. It has been leveraged to obtain constructions of fully secure IBE and HIBE from simple assumptions [66], fully secure HIBE with short ciphertexts [52], fully secure ABE and Inner Product Encryption (IPE) [51], and fully secure functional encryption combining ABE and IPE [56].

Independently, Alwen and Ibraimi [4] 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.

1.2 Organization

In Section 2, we provide the necessary definitions and state our complexity assumptions. In Section 3, we define the dual system encryption methodology for IBE schemes as an abstraction. In Section 4, we formalize the relationship between our security model for leakage resilience and the model of continual leakage resilience for IBE schemes. In Section 5, we present our IBE scheme. In Section 6, we prove its security. In Section 7, we present our HIBE scheme. In Section 8, we present our Ciphertext-Policy Attribute-Based Encryption (CP-ABE) scheme. In Section 9, we discuss the leakage bounds for our schemes.

2 Preliminaries

2.1 Notation

We denote by $s \xleftarrow{\$} S$ the fact that s is picked uniformly at random from a finite set S and by $x, y, z \xleftarrow{\$} S$ that all x, y, z are picked independently and uniformly at random from S. By $\mathsf{negl}(\lambda)$ we denote a negligible function of λ, i.e. a function $f : \mathbb{N} \to \mathbb{R}$ such that for every $c > 0$ and for all but a finite number of λ's: $f(\lambda) \leq \lambda^{-c}$. By $|x|$ we denote the size/number of bits of term x. Also, the special symbol $\perp$ is meant to serve as a unique dummy value in all our systems. Finally, by PPT we denote a probabilistic polynomial-time algorithm.

2.2 Composite Order Bilinear Groups

Our construction uses composite order bilinear groups, first introduced in [16]. We use groups of order $N = p_1p_2p_3$, where p_1, p_2, p_3 are three distinct prime numbers. We require that these groups, denoted $\mathbb{G}$ from now on, admit an efficiently computable non-degenerate bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$, where $\mathbb{G}_T$ is a target group of the same order:

Non-degenerate:	For all generators $g \in \mathbb{G}$	$e(g, g) \neq 1_{\mathbb{G}_T}$
Bilinear:	For all $a, b \in \mathbb{Z}_N$	$e(g^a, g^b) = e(g, g)^{ab}$

Since $\mathbb{G}$ is of composite order, it includes three subgroups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ of order p_1, p_2, p_3, respectively. Any element of $\mathbb{G}$ is of the form $g_1^{x_1} g_2^{x_2} g_3^{x_3}$, where g_i is a generator of subgroup $\mathbb{G}_i$ and $x_i \in \mathbb{Z}_{p_i}$. We will refer to $g_1^{x_1}, g_2^{x_2}, g_3^{x_3}$ as the $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ part of the term, respectively. It is easily proved that if an element of $\mathbb{G}_i$ (i.e. $x_k = 0$ for $k \neq i$) is paired with an element of $\mathbb{G}_j$ with $i \neq j$, the result of the pairing is the identity element.

To see this, suppose that g is a generator of $\mathbb{G}$. Then $g^{p_1p_2}$ generates $\mathbb{G}_3$, $g^{p_1p_3}$ generates $\mathbb{G}_2$, and $g^{p_2p_3}$ generates $\mathbb{G}_1$. Suppose that $u \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. Then for some a, b we have that $u = (g^{p_2p_3})^a$ and $h = (g^{p_1p_3})^b$. But then

$$e(u, h) = e(g^{p_2p_3a}, g^{p_1p_3b}) = e(g^{p_3a}, g^b)^{p_1p_2p_3} = 1_{\mathbb{G}_T}$$

We will make heavy use of this property in our constructions.

2.3 Complexity Assumptions

To prove the security of our system, we will use the following three assumptions in composite order groups, also used in [52, 51]. 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 [52].

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 $N = p_1p_2p_3$, a map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ 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.

The three assumptions are the following:

Assumption 2.1. Given $D^1 = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_3)$ no PPT adversary has a non-negligible advantage in distinguishing

$$T_0^1 = g_1^z \text{ from } T_1^1 = g_1^z g_2^\nu,$$

where $z, \nu \xleftarrow{\$} \mathbb{Z}_N$. The advantage of an algorithm $\mathcal{A}$ in solving Assumption 2.1 is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{2.1}(\lambda) = \left|Pr[\mathcal{A}(D^1, T_0^1) = 1] - Pr[\mathcal{A}(D^1, T_1^1) = 1]\right|.$$

We say that Assumption 2.1 holds if for all PPT $\mathcal{A}$, $Adv_{\mathcal{A}}^{2.1}(\lambda) \leq \mathsf{negl}(\lambda)$.

Assumption 2.2. Given $D^2 = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_3, g_1^z g_2^\nu, g_2^\mu g_3^\rho)$ where $z, \nu, \mu, \rho \xleftarrow{\$} \mathbb{Z}_N$, no PPT adversary has a non-negligible advantage in distinguishing

$$T_0^2 = g_1^w g_3^\sigma \text{ from } T_1^2 = g_1^w g_2^\kappa g_3^\sigma,$$

where $w, \kappa, \sigma \xleftarrow{\$} \mathbb{Z}_N$. The advantage of an algorithm $\mathcal{A}$ in solving Assumption 2.2 is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{2.2}(\lambda) = \left|Pr[\mathcal{A}(D^2, T_0^2) = 1] - Pr[\mathcal{A}(D^2, T_1^2) = 1]\right|.$$

We say that Assumption 2.2 holds if for all PPT $\mathcal{A}$, $Adv_{\mathcal{A}}^{2.2}(\lambda) \leq \mathsf{negl}(\lambda)$.

Assumption 2.3. Given $D^3 = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_2, g_3, g_1^{\alpha} g_2^{\nu}, g_1^z g_2^{\mu})$ where $\alpha, \nu, z, \mu \overset{\$}{\leftarrow} \mathbb{Z}_N$, no PPT adversary has a non-negligible advantage in distinguishing

$$T_0^3 = e(g_1, g_1)^{\alpha z} \in \mathbb{G}_T \text{ from } T_1^3 \overset{\$}{\leftarrow} \mathbb{G}_T.$$

The advantage of an algorithm $\mathcal{A}$ in solving Assumption 2.3 is defined as:

$$\mathsf{Adv}_{\mathcal{A}}^{2.3}(\lambda) = \left| Pr[\mathcal{A}(D^3, T_0^3) = 1] - Pr[\mathcal{A}(D^3, T_1^3) = 1] \right|.$$

We say that Assumption 2.3 holds if for all PPT $\mathcal{A}$, $Adv_{\mathcal{A}}^{2.3}(\lambda) \leq \mathsf{negl}(\lambda)$.

2.4 Vector Notation

We will use angle brackets $\langle \cdot, \cdot, \cdot, \cdot \rangle$ to denote vectors and parentheses $(\cdot, \cdot, \cdot, \cdot)$ to denote collections of elements of different types. The dot product of vectors is denoted by $\cdot$ and component-wise multiplication is denoted by $*$.

We define the exponentiation operator for vectors in the following way: For $u \in \mathbb{G}$, $\vec{u} = \langle u_1, u_2, \ldots, u_n \rangle \in \mathbb{G}^n$, $a \in \mathbb{Z}_N$, and $\vec{a} \in \mathbb{Z}_N^n$ where $\mathbb{G}$ is a group and N, n are integers, we define

$$u^{\vec{a}} := \langle u^{a_1}, u^{a_2}, \ldots, u^{a_n} \rangle, \qquad \vec{u}^a := \langle u_1^a, u_2^a, \ldots, u_n^a \rangle$$

The resulting terms are elements of $\mathbb{G}^n$ and all the above operations are efficiently computable if the underlying operations of $\mathbb{G}$ are efficiently computable.

For a bilinear group $\mathbb{G}$, we define a pairing operation of vectors in $\mathbb{G}^n$: For $\vec{v}_1 = \langle v_{11}, v_{12}, \ldots, v_{1n} \rangle \in \mathbb{G}^n$ and $\vec{v}_2 = \langle v_{21}, v_{22}, \ldots, v_{2n} \rangle \in \mathbb{G}^n$, their pairing is

$$e_n(\vec{v}_1, \vec{v}_2) = \prod_{i=1}^{n} e(v_{1i}, v_{2i}) \in \mathbb{G}_T,$$

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 $\mathbb{G}_T$.

2.5 Identity-Based Encryption

An Identity-Based Encryption system (first introduced in [62]) 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.

Setup $(1^{\lambda}) \to (\text{PP}, \text{MK})$ The setup algorithm takes an integer security parameter, λ, as input and outputs the public parameters, PP, and the original master key, MK. 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.

Keygen$(\text{MK}', X) \to K$ The key generation algorithm takes in a master key, MK$'$, and either $X = I$, an identity, or $X = \epsilon$, the empty string[2]. In the former case, it outputs a secret key, $K = \text{SK}$, for the identity I. In the latter case, it outputs another master key, $K = \text{MK}''$, such that $|\text{MK}''| = |\text{MK}'|$ [3]. This new master key can now be used instead of the original key in calls of **Keygen** (either with I or with ϵ as input).

[2]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 4).

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

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

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 I 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 MK, PP generated by a call to **Setup**, for all master keys MK' generated by applying the **Keygen** algorithm with the empty string and a previously generated master key, and for all M, I

$$\mathbf{Decrypt}(\mathbf{Encrypt}(M, I), \mathbf{Keygen}(\mathrm{MK}', I)) = M.$$

2.6 Security Definition

The security of our 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_{MK} bits per master key (remember we can have many master keys) and ℓ_{SK} bits per secret key, where $\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}}$ 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 $\ell_{\mathrm{MK}} = \ell_{\mathrm{MK}}(\lambda)$, $\ell_{\mathrm{SK}} = \ell_{\mathrm{SK}}(\lambda)$. 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 MasterLeak consists of the following phases:

Setup: The challenger makes a call to **Setup**(1^λ) and gets a master key MK and the public parameters PP. It gives PP to the attacker. Also, it sets $\mathcal{R} = \emptyset$ and $\mathcal{T} = \{(0, \epsilon, \mathrm{MK}, 0)\}$. 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 I 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 $(h, \epsilon, \mathrm{MK}', L)$. Then the challenger makes a call to **Keygen**$(\mathrm{MK}', X) \rightarrow K$ and adds the tuple $(H + 1, X, K, 0)$ to the set $\mathcal{T}$. K is either a secret key for identity I 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 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, I, SK, L) or $\big(h, \epsilon, \mathrm{MK}', L\big)$ [4].

  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 MK', it checks if $L + \big|f(\mathrm{MK}')\big| \leq \ell_{\mathrm{MK}}$. If this is true, it responds with $f(\mathrm{MK}')$ and updates the L with $L + \big|f(\mathrm{MK}')\big|$. 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, I, SK, L). The challenger responds with SK and adds the identity I to the set $\mathcal{R}$.

Challenge: The adversary submits a challenge identity $I^* \notin \mathcal{R}$ and two messages M_0, M_1 of equal size. The challenger flips a uniform coin $c \xleftarrow{\$} \{0, 1\}$ and encrypts M_c under I^* with a call to **Encrypt**(M_c, I^*). 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 I^*. The reason for forbidding **Leak** queries on a master key and on I^* 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 I^* to the adversary and hence no **Reveal** queries involving I^* are allowed too. **Leak** queries on keys of identities other than I^* 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 2.4. An IBE encryption system Π is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure if for all PPT adversaries $\mathcal{A}$ it is true that

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

[4] It can be the case that MK' is the original master key.

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

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

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

3 Dual System IBE

We now define dual system IBE schemes as an abstraction and define three security properties which will ensure leakage resilience. We show that these properties imply that a dual system IBE scheme is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure.

3.1 Definition

A dual system IBE scheme Π_D has the following algorithms:

Setup$(1^\lambda) \to (\mathrm{PP}, \mathrm{MK})$ The setup algorithm takes in the security parameter, λ, and outputs the public parameters, PP, and a normal master key, MK.

Keygen$(\mathrm{MK}', X) \to K$ The key generation algorithm takes in a normal master key, MK$'$, and either an identity, I, or the empty string ϵ. In the first case, it outputs a normal secret key, SK, for the identity I, and in the second case, it outputs another normal master key, MK$''$.

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

Decrypt$(\mathrm{CT}, \mathrm{SK}) \to M$ The decryption algorithm takes in a ciphertext CT encrypted to identity I, and a secret key SK for identity I. It outputs the message M, unless both the key and the ciphertext are semi-functional.

KeygenSF$(\mathrm{MK}', X) \to \widetilde{K}$ The semi-functional key generation algorithm works in a similar way to **Keygen** but outputs semi-functional keys. If $X = I$, an identity, it outputs a semi-functional secret key, $\widetilde{SK}$ for identity I. If $X = \epsilon$, the empty string, it outputs a semi-functional master key, $\widetilde{MK}$.

Notice that this algorithm takes in a *normal* master key; not a semi-functional one. Also, this algorithm *need not be polynomial time computable*, in contrast to **Setup**, **Keygen**, **Encrypt**, and **Decrypt**.

EncryptSF$(M, I) \to \widetilde{\mathrm{CT}}$ The semi-functional encryption algorithm takes in a message M, and an identity I, and outputs a semi-functional ciphertext, CT. This algorithm *need not be polynomial time computable.*

3.2 Security Properties for Leakage Resilience

We now define three security properties for a dual system IBE scheme. For this, we define two additional games which are modifications of the $\mathsf{MasterLeak}$ game.

The first game, called $\mathsf{MasterLeakC}$, is exactly the same as the $\mathsf{MasterLeak}$ 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 second new game, called $\mathsf{MasterLeakCK}$, 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. [5] More precisely, the challenger additionally stores a semi-functional master key in tuple 0 by calling **KeygenSF**(MK, ϵ) after calling **Setup**. Thereafter, for all **Create**(h, X) queries, the challenger makes an additional call to **KeygenSF**(MK', X), where MK' 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.

Finally, notice that the same attackers that play game MasterLeak can play games MasterLeakC and MasterLeakCK without any change in their algorithms - queries etc. The change in set $\mathcal{T}$ is irrelevant to the attacker.

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

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

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

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

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

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

Theorem 3.1. *If a dual system IBE scheme Π_D =(**Setup**, **Keygen**, **Encrypt**, **Decrypt**, **KeygenSF**, **EncryptSF**) has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional ciphertext invariance, $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional key invariance, and $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional security, then Π =(**Setup**, **Keygen**, **Encrypt**, **Decrypt**) is a $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure IBE scheme.*

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

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

By semi-functional ciphertext invariance, we have that:

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

By semi-functional key invariance, we have that:

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

[5] As one should notice, we will never use the normal versions of non-master keys in this game. However, we have them here because we will need them in the game of the next section and when we move to the HIBE setting.

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

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

By semi-functional security, we know that

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

Hence,

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

which implies that

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

□

3.3 An Alternate Security Property

We additionally define a property called *One Semi-functional Key Invariance*. We will show that this implies semi-functional key invariance, and so can be substituted for semi-functional key invariance in proving that a system is $(\ell_{\mathrm{MK}},\ell_{\mathrm{SK}})$-master-leakage secure. The motivation for this is that proving semi-functional key invariance directly will often involve a hybrid argument, and defining one semi-functional key invariance allows us to include this hybrid as part of our abstraction and hence avoid repeating it for each system.

To define this property, we first define one more variation of our security game, called $\mathsf{MasterLeak}_b$. This is similar to the $\mathsf{MasterLeakCK}$ 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 - identity or empty string - normal key - semi-functional key - leakage - lock. For example, the original master key is stored at the beginning of the game in the tuple $(0, \epsilon, \mathrm{MK}, \mathbf{KeygenSF}(\mathrm{MK}, \epsilon), 0, -1)$.

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 $b \xleftarrow{\$} \{0,1\}$ 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).

One Semi-functional Key Invariance: We say that a dual system IBE scheme Π_D has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-*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:

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

Theorem 3.2. *If a dual system IBE scheme Π_D has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-one semi-functional key invariance, then it also has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-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

$$\left|\mathsf{Adv}_{\mathcal{A},\Pi_D}^{\mathsf{MasterLeakC}}(\lambda, \ell_{\mathrm{MK}}, \ell_{\mathrm{SK}}) - \mathsf{Adv}_{\mathcal{A},\Pi_D}^{\mathsf{MasterLeakCK}}(\lambda, \ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})\right| \tag{1}$$

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 (1) and Q is a polynomial in λ, there exists a $q^* \in [0, Q-1]$ such that the difference

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

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

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

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

4 Continual Leakage

If an IBE 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 our security definition 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 [18] and [27]. 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 IBE scheme in question has a specific re-randomization property (our scheme will 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 **UpdateMK** and **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 scheme of Section 3.1 has an extra **UpdateSK** algorithm and a specific re-randomization property, then our definition of security implies security in the CLM. The **UpdateMK** algorithm is going to be implemented by our **Keygen** algorithm with the empty string as input. The additional algorithm is:

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

Definition 4.1. An IBE scheme $\Pi = (\mathbf{Setup}, \mathbf{Keygen}, \mathbf{Encrypt}, \mathbf{Decrypt}, \mathbf{UpdateSK})$ is called an *IBE with re-randomization* if the following property holds:

For all MK, PP generated by a call to **Setup**, for all master keys $\mathrm{MK}', \mathrm{MK}''$ generated by applying the **Keygen** algorithm with the empty string and a previously generated master key, for all identities I, the distribution of a secret key SK' generated by the $\mathbf{UpdateSK}(\mathbf{Keygen}(\mathrm{MK}', I))$ method is indistinguishable from the distribution of a secret key SK generated by $\mathbf{Keygen}(\mathrm{MK}'', I)$.

The security definition of IBE schemes in the Continual Leakage Model is defined via the following game, called ClmIbe. This is proposed (informally) in [18]. 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 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 algorithm, 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 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_{MK}. The phases of the game are:

Setup - CLM: The challenger chooses "secret randomness" r and "public randomness" p and calls $\mathbf{Setup}(1^\lambda; r, p) \to (\mathrm{PP}, \mathrm{MK})$. The adversary specifies a polynomial-time computable function f 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 $L_{\mathrm{MK}} = |f(r,p)|$ 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).

- $\mathbf{Keygen}(I)$: The challenger adds the identity I to $\mathcal{R}$, since it should be considered "revealed" from now on and responds with the output of a call to $\mathbf{Keygen}(\mathrm{MK}, I)$.
- $\mathbf{MasterLeak}(f)$: In this query, f is a function such that $L_{\mathrm{MK}} + |f(\mathrm{MK})| \leq \rho_M \cdot |\mathrm{MK}|$. The adversary requests leakage from the master key here. The challenger responds with the value $f(\mathrm{MK})$ and updates L_{MK} to $L_{\mathrm{MK}} + |f(\mathrm{MK})|$. If $L_{\mathrm{MK}} + |f(\mathrm{MK})| > \rho_M \cdot |\mathrm{MK}|$, it responds with the dummy value $\perp$.

- **UpdateMK**(f): Now the attacker requests an update (and leakage) on the master key. It should be true that $|f(\mathrm{MK}, r, p)| \leq \rho_{UM} \cdot (|\mathrm{MK}| + |r|)$ for all MK, 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, $\widehat{\mathrm{MK}} \leftarrow \textbf{Keygen}(\mathrm{MK}, \epsilon; r, p)$. If $L_{\mathrm{MK}} + |f(\mathrm{MK}, r, p)| \leq \rho_M \cdot |\mathrm{MK}|$, it gives to the attacker $f(\mathrm{MK}, r, p)$. Finally, it sets $L_{\mathrm{MK}} = |f(\mathrm{MK}, r, p)|$ and $\mathrm{MK} \leftarrow \widehat{\mathrm{MK}}$, in that order.

Challenge Identity - CLM: In this phase, the attacker chooses the challenge identity $I^* \notin \mathcal{R}$ and the challenger creates a secret key for it: $\mathrm{SK}_{I^*} \leftarrow \textbf{Keygen}(MK, I^*)$. 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 similar to the respective ones of **Phase 1 -CLM**.

- **Keygen**(I): Obviously $I \neq I^*$ is required.
- **MasterLeak**(f)
- **UpdateMK**(f)
- **Leak**(f): In this query, f is a function such that $L_{\mathrm{SK}} + |f(\mathrm{SK}_{I^*})| \leq \rho_S \cdot |\mathrm{SK}_{I^*}|$. The adversary requests leakage from the secret key of I^* here. The challenger responds with the value $f(\mathrm{SK}_{I^*})$ and updates L_{SK} to $L_{\mathrm{SK}} + |f(\mathrm{SK}_{I^*})|$. If $L_{\mathrm{SK}} + |f(\mathrm{SK}_{I^*})| > \rho_S \cdot |\mathrm{SK}_{I^*}|$, it responds with the dummy value $\perp$.
- **UpdateSK**(f): Now the attacker requests an update (and leakage) on the secret key of I^*. It should be true that $|f(\mathrm{SK}_{I^*}, r, p)| \leq \rho_{US} \cdot (|\mathrm{SK}| + |r|)$ 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, $\widehat{\mathrm{SK}_{I^*}} \leftarrow \textbf{Keygen}(\mathrm{MK}, I^*; r, p)$. If $L_{\mathrm{SK}} + |f(\mathrm{SK}_{I^*}, r, p)| \leq \rho_S \cdot |\mathrm{SK}_{I^*}|$, it gives to the attacker $f(\mathrm{SK}_{I^*}, r, p)$. Finally, it sets $L_{\mathrm{SK}} = |f(\mathrm{SK}_{I^*}, r, p)|$ and $\mathrm{SK}_{I^*} \leftarrow \widehat{\mathrm{SK}_{I^*}}$, in that order.

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

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

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

We say that a scheme $\Pi = (\textbf{Setup}, \textbf{Keygen}, \textbf{Encrypt}, \textbf{Decrypt}, \textbf{UpdateSK})$ 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 Clmlbe game.

We will prove the following theorem:

Theorem 4.2. *If an IBE system $\Pi = ($**Setup**, **Keygen**, **Encrypt**, **Decrypt**, **UpdateSKey**$)$ with re-randomization is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure, then it is also*

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

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 $\left(0, 0, \frac{\ell_{\mathrm{MK}}}{|\mathrm{MK}|}, 0, \frac{\ell_{\mathrm{SK}}}{|\mathrm{SK}|}\right)$. 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 $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage security of our system. $\mathcal{B}$ will play the role of $\mathcal{A}$'s challenger in the Clmlbe 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 MasterLeak game. It will use a handle h_{MK} to denote the "current" master key and a handle h_{I^*} 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 $h_{\text{MK}} = 0$. $\mathcal{B}$ works as follows:

Setup: $\mathcal{B}$ executes the setup phase of 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 **Keygen**(I) query made by $\mathcal{A}$, $\mathcal{B}$ makes a **Create**$(h_{\text{MK}}, I) \to h'$ 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**(MK, I), where MK is the current master key.

For every **MasterLeak**(f) query made by $\mathcal{A}$, $\mathcal{B}$ makes a **Leak**(h_{MK}, f) query. Since $L_{\text{MK}} + |f(\text{MK})| \leq \rho_M \cdot |\text{MK}| \implies L_{\text{MK}} + |f(\text{MK})| \leq \ell_{\text{MK}}$, the challenger of the MasterLeak game has to provide $\mathcal{B}$ with the requested leakage $f(\text{MK})$. 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_{MK} - 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**$(h_{\text{MK}}, \epsilon) \to h'$ query. It sets $h_{\text{MK}} \leftarrow h'$, which changes the current master key to the new one. The method called is exactly the same, i.e. **Keygen**(MK, ϵ); hence $\mathcal{A}$ sees no difference.

At some point, $\mathcal{A}$ reaches the challenge phase. After sending the challenge identity I^*, $\mathcal{B}$ makes a **Create**$(h_{\text{MK}}, I^*) \to h_{I^*}$ query. The handle h_{I^*} 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 **Leak**(f) query, $\mathcal{B}$ makes a **Leak**(h_{I^*}, f) query. Since $L_{\text{SK}} + |f(\text{SK}_{I^*})| \leq \rho_S \cdot |\text{SK}_{I^*}| \implies L_{\text{SK}} + |f(\text{SK}_{I^*})| \leq \ell_{\text{SK}}$, the challenger of the MasterLeak game has to provide $\mathcal{B}$ with the requested leakage $f(\text{SK}_{I^*})$. 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 **UpdateSK**(f) query, $\mathcal{B}$ has only to update the secret key of I^*. That is because $\rho_{US} = 0$ and thus f outputs nothing. Instead of updating, it makes a **Create**$(h_{\text{MK}}, I^*) \to h'$ query. It sets $h_{I^*} \leftarrow h'$, 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 $I \neq I^*$. For each such query, $\mathcal{B}$ makes a **Create**$(h_{\text{MK}}, I) \to h'$ 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**(MK, I), where MK is the current master key.

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

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

4.1 Leakage from Updates

We note here that using the same "guess and check" method of [18], 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 [18].

5 Master-Leakage Secure IBE Scheme

The IBE scheme we give here is an augmented version of the Lewko-Waters IBE [52], designed to sustain master and secret key leakage from an arbitrary number of keys. The original version of the Lewko-Waters IBE was proven secure (without leakage) using the dual system encryption framework, i.e. security was proven by a hybrid argument where first the ciphertext was changed to the semi-functional, and then the revealed keys were changed to semi-functional one at a time. At the step where a particular key is being changed to semi-functional, it is crucial that the simulator cannot determine the nature of this key for itself, and yet it seems that the simulator can make a semi-functional ciphertext for the same identity and test for semi-functionality of the key by attempting to decrypt. This apparent paradox is resolved by only allowing the simulator to make this key *nominally semi-functional*, meaning that even if it has some semi-functional components, they will cancel out upon decryption. To the attacker, the key will still appear to be distributed as a regular semi-functional key, since it cannot be for the same identity as the challenge ciphertext.

Now that we allow the attacker to request leakage on a secret key for the same identity as the challenge ciphertext, it is more challenging to hide nominal semi-functionality in the attacker's view. To accomplish this, we add vectors of dimension n to the front of the ciphertexts and secret keys of 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 then use the algebraic lemma of [18] to assert that this orthogonality is hidden from attackers with suitably bounded leakage. Essentially, the attacker can learn limited information about the secret key vector through leakage, and then cannot determine whether this vector is orthogonal to the ciphertext vector. It is crucial here that the secret key leakage must occur *before* the attacker sees the ciphertext. This same strategy of adding vectors to the front of the system to allow leakage is also sufficient to yield leakage-resilient HIBE and ABE constructions, as we show in subsequent sections.

Finally, to allow leakage on the master key, we designed the master key to be similar in form to regular secret keys, i.e. we added many new elements and randomness. As we will see, the only knowledge needed to create secret keys is just an integer, denoted by α. However, small leakage from this integer compromises the security of our system[6]. By setting up the master key in the form of the secret keys, we allowed our simulator to create keys using α and use the challenge terms in the extra randomness we added. Essentially, we set up the master key like a secret key of level 0 in an HIBE scheme. Obviously, the above discussion implies that the master key authority should not store α in memory and should only store the master key as we have defined it below. Hence, the authority now stores $n + 3$ elements of the group as the master key.

5.1 Construction

Our dual system IBE scheme consists of the following algorithms:

[6]It can only sustain leakage logarithmic in the security parameter (see Section 4.1).

Setup(1^λ) The setup algorithm generates a bilinear group $\mathbb{G}$ of composite order $N = p_1p_2p_3$, where p_1, p_2, p_3 are three different λ_1,λ_2,λ_3-bit prime numbers respectively[7]. Therefore, for every $i \in \{1, 2, 3\}$ we have that $2^{\lambda_i - 1} \le p_i < 2^{\lambda_i}$. 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 Section 9), while lower values of n will yield a system with fewer group elements in the keys and ciphertexts.

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

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

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

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

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

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

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

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 $r, y_1, \ldots, y_n, \rho_{n+3}, \vec{\rho}$ 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 $\mathbb{G}$.

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

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

[7]The three λ's depend on the security parameter and are chosen appropriately to get a better leakage fraction (see Section 9 for details).

Decrypt(CT, 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:

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

(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:

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

5.2 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:

KeygenSF(MK, X) $\to \widetilde{K}$ This algorithm calls first the normal key generation algorithm **Keygen**(MK, X) to get a normal key MK $= \left(\vec{K^*}, K^*\right)$ or SK $= \vec{K}_1$, depending on X.

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

$$\widetilde{\text{MK}} = \left(\vec{K^*} * g_2^{\vec{\theta}}, K^* g_2^{\theta}\right).$$

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

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

EncryptSF$(M, I) \to \widetilde{\text{CT}}$ This algorithm calls first the normal encryption algorithm **Encrypt**(M, I) to get the ciphertext CT $= \left(C_0, \vec{C}_1\right)$. Then it picks $\vec{\delta} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$ and outputs

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

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 $\widetilde{\text{MK}}$, with parameters $\left(\vec{\theta}, \theta\right)$, to construct a secret key for identity I with **Keygen**, then this will be semi-functional with parameters $\vec{\gamma} = \vec{\theta} + \langle 0, \ldots, 0, -I\theta, 0\rangle$. 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.

As a result, a semi-functional secret key of identity I_k with parameters $\vec{\gamma}$ is nominal with respect to a ciphertext for identity I_c with parameters $\vec{\delta}$ if and only if

$$\vec{\gamma} \cdot \vec{\delta} = 0 \bmod p_2 \quad \text{and} \quad I_k = I_c.$$

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 I with parameters $\vec{\delta}$ if and only if

$$\vec{\delta} \cdot \left(\vec{\theta} + \langle 0, \ldots, 0, -I\theta, 0 \rangle \right) = 0 \bmod p_2.$$

5.3 Continual Leakage

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

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

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

6 Security Proof

We now prove the following theorem:

Theorem 6.1. *For* $(\ell_{\mathrm{MK}} = (n-1-2c)\log(p_2), \ell_{\mathrm{SK}} = (n-1-2c)\log(p_2))$*, where* $c > 0$ *is a fixed positive constant, our dual system IBE scheme is* $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$*-master-leakage secure.*

In order to prove that our system is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-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 3.1 and 3.2, it is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure. We will base each of properties on one of our three complexity assumptions of subsection 2.3.

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

6.1 A Useful Lemma for Leakage Analysis

Our analysis of the leakage resilience of our system will rely on the following lemma from [18], which is proven using the techniques from [9]. Below, we let *dist*(X_1, X_2) denote the statistical distance of two random variables X_1 and X_2.

Lemma 6.2. *Let* $m, \ell, d \in \mathbb{N}$, $m \geq \ell \geq 2d$ *and let* p *be a prime. Let* $X \xleftarrow{\$} \mathbb{Z}_p^{m \times \ell}$*, let* $Y \xleftarrow{\$} \mathbb{Z}_p^{m \times d}$*, and let* $T \xleftarrow{\$} Rk_d\left(\mathbb{Z}_p^{\ell \times d}\right)$*, 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 6.3. *Let $m \in \mathbb{N}$, $m \geq 3$, and let p be a prime. Let $\vec{\delta} \xleftarrow{\$} \mathbb{Z}_p^m$, $\vec{\tau} \xleftarrow{\$} \mathbb{Z}_p^m$, 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 6.2 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 $T \xleftarrow{\$} Rk_1\left(\mathbb{Z}_p^{m-1\times 1}\right)$. We note that X is determined by $\vec{\delta}$, and is distributed as $X \xleftarrow{\$} \mathbb{Z}_p^{m\times m-1}$, since $\vec{\delta}$ is chosen uniformly randomly from $\mathbb{Z}_p^m$. It follows that:

$$dist\left((\vec{\delta}, f(\vec{\tau'})), (\vec{\delta}, f(\vec{\tau}))\right) = dist\left((X, f(X \cdot T)), (X, f(Y))\right) \leq \epsilon.$$

□

This corollary allows us to set $\ell_{\mathrm{MK}} = \ell_{\mathrm{SK}} = (n-1-2c)\log(p_2)$ for our construction (we'll have $n+1 = m$), where c is any fixed positive constant (so that $\epsilon := p_2^{-c}$ is negligible).

6.2 Semi-functional Ciphertext Invariance

Theorem 6.4. *If assumption 2.1 holds, our system has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional ciphertext invariance.*

Proof. We will build a PPT simulator $\mathcal{B}$ that breaks assumption 2.1 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 $\langle \alpha, x_1, x_2, \dots, x_n, a, b\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+3}$. 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 PP $= (N, g_1, g_3, u_1, h_1, e(g_1, g_1)^\alpha, g_1^{x_1}, g_1^{x_2}, \dots, g_1^{x_n})$ to $\mathcal{A}$ where N, g_1 and g_3 are given by the challenger.

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

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

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 I^*. The simulator $\mathcal{B}$ chooses $c \xleftarrow{\$} \{0,1\}$ and outputs the ciphertext:

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

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

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

This implicitly sets $s = z$ and $\vec{\delta} = (\nu x_1, \nu x_2, \ldots, \nu x_n, \nu, \nu(aI^*+b))$. Obviously, s is properly distributed since $z \overset{\$}{\leftarrow} \mathbb{Z}_N$ 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 assumption 2.1 with non-negligible advantage. □

6.3 One Semi-functional Key Invariance

Theorem 6.5. *If assumptions 2.1 and 2.2 hold, our system has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-one semi-functional key invariance.*

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

Lemma 6.6. *If assumptions 2.1 and 2.2 hold, then for any PPT adversary $\mathcal{A}$, $\mathcal{A}$'s advantage in the $\mathsf{MasterLeak}_b$ game 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 either Assumption 2.1 or Assumption 2.2 (same proof as [52]). □

Lemma 6.7. *We suppose the leakage is at most $(\ell_{\mathrm{MK}} = (n-1-2c)\log(p_2), \ell_{\mathrm{SK}} = (n-1-2c)\log(p_2))$, 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 6.3 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 $2^{\ell_{\text{SK}}}$. 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 $\left\langle \vec{\Gamma}, 0 \right\rangle + \langle 0, \ldots, 0, r_1, r_2 \rangle$ in the case of a key which is not a master key, and is defined to be $\left\langle \vec{\Gamma}, 0, 0 \right\rangle + \langle 0, \ldots, 0, r_1, r_2, 0 \rangle + \langle 0, \ldots, 0, \theta \rangle$ 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 $\delta_{n+1}(r_1 - I\theta) + t_2 r_2 \equiv 0 \mod p_2$, where I is the challenge ciphertext identity. If the challenge key is for the identity I, 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 $\left\langle \vec{\delta}, 0 \right\rangle + \langle 0, \ldots, 0, 0, t_2 \rangle$ 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 6.3, since these distributions have a negligible statistical distance for f with this output size. $\square$

To prove Theorem 6.5, we will build a PPT simulator $\mathcal{B}$ that breaks assumption 2.2 with the help of a PPT attacker $\mathcal{A}$ that breaks one semi-functional key invariance of our system. Notice that we included assumption 2.1 in the theorem's premise only for Lemma 6.6; not for the main body of the proof.

$\mathcal{B}$ 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 $\langle \alpha, x_1, x_2, \ldots, x_n, a, b \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+3}$. It computes $u_1 = g_1^a, h_1 = g_1^b$, $e(g_1, g_1)^\alpha$, $g_1^{x_1}$, $g_1^{x_2}$, $\ldots$, and $g_1^{x_n}$. It gives the public parameters $\text{PP} = (N, g_1, g_3, u_1, h_1, e(g_1, g_1)^\alpha, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$ 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 $\langle r, y_1, y_2, \ldots, y_n \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, a random vector $\vec{\rho} = \langle \rho_1, \ldots, \rho_{n+2} \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, and a random element $\rho_{n+3} \xleftarrow{\$} \mathbb{Z}_N$. It stores the following key in the tuple along with lock-value $V = 0$:

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

  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 $\langle r, y_1, y_2, \ldots, y_n \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\vec{\rho} = \langle \rho_1, \ldots, \rho_{n+2} \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, and $\rho_{n+3} \xleftarrow{\$} \mathbb{Z}_N$. Also it picks $\vec{\theta'} \xleftarrow{\$} \mathbb{Z}_N^{n}$ and $\theta' \xleftarrow{\$} \mathbb{Z}_N$ and generates the following key:

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

  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 $\langle r', z_1, z_2, \ldots, z_n \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, and a random vector $\vec{\rho'} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$. It creates the following key:

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

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

$$\widetilde{\mathrm{SK}} = \vec{K_1} = \left\langle g_1^{z_1}, \ldots, g_1^{z_n}, g_1^{\alpha} (u_1^I h_1)^{-r'} \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r'} \right\rangle * (g_2^{\mu} g_3^{\rho})^{\vec{\gamma'}} * g_3^{\vec{\rho'}}$$

  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 $\langle y'_1, y'_2, \ldots, y'_n \rangle \xleftarrow{\$} \mathbb{Z}_N^{n}$, $\vec{\rho} = \langle \rho_1, \ldots, \rho_{n+2} \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, and $\rho_{n+3} \xleftarrow{\$} \mathbb{Z}_N$ and generates:

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

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:

$$\vec{\theta} = \kappa \left\langle y_1', \ldots, y_n', -b - \sum x_i y_i', 1 \right\rangle \quad \text{and} \quad \theta = \kappa a.$$

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 $\langle z_1', z_2', \ldots, z_n' \rangle \xleftarrow{\$} \mathbb{Z}_N^n$, and a random vector $\vec{\rho'} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$. It stores the following key:

$$\mathrm{SK} = \vec{K_1} = \left\langle T^{z_1'}, T^{z_2'}, \ldots, T^{z_n'}, g_1^\alpha T^{-(aI+b)} \prod_{i=1}^{n} T^{-x_i z_i'}, T \right\rangle * g_3^{\vec{\rho'}},$$

  where I 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

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

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

  For the same reasons as before, all vectors seem random in $\mathcal{A}$'s view[8].

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 **EncryptSF**. It gets two messages M_0 and M_1 and the challenge identity I^* from $\mathcal{A}$ and chooses $c \xleftarrow{\$} \{0, 1\}$. Then it generates the following ciphertext:

$$\begin{aligned} \widetilde{CT} &= \left(C_0, \vec{C_1} \right) = \\ &= \left(M_c \cdot e\left((g_1^z g_2^\nu), g_1^\alpha\right), \left\langle (g_1^z g_2^\nu)^{x_1}, \ldots, (g_1^z g_2^\nu)^{x_n}, (g_1^z g_2^\nu), (g_1^z g_2^\nu)^{aI^*+b} \right\rangle \right) \end{aligned}$$

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

$$s = z \quad \text{and} \quad \vec{\delta} = \nu \left\langle x_1, \ldots, x_n, 1, aI^* + b \right\rangle.$$

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 $x_1, \ldots, x_n$ seem random modulo p_2 to the adversary (and ν) as before, but we can not do the same for $aI^* + b$. This happens, because

[8]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.

it might be the case that a, b have been seen modulo p_2 (if the challenge key is the master key) or $aI^* + b$ is seen (if the identity of the challenge key I is equal to I^* modulo p_2 or the identity of the challenge key is the challenge identity). However, lemmas 6.6 and 6.7 assert that the change in any adversary's advantage is negligible.

Lemma 6.6 states that if the simulator $\mathcal{B}$ aborts and guesses a random value for the assumption in case it detects that $I = I^* \bmod p_2$ and $I \neq I^*$ (it can do that with N), the loss in advantage is only a negligible amount. Otherwise, the ciphertext is well-distributed when $I \neq I^*$, because the $aI + b$ in the secret key is uncorrelated to the $aI^* + b$ 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:

$$\vec{\delta} \cdot \left(\vec{\theta} + \langle 0, \ldots, 0, -I^*\theta, 0\rangle\right) = \nu \langle x_1, \ldots, x_n, 1, aI^* + b\rangle \cdot \kappa \left\langle y'_1, \ldots, y'_n, -aI^* - b - \sum x_i y'_i, 1\right\rangle = 0 \bmod p_2.$$

The same happens when the challenge key is a secret key for identity I^*:

$$\vec{\delta} \cdot \vec{\gamma} = \nu \langle x_1, \ldots, x_n, 1, aI^* + b\rangle \cdot \kappa \left\langle z'_1, \ldots, z'_n, -(aI^* + b) - \sum x_i z'_i, 1\right\rangle = 0 \bmod p_2.$$

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 6.7, 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 Assumption 2.2 with non-negligible advantage. □

6.4 Semi-functional Security

Theorem 6.8. *If assumption 2.3 holds, our system has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional security.*

Proof. We will build a PPT simulator $\mathcal{B}$ that breaks assumption 2.3 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 $\langle x_1, x_2, \ldots, x_n, a, b\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+2}$. It computes $u_1 = g_1^a, h_1 = g_1^b$, and $g_1^{x_1}$, $g_1^{x_2}$, …, $g_1^{x_n}$. 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 $\mathrm{PP} = (N, g_1, g_3, u_1, h_1, e(g_1, g_1)^\alpha, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$ to $\mathcal{A}$.

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 $\langle r, y_1, y_2, \ldots, y_n, \rho_{n+3}, \theta'\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+3}$ and two random vectors $\vec{\rho}, \vec{\theta'} \xleftarrow{\$} \mathbb{Z}_N^n$ and constructs:

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

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 $\vec{\theta} = \langle 0, \ldots, 0, \nu, 0\rangle + \vec{\theta'}$ and $\theta = \theta'$.

For all secret keys requested by the adversary on identity I, the simulator creates and stores the following semi-functional keys:

$$SK = \vec{K_1} = \left\langle g_1^{z_1}, \ldots, g_1^{z_n}, (g_1^{\alpha} g_2^{\nu})(u_1^I h_1)^{-r'} \prod_{i=1}^{n} g_1^{-z_i x_i}, g_1^{r'} \right\rangle * g_2^{\vec{\gamma'}} * g_3^{\vec{\rho'}},$$

where the vectors $\vec{\rho'}, \vec{\gamma'} \xleftarrow{\$} \mathbb{Z}_N^n$ and $\langle r', z_1, \ldots, z_n\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$ 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 $\vec{\gamma} = \langle 0, \ldots, 0, \nu, 0\rangle + \vec{\gamma'}$.

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

$$\begin{aligned} CT &= \left(C_0, \vec{C_1}\right) = \\ &= \left(M_c \cdot T, \left\langle (g_1^z g_2^{\mu})^{x_1}, \ldots, (g_1^z g_2^{\mu})^{x_n}, (g_1^z g_2^{\mu}), (g_1^z g_2^{\mu})^{aI^*+b} \right\rangle\right), \end{aligned}$$

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:

$$s = z \quad \text{and} \quad \vec{\delta} = \langle \mu x_1, \ldots, \mu x_n, \mu, \mu(aI^* + b)\rangle$$

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

On the other hand, if $T \xleftarrow{\$} \mathbb{G}_T$, 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 $c \xleftarrow{\$} \{0,1\}$. Thus, $\mathcal{B}$ can use the output of $\mathcal{A}$ to break Assumption 2.3 with non-negligible advantage. □

This concludes the proof of Theorem 6.1.

7 Master-Leakage Secure HIBE Scheme

In a Hierarchical Identity-Based Encryption Scheme (HIBE) (first introduced in [44]), users have vectors of identities which represent their place in a hierarchy. A user has the ability to delegate secret keys to its subordinates on lower levels, and hence can decrypt messages encrypted to these subordinates. The formal definition of a HIBE scheme can be found in Appendix A.

We now present our construction of a HIBE scheme which is resilient to leakage from many secret keys per user and many master keys. Our definition and proof of security can be found in Appendix A. Our construction and proof employ the same techniques used in our IBE system: we modify the HIBE scheme of [52] (which is a composite-order group version of the scheme in [12]) by adjoining n-dimensional vectors to allow leakage. The parameter n can be varied to achieve desired leakage resilience and size of keys/ciphertexts. As in our IBE scheme, nominal semi-functionality will correspond to orthogonality of vectors of exponents in the G_{p_2} components. This will be hidden from the adversary for keys that are incapable of decrypting by pairwise independence, and will be hidden for other keys by the leakage bound. We note that this scheme has ciphertexts which are a constant number of group elements (i.e. independent of the level of the hierarchy).

7.1 Construction

Setup$(1^\lambda) \to (\mathrm{PP}, \mathrm{MK})$ The setup algorithm chooses a bilinear group $\mathbb{G}$ of order $N = p_1p_2p_3$. We will assume that users are associated with vectors of identities whose components are elements of $\mathbb{Z}_N$. If the maximum depth of the HIBE is D, the setup algorithm chooses $D+2$ generators of $\mathbb{G}_1$: $\langle g_1, h, u_1, \ldots, u_D\rangle \xleftarrow{\$} \mathbb{G}_1^{D+2}$ and a generator $g_3 \xleftarrow{\$} \mathbb{G}_3$. It also picks $n+1$ random exponents $\langle \alpha, x_1, x_2, \ldots, x_n\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$ for the public parameters. For the master key, it picks $\langle r, y_1, y_2, \ldots, y_n\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, a random vector $\vec{\rho} = \langle \rho_1, \ldots, \rho_{n+2}\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, and D random elements $\rho_{n+3}, \rho_{n+4}, ..., \rho_{n+D+2} \xleftarrow{\$} \mathbb{Z}_N$. It outputs the following public parameters and master key:

$$\mathrm{PP} = (N, g_1, g_3, h, u_1, u_2, \ldots, u_D, e(g_1,g_1)^\alpha, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$$

$$\mathrm{MK} = \left(\vec{K}^*, E_1^*, \ldots, E_D^*\right) = \left(\left\langle g_1^{y_1}, \ldots, g_1^{y_n}, g_1^\alpha h^{-r} \prod_{i=1}^{n} g_1^{-x_i y_i}, g_1^r \right\rangle * g_3^{\vec{\rho}}, u_1^r g_3^{\rho_{n+3}}, \ldots, u_D^r g_3^{\rho_{n+D+2}}\right)$$

Keygen$(\mathrm{MK}, (I_1, I_2, \ldots, I_j)), \mathrm{PP}$ The key generation algorithm picks $n+1$ random exponents $\langle r', z_1, z_2, \ldots, z_n\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\vec{\rho'} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$ and $\rho'_{n+3}, \ldots, \rho'_{n+2+D-j} \xleftarrow{\$} \mathbb{Z}_N$. It outputs the secret key

$$\begin{aligned}\mathrm{SK} &= \left(\vec{K}_1, E_{j+1}, \ldots, E_D\right) = \\ &= \Bigg(\vec{K}^* * \left\langle g_1^{z_1}, g_1^{z_2}, \ldots, g_1^{z_n}, h^{-r'} \cdot \prod_{i=1}^{j} (E_i^*)^{-I_i} \cdot \left(\prod_{i=1}^{j} u_i^{I_i}\right)^{-r'} \cdot \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r'} \right\rangle * g_3^{\vec{\rho'}}, \\ &\quad E_{j+1}^* u_{j+1}^{r'} g_3^{\rho'_{n+3}}, \ldots, E_D^* u_D^{r'} g_3^{\rho'_{n+2+D-j}}\Bigg)\end{aligned}$$

The terms $g_1^{-x_i z_i}$ are calculated by using the g^{x_i} terms of PP.

Notice that if the second argument of the algorithm is the empty string (i.e. $j = 0$), this algorithm outputs a re-randomized master key.

Delegate$((I_1, I_2, \ldots, I_j), \mathrm{SK}', I_{j+1})$ Given a secret key $\mathrm{SK}' = \left(\vec{K}'_1, E'_{j+1}, \ldots, E'_D\right)$ for identity $(I_1, I_2, \ldots, I_j)$, this algorithm outputs a key for $(I_1, I_2, \ldots, I_{j+1})$. It works similar to **Keygen**: It picks $n+1$ random exponents $\langle r', z_1, z_2, \ldots, z_n\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\vec{\rho'} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, and $\rho'_{n+3}, \ldots, \rho'_{n+1+D-j} \xleftarrow{\$} \mathbb{Z}_N$. It outputs the secret key

$$\begin{aligned}\mathrm{SK} &= \left(\vec{K}_1, E_{j+2}, \ldots, E_D\right) = \\ &= \Bigg(\vec{K}'_1 * \left\langle g_1^{z_1}, g_1^{z_2}, \ldots, g_1^{z_n}, h^{-r'} (E'_{j+1})^{-I_{j+1}} \cdot \left(\prod_{i=1}^{j+1} u_i^{I_i}\right)^{-r'} \cdot \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r'} \right\rangle * g_3^{\vec{\rho'}}, \\ &\quad E'_{j+2} u_{j+2}^{r'} g_3^{\rho'_{n+3}}, \ldots, E'_D u_D^{r'} g_3^{\rho'_{n+1+D-j}}\Bigg)\end{aligned}$$

Encrypt$(M, (I_1, I_2, \ldots, I_j))$ The encryption algorithm chooses $s \xleftarrow{\$} \mathbb{Z}_N$ and outputs the ciphertext:

$$\begin{aligned}\mathrm{CT} &= \left(C_0, \vec{C}_1\right) = \\ &= \left(M \cdot (e(g_1,g_1)^\alpha)^s, \left\langle (g_1^{x_1})^s, \ldots, (g_1^{x_n})^s, g_1^s, (u_1^{I_1} \cdot \ldots \cdot u_j^{I_j} h)^s \right\rangle\right) \in \mathbb{G}_T \times \mathbb{G}^{n+2}\end{aligned}$$

Decrypt(CT, SK) To calculate the blinding factor, one computes $e_{n+2}(\vec{K_1}, \vec{C_1})$. If the encryption and decryption are correct, we get:

$$\begin{aligned} e_{n+2}(\vec{K_1}, \vec{C_1}) &= e(g_1, g_1)^{\alpha s} e(g_1, u_1^{I_1} \cdot \ldots \cdot u_j^{I_j} h)^{-rs} e(g_1, u_1^{I_1} \cdot \ldots \cdot u_j^{I_j} h)^{rs} \cdot \\ &\cdot \prod_{i=1}^{n} e(g_1, g_1)^{-x_i z_i s} \prod_{i=1}^{n} e(g_1, g_1)^{x_i z_i s} \\ &= e(g_1, g_1)^{\alpha s}, \end{aligned}$$

where by r, z_i we mean the combined exponents coming from the master key, the **Keygen**, and possibly the **Delegate** algorithm. (The $\mathbb{G}_3$ parts do not contribute because they are orthogonal to the ciphertext under e.) Notice that in order for the decryption algorithm to work correctly, the identity vector of the ciphertext has to be the same as the identity vector of the secret key. However, if a user's identity is a prefix of this identity vector, he can use the delegate algorithm to get the same identity vector as the ciphertext and then decrypt correctly.

7.2 Semi-functionality

KeygenSF(MK, $(I_1, I_2, \ldots, I_j)$) $\to \widetilde{K}$ To create a semi-functional key, $\widetilde{K}$, this algorithms calls first **Keygen**(MK, $(I_1, I_2, \ldots, I_j)$) $\to K$ (Notice that this can be a master key, if $j = 0$). Then, for K of the form $K = \left(\vec{K_1}, E_{j+1}, \ldots, E_D\right)$, the algorithm picks $\vec{\gamma} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$ and $\theta_{j+1}, \ldots, \theta_D \xleftarrow{\$} \mathbb{Z}_N$. It outputs

$$\widetilde{K} = \left(\vec{K_1} * g_2^{\vec{\gamma}}, E_{j+1} g_2^{\theta_{j+1}}, \ldots, E_D g_2^{\theta_D}\right).$$

EncryptSF$(M, \vec{I}) \to \widetilde{\mathrm{CT}}$ This algorithm first calls the normal encryption algorithm **Encrypt**$(M, \vec{I})$ to get the ciphertext CT $= \left(C_0, \vec{C_1}\right)$. Then it picks $\vec{\delta} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$ and outputs

$$\widetilde{\mathrm{CT}} = \left(C_0, \vec{C_1} * g_2^{\vec{\delta}}\right).$$

It is easy to see that a semi-functional key will correctly decrypt a semi-functional ciphertext (i.e. it is nominal) if and only if $\vec{\gamma} \cdot \vec{\delta} = 0 \bmod p_2$, assuming they have the same identity vectors. If the identity vector of the secret key, say $\vec{I_{\mathrm{SK}}} = \langle I_1, I_2, \ldots, I_k \rangle$, is a prefix of the identity vector of the ciphertext, say $\vec{I_{\mathrm{CT}}} = \vec{I_{\mathrm{SK}}} || \langle I_{k+1}, \ldots, I_j \rangle$, then the user can use the delegate algorithm to get a secret key for identity vector $\vec{I_{\mathrm{CT}}}$. Then the semi-functional parameters will become:

$$\vec{\gamma} + \left\langle 0, \ldots, 0, -\sum_{i=k+1}^{j} \theta_i I_i, 0 \right\rangle.$$

Thus, we say that this key is nominally semi-functional if $\left(\vec{\gamma} + \left\langle 0, \ldots, 0, -\sum_{i=k+1}^{j} \theta_i I_i, 0 \right\rangle\right) \cdot \vec{\delta} = 0 \bmod p_2$. We note that our IBE construction in section 5.1 is just a special case of this HIBE scheme for $D = 1$.

8 Attribute-Based Encryption

Attribute-Based Encryption was first introduced in [61]. In [42], two kinds of ABE systems are defined: Ciphertext-Policy ABE and Key-Policy ABE. In a Ciphertext-Policy Attribute-Based Encryption scheme, ciphertexts are associated with access policies and keys are associated with sets of attributes. A key can decrypt a ciphertext if its set of attributes satisfies the access policy of the ciphertext. We now provide a construction of a Ciphertext-Policy Attribute-Based Encryption scheme which is resilient to leakage from many keys capable of decrypting the challenge ciphertext, including master keys. Our system allows for ciphertext policies which are expressed as LSSS matrices. (For background on these access structures, see Appendix B.) Our construction is a modified version of the adaptively secure (without leakage) construction of [51], which is a composite-order version of the construction in [65]. As for our IBE and HIBE constructions, the modification consists of adjoining vectors of length n to the keys and ciphertexts. The formal definition of CP-ABE, our security definition, and the proof of security for our system are in Appendix B. The parameter n can be varied to achieve desired leakage resilience and ciphertext/key size. We note that these same techniques could be applied to obtain an analogous Key-Policy Attribute-Based Encryption scheme.

8.1 Construction

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

Setup$(1^\lambda, U) \rightarrow (\mathrm{PP}, \mathrm{MK})$**:** The setup algorithm chooses a bilinear group of composite order $N = p_1p_2p_3$, where p_1, p_2, p_3 are three distinct primes.

It picks random exponents $\alpha, a \xleftarrow{\$} \mathbb{Z}_N$, and a random subgroup generator $g_1 \in \mathbb{G}_1$. We note that U denotes the universe of attributes. For each attribute $i \in U$, it chooses random $s_i \xleftarrow{\$} \mathbb{Z}_N$. It also picks n random exponents $x_1, x_2, \ldots, x_n \xleftarrow{\$} \mathbb{Z}_N$ to get the required vectors. For the master key, it picks $t^*, y_1, \ldots, y_n \in \mathbb{Z}_N$ and $\vec{\rho} \xleftarrow{\$} \mathbb{Z}_N^{n+1}, \rho_{n+2} \xleftarrow{\$} \mathbb{Z}_N, \forall i \in U \;\; \rho'_i \xleftarrow{\$} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part.

It outputs the following public parameters and master key:

$$\mathrm{PP} = (N, g_1, g_3, g_1^a, e(g_1, g_1)^\alpha, g_1^{x_1}, \ldots, g_1^{x_n}, \forall i \in U \;\; T_i = g_1^{s_i})$$
$$\mathrm{MK} = \left(U, \vec{K_1^*}, L^*, \forall i \in U \;\; K_i^*\right) =$$
$$= \left(U, \left\langle g_1^{y_1}, \ldots, g_1^{y_n}, g_1^\alpha g_1^{at^*} \prod_{i=1}^{n} g_1^{-x_i y_i} \right\rangle * g_3^{\vec{\rho}}, g_1^{t^*} g_3^{\rho_{n+2}}, \forall i \in U \;\; T_i^{t^*} g_3^{\rho'_i}\right)$$

Notice that $\vec{K_1^*}$ has $n + 1$ elements.

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

$$\mathrm{SK} = \left(S, \vec{K_1}, L, \forall i \in S \;\; K_i\right) =$$
$$= \left(S, \vec{K_1^*} * \left\langle g_1^{z_1}, \ldots, g_1^{z_n}, g_1^{at} \prod_{i=1}^{n} g_1^{-x_i z_i} \right\rangle * g_3^{\vec{\rho}}, L^* g_1^t g_3^{\rho_{n+2}}, \forall i \in S \;\; K_i^* T_i^t g_3^{\rho'_i}\right)$$

In case we want to re-randomize a master key, we use $S = U$ (instead of the empty string as we used for IBE and HIBE).

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

$$\mathrm{CT} = \left((A, \rho), C_0, \vec{C_1}, \forall x \;\; C_x, \forall x \;\; D_x\right) =$$
$$= \left((A, \rho), M \cdot (e(g_1, g_1)^\alpha)^s, \langle (g_1^{x_1})^s, \ldots, (g_1^{x_n})^s, g_1^s\rangle, \forall x \;\; g_1^{aA_x \cdot \vec{v}} T_{\rho(x)}^{-r_x}, \forall x \;\; g_1^{r_x}\right).$$

Decrypt(CT, 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 $\sum_{\rho(x)\in S} \omega_x A_x = \langle 1, 0, \dots, 0\rangle \in \mathbb{Z}_N^{n_2}$. To calculate the blinding factor, it computes:

$$\frac{e_{n+1}(\vec{C_1}, \vec{K_1})}{\prod_{\rho(x)\in S}\left(e(C_x, L)e(D_x, K_{\rho(x)})\right)^{\omega_x}} =$$
$$= \frac{e(g_1,g_1)^{\alpha s} e(g_1,g_1)^{sat} \cdot \prod_{i=1}^{n} e(g_1,g_1)^{-sx_iz_i} \cdot \prod_{i=1}^{n} e(g_1,g_1)^{sx_iz_i}}{\prod_{\rho(x)\in S}\left(e(g_1,g_1)^{atA_x\cdot\vec{v}} e(g_1,g_1)^{-r_x s_{\rho(x)} t} e(g_1,g_1)^{r_x s_{\rho(x)} t}\right)^{\omega_x}} =$$
$$= \frac{e(g_1,g_1)^{\alpha s} e(g_1,g_1)^{sat}}{e(g_1,g_1)^{at(\sum_{\rho(x)\in S}\omega_x A_x)\cdot\vec{v}}} =$$
$$= e(g_1,g_1)^{\alpha s}.$$

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

8.2 Semi-functionality

In this section, we present the algorithms for creating semi-functional ciphertexts and secret keys for our CP-ABE system. 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 **KeygenSF** algorithms. Another difference is that for every attribute $i \in U$, random values $q_i \xleftarrow{\$} \mathbb{Z}_N$ 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:

KeygenSF1(MK, S) $\to \widetilde{K}$ To create a semi-functional key of type 1, this algorithm first calls **Keygen**(MK, S) and gets the key $K = (S, \vec{K_1}, L, \forall i \in S \;\; K_i)$ (Notice that this can be a master key, if $S = U$). Then it picks $\vec{\gamma} \xleftarrow{\$} \mathbb{Z}_N^{n+1}$ and $\theta \xleftarrow{\$} \mathbb{Z}_N$ and outputs

$$\widetilde{K} = \left(S, \vec{K_1} * g_2^{\vec{\gamma}}, Lg_2^{\theta}, \forall i \in S \;\; K_i g_2^{\theta q_i}\right)$$

KeygenSF2(MK, 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

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

EncryptSF($M, (A, \rho)$) $\to \widetilde{\text{CT}}$ This algorithm first calls the normal encryption algorithm **Encrypt**($M, (A, \rho)$) to get the ciphertext CT $= \left((A, \rho), C_0, \vec{C_1}, \forall x \;\; C_x, \forall x \;\; D_x\right)$. Then it picks $\vec{\delta} \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, a random vector $\vec{u} \xleftarrow{\$} \mathbb{Z}_N^{n_2}$ (recall n_2 is the number of columns of A), and for every row A_x of A, it chooses $\delta'_x \xleftarrow{\$} \mathbb{Z}_N$. It outputs

$$\widetilde{\text{CT}} = \left((A, \rho), C_0, \vec{C_1} * g_2^{\vec{\delta}}, \forall x \;\; C_x g_2^{A_x\cdot\vec{u} + \delta'_x q_{\rho(x)}}, \forall x \;\; D_x g_2^{-\delta'_x}\right)$$

Notice the use of $q_{\rho(x)}$, which are the same q's used by the **KeygenSF1** algorithm.

If we use the **Decrypt** algorithm to decrypt a semi-functional ciphertext with a semi-functional key, we get the extra term

$$e(g_2, g_2)^{\vec{\gamma}\cdot\vec{\delta} - \theta u_1},$$

where u_1 denotes the first coordinate of vector $\vec{u}$ picked during **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 $\vec{\gamma}\cdot\vec{\delta} - \theta u_1 = 0 \bmod p_2$.

8.3 Unique Attribute Restriction

In order to prove leakage resilience of the above system using our assumptions, we have to impose an extra requirement on the structure of the LSSS matrices (A, ρ) used during the encryption (for a more elaborate discussion see [52]). We require that in any access structure (A, ρ), no two rows of the matrix A are mapped to the same attribute via ρ. That is, for every two different rows x_1, x_2 of A it is true that $\rho(x_1) \neq \rho(x_2)$. If we wanted to instead allow an attribute to be used up to k times for some fixed k, we could encode each attribute, say B, as k different attributes $B : 1, B : 2, \ldots, B : k$. When a user has this attribute, he gets instead the entire set $\{B : i | i \in [1, k]\}$. In an access matrix, the first occurrence is mapped by ρ to $B : 1$, the second to $B : 2$, and so on.

9 Our Leakage Bound

Our systems allow the same absolute amount of leakage for both the master and the secret keys. That is, $\ell_{\mathrm{MK}} = \ell_{\mathrm{SK}} = (n - 1 - 2c) \log p_2$ 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 $N = p_1 p_2 p_3$ 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:

Scheme	Master Key	Secret Key
IBE	$\frac{n-1-2c}{n+3} \cdot \frac{1}{1+c_1+c_3}$	$\frac{n-1-2c}{n+2} \cdot \frac{1}{1+c_1+c_3}$
HIBE	$\frac{n-1-2c}{n+2+D-i} \cdot \frac{1}{1+c_1+c_3}$ for key of depth i in the hierarchy	
ABE	$\frac{n-1-2c}{n+2+\lvert U\rvert} \cdot \frac{1}{1+c_1+c_3}$	$\frac{n-1-2c}{n+2+\lvert S\rvert} \cdot \frac{1}{1+c_1+c_3}$

Table 1: c, c_1, c_3 are arbitrary positive constants and n is an integer greater than 2. For the HIBE scheme, D is the maximum depth of the hierarchy and i is the depth of the key in question. The master key is considered to be the root of the hierarchy tree and it is of depth 0. The keys at the leaves have depth D. For the ABE scheme, $|U|$ is the total number of attributes in the system, i.e. the size of the universe, and $|S|$ is the number of attributes of the key in question. Notice that in the ABE scheme we ignored the size of the representations of U and S. They are included in the keys, but they are considered public; thus not included in the leakage fraction.

One notable property of our HIBE scheme is that the higher our keys are in the hierarchy, the less leakage is allowed from them. The master key which is at the top allows for a leakage fraction of $(n - 1 - 2c)/((n + 2 + D)(1 + c_1 + c_3))$. This is because the base system we adapted, a HIBE system with short ciphertexts, has keys which contain more group elements for users which are higher in the hierarchy. This feature could be avoided by choosing a different base system.

As one can see, the leakage fraction can be made arbitrarily close to 1 by modifying n, c_1 and c_3 (if we assume a fixed maximum depth for HIBE and a fixed universe size for ABE). As we have noted previously, 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.

References

[1] A. Akavia, S. Goldwasser, and V. Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[2] 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.

[3] J. Alwen, Y. Dodis, and D. Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[4] J. Alwen and L. Ibraimi. Leakage resilient ciphertext-policy attribute-based encryption. manuscript, 2010.

[5] A. Beimel. Secure schemes for secret sharing and key distribution. PhD thesis, Israel Institute of Technology, Technion, Haifa, Israel, 1996.

[6] J. Bethencourt, A. Sahai, and B. Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334, 2007.

[7] E. Biham, Y. Carmeli, and A. Shamir. Bug attacks. In *CRYPTO*, pages 221–240, 2008.

[8] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[9] 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.

[10] D. Boneh and X. Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

[11] D. Boneh and X. Boyen. Secure identity based encryption without random oracles. In *CRYPTO*, pages 443–459, 2004.

[12] D. Boneh, X. Boyen, and E. Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, pages 440–456, 2005.

[13] D. Boneh and D. Brumley. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[14] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In *EUROCRYPT*, pages 37–51, 1997.

[15] D. Boneh and M. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[16] D. Boneh, E. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[17] Z. Brakerski and S. Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability (or: Quadratic residuosity strikes back). In *CRYPTO*, 2010.

[18] Z. Brakerski, Y. T. Kalai, J. Katz, and V. Vaikuntanathan. Cryptography resilient to continual memory leakage. In *FOCS*, 2010.

[19] 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.

[20] R. Canetti, S. Halevi, and J. Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, pages 255–271, 2003.

[21] 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.

[22] L. Cheung and C. Newport. Provably secure ciphertext policy abe. In *ACM Conference on Computer and Communications Security*, pages 456–465, 2007.

[23] 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*, 2010.

[24] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA Int. Conf.*, pages 360–363, 2001.

[25] R. Cramer and V. Shoup. Universal hash proofs and a paradigm for adaptive chosen ciphertext secure public-key encryption. In *EUROCRYPT*, pages 45–64, 2002.

[26] D. Di Crescenzo, R. J. Lipton, and S. Walfish. Perfectly secure password protocols in the bounded retrieval model. In *TCC*, pages 225–244, 2006.

[27] Y. Dodis, K. Haralambiev, A. Lopez-Alt, and D. Wichs. Cryptography against continuous memory attacks. In *FOCS*, 2010.

[28] Y. Dodis, Y. Kalai, and S. Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[29] Y. Dodis and K. Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, 2010.

[30] Y. Dodis, A. Sahai, and A. Smith. On perfect and adaptive security in exposure-resilient cryptography. In *EUROCRYPT*, pages 301–324, 2001.

[31] S. Dziembowski. Intrusion-resilience via the bounded-storage model. In *TCC*, pages 207–224, 2006.

[32] S. Dziembowski and K. Pietrzak. Intrusion-resilient secret sharing. In *FOCS*, pages 227–237, 2007.

[33] S. Dziembowski and K. Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302, 2008.

[34] S. Faust, E. Kiltz, K. Pietrzak, and G. N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[35] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In *CHES*, pages 251–261, 2001.

[36] C. Gentry. Practical identity-based encryption without random oracles. In *EUROCRYPT*, pages 445–464, 2006.

[37] C. Gentry and S. Halevi. Hierarchical identity based encryption with polynomially many levels. In *TCC*, pages 437–456, 2009.

[38] C. Gentry, C. Peikert, and V. Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *Proceedings of the 40th annual ACM Symposium on Theory of Computing*, pages 197–206. ACM, 2008.

[39] C. Gentry and A. Silverberg. Hierarchical id-based cryptography. In *ASIACRYPT*, pages 548–566, 2002.

[40] S. Goldwasser and G. Rothblum. How to play mental solitaire under continuous side-channels: A completeness theorem using secure hardware. In *CRYPTO*, 2010.

[41] V. Goyal, A. Jain, O. Pandey, and A. Sahai. Bounded ciphertext policy attribute based encryption. In *ICALP (2)*, pages 579–591, 2008.

[42] V. Goyal, O. Pandey, A. Sahai, and B. Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM Conference on Computer and Communications Security*, pages 89–98, 2006.

[43] 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.

[44] J. Horwitz and B. Lynn. Toward hierarchical identity-based encryption. In *EUROCRYPT*, pages 466–481, 2002.

[45] Y. Ishai, A. Sahai, and D. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[46] A. Juma and Y. Vahlis. On protecting cryptographic keys against side-channel attacks. In *CRYPTO*, 2010.

[47] J. Kamp and D. Zuckerman. Deterministic extractors for bit-fixing sources and exposure-resilient cryptography. In *FOCS*, pages 92–101, 2003.

[48] J. Katz and V. Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[49] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, pages 104–113, 1996.

[50] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[51] A. 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.

[52] A. Lewko and B. Waters. New techniques for dual system encryption and fully secure hibe with short ciphertexts. In *TCC*, pages 455–479, 2010.

[53] S. Micali and L. Reyzin. Physically observable cryptography. In *TCC*, pages 278–296, 2004.

[54] M. Naor and G. Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[55] P. Q. Nguyen and I. Shparlinski. The insecurity of the digital signature algorithm with partially known nonces. *J. Cryptology*, 15(3):151–176, 2002.

[56] T. Okamoto and K. Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, 2010.

[57] R. Ostrovksy, A. Sahai, and B. Waters. Attribute based encryption with non-monotonic access structures. In *ACM conference on Computer and Communications Security*, pages 195–203, 2007.

[58] 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.

[59] K. Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[60] J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-smart*, pages 200–210, 2001.

[61] A. Sahai and B. Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[62] A. Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[63] 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.

[64] B. Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

[65] B. Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. Cryptology ePrint Archive, Report 2008/290, 2008.

[66] B. Waters. Dual system encryption: Realizing fully secure ibe and hibe under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

Appendices

A Hierarchical Identity Based Encryption

A.1 Preliminaries

Hierarchical Identity-Based Encryption was first introduced in [44] and is a generalization of Identity-Based Encryption. In an HIBE system, identities form a structured hierarchy: a user can delegate keys to its subordinate identities, and hence also decrypt any messages encrypted to its subordinate identities. More specifically, the algorithms use identity vectors $\vec{I} = \langle I_1, I_2 \ldots, I_j \rangle$, and we organize these vectors in a tree-like structure where the immediate children of $\vec{I}$ are all vectors $\vec{I}||I_x$, where $||$ denotes concatenation and I_x is any identity. A user that has a secret key for vector $\vec{I}$ can create secret keys for all users that have identity vectors whose prefix is $\vec{I}$; i.e. in the subtree with root $\vec{I}$.

The maximum number of levels in this tree, denoted by D, is the maximum length of the identity vectors and is called the depth of the hierarchy. The master key is formulated as a secret key of level 0. This means that the identity vector of the master key is the empty vector; thus a prefix of all vectors.

An HIBE scheme consists of the following five PPT algorithms:

Setup $(1^\lambda) \to (\mathrm{PP}, \mathrm{MK})$ The setup algorithm takes in the security parameter λ and outputs the public parameters PP and the master key MK.

Keygen$(\mathrm{MK}, \vec{I}, \mathrm{PP}) \to K$ The key generation algorithm takes in the master key, and identity vector $\vec{I}$, and the public parameters. It outputs a secret key K for the identity vector. (Similar to our previous systems, K can be either a secret key for identity vector $\vec{I}$, or a master key if $\vec{I}$ is empty; $\vec{I} = \langle\rangle$.)

Delegate$(\vec{I}, \mathrm{SK}_{\vec{I}}, I, \mathrm{PP}) \to \mathrm{SK}_{\vec{I}||I}$ The delegation algorithm allows a user that has a secret key for identity vector $\vec{I}$ to construct secret keys down the hierarchy[9]. The algorithm takes in an identity vector $\vec{I}$, a secret key for that identity vector, an identity I, and the public parameters. It outputs a secret key for the identity vector formed by concatenating $\vec{I}$ with I. Notice that this algorithm does not require the master key.

[9]Actually, one level down. To construct key further down the hierarchy, one can repeat the algorithm.

Encrypt$(M, \vec{I}, \mathrm{PP}) \rightarrow \mathrm{CT}$ The encryption algorithm takes in a message M, an identity vector $\vec{I}$, and the public parameters PP. It outputs a ciphertext encrypted to $\vec{I}$.

Decrypt$(\mathrm{CT}, \mathrm{SK}) \rightarrow M$ The decryption takes in a ciphertext for identity vector $\vec{I}$, and a secret key for identity vector $\vec{I}'$. If $\vec{I}'$ is a prefix of $\vec{I}$, the algorithm will output the message M.

For a Dual System Encryption HIBE scheme, the following two algorithms are added:

KeygenSF$(\mathrm{MK}, \vec{I}, \mathrm{PP}) \rightarrow \widetilde{K}$ The semi-functional key generation algorithm takes in the master key, an identity vector $\vec{I}$, and the public parameters. It outputs a semi-functional key for this identity vector. (Note that this algorithm is not required to run in polynomial time.)

EncryptSF$(M, \vec{I}, \mathrm{PP}) \rightarrow \widetilde{\mathrm{CT}}$ The semi-functional encryption algorithm takes in a message M, and identity vector $\vec{I}$, and the public parameters. It outputs a semi-functional ciphertext for this identity vector. (Note that this algorithm is not required to run in polynomial time.)

A.2 Security Definition

Our security game in the HIBE setting is similar to the MasterLeak game in the IBE setting. Instead of single identities, we now have identity vectors and the set $\mathcal{R}$ with revealed keys holds identity vectors of length up to D, where D is the maximum depth of the hierarchy. We let $\mathcal{I}^*$ denote the set of all possible identity vectors. The other difference is that we allow the attacker **Delegate** queries that work similar to **Create** queries, but instead of using the **Keygen** algorithm to create a key, they use the **Delegate** algorithm. The new security game, which is called MasterLeakHibe, goes as follows:

Setup: The challenger makes a call to **Setup**(1^λ) and gets a master key MK and the public parameters PP. It gives PP to the attacker. Also, it sets $\mathcal{R} = \emptyset$ and $\mathcal{T} = \{(0, \langle\rangle, \mathrm{MK}, 0)\}$. Similarly to IBE, we note 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 - identity vectors - keys - leaked bits). Thus initially the set $\mathcal{T}$ holds a record of the master key (empty identity vector 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 any 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).

- **Create**$(h, \vec{I})$: h is a handle to a tuple of $\mathcal{T}$ that has to refer to a master key. The identity vector $\vec{I}$ can be the empty vector $\langle\rangle$. In this case, the attacker asks for the creation of another master key.

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

  Otherwise, the tuple is of the form $(h, \langle\rangle, \mathrm{MK}', L)$. Then the challenger makes a call to **Keygen**$(\mathrm{MK}', \vec{I}) \rightarrow K$ and adds the tuple $\left(H+1, \vec{I}, K, 0\right)$ 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 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 $\left(h, \vec{I}, \mathrm{SK}, L\right)$ or $(h, \langle\rangle, \mathrm{MK}', L)$ [10].

[10] It can be the case that MK′ is the original master key.

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

If the tuple holds a master key MK', it checks if $L + \left|f(\text{MK}')\right| \leq \ell_{\text{MK}}$. If this is true, it responds with $f(\text{MK}')$. Now it updates the L with $L + \left|f(\text{MK}')\right|$. 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, let's say the tuple is $(h, \vec{I}, \text{SK}, L)$. The challenger responds with SK and adds the identity vector $\vec{I}$ to the set $\mathcal{R}$.
- **Delegate**(h, I'): The challenger initially scans $\mathcal{T}$ to find the tuple with handle h. Let's say it is $\left(h, \vec{I}, K, L\right)$. It makes a call to **Delegate**$(\vec{I}, K, I') \to K'$ and adds the tuple $\left(H+1, \vec{I}||I', K', 0\right)$ to the set $\mathcal{T}$. After that, it updates the handle counter to $H \leftarrow H + 1$.

Challenge: The adversary submits a challenge identity vector $\vec{I^*}$ with the restriction that no identity vector in $\mathcal{R}$ is a prefix of it. It also submits two messages M_0, M_1 of equal size. The challenger flips a uniform coin $c \xleftarrow{\$} \{0,1\}$ and encrypts M_c under $\vec{I^*}$ with a call to **Encrypt**$(M_c, \vec{I^*})$. It sends the resulting ciphertext CT^* to the adversary.

Phase 2: This is the same as **Phase 1**, except the only allowed queries are **Create** and **Reveal** queries for secret keys with identity vectors which are not prefixes of $\vec{I^*}$.

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

Definition A.1. A Hierarchical Identity Based Encryption scheme is $(\ell_{\text{MK}}, \ell_{\text{SK}})$-master leakage secure if all PPT adversaries have at most a negligible advantage in the above security game.

A.3 Security properties

To prove security of our system, we will first define properties similar to those in Sections 3.2, 3.3. Namely, we will give versions of semi-functional ciphertext invariance, the one semi-functional key invariance, and semi-functional security adapted to the HIBE setting. However, now our security game has the extra **Delegate** queries, which require one more property of our system, called *delegate invariance*. We define a game the same as the $\mathsf{MasterLeakHibe}$, with the new feature being that all **Delegate**$(\vec{I}, K, I')$ calls are substituted by **Keygen**$(\text{MK}, \vec{I}||I')$ calls, where MK is any master key of the system. The new game is denoted by $\mathsf{MasterLeakHibe}^*$.

Delegation Invariance: We say that a dual system HIBE scheme Π_D has *$(\ell_{\text{MK}}, \ell_{\text{SK}})$- delegate invariance* if for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeakHibe}$ game is negligibly close to the advantage of $\mathcal{A}$ in the $\mathsf{MasterLeakHibe}^*$ game. We denote this by:

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

We note that any HIBE system where the distribution of delegated keys is identical to the distribution of keys produced by fresh calls to **Keygen** automatically has delegation invariance. The remaining properties are defined the same way as in Sections 3.2, 3.3 with $\mathsf{MasterLeakHibe}^*$ instead of $\mathsf{MasterLeak}$.

The $\mathsf{MasterLeakC}$ game is exactly the same as the $\mathsf{MasterLeakHibe}^*$ 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 $\mathbf{KeygenSF}(\mathrm{MK}, \epsilon)$ after calling **Setup**. Thereafter, for all $\mathbf{Create}(h, X)$ queries, the challenger makes an additional call to $\mathbf{KeygenSF}(\mathrm{MK}', X)$, where MK' 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 - identity or empty string - normal key - semi-functional key - leakage - lock. For example, the original master key is stored at the beginning of the game in the tuple $(0, \langle\rangle, \mathrm{MK}, \mathbf{KeygenSF}(\mathrm{MK}, \langle\rangle), 0, -1)$.

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 $b \xleftarrow{\$} \{0, 1\}$ 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 HIBE scheme Π_D has *($\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}}$)-semi-functional ciphertext invariance* if for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in the MasterLeakHibe* game is negligibly close to the advantage of $\mathcal{A}$ in the MasterLeakC game. We denote this by:

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

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

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

One Semi-functional Key Invariance: We say that a dual system HIBE scheme Π_D has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-*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:

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

Semi-functional Security: We say that a dual system HIBE scheme Π_D has *$(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-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:

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

The following theorems are proved the same way as theorems 3.1 and 3.2.

Theorem A.2. *If a dual system encryption HIBE scheme Π_D =(**Setup**, **Keygen**, **Encrypt**, **Decrypt**, **KeygenSF**, **EncryptSF**, **Delegate**) has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$- delegation invariance, $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional ciphertext invariance, $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional key invariance, and $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional security, then Π =(**Setup**, **Keygen**, **Encrypt**, **Decrypt**) is a $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure HIBE scheme.*

Theorem A.3. *If a dual system encryption HIBE scheme Π_D has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-one semi-functional key invariance, then it also has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional key invariance.*

A.4 Proof

We will prove the following theorem for the HIBE system of subsection 7.1:

Theorem A.4. *For $(\ell_{\mathrm{MK}} = (n-1-2c)\log(p_2), \ell_{\mathrm{SK}} = (n-1-2c)\log(p_2))$, where $c > 0$ is any fixed positive constant, our HIBE scheme is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure.*

As in the IBE case, we have to prove that it has semi-functional ciphertext invariance, one semi-functional key invariance, and semi-functional security. For HIBE, we also have to prove delegation invariance. We will base each of the properties on our three complexity assumptions of subsection 2.3. In all proofs, we will treat the master key as a special form of secret keys.

Theorem A.5. *Our system has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-delegation invariance.*

Proof. It is easy to verify that the output of the **Delegate** algorithm for input $(\vec{I}, \mathrm{SK}_{\vec{I}}, I')$ is identically distributed to the output of **Keygen**$(\mathrm{MK}, \vec{I}||I')$. □

Theorem A.6. *If assumption 2.1 holds, our system has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional ciphertext invariance.*

Proof. As in the proof of theorem 6.4, we will build a PPT simulator $\mathcal{B}$ that breaks assumption 2.1 using an attacker $\mathcal{A}$ that breaks the semi-functional ciphertext invariance of the system.

$\mathcal{B}$ receives $D^1 = (N, \mathbb{G}, \mathbb{G}_T, e, g_1, g_3)$ and a challenge term T. Then it plays the $\mathsf{MasterLeak}$ or the $\mathsf{MasterLeakC}$ game with $\mathcal{A}$ in the following way:

Setup phase: $\mathcal{B}$ picks $\langle \alpha, x_1, x_2, \ldots, x_n, a_1, \ldots, a_D, b\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+D+2}$. It computes $u_i = g_1^{a_i}$ for i from 1 to D, $h = g_1^b$, $e(g_1, g_1)^\alpha$, and $g_1^{x_1}$, $g_1^{x_2}$, …, $g_1^{x_n}$. It gives the public parameters PP = $(N$,g_1,g_3,h,u_1,u_2,…,u_D,$e(g_1, g_1)^\alpha$, $g_1^{x_1}$, $g_1^{x_2}$, …, $g_1^{x_n})$ to $\mathcal{A}$ where N, g_1, and g_3 are given by the challenger.

Phase 1: Knowing α, the simulator can generate a master key and execute all secret key queries (create, leaked, keygen) with its master key.

Challenge Phase: The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages M_0 and M_1 and a challenge identity vector $\langle I_1^*, I_2^*, \ldots, I_j^* \rangle$. The simulator $\mathcal{B}$ chooses $c \xleftarrow{\$} \{0,1\}$ and outputs the ciphertext:

$$CT = \left(C_0, \vec{C_1}\right) = \left(M_c \cdot e\left(T, g_1^{\alpha}\right), \left\langle T^{x_1}, T^{x_2}, \ldots, T^{x_n}, T, T^{a_1 I_1^* + a_2 I_2^* + \ldots + a_j I_j^* + b} \right\rangle\right),$$

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

$$\begin{aligned}
&C_0 = M_c \cdot e\left(g_1^z g_2^{\nu}, g_1^{\alpha}\right) = M \cdot e(g_1, g_1)^{\alpha z} \\
&T^{x_i} = (g_1^{x_i})^z g_2^{\nu x_i} && \text{for } i \in \{1, 2, \ldots, n\} \\
&T = g_1^z g_2^{\nu} \\
&T^{b + \sum a_i I_i^*} = (h \prod u_i^{I_i^*})^z g_2^{\nu(b + \sum a_i I_i^*)}
\end{aligned}$$

This implicitly sets $s = z$ and $\vec{\delta} = (\nu x_1, \nu x_2, \ldots, \nu x_n, \nu, \nu(b + \sum a_i I_i^*))$. Obviously, s is properly distributed since $z \xleftarrow{\$} \mathbb{Z}_N$ according to the assumption. The vector $\vec{\delta}$ is properly distributed in the attacker's view because the multiplying factors $(b + \sum a_i I_i^*), x_1, x_2, \ldots, x_n$ are only determined modulo p_1 by the public parameters and not modulo p_2. Thus they are random modulo p_2 in $\mathcal{A}$'s view. 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 $\mathbb{G}_2$ parts. In this case, $\mathcal{B}$ has properly simulated the $\mathsf{MasterLeakHibe}^*$ game. □

Theorem A.7. *If assumptions 2.1 and 2.2 hold, our system has* $(\ell_{\text{MK}}, \ell_{\text{SK}})$*-one semi-functional key invariance.*

Proof. We suppose there exists a PPT attacker $\mathcal{A}$ with non-negligible advantage in breaking one semi-functional key invariance. $\mathcal{B}$ will simulate the game $\mathsf{MasterLeak}_b$ and play it with $\mathcal{A}$ to break assumption 2.2. Initially, $\mathcal{B}$ 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: It picks $\langle \alpha, x_1, x_2, \ldots, x_n, a_1, \ldots, a_D, b \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+D+2}$. It computes $u_i = g_1^{a_i}$ for i from 1 to D, $h = g_1^b$, $e(g_1, g_1)^{\alpha}$, and $g_1^{x_1}$, $g_1^{x_2}$, ..., $g_1^{x_n}$. It gives the public parameters PP $= (N, g_1, g_3, h, u_1, u_2, \ldots, u_D, e(g_1, g_1)^{\alpha}$, $g_1^{x_1}$, $g_1^{x_2}$, ..., $g_1^{x_n})$ to $\mathcal{A}$, where N, g_1, and g_3 are given by the challenger.

Phase 1: In the HIBE setting, we treat the master key simply as a secret key at level 0 of the hierarchy. Thus, the simulator has to answer successfully three different types of queries by $\mathcal{A}$. In this game, all **Delegate** calls have been substituted by **Keygen** calls (which is justified by delegation invariance).

- *$\mathcal{A}$ requested a normal secret key leakage or reveal:* $\mathcal{B}$ picks (once for each key) $n+1$ random exponents $\langle r, z_1, z_2, \ldots, z_n \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\vec{\rho} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, and $\rho_{n+3}, \ldots, \rho_{n+2+D-j} \xleftarrow{\$} \mathbb{Z}_N$. If the identity vector given by $\mathcal{A}$ is $\langle I_1, \ldots, I_j \rangle$, then $\mathcal{B}$ generates the following secret key:

$$\begin{aligned}
\text{SK} &= \left(\vec{K_1}, E_{j+1}, \ldots, E_D\right) = \\
&= \left(\left\langle g_1^{z_1}, g_1^{z_2}, \ldots, g_1^{z_n}, g^{\alpha} h^{-r} \cdot \left(\prod_{i=1}^{j} u_i^{I_i}\right)^{-r} \cdot \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^r \right\rangle * g_3^{\vec{\rho}},\right. \\
&\left. u_{j+1}^r g_3^{\rho_{n+3}}, \ldots, u_D^r g_3^{\rho_{n+2+D-j}}\right)
\end{aligned}$$

It is easy to see that this is a properly distributed secret key. We remind here that if this is a master key, then $j = 0$ and the adversary gives an empty identity vector.

- *$\mathcal{A}$ requested a semi-functional secret key leakage or reveal:* Now $\mathcal{B}$ picks (once for each key) $n+1$ random exponents $\langle r, z_1, z_2, \dots, z_n \rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\vec{\rho} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, $\rho_{n+3}, \dots, \rho_{n+2+D-j} \xleftarrow{\$} \mathbb{Z}_N$, and for the semi-functional parameters, $\vec{\gamma'} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$ and $\theta'_{n+3}, \dots, \theta'_{n+2+D-j} \xleftarrow{\$} \mathbb{Z}_N$. It uses the secret key

$$\begin{aligned}
\mathrm{SK} &= \left(\vec{K}_1, E_{j+1}, \dots, E_D\right) = \\
&= \left(\left\langle g_1^{z_1}, g_1^{z_2}, \dots, g_1^{z_n}, g^{\alpha} h^{-r} \cdot \left(\prod_{i=1}^{j} u_i^{I_i}\right)^{-r} \cdot \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r} \right\rangle * (g_2^{\mu} g_3^{\rho})^{\vec{\gamma'}} * g_3^{\vec{\rho}}, \right. \\
&\quad \left. u_{j+1}^{r} (g_2^{\mu} g_3^{\rho})^{\theta'_{n+3}} g_3^{\rho_{n+3}}, \dots, u_D^{r} (g_2^{\mu} g_3^{\rho})^{\theta'_{n+2+D-j}} g_3^{\rho_{n+2+D-j}} \right)
\end{aligned}$$

  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 $\vec{\gamma} = \mu \vec{\gamma'}$ and $\theta_i = \mu \theta'_i$ for $i \in [n+3, n+2+D-j]$. These are properly distributed as well.

- *$\mathcal{A}$ requested to be challenged on a secret key:* Now $\mathcal{B}$ has to pick a bit b and give leakage to $\mathcal{A}$ either from a normal or from a semi-functional secret key. Instead of doing this, it will use the assumption's challenge term T to generate the secret key. It picks $\langle z'_1, z'_2, \dots, z'_n \rangle \xleftarrow{\$} \mathbb{Z}_N^{n}$, and a random vector $\vec{\rho'} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$. It uses the following key for all subsequent queries on the same key:

$$\begin{aligned}
\mathrm{SK} &= \left(\vec{K}_1, E_{j+1}, \dots, E_D\right) = \\
&= \left(\left\langle T^{z'_1}, T^{z'_2}, \dots, T^{z'_n}, g^{\alpha} T^{-b} \cdot T^{-\sum_{i=1}^{j} a_i I_i} \cdot \prod_{i=1}^{n} T^{-x_i z'_i}, T \right\rangle * g_3^{\vec{\rho}}, \right. \\
&\quad \left. T^{a_{j+1}} g_3^{\rho_{n+3}}, \dots, T^{a_D} g_3^{\rho_{n+2+D-j}} \right)
\end{aligned}$$

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

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

$$\vec{\gamma} = \kappa \left\langle z'_1, \dots, z'_n, -b - \sum_{i=1}^{j} a_i I_i - \sum x_i z'_i, 1 \right\rangle \quad \text{and} \quad \theta_i = \kappa a_i \quad \forall i \in [j+1, D]$$

  Since the public parameters only determine the a's and b modulo p_1, the semi-functional parameters here are random modulo p_2 in $\mathcal{A}$'s view.

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 **EncryptSF**. It gets two messages M_0 and M_1 and a challenge identity vector $\langle I_1^*, I_2^*, \dots, I_j^* \rangle$ from $\mathcal{A}$ and chooses $c \xleftarrow{\$} \{0,1\}$. Then it generates the following ciphertext:

$$\begin{aligned}
\widetilde{CT} &= \left(C_0, \vec{C}_1\right) = \\
&= \left(M_c \cdot e\left((g_1^{z} g_2^{\nu}), g_1^{\alpha}\right), \left\langle (g_1^{z} g_2^{\nu})^{x_1}, \dots, (g_1^{z} g_2^{\nu})^{x_n}, (g_1^{z} g_2^{\nu}), (g_1^{z} g_2^{\nu})^{a_1 I_1^* + \dots + a_j I_j^* + b} \right\rangle\right)
\end{aligned}$$

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

$$s = z \quad \text{and} \quad \vec{\delta} = \nu \left\langle x_1, \dots, x_n, 1, b + \sum_{i=1}^{j} a_i I_i^* \right\rangle$$

The situation is similar to the one encountered in the proof of theorem 6.5: s is properly distributed, but the semi-functional parameters are not (if the challenge key is capable of decrypting this ciphertext). One can see that if the challenge key identity vector is a prefix of the challenge ciphertext identity vector, the secret key is nominal with respect to the ciphertext. In this case (or if the prefix is equal to the challenge identity's prefix modulo p_2), we have that

$$\left(\vec{\gamma} + \left\langle 0, \ldots, 0, -\sum_{i=k+1}^{j} \theta_i I_i, 0 \right\rangle\right) \cdot \vec{\delta} = 0 \bmod p_2$$

By modifying lemmas 6.6 and 6.7 in the following way, we get that the change in the simulator's advantage is only negligible.

Lemma A.8. *If assumptions 2.1 and 2.2 hold, then for any PPT adversary $\mathcal{A}$, $\mathcal{A}$'s advantage in the* $\mathsf{MasterLeak}_b$ *game changes only by a negligible amount if we restrict it to make queries only on the challenge identity vector and on identity vectors such that no component of them is equal to a respective component from the challenge identity vector modulo p_2 and not also equal modulo N.*

Proof. If there exists such an adversary, we can find a non-trivial factor of N. This non-trivial factor can be used to break either Assumption 2.1 or Assumption 2.2. (Same proof as [52]) □

Lemma A.9. *We suppose the leakage is at most ($\ell_{\mathrm{MK}} = (n-1-2c)\log(p_2), \ell_{\mathrm{SK}} = (n-1-2c)\log(p_2)$), 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 an identity vector which is a prefix of the challenge ciphertext identity vector.*

Proof. The proof is similar to the proof of lemma 6.7, but the way we manipulate vectors $\vec{\Gamma}$ and $\vec{\delta}$ is generalized. Throughout this proof we treat the master keys as a special form of secret keys, i.e. the ones with empty identity vector, which we consider to be a prefix of all identity vectors.

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 6.3 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** with its identity vector being a prefix of the challenge ciphertext's identity vector. (If it only did this with negligible probability, then the difference in advantages whenever it gave a prefix 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 $2^{\ell_{\mathrm{SK}}}$ (or $2^{\ell_{\mathrm{MK}}}$). 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 $r_1, r_2, \theta_{k+1}, \ldots, \theta_D \in \mathbb{Z}_{p_2}$. 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

$$\vec{\Gamma'} = \left\langle \vec{\Gamma}, \overbrace{0, \ldots, 0}^{D-k+1} \right\rangle + \left\langle \overbrace{0, \ldots, 0}^{n}, r_1, r_2, \theta_{k+1}, \ldots, \theta_D \right\rangle$$

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 identity for the challenge ciphertext. If the challenge key had an identity vector which was not a prefix of the challenge ciphertext's identity vector, 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. Suppose the challenge key's identity vector is $\langle I_1, I_2, \ldots, I_k\rangle$ and the challenge ciphertext's identity vector is $\langle I_1, \ldots, I_k, \ldots, I_j\rangle$. The challenge key is master if $k = 0$.

$\mathcal{B}$ chooses a random element $t_2 \in \mathbb{Z}_{p_2}$ subject to the constraint $\delta_{n+1}(r_1 - \sum_{i=k+1}^{j} I_i\theta_i) + t_2 r_2 = 0 \bmod p_2$. It then constructs the challenge ciphertext, using $\left\langle \vec{\delta}, 0\right\rangle + \langle 0, \ldots, 0, 0, t_2\rangle$ 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's identity vector is a prefix of the ciphertext's identity vector. 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 6.3, since these distributions have a negligible statistical distance for f with this output size. $\square$

The above lemmas conclude the theorem. $\square$

Theorem A.10. *If assumption 2.3 holds, our system has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional security.*

Proof. We will use a PPT attacker $\mathcal{A}$ with non-negligible advantage in breaking the semi-functional security of our system to build a PPT simulator $\mathcal{B}$ that breaks assumption 2.3 with non-negligible advantage. 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: It picks $\langle x_1, x_2, \ldots, x_n, a_1, \ldots, a_D, b\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+D+1}$. Notice that now α is unknown. It computes $u_i = g_1^{a_i}$ for i from 1 to D, $h = g_1^{b}$, and $g_1^{x_1}$, $g_1^{x_2}$, …, $g_1^{x_n}$. The term $e(g_1, g_1)^{\alpha}$ is computed as $e(g_1^{\alpha} g_2^{\nu}, g_1)$. It gives the public parameters $\mathrm{PP} = (N, g_1, g_3, h, u_1, u_2, \ldots, u_D, e(g_1, g_1)^{\alpha}, g_1^{x_1}, g_1^{x_2}, \ldots, g_1^{x_n})$ to $\mathcal{A}$, where N, g_1, and g_3 are given by the challenger.

Phase 1: For all secret keys requested by the adversary on identity vector $\langle I_1, \ldots, I_j\rangle$, the simulator $\mathcal{B}$ picks (once for each key) $n+1$ random exponents $\langle r, z_1, z_2, \ldots, z_n\rangle \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\vec{\rho} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$, $\rho_{n+3}, \ldots, \rho_{n+2+D-j} \xleftarrow{\$} \mathbb{Z}_N$, and for the semi-functional parameters $\vec{\gamma'} \xleftarrow{\$} \mathbb{Z}_N^{n+2}$ and $\theta'_{n+3}, \ldots, \theta'_{n+2+D-j} \xleftarrow{\$} \mathbb{Z}_N$. It uses the secret key

$$\mathrm{SK} = \left(\vec{K}_1, E_{j+1}, \ldots, E_D\right) =$$
$$= \left(\left\langle g_1^{z_1}, g_1^{z_2}, \ldots, g_1^{z_n}, (g^{\alpha} g_2^{\nu}) h^{-r} \cdot \left(\prod_{i=1}^{j} u_i^{I_i}\right)^{-r} \cdot \prod_{i=1}^{n} g_1^{-x_i z_i}, g_1^{r}\right\rangle * g_2^{\vec{\gamma'}} * g_3^{\vec{\rho}},\right.$$
$$\left. u_{j+1}^{r} g_2^{\theta'_{n+3}} g_3^{\rho_{n+3}}, \ldots, u_D^{r} g_2^{\theta'_{n+2+D-j}} g_3^{\rho_{n+2+D-j}}\right)$$

It is easy to see that the above is a properly distributed semi-functional key with semi-functional parameters $\vec{\gamma} = \langle 0, \ldots, 0, \nu, 0\rangle + \vec{\gamma'}$ and $\theta_i = \mu\theta'_i$ for $i \in [n+3, n+2+D-j]$.

Challenge Phase: The adversary $\mathcal{A}$ gives $\mathcal{B}$ two messages M_0 and M_1 and a challenge identity vector $\langle I_1^*, I_2^*, \ldots, I_j^*\rangle$. The simulator chooses $c \xleftarrow{\$} \{0,1\}$ and outputs the following ciphertext:

$$CT = \left(C_0, \vec{C_1}\right) =$$
$$= \left(M_c \cdot T, \left\langle (g_1^{z} g_2^{\mu})^{x_1}, \ldots, (g_1^{z} g_2^{\mu})^{x_n}, (g_1^{z} g_2^{\mu}), (g_1^{z} g_2^{\mu})^{a_1 I_1^* + \ldots + a_j I_j^* + b}\right\rangle\right),$$

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:

$$s = z \quad \text{and} \quad \vec{\delta} = \left\langle \mu x_1, \ldots, \mu x_n, \mu, \mu(b + \sum_{i=1}^{j} a_i I_i^*) \right\rangle$$

As before, $\vec{\delta}$ is properly distributed since all terms $x_1, \ldots, x_n, b+\sum_{i=1}^{j} a_i I_i^*$ are random modulo p_2. Therefore, $\mathcal{B}$ has properly simulated game $\mathsf{MasterLeakCK}$.

On the other hand, if $T \xleftarrow{\$} \mathbb{G}_T$ 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 $c \xleftarrow{\$} \{0, 1\}$. Hence, $\mathcal{B}$ can use the output of $\mathcal{A}$ to break 2.3 with non-negligible advantage. $\square$

B Attribute Based Encryption

B.1 Background

In this section, we define access structures and linear secret-sharing schemes (LSSS). Then we formally define CP-ABE systems.

B.1.1 Access Structures

In the attribute-based encryption setting, a user has the ability to encrypt a message so that only users that satisfy a specified access structure can decrypt (Ciphertext Policy). In our setting, we will deal with monotone access structures, i.e. negated attributes are not allowed. We note that it is possible to (inefficiently) realize general access structure with our techniques by having the negation of an attribute be a separate attribute (so the total number of attributes will be doubled). Each access structure is defined as a set of subsets of attributes, such that each subset "satisfies" the structure. More formally:

Definition B.1. (Access Structure [5]) Let $\{A_1, \ldots, A_n\}$ be a set of attributes. A collection $\mathbb{A} \subseteq 2^{\{A_1,\ldots,A_n\}}$ is monotone if $\forall B, C$: if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$. An *access structure* (respectively, monotone access structure) is a collection (respectively, monotone collection) $\mathbb{A}$ of non-empty subsets of $\{A_1, \ldots, A_n\}$, i.e., $\mathbb{A} \subseteq 2^{\{A_1,\ldots,A_n\}} \setminus \emptyset$. The sets in $\mathbb{A}$ are called the *authorized sets*, and the sets not in $\mathbb{A}$ are called the *unauthorized sets*.

B.1.2 Linear Secret-Sharing Schemes

The access structures of our system will be linear secret-sharing schemes. We adapt the definition from [5].

Definition B.2. (Linear Secret-Sharing Schemes (LSSS)) A secret sharing scheme over a set of attributes $\mathcal{A}$ is called *linear* (over $\mathbb{Z}_p$) if

1. The shares for each attribute form a vector over $\mathbb{Z}_p$.
2. There exists a matrix A called the share-generating matrix for the scheme. The matrix A has n_1 rows and n_2 columns. For all $i = 1, \ldots, n_1$, the i^{th} row of A is labeled by an attribute $\rho(i)$ (ρ is a function from $\{1, \ldots, n_1\}$ to $\mathcal{A}$). When we consider the column vector $v = (s, r_2, \ldots, r_{n_2})$, where $s \in \mathbb{Z}_p$ is the secret to be shared and $r_2, \ldots, r_{n_2} \in \mathbb{Z}_p$ are randomly chosen, then Av is the vector of n_1 shares of the secret s according to the scheme.

We note the *linear reconstruction* property: we suppose that Π is an LSSS for access structure $\mathbb{A}$. We let S denote an authorized set, and define $I \subseteq \{1, \ldots, n_1\}$ as $I = \{i | \rho(i) \in S\}$. There exist constants $\{\omega_i \in \mathbb{Z}_p\}_{i \in I}$ such that, for any valid shares $\{\lambda\}_i$ of a secret s according to Π, we have: $\sum_{i \in I} \omega_i \lambda_i = s$. These constants $\{\omega_i\}$ can be found in time polynomial in the size of the share-generating matrix A [5].

Boolean Formulas Access structures might also be described in terms of monotonic boolean formulas. LSSS access structures are more general and can be derived from such representations. More precisely, one can use standard techniques to convert any monotonic boolean formula into a corresponding LSSS matrix. We can represent a monotone boolean formula as an access tree, where the interior nodes are AND and OR gates, and the leaf nodes correspond to attributes. The number of rows in the corresponding LSSS matrix will be same as the number of leaf nodes in the access tree.

B.2 CP-ABE

A ciphertext-policy attribute-based encryption system consists of four algorithms: **Setup**, **Encrypt**, **Keygen**, and **Decrypt**.

Setup$(1^\lambda, U) \rightarrow (\mathrm{PP}, \mathrm{MK})$ This algorithms takes in the security parameter λ and a description of the attributes' universe U. It outputs the public parameters and the master key of the system.

Keygen$(\mathrm{MK}, S) \rightarrow \mathrm{SK}$ The key generation algorithm takes in the master key MK, and a set of attributes $S \subseteq U$. It outputs a private key SK. If the set of attributes is the entire universe U, this algorithm outputs a master key.

Encrypt$(M, \mathbb{A}) \rightarrow \mathrm{CT}$ The encryption algorithm takes the message M, and an access structure $\mathbb{A}$ over the universe of attributes. It outputs a ciphertext CT.

Decrypt$(\mathrm{CT}, \mathrm{SK}) \rightarrow M$ The decryption algorithm takes in a ciphertext CT, and a private key SK. If the set of attributes of the private key satisfies the access structure of the ciphertext, it outputs the message M.

For a Dual System Encryption ABE scheme, the following two algorithms are added:

KeygenSF$(\mathrm{MK}, S, \mathrm{PP}) \rightarrow \widetilde{K}$ The semi-functional key generation algorithm takes in the master key, a set of attributes S, and the public parameters. It outputs a semi-functional key for this set of attributes. (Note that this algorithm is not required to run in polynomial time.)

EncryptSF$(M, \mathbb{A}, \mathrm{PP}) \rightarrow \widetilde{\mathrm{CT}}$ The semi-functional encryption algorithm takes in a message M, and an access structure $\mathbb{A}$, and the public parameters. It outputs a semi-functional ciphertext for this access structure. (Note that this algorithm is not required to run in polynomial time.)

B.3 Security Definition

As in the HIBE case, we will define a new security game called MasterLeakAbe, which is a modification of the usual MasterLeak game. The only difference is that now instead of identities we have sets of attributes. Thus, $\mathcal{R} \subseteq 2^U$. The game is the following:

Setup: The challenger makes a call to **Setup**$(1^\lambda, U)$ and gets a master key MK and the public parameters PP. It gives PP to the attacker. Also, it sets $\mathcal{R} = \emptyset$ and $\mathcal{T} = \{(0, U, \mathrm{MK}, 0)\}$. Here, $\mathcal{R} \subseteq 2^U$ and $\mathcal{T} \subseteq \mathcal{H} \times 2^U \times (\mathcal{MK} \cup \mathcal{SK}) \times \mathbb{N}$ (handles - sets of attributes - keys - leaked bits). 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).

- **Create**(h, S): h is a handle to a tuple of $\mathcal{T}$ that has to refer to a master key. The attribute set S can be any subset of the universe U, including U itself. If $S = U$, the attacker asks for the creation 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 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, U, MK', L). Then the challenger makes a call to **Keygen**$(\mathrm{MK}', S) \rightarrow K$ and adds the tuple $(H + 1, S, K, 0)$ 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 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, S, SK, L) or $\big(h, U, \mathrm{MK}', L\big)$ [11].

  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 checks fails, it returns $\perp$ to the adversary.

  If the tuple holds a master key MK', it checks if $L + \big|f(\mathrm{MK}')\big| \leq \ell_{\mathrm{MK}}$. If this is true, and responds with $f(\mathrm{MK}')$ and updates the L with $L + \big|f(\mathrm{MK}')\big|$. 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, let's say the tuple is (h, S, SK, L). The challenger responds with SK and adds the subset 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 $c \xleftarrow{\$} \{0, 1\}$ and encrypts M_c under $\mathbb{A}^*$ with a call to **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 B.3. An Attribute-Based Encryption scheme is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master leakage secure if all PPT adversaries have at most a negligible advantage in the above security game.

B.4 Security properties

To prove security of our system, we will first define properties similar to those in Sections 3.2, 3.3. They are defined exactly the same way with the only difference that instead of MasterLeak we use MasterLeakAbe. Namely, we will give versions of semi-functional ciphertext invariance, the one semi-functional key invariance, and semi-functional security adapted to 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.

[11] It can be the case that MK' is the original master key.

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 $\mathbf{KeygenSF}(\mathrm{MK}, \epsilon)$ after calling **Setup**. Thereafter, for all $\mathbf{Create}(h, X)$ queries, the challenger makes an additional call to $\mathbf{KeygenSF}(\mathrm{MK}', X)$, where MK' 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 $(0, U, \mathrm{MK}, \mathbf{KeygenSF}(\mathrm{MK}, U), 0, -1)$.

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 $b \xleftarrow{\$} \{0, 1\}$ 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 *$(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional ciphertext invariance* if for any probabilistic polynomial time algorithm $\mathcal{A}$, the advantage of $\mathcal{A}$ in the MasterLeakAbe game is negligibly close to the advantage of $\mathcal{A}$ in the MasterLeakC game. We denote this by:

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

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

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

One Semi-functional Key Invariance: We say that a dual system ABE scheme Π_D has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-*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:

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

Semi-functional Security: We say that a dual system ABE scheme Π_D has *$(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-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:

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

The following theorems are proved the same way as theorems 3.1 and 3.2.

Theorem B.4. *If a dual system ABE scheme Π_D =(**Setup, Keygen, Encrypt, Decrypt, KeygenSF, EncryptSF**) has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional ciphertext invariance, $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional key invariance, and $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional security, then Π =(**Setup, Keygen, Encrypt, Decrypt**) is a $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure ABE scheme.*

Theorem B.5. *If a dual system ABE scheme Π_D has* $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-one semi-functional key invariance*, then it also has* $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional key invariance*.*

B.5 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 B.6. *If assumption 2.1 holds, our system has $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-semi-functional ciphertext invariance.*

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 2.1 with non-negligible advantage. The simulator $\mathcal{B}$ plays the $\mathsf{MasterLeakAbe}$ or the $\mathsf{MasterLeakC}$ game with the attacker $\mathcal{A}$ in the following way:

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

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

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^*, ρ^*). The simulator $\mathcal{B}$ chooses random values $v'_2, \ldots, v'_{n_2} \xleftarrow{\$} \mathbb{Z}_N$ and for each row A^*_x of A^* one value $r_x \xleftarrow{\$} \mathbb{Z}_N$. Using the v' values, it creates the vector $\vec{v'} = \langle 1, v'_2, \ldots, v'_{n_2} \rangle$. It flips a random coin $c \xleftarrow{\$} \{0,1\}$ and outputs the ciphertext:

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

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

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

For the $\mathbb{G}_1$ part, this implicitly sets $s = z$, $\vec{v} = z\vec{v'}$ and $r_x = zr'_x$. 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 $\vec{\delta} = \nu \langle x_1, \ldots, x_n, 1\rangle$, $\vec{u} = a\nu\vec{v'}$, $\delta'_x = -\nu r'_x$, 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: $x_1, x_2, \ldots, x_n, \nu, a, v'_2, \ldots, v'_{n_2}, r'_x, s_{\rho^*(x)}$. So this is a properly distributed semi-functional ciphertext, and $\mathcal{B}$ has properly simulated the 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 MasterLeakAbe game. □

Theorem B.7. *If assumption 2.2 holds, the difference between the advantages of any PPT attacker when playing the* $\mathsf{MasterLeak}_0$ *and* $\mathsf{MasterLeak}_{1/2}$ *games with leakage* $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$ *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 2.2 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 $\alpha, a \xleftarrow{\$} \mathbb{Z}_N$ and for each attribute $i \in U$, it chooses random $s_i \xleftarrow{\$} \mathbb{Z}_N$. It also picks n random exponents $x_1, x_2, \ldots, x_n \xleftarrow{\$} \mathbb{Z}_N$. It gives the public parameters

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

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 $\vec{\rho} \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\rho_{n+2} \xleftarrow{\$} \mathbb{Z}_N, \forall i \in S \;\; \rho'_i \xleftarrow{\$} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part. It uses the following secret key:

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

where 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 $\vec{\rho} \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\rho_{n+2} \xleftarrow{\$} \mathbb{Z}_N, \forall i \in S \;\; \rho'_i \xleftarrow{\$} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part. It uses the secret key:

$$\begin{aligned}\mathrm{SK} &= \left(S, \vec{K_1}, L, \forall i \in S \;\; K_i\right) = \\ &= \left(S, \left\langle T^{z'_1}, \ldots, T^{z'_n}, g_1^{\alpha} T^{a} \prod_{i=1}^{n} T^{-x_i z'_i} \right\rangle * g_3^{\vec{\rho}}, T g_3^{\rho_{n+2}}, \forall i \in S \;\; T^{s_i} g_3^{\rho'_i}\right)\end{aligned}$$

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:

$$\vec{\gamma} = \kappa \left\langle z'_1, \ldots, z'_n, a - \sum x_i z'_i \right\rangle \quad , \quad \theta = \kappa \;\; \text{and} \;\; q_i = s_i.$$

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^*, ρ^*). The simulator $\mathcal{B}$ chooses random values $v'_2, \ldots, v'_{n_2} \xleftarrow{\$} \mathbb{Z}_N$ and for each row A^*_x of A^* and one value $r'_x \xleftarrow{\$} \mathbb{Z}_N$. Using the v' values, it creates the vector $\vec{v'} = \left\langle 1, v'_2, \ldots, v'_{n_2} \right\rangle$. It flips a random coin $c \xleftarrow{\$} \{0, 1\}$ and outputs the ciphertext:

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

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

The ciphertext is semi-functional since

$$\begin{aligned}
&C_0 = M_c \cdot e\left(g_1^z g_2^{\nu}, g_1^{\alpha}\right) = M \cdot e(g_1, g_1)^{\alpha z} && \\
&(g_1^z g_2^{\nu})^{x_i} = (g_1^{x_i})^z g_2^{\nu x_i} && \text{for } i \in \{1, 2, \ldots, n\} \\
&(g_1^z g_2^{\nu}) = g_1^z g_2^{\nu} && \\
&(g_1^z g_2^{\nu})^{a A^*_x \cdot \vec{v'}} (g_1^z g_2^{\nu})^{-r'_x s_{\rho^*(x)}} = g_1^{a A^*_x \cdot z \vec{v'}} T_{\rho^*(x)}^{-z r'_x} \cdot g_2^{A^*_x \cdot a \nu \vec{v'} - \nu r'_x s_{\rho^*(x)}} && \text{for every row } x \text{ of } A^* \\
&(g_1^z g_2^{\nu})^{r'_x} = g_1^{z r'_x} \cdot g_2^{\nu r'_x} && \text{for every row } x \text{ of } A^*
\end{aligned}$$

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 $\vec{\delta} = \nu \langle x_1, \dots, x_n, 1\rangle$, $\vec{u} = a\nu\vec{v'}$, $\delta'_x = -\nu r'_x$ and $q_{\rho*(x)} = s_{\rho*(x)}$.

First, notice that $q_{\rho*(x)} = s_{\rho*(x)}$ 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 **KeygenSF1** and **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 in the following table:

Secret key

$$\vec{\gamma} = \kappa \langle z'_1, \dots, z'_n, a - \sum x_i z'_i\rangle \quad \theta = \kappa$$

Ciphertext

$$\vec{\delta} = \nu \langle x_1, \dots, x_n, 1\rangle \quad \vec{u} = a\nu \left\langle 1, v'_2, \dots, v'_{n_2}\right\rangle \quad \delta_x = -\nu r'_x$$

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:

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

According to the rules of the game, this key can not be revealed to the adversary[12], but only leakage is allowed on it. Then we can modify lemma 6.7 and show that no PPT attacker can have more than negligible advantage in distinguishing these two keys. The modified lemma is the following:

Lemma B.8. *We suppose the leakage is at most* $(\ell_{\text{MK}} = (n-1-2c)\log(p_2), \ell_{\text{SK}} = (n-1-2c)\log(p_2))$, *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. The proof is similar to the proof of lemma 6.7, but the way we manipulate vectors $\vec{\Gamma}$ and $\vec{\delta}$ is different. 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 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 6.3 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** 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 $2^{\ell_{\text{MK}}}$). It can do this by fixing the values of all other keys and fixing all other variables involved in the

[12]This situation is similar to IBE where the adversary can not get an entire key for I^* or the master key.

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

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

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 $\delta_{n+1} r_1 - t_2 r_2 = 0 \bmod p_2$. 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 6.3, since these distributions have a negligible statistical distance for f with this output size. □

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. Remember that in the IBE case, we used the fact that the identities the attacker asks for are not equal to the challenge identity modulo p_2. This gave us the extra random term. 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 $\langle 1, 0, \ldots, 0 \rangle$. Otherwise, one could find ω_x's such that $\sum_{\rho^*(x) \in S} \omega_x A_x^* = \langle 1, 0, \ldots, 0 \rangle$ and decrypt. This means that there is a vector $\vec{w}$ that is in the orthogonal complement of R, but not orthogonal to $\langle 1, 0, \ldots, 0 \rangle$. 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

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

Since $\vec{u'}$ reveals no information about f and $\vec{w}$ is not orthogonal to $\langle 1, 0, \ldots, 0 \rangle$, 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)

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

However, not all of these are affected by the value of f. More specifically, we know that the rows for which the attribute $\rho^*(x)$ is in 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 $\delta'_x q_{\rho^*(x)}$. Here is where we use the restriction that each attribute in the access structure appears only once: Each of these $q_{\rho^*(x)}$ 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 [51].)

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 assumption 2.2 with non-negligible advantage. □

Theorem B.9. *If assumption 2.2 holds, the difference between the advantages of any PPT attacker when playing the* $\mathsf{MasterLeak}_{1/2}$ *and* $\mathsf{MasterLeak}_1$ *games with leakage* $(\ell_{\text{MK}}, \ell_{\text{SK}})$ *on our ABE system is negligible in* λ.

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 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 $\vec{h} \xleftarrow{\$} \mathbb{Z}_N^{n+1}$ and generates the challenge key as:

$$\begin{aligned}\text{SK} &= \left(S, \vec{K_1}, L, \forall i \in S \;\; K_i\right) = \\ &= \left(S, \left\langle T^{z'_1}, \ldots, T^{z'_n}, g_1^{\alpha} T^{a} \prod_{i=1}^{n} T^{-x_i z'_i} \right\rangle * (g_2^{\mu} g_3^{\rho})^{\vec{h}} * g_3^{\vec{\rho}}, T g_3^{\rho_{n+2}}, \forall i \in S \;\; T^{s_i} g_3^{\rho'_i} \right)\end{aligned}$$

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:

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

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). □

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

Theorem B.10. *If assumption 2.2 holds*[13]*, our system has* $(\ell_{\text{MK}}, \ell_{\text{SK}})$*-one semi-functional key invariance.*

Proof. By theorems B.7 and B.9, we have that for any PPT adversary $\mathcal{A}$

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

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

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

Theorem B.11. *If assumption 2.3 holds, our system has* $(\ell_{\text{MK}}, \ell_{\text{SK}})$*-semi-functional security.*

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 assumption 2.3 with non-negligible advantage. As always, $\mathcal{B}$ that plays either the $\mathsf{MasterLeakCK}$ 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.

[13]Notice that in this case we don't need assumption 2.1 - we have no identities.

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

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

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 $\vec{h} \xleftarrow{\$} \mathbb{Z}_N^{n+1}$ for the $\mathbb{G}_2$ part and $\vec{\rho} \xleftarrow{\$} \mathbb{Z}_N^{n+1}$, $\rho_{n+2} \xleftarrow{\$} \mathbb{Z}_N, \forall i \in S \;\; \rho'_i \xleftarrow{\$} \mathbb{Z}_N$ for the $\mathbb{G}_3$ part. It creates the following secret key:

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

where 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 $\vec{\gamma} = \vec{h} + \langle 0, \ldots, 0, \nu \rangle$.

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^*, ρ^*). The simulator $\mathcal{B}$ chooses random values $v'_2, \ldots, v'_{n_2} \xleftarrow{\$} \mathbb{Z}_N$ and for each row A^*_x of A^*, one value $r'_x \xleftarrow{\$} \mathbb{Z}_N$. Using the v' values, it creates the vector $\vec{v'} = \left\langle 1, v'_2, \ldots, v'_{n_2} \right\rangle$. It flips a random coin $c \xleftarrow{\$} \{0, 1\}$ and outputs the ciphertext:

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

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

The ciphertext is semi-functional since

$$\begin{aligned}&(g_1^z g_2^\mu)^{x_i} = (g_1^{x_i})^z g_2^{\mu x_i} && \text{for } i \in \{1, 2, \ldots, n\} \\ &(g_1^z g_2^\mu)^{a A^*_x \cdot \vec{v'}} (g_1^z g_2^\mu)^{-r'_x s_{\rho^*(x)}} = g_1^{a A^*_x \cdot z \vec{v'}} T_{\rho^*(x)}^{-z r'_x} \cdot g_2^{A^*_x \cdot a \mu \vec{v'} - \mu r'_x s_{\rho^*(x)}} && \text{for every row } x \text{ of } A^* \\ &(g_1^z g_2^\mu)^{r'_x} = g_1^{z r'_x} \cdot g_2^{\mu r'_x} && \text{for every row } x \text{ of } A^*\end{aligned}$$

For the $\mathbb{G}_1$ parts, this implicitly sets $s = z$, $\vec{v} = z\vec{v'}$, and $r_x = z r'_x$. 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 $\vec{\delta} = \mu \langle x_1, \ldots, x_n, 1 \rangle$, $\vec{u} = a\mu\vec{v'}$, $\delta'_x = -\mu r'_x$ and $q_{\rho*(x)} = s_{\rho*(x)}$. These are properly distributed modulo p_2 because the terms $x_1, \ldots, x_n, \mu, a, v'_2, \ldots, v'_{n_2}, r'_x, s_{\rho^*(x)}$ 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. □

By theorems B.6, B.10, B.11 we get that:

Theorem B.12. *For $(\ell_{\mathrm{MK}} = (n - 1 - 2c)\log(p_2), \ell_{\mathrm{SK}} = (n - 1 - 2c)\log(p_2))$, where $c > 0$ is a fixed positive constant, our dual system encryption ABE scheme is $(\ell_{\mathrm{MK}}, \ell_{\mathrm{SK}})$-master-leakage secure.*