Multi-Authority Attribute-Based Encryption from LWE in the OT Model

Sam Kim
Stanford University
skim13@cs.stanford.edu

Abstract

In a (ciphertext policy) attribute-based encryption (ABE) scheme, a ciphertext is associated with a predicate ϕ and a secret key is associated with a string x such that a key decrypts a ciphertext if and only of $\phi(x) = 1$. Moreover, the scheme should be collusion-resistant meaning that no colluding set of users can learn about the message if none of their secret keys can individually decrypt the ciphertext. Traditionally, in an ABE scheme, there exists a central authority that generates the keys for each users. In a multi-authority attribute-based encryption (MA-ABE) scheme, individual components of the secret keys are generated by different key-generating authorities.

Although the notion of MA-ABE is a natural extension of the standard ABE, its realization has so far been limited. Indeed, all existing MA-ABE constructions rely solely on bilinear maps and can only support predicates that are computable by monotone boolean formulas. In this work, we construct the first collusion-resistant MA-ABE scheme that can support circuit predicates from the Learning with Errors (LWE) assumption. Our construction works in a new model that we call the *OT model*, which can be viewed as a direct relaxation of the traditional GID model that previous MA-ABE constructions consider. We believe that the new OT model is a compelling alternative to the traditional GID model as it captures the core requirements for an MA-ABE scheme. The techniques that are used to construct MA-ABE in this model can also be used as a stepping stone towards constructing MA-ABE in the stronger GID model in the future.

1 Introduction

A (ciphertext-policy) attribute-based encryption scheme [SW05, GPSW06] is an advanced form of public-key encryption where a ciphertext for a message μ is bound under a policy predicate $\phi : \{0,1\}^\ell \to \{0,1\}$, and a decryption key sk_x is associated with an attribute string $x \in \{0,1\}^\ell$. A key sk_x can be used to decrypt a ciphertext ct_ϕ if and only if $\phi(x) = 1$. The security requirement of an attribute-based encryption scheme states that no colluding set of users can learn about the plaintext μ if none of their keys are authorized to decrypt the ciphertext.

Since its introduction, attribute-based encryption (ABE) has become a well-studied notion in cryptography due to the many advantages that it provides for enforcing complex access control of encrypted data. Even beyond applications, ABE is an important theoretical object of study as it serves as a natural "middle-ground" between the two popular notions of identity-based encryption [Sha84, BF01, Coc01] and general-purpose functional encryption [BSW11]. As a result, significant amount of effort has been put into constructing ABE in the past two decades. By now,

we know how to construct ABE schemes from a variety of standard cryptographic assumptions. In particular, we can construct ABE schemes that support predicates computable by boolean formulas from standard assumptions on bilinear groups [GPSW06, LOS+10, OT10, Wat12, HW13] and schemes that support predicates computable by bounded depth circuits using the Learning with Errors (LWE) assumption [Boy13, GVW13, BGG+14, BV14].

Multi-authority attribute-based encryption. Although the traditional notion of attribute-based encryption is useful for both theory and practice, it suffers from a fundamental limitation: the decryption keys sk_x must be issued by a single central authority who holds a master secret key msk. The need for a central authority (CA) can be problematic for several reasons. As the CA can generate a decryption key for any attribute string $x \in \{0,1\}^\ell$, it can decrypt all ciphertexts that are generated in the system, forcing complete trust in the CA. Furthermore, in many practical scenarios, each of the bits of an attribute string $x \in \{0,1\}^\ell$ are naturally associated with separate authorizing parties in the real-world. For example, a predicate that one might consider in an application is as follows:

$$\phi(x_{\texttt{[PhD]}}, x_{\texttt{[Work Experience]}}) = x_{\texttt{[PhD]}} \vee x_{\texttt{[Work Experience]}}.$$

The predicate ϕ allows decryption of a ciphertext if and only if a decryptor either holds a PhD or has some years of work experience. In this situation, instead of having a single trusted authority who generates the entire ABE key, it is more natural for a university to issue a key component $\mathsf{sk}_{\texttt{[PhD]}}$ and for a company to issue a key component $\mathsf{sk}_{\texttt{[Work Experience]}}$. In the case where the attribute bits $x_{\texttt{[Phd]}}, x_{\texttt{[Work Experience]}}$ are sensitive information, it also makes sense for the university to not learn any information about a user's work experience, and an employer to not learn any information about a user's education. Motivated by these type of situations, Sahai and Waters [SW05] proposed the problem of constructing ABE with multiple authorities.

In theory, an ABE scheme can be decentralized generically using secure multiparty computation (MPC) techniques. For instance, the university and the employer can hold a secret share of a standard ABE master secret key msk. Whenever a user requests a key, the parties can participate in an MPC protocol that securely combines the shares to msk and runs the ABE key generation algorithm for the user. This solution, however, is not viable for many practical situations as all key generating authorities must interact in an MPC protocol for *each* user in the whole system.

To solve this problem, Chase [Cha07] as well as Lewko and Waters [LW11] proposed a natural extension to the standard notion of ABE called *multi-authority attribute-based encryption* (MA-ABE). In an MA-ABE scheme, the master secret key of a standard ABE scheme msk can be further divided into multiple master secret keys $\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell$ that are assigned to separate key-generating authorities. An authority who holds msk_j can generate a decryption key component $\mathsf{sk}_{j,x}$ for a single attribute bit $x_j \in \{0,1\}$ *without* interacting with any other key-generating authorities. The collection of these individual key components that are generated by each authorities $\mathsf{sk}_x = (\mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell})$ make up a single decryption key for the attribute string $x \in \{0,1\}^\ell$. The key sk_x can be used to decrypt a ciphertext ct_ϕ if and only if $\phi(x) = 1$. The key security requirement of an MA-ABE scheme states that a ciphertext ct_ϕ remains secure even when some of the key-generating authorities are corrupt and therefore, collude with the adversary.

Over the years, there have been great progress in constructing MA-ABE from standard cryptographic assumptions [Cha07, CC09, LW11]. In particular, the state of the art construction of Lewko and Waters [LW11] provide an MA-ABE scheme that satisfies a very strong form of functionality and security. However, all of these previous work on MA-ABE have so far been limited to the setting of bilinear groups and are restricted to support only predicates that are computable by monotone

boolean formulas. Constructing an MA-ABE scheme for circuits from lattices that satisfies any reasonable notion of functionality and security has so far been a challenging task. Very recently, some progress was made by Wang et al. [WFL19]; however, their construction provides security only against bounded collusion of parties.

1.1 Our Contributions

In this work, we propose a new multi-authority attribute-based encryption scheme from the LWE problem. Our construction supports predicates that are computable by *bounded depth circuits* and works in a new model called the *oblivious transfer (OT) model*, which can be viewed as a relaxation of the traditional GID model that previous MA-ABE constructions consider. We elaborate on the new OT model below and provide the high level ideas of our construction in Section 2.

Traditional GID model. In an MA-ABE scheme, each of the decryption key components $\mathsf{sk}_x = (\mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell})$ must be generated independently by the ℓ authorities who each hold master secret keys $\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell$. Since the model does not allow any communication among the key-generating authorities to generate these key components, there is no way for the authorities to even agree on the specific recipient of a decryption key. Therefore, an MA-ABE scheme is traditionally defined in the *global identifier* (GID) model [Cha07, LW11] where each recipient of a decryption key is identified by some *global identity string* $\mathtt{gid} \in \{0,1\}^*$. For example, $\mathtt{gid}$ can consist of a user's driver's license, student ID, or voter registration number that the users provide to each key-generating authorities. To generate a key, each authority with master secret key msk_j takes in as input a recipient's identity string $\mathtt{gid}$ and produces a decryption key component $\mathsf{sk}_{\mathtt{gid},j,x_j}$ that is bound under $\mathtt{gid}$ and an attribute bit $x_j \in \{0,1\}$. The complete key $\mathsf{sk}_{\mathtt{gid},x} = (\mathsf{sk}_{\mathtt{gid},1,x_1}, \ldots, \mathsf{sk}_{\mathtt{gid},\ell,x_\ell})$ for a *single* $\mathtt{gid}$ can then be used to decrypt a ciphertext ct_ϕ if and only if $\phi(x) = 1$.[1]

OT model. The OT model is similar to the GID model where individual key components are bound under some string $\mathtt{gid} \in \{0,1\}^*$. However, in contrast to the GID model where a $\mathtt{gid}$ can be an *arbitrary* string, we allow a $\mathtt{gid}$ to be a *structured* string that is generated by the key-recipients as a form of a *key-request*. Specifically, to receive a key sk_x for an attribute string $x \in \{0,1\}^\ell$, each key-recipient generates a formal key-request $\mathsf{req}_x \in \{0,1\}^*$ to provide to each of the key-generating authorities. Upon request for a decryption key req_x, each authority with msk_j generates a key component $\mathsf{sk}_{\mathsf{req},j,x_j}$ to provide to the corresponding recipient. The components that are generated for the same request string req can be combined to form a complete decryption key $\mathsf{sk}_{\mathsf{req},x} = (\mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell})$. Just like before, the key $\mathsf{sk}_{\mathsf{req},x}$ can decrypt a ciphertext ct_ϕ if and only if $\phi(x) = 1$. We note that in the OT model, the key-generating authorities still generate the key components independently without the need to communicate with each other. In fact, if we define a user's request string $\mathsf{req}_x \in \{0,1\}^*$ to simply be the global identity string $\mathtt{gid} \in \{0,1\}^*$, then this is equivalent to the GID model. In this respect, the OT model can be viewed as a generalization/relaxation of the GID model.

We note that one of the motivations for a multi-authority ABE scheme is to attain *privacy* of attributes. Namely, a user with an attribute bit $x_j \in \{0,1\}$ for an index $j \in [\ell]$ must not be forced to reveal x_j to any other key-generating authority other than the authority who holds msk_j. In the GID model, the key-generating authorities only require a global identity string $\mathtt{gid}$ of the user to generate the key components. Therefore, privacy is achieved innately. In the OT model,

[1] In the technical sections, we revert this convention such that decryption is allowed if and only if $\phi(x) = 0$ as is commonly done in lattice-based ABE papers.

however, a user submits a formal request string req_x that may reveal the entire attribute string $x \in \{0,1\}^\ell$. Therefore, in addition to the standard ciphertext security condition for an MA-ABE scheme, we define a *receiver privacy* requirement for the OT model. We refer to this model as the OT model as both the syntax of the key generation procedure as well as the security requirements (ciphertext security and receiver privacy) for the MA-ABE scheme largely resembles those of an *oblivious transfer protocol*. In fact, our construction of MA-ABE relies on an oblivious transfer protocol as a fundamental building block. We provide the formal definitions of MA-ABE in the OT model in Section 5.

Decomposable Attribute-based Encryption. The main intermediate notion that we introduce to construct our MA-ABE schemes is called *decomposable attribute-based encryption* (DABE), which can be viewed as a standard ABE scheme with an additional decomposability property on the decryption keys. The syntax of the algorithms for a decomposable ABE scheme largely remains identical to that of a standard ABE scheme. In particular, the master secret key msk still remains a single object that is needed to generate the entire decryption key sk_x. The only additional requirement that we make on a DABE scheme is that the decryption keys sk_x are *decomposable* into multiple components $\mathsf{sk}_0, \mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell}$. The components sk_{x_j} for $j \in [\ell]$ are part of the key sk_x that correspond to each bit of the attribute string $x_j \in \{0,1\}$. The component sk_0 is a part of the key that is independent of the attribute $x \in \{0,1\}^\ell$ and its only functionality is to bind the attribute components $\mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell}$ together. We refer to the key component sk_0 as the *binding component* of the key and the key components $\mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell}$ as the *attribute components*. We provide the precise definitions in Section 6.

To construct an MA-ABE scheme, we first show that a DABE scheme can be upgraded to an MA-ABE scheme using a fully-homomorphic encryption (FHE) scheme, a key-homomorphic PRF, and an oblivious transfer protocol. At a conceptual level, the main additional property that an MA-ABE scheme must provide over a DABE scheme is the decomposability of master secret keys. To achieve this decomposability of master secret keys in a DABE scheme, we use the ideas in the recent work of Boneh et al. [BGG+18]. Specifically, we define the setup algorithm to encrypt msk using FHE and include the FHE ciphertext as part of the public parameters. Each key-generating authorities are then provided a share of the FHE decryption key that allows threshold decryption on any FHE ciphertexts. With these shares, the key-generating authorities can generate the key components of a DABE decryption key in a decentralized way. Namely, each authority can homomorphically evaluate the DABE key generation algorithm on the FHE ciphertext and provide its partial decryption using the FHE key share. There are a number of technical difficulties that must be overcome to make this approach work and we provide a detailed overview in Section 2. We provide the full construction in Section 6.2.

Finally, we construct DABE from the LWE problem. Our construction is a modification of the standard ABE scheme of Boneh et al. [BGG+14]. We provide a detailed overview in Section 2 and the full construction in Section 7.

1.2 Comparison to Lewko-Waters Construction

In addition to the fact that our MA-ABE construction works in the OT model (as opposed to the GID model), there are two additional desirable properties of the state-of-the-art Lewko-Waters construction [LW11] that our MA-ABE construction do not satisfy. We explain these two limitations of our construction in the following:

1. **Static vs. dynamic set of authorities**: In the Lewko-Waters construction, even the setup algorithm is decentralized. Namely, each key-generating authority generates its own master secret key msk_j *independently* without the need to communicate with any other authority. There does exist a global setup algorithm, but it is only used to generate a set of public parameters for the system.

   In our construction, all master secret keys $\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell$ must be generated under a single setup algorithm. If the set of key-generating authorities are *static*, then the setup algorithm can just be run once in the beginning and therefore, the authorities can use general MPC to generate the master secret keys $\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell$ in a secure way. When the set of authorities are *dynamic*, however, the setup algorithm must be run multiple times. For instance, if a new key-generating authority joins the system, then all existing key-generating authorities must participate in a new MPC protocol. This limitation is not satisfied by the Lewko-Waters construction since a new key-generating authority can generate its own master secret key.

2. **Flexibility in decryption keys**: Let $\mathsf{sk}_{\mathtt{gid},x} = (\mathsf{sk}_{\mathtt{gid},1,x_1}, \ldots, \mathsf{sk}_{\mathtt{gid},\ell,x_\ell})$ be a decryption key for a user corresponding to an attribute string $x \in \{0,1\}^\ell$ (in the GID model). Then, in the Lewko-Waters construction, the key components $\mathsf{sk}_{\mathtt{gid},j,x_j}$ for $j \in [\ell]$ that correspond to the zero-bit attribute $x_j = 0$ are just empty strings $\mathsf{sk}_{\mathtt{gid},j,0} = \varepsilon$. Therefore, for all indices $j \in [\ell]$ for which $x_j = 0$, there is no need for the j^{th} key-generating authority to participate in the key generation procedure.

   In our construction, the key components $\mathsf{sk}_{\mathsf{req},j,0}$ are not empty strings and therefore, must be generated by the corresponding key-generating authorities. This means that for any attribute string $x \in \{0,1\}^\ell$, *all* ℓ key-generating authorities in the system must provide a decryption key component to a key recipient. If any single authority fails to provide its key component, then the rest of the recipient's key components are incomplete and become useless.

We believe that despite these limitations, our constructions still satisfy the core properties that are needed for an MA-ABE scheme. Furthermore, while the pairing-based constructions rely on random oracles, our construction works in the standard model. Given the fact that no collusion-resistant MA-ABE scheme from lattices was known prior to this work, we believe that our construction represents a significant step forward in decentralizing trust in lattice-based attribute-based encryption schemes.

2 Technical Overview

In this section, we provide a technical overview of our constructions. We begin by recalling the learning with errors assumption.

The LWE assumption. The learning with errors (LWE) assumption [Reg05], parameterized by n, m, q, χ, states that for a uniformly random vector $\mathbf{s} \in \mathbb{Z}_q^n$ and a uniformly random matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, the distribution $(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T)$ is computationally indistinguishable from the uniform distribution over $\mathbb{Z}_q^{n \times m} \times \mathbb{Z}_q^m$, where $\mathbf{e}$ is sampled from a (low-norm) error distribution χ. To simplify the presentation in this section, we will ignore the precise generation and evolution of the error term $\mathbf{e}$ and just refer to it as "noise."

2.1 General Approach towards MA-ABE in the GID Model

We demonstrate the main ideas of our construction by first describing a natural approach to constructing a lattice-based MA-ABE scheme in the GID model. To do so, we first review the existing ABE constructions of Boneh et al. [BGG+14] and the technique of *matrix embeddings*.

Matrix embeddings. The matrix embedding technique, which was first formalized by Boneh et al. [BGG+14] is a way of encoding a sequence of bits $y_1, \ldots, y_N \in \{0,1\}$ into vectors

$$\mathbf{s}^T\big(\mathbf{A}_1 + y_1 \cdot \mathbf{G} \mid \cdots \mid \mathbf{A}_N + y_N \cdot \mathbf{G}\big) + \mathsf{noise} \in \mathbb{Z}_q^{mN}, \tag{2.1}$$

where $\mathbf{A}_1, \ldots, \mathbf{A}_N \in \mathbb{Z}_q^{n \times m}$ are uniformly random matrices, $\mathbf{s} \in \mathbb{Z}_q^n$ is a uniformly random vector, and $\mathbf{G} \in \mathbb{Z}_q^{n \times m}$ is a special fixed matrix (called the "gadget matrix"). Embedding bits in this way enables homomorphic operations on these bits while keeping the noise small. In particular, given an input $y \in \{0,1\}^N$ and any polynomial-size circuit $C : \{0,1\}^N \to \{0,1\}$, there is a public operation that allows computing the following vector from Eq. (2.1):

$$\mathbf{s}^T\big(\mathbf{A}_C + C(y) \cdot \mathbf{G}\big) + \mathsf{noise} \in \mathbb{Z}_q^m, \tag{2.2}$$

where the matrix $\mathbf{A}_C \in \mathbb{Z}_q^{n \times m}$ depends only on the circuit C, and *not* on the encoded input y. Thus, we can define a homomorphic operation $\mathsf{Eval}_{\mathsf{pk}}$ on the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_N$ where on input a sequence of matrices $\mathbf{A}_1, \ldots, \mathbf{A}_N$ and a circuit C, $\mathsf{Eval}_{\mathsf{pk}}(C, \mathbf{A}_1, \ldots, \mathbf{A}_N) \to \mathbf{A}_C$.

ABE from LWE. The matrix embeddings technique can be used to construct a ciphertext policy ABE scheme [BGG+14] as follows. We define the master secret key of the ABE scheme to consist of a trapdoor for a public matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ denoted $\mathsf{td}_{\mathbf{A}} = \mathbf{A}^{-1}$. The ABE ciphertext for a predicate $\phi : \{0,1\}^\ell \to \{0,1\}$ and a message μ is defined to be a set of vectors $\mathsf{ct}_\phi = (\mathsf{ct}_0, \mathsf{ct}_1, \mathsf{ct}_2)$ that are defined as follows:

- $\mathsf{ct}_0 = \mathbf{s}^T\mathbf{d} + \mathsf{noise} + \lfloor q/2 \rceil \cdot \mu$,
- $\mathsf{ct}_1 = \mathbf{s}^T\mathbf{A} + \mathsf{noise}$,
- $\mathsf{ct}_2 = \mathbf{s}^T\big(\mathbf{A}_1 + \phi_1 \cdot \mathbf{G} \mid \cdots \mid \mathbf{A}_N + \phi_N \cdot \mathbf{G}\big) + \mathsf{noise}$,

for a secret vector $\mathbf{s} \in \mathbb{Z}_q^n$, a set of public matrices $\mathbf{A}_1, \ldots, \mathbf{A}_N \in \mathbb{Z}_q^{n \times m}$, and public vector $\mathbf{d} \in \mathbb{Z}_q^n$. A decryption key for an attribute string $x \in \{0,1\}^\ell$ then corresponds to a short vector $\mathbf{u}_x \in \mathbb{Z}^{2m}$ such that

$$\big(\mathbf{A} \mid \mathbf{A}_x\big) \cdot \mathbf{u}_x = \mathbf{d},$$

where $\mathbf{A}_x \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathcal{U}_x, \mathbf{A}_1, \ldots, \mathbf{A}_N)$ and $\mathcal{U}_x$ is the universal circuit $\mathcal{U}_x(\phi) = \phi(x)$. The vector $\mathbf{u}_x$ can be generated from trapdoor $\mathbf{A}^{-1}$ using standard lattice trapdoor techniques. To decrypt the ciphertext ct_ϕ, a user homomorphically evaluates the universal circuit $\mathcal{U}_x$ on the ciphertext vector ct_2 to produce

$$\mathbf{s}^T\big(\mathbf{A} \mid \mathbf{A}_x + \mathcal{U}_x(\phi) \cdot \mathbf{G}\big) + \mathsf{noise} \in \mathbb{Z}_q^m,$$

which can then be combined with $\mathbf{u}_x$ to unmask the message in ct_0 as long as $\phi(x) = 0$.[2]

Decentralization via FHE. One approach in constructing an MA-ABE scheme is to use fully-homomorphic encryption (FHE)[3] to divide the master secret key into multiple components. We note

[2] As is commonly done in lattice-based ABE papers, we revert the convention on the predicate ϕ such that a key sk_x can decrypt a ciphertext ct_ϕ bound under ϕ if and only if $\phi(x) = 0$.

[3] In our actual construction, we use a (leveled) homomorphic encryption scheme as opposed to a fully homomorphic encryption scheme.

that in existing LWE-based FHE constructions, a decryption key $\mathsf{fhe.sk}$ and a (homomorphically evaluated) ciphetext $\mathsf{fhe.ct}$ are vectors in $\mathbb{Z}_q^n$, and the decryption algorithm consists of simply taking their inner product $\langle \mathsf{fhe.sk}, \mathsf{fhe.ct} \rangle$ and rounding the result. Since inner products are linear, we can divide the FHE secret key vector $\mathsf{fhe.sk}$ into multiple additive shares $\mathsf{fhe.sk}_1, \ldots, \mathsf{fhe.sk}_\ell$ in $\mathbb{Z}_q^n$ such that $\langle \mathsf{fhe.sk}_1, \mathsf{fhe.ct} \rangle + \ldots + \langle \mathsf{fhe.sk}_\ell, \mathsf{fhe.ct} \rangle = \langle \mathsf{fhe.sk}, \mathsf{fhe.ct} \rangle$.

Using this property on FHE, our idea in constructing an MA-ABE scheme is as follows. We first define the setup algorithm such that it first generates an FHE secret key $\mathsf{fhe.sk}$, encrypts the trapdoor $\mathsf{fhe.ct}_{\mathbf{A}^{-1}} \leftarrow \mathsf{FHE.Encrypt}(\mathsf{fhe.sk}, \mathbf{A}^{-1})$, and includes $\mathsf{fhe.ct}_{\mathbf{A}^{-1}}$ as part of the public parameters. In addition, the setup algorithm divides the key $\mathsf{fhe.sk}$ into ℓ shares $\mathsf{fhe.sk}_1, \ldots, \mathsf{fhe.sk}_\ell$ and distributes them to each of the ℓ key-generating authorities. Now, to generate an attribute key sk_x, each authority with $\mathsf{fhe.sk}_j$ can homomorphically evaluate the ABE key generation algorithm on the FHE ciphertext to produce an encryption of the vector $\mathbf{u}_x$,

$$\mathsf{fhe.ct}_{\mathbf{A}^{-1}} \longrightarrow \mathsf{fhe.ct}_{\mathbf{u}_x},$$

and provide $\langle \mathsf{fhe.sk}_j, \mathsf{fhe.ct}_{\mathbf{u}_x} \rangle + \mathsf{noise}$ to the key recipient. The recipient can recover the key $\mathbf{u}_x$ by linearly combining each of these partial decryptions.

Problem with functionality. The problem with the approach above is that to generate the key $\mathbf{u}_x$, each key-generating authority $j \in [\ell]$ must be provided the entire string $x \in \{0,1\}^\ell$. The MA-ABE functionality requires that each authority $j \in [\ell]$ generates the key components only given an attribute bit $x_j \in \{0,1\}$ that it controls without interacting with any of the other authorities. To fix this problem, we modify the construction above as follows. First, we modify the setup algorithm to generate an additional set of trapdoor matrices $(\mathbf{B}_1, \mathbf{B}_1^{-1}), \ldots, (\mathbf{B}_\ell, \mathbf{B}_\ell^{-1})$ that are to be distributed to the key-generating authorities. Next, we include an extra vector in the ciphertext

- $\mathsf{ct}_3 = \mathbf{s}^T \big(\mathbf{B}_1 \mid \cdots \mid \mathbf{B}_\ell \big) + \mathsf{noise}$.

Now, to generate a key component sk_{j,x_j} for an index $j \in [\ell]$ and a bit $x_j \in \{0,1\}$, the j^{th} key authority who holds a share of the FHE secret key $\mathsf{fhe.sk}_j$ and the trapdoor $\mathbf{B}_j^{-1}$ proceeds as follows:

1. Derive a set of matrices $(\mathbf{C}_1, \ldots, \mathbf{C}_\ell) \leftarrow H(\mathtt{gid})$ from a public hash function
$$H : \{0,1\}^* \to (\mathbb{Z}_q^{n \times m})^\ell.$$
2. Use $\mathbf{B}_j^{-1}$ to sample a short matrix $\mathbf{R}_{j,x_j} \in \mathbb{Z}^{m \times m}$ such that
$$\mathbf{B}_j \cdot \mathbf{R}_{j,x_j} = \mathbf{C}_j + x_j \cdot \mathbf{G}.$$
3. Compute $\mathbf{A}_\mathcal{U} \leftarrow \mathsf{Eval}_{\mathsf{pk}}(\mathcal{U}, \mathbf{A}_1, \ldots, \mathbf{A}_N, \mathbf{C}_1, \ldots, \mathbf{C}_\ell)$ where $\mathcal{U}$ is the universal circuit
$$\mathcal{U}(\phi, x) = \phi(x).$$
4. Homomorphically evaluate the FHE ciphertext $\mathsf{fhe.ct}_{\mathbf{A}^{-1}}$ to produce an FHE ciphertext $\mathsf{fhe.ct}_{\mathbf{u}}$ that encrypts the vector $\mathbf{u} \in \mathbb{Z}^{2m}$ satisfying
$$\big(\mathbf{A} \mid \mathbf{A}_\mathcal{U} \big) \cdot \mathbf{u} = \mathbf{d}.$$

5. Define the short matrix $\mathbf{R}_{j,x_j}$ and the partial FHE decryption $\mathsf{p}_j = \langle \mathsf{fhe.sk}_j, \mathsf{fhe.ct}_{\mathbf{u}} \rangle + \mathsf{noise}$ as the key component

$$\mathsf{sk}_{\mathtt{gid},j,x_j} = \big(\mathbf{R}_{j,x_j}, \mathsf{p}_j\big).$$

Now, given all the key components $\mathsf{sk}_{\mathtt{gid},x} = \big(\mathsf{sk}_{\mathtt{gid},1,x_1}, \ldots, \mathsf{sk}_{\mathtt{gid},\ell,x_\ell}\big)$, a decryptor can first recombine the FHE partial decryption shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ to recover $\mathbf{u}$. Then, it can apply the short matrices $\mathbf{R}_{1,x_1}, \ldots, \mathbf{R}_{\ell,x_\ell}$ to ct_3 to produce the vector

$$\mathbf{s}^T\big(\ \mathbf{A} \ \big|\ \mathbf{A}_1 + \phi_1 \cdot \mathbf{G} \ \big|\ \cdots \big|\ \mathbf{A}_N + \phi_N \cdot \mathbf{G} \ \big|\ \mathbf{C}_1 + x_1 \cdot \mathbf{G} \ \big|\ \cdots \ \big|\ \mathbf{C}_\ell + x_\ell \cdot \mathbf{G}\ \big) + \mathsf{noise}.$$

Finally, the decryptor can homomorphically derive the vector

$$\mathbf{s}^T\big(\ \mathbf{A} \ \big|\ \mathbf{A}_{\mathcal{U}} + \mathcal{U}(\phi, x) \cdot \mathbf{G}\ \big) + \mathsf{noise},$$

which can be combined with $\mathbf{u}$ to unmask the message in ct_0 as long as $\phi(x) = 0$.

Problem with security. The MA-ABE scheme above satisfies the functionality requirements for an MA-ABE scheme. However, it still fails to satisfy the security requirements. In an MA-ABE scheme, even a colluding set of *key-generating authorities* should not be able to decrypt a ciphertext that they are not authorized to do so. For instance, if the policy predicate ϕ is an all-one function $\mathbf{1}(x) = 1$ for all $x \in \{0,1\}^\ell$, then even an adversary who holds all the master secret keys $\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell$ must not be able to decrypt a ciphertext $\mathsf{ct}_{\mathbf{1}}$ that is bound under $\mathbf{1}$. In the construction above, any single authority can decrypt any ciphertext ct_ϕ as it holds a trapdoor $\mathbf{B}_j^{-1}$ for some $j \in [\ell]$. This trapdoor information can be used to recover the secret vector $\mathbf{s}$ from the ciphertext component $\mathbf{s}^T\mathbf{B}_j + \mathsf{noise}$, which can be used to unmask the message.

To prevent this problem, we modify the construction above further. Instead of distributing the trapdoor information $\mathbf{B}_1^{-1}, \ldots, \mathbf{B}_\ell^{-1}$ to each of the key-generating authorities in the clear, we move these trapdoor information inside the FHE ciphertext. Specifically, the setup algorithm groups all trapdoor information into a single master secret key

$$\mathsf{msk}' = (\mathbf{A}^{-1}, \mathbf{B}_1^{-1}, \ldots, \mathbf{B}_\ell^{-1}),$$

and generates a ciphertext $\mathsf{fhe.ct}_{\mathsf{msk}'} \leftarrow \mathsf{FHE.Encrypt}(\mathsf{fhe.sk}, \mathsf{msk}')$ to be included as part of the public parameters. Now, each key-generating authority, who holds an FHE secret key share $\mathsf{fhe.sk}_j$ for some $j \in [\ell]$, homomorphically evaluates each steps of the key-generation procedure described above on $\mathsf{fhe.ct}_{\mathsf{msk}'}$ to produce the following ciphertexts:

- $\mathsf{fhe.ct}_{\mathbf{u}}$ that encrypts the vector $\mathbf{u}$,
- $\mathsf{fhe.ct}_{\mathbf{R}_{j,x_j}}$, which encrypts the short matrix $\mathbf{R}_{j,x_j} \in \mathbb{Z}^{m \times m}$ such that

$$\mathbf{B}_j \cdot \mathbf{R}_{j,x_j} = \mathbf{C}_j + x_j \cdot \mathbf{G}.$$

Then, it computes the partial decryption of these two ciphertexts and includes them in the key component sk_{j,x_j}.[4]

[4] In the actual construction, we also include a PRF key as part of the master secret key msk' and replace the public hash function H with a PRF. This allows us to remove the dependence on random oracles.

If the key component $\mathsf{sk}_{\mathsf{gid},j,x_j}$ consisted only of these two components, then there is no way for a recipient to recover the matrix $\mathbf{R}_{j,x_j}$ from the partial decryptions. Therefore, each key-generating authority $j \in [\ell]$ homomorphically computes (under FHE) all matrices

$$\begin{pmatrix} \mathbf{R}_{1,0} & \cdots & \mathbf{R}_{j-1,0} & \mathbf{R}_{j+1,0} & \cdots & \mathbf{R}_{\ell,0} \\ \mathbf{R}_{1,1} & & \mathbf{R}_{j-1,1} & \mathbf{R}_{j+1,1} & & \mathbf{R}_{\ell,1} \end{pmatrix},$$

such that

$$\mathbf{B}_i \cdot \mathbf{R}_{i,b} = \mathbf{C}_i + b \cdot \mathbf{G} \quad \forall i \in [\ell] \backslash \{j\}, b \in \{0,1\},$$

to produce the set of ciphertexts $\{\mathsf{fhe.ct}_{\mathbf{R}_{i,b}}\}_{i \in [\ell]\backslash\{j\}, b \in \{0,1\}}$. It includes the partial decryption of all of these ciphertexts as part of $\mathsf{sk}_{\mathsf{gid},j,x_j}$.

Now, given the decryption key components $\mathsf{sk}_{\mathsf{gid},x} = (\mathsf{sk}_{\mathsf{gid},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{gid},\ell,x_\ell})$, the recipient can recover the vector $\mathbf{u}$ as well as the matrices $\mathbf{R}_{1,x_1}, \ldots, \mathbf{R}_{\ell,x_\ell}$ by combining the ℓ FHE partial decryptions from the authorities. The matrices $\mathbf{R}_{1,1-x_1}, \ldots, \mathbf{R}_{\ell,1-x_\ell}$ that does not correspond to the attribute bits $x_1, \ldots, x_\ell$ still remain hidden from the recipient as the key components only include $\ell - 1$ partial decryption of their ciphertexts.

2.2 MA-ABE in the OT Model

Additional problems with security. The construction above comes close to a fully secure MA-ABE scheme, but not quite enough. The insecurity is not due to the way we are using FHE to split the master secret key msk', but due to the algebraic structure of the ABE key itself. Consider a user who holds two keys $\mathsf{sk}_{\mathsf{gid},x} = (\mathsf{sk}_{\mathsf{gid},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{gid},\ell,x_\ell})$ and $\mathsf{sk}_{\mathsf{gid},x'} = (\mathsf{sk}_{\mathsf{gid},1,x'_1}, \ldots, \mathsf{sk}_{\mathsf{gid},\ell,x'_\ell})$ for $x \neq x' \in \{0,1\}^\ell$. The security requirement of an MA-ABE scheme requires that the user cannot decrypt a ciphertext $\mathsf{ct}_{\mathbf{1}}$ that is encrypted under the all-one function $\mathbf{1}$. However, since $x \neq x'$, there exists an index $j \in [\ell]$ for which the user can derive two short matrices $\mathbf{R}_{j,0}, \mathbf{R}_{j,1} \in \mathbb{Z}^{m \times m}$ such that

$$\mathbf{B}_j \cdot \mathbf{R}_{j,0} = \mathbf{C}_j + 0 \cdot \mathbf{G}$$
$$\mathbf{B}_j \cdot \mathbf{R}_{j,1} = \mathbf{C}_j + 1 \cdot \mathbf{G}$$

The matrix $\Delta\mathbf{R} = \mathbf{R}_{j,0} - \mathbf{R}_{j,1}$ directly translates into a trapdoor matrix for $\mathbf{B}_j$, which can be used to decrypt any ciphertext.

One can hope that adding extra components or additionally modifying the construction may prevent a user from deriving the trapdoor information $\Delta\mathbf{R}$. However, as long as we resort to the algebraic structure of the Boneh et al. ABE construction, this insecurity seems inevitable as any single corrupt authority must be able to generate both key components $\mathsf{sk}_{\mathsf{gid},j,0}$ and $\mathsf{sk}_{\mathsf{gid},j,1}$ for some $j \in [\ell]$ simply by the functionality of MA-ABE.

Using oblivious transfer. We get around this limitation using oblivious transfers. The reason why an adversary can extract trapdoor information $\Delta\mathbf{R}$ from duplicate key components is that each key generating authority $j \in [\ell]$ homomorphically evaluates (under FHE) each of the matrices

$$\begin{pmatrix} \mathbf{R}_{1,0} & \cdots & \mathbf{R}_{j-1,0} & \mathbf{R}_{j,x_j} & \mathbf{R}_{j+1,0} & \cdots & \mathbf{R}_{\ell,0} \\ \mathbf{R}_{1,1} & & \mathbf{R}_{j-1,1} & & \mathbf{R}_{j+1,1} & & \mathbf{R}_{\ell,1} \end{pmatrix},$$

and provide the partial decryption of each of these $2\ell - 1$ ciphertexts as part of a single key component $\mathsf{sk}_{\mathsf{gid},j,x_j}$. This means that any user with key components $\{\mathsf{sk}_{\mathsf{gid},i,x_i}\}_{i \in [\ell]\backslash\{j\}}$ already has

access to $\ell - 1$ decryption shares for the ciphertexts of each of the matrices $\mathbf{R}_{j,0}$ and $\mathbf{R}_{j,1}$. The duplicate key components $\mathsf{sk}_{\mathsf{gid},j,0}$ and $\mathsf{sk}_{\mathsf{gid},j,1}$ provide the ℓ^{th} decryption shares for $\mathbf{R}_{j,0}$ and $\mathbf{R}_{j,1}$.

Now, the reason why a key-generating authority $j \in [\ell]$ must provide the partial decryption of all $2\ell - 2$ ciphertexts for the matrices

$$\begin{pmatrix} \mathbf{R}_{1,0} & \dots & \mathbf{R}_{j-1,0} & \mathbf{R}_{j+1,0} & \dots & \mathbf{R}_{\ell,0} \\ \mathbf{R}_{1,1} & & \mathbf{R}_{j-1,1} & \mathbf{R}_{j+1,1} & & \mathbf{R}_{\ell,1} \end{pmatrix},$$

is that it only knows the single attribute bit $x_j \in \{0,1\}$ that it controls and not the rest of the bits $x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_\ell \in \{0,1\}$. Our observation is that if the key-recipient can somehow provide a "hint" on the rest of the attribute bits $\{x_i\}_{i \neq j}$ to the key-generating authority, then the authority may not need to provide all $2\ell - 2$ partial decryptions, but only the $\ell - 1$ partial decryptions that correspond to the key-recipient's attribute $x \in \{0,1\}^\ell$. A natural tool for such situation is an oblivious transfer protocol.

Using an oblivious transfer (OT) protocol, we can make the following modification to our construction. We first require that to generate a key, each key-recipient sends an OT commitment of its attribute bits $x \in \{0,1\}^\ell$ as a form of a *key-request* to be sent to the key-generating authorities. Each authority $j \in [\ell]$ still computes the partial decryption of the FHE ciphertexts that encrypts the matrices

$$\begin{pmatrix} \mathbf{R}_{1,0} & \dots & \mathbf{R}_{j-1,0} & \mathbf{R}_{j,x_j} & \mathbf{R}_{j+1,0} & \dots & \mathbf{R}_{\ell,0} \\ \mathbf{R}_{1,1} & & \mathbf{R}_{j-1,1} & & \mathbf{R}_{j+1,1} & & \mathbf{R}_{\ell,1} \end{pmatrix},$$

like before, and provides the partial decryption of the ciphertexts that encrypts $\mathbf{R}_{j,x_j}$. However, for the partial decryption of the ciphertexts that correspond to the rest of the matrices $\{\mathbf{R}_{i,b}\}_{j \in [\ell] \setminus \{j\}, b \in \{0,1\}}$, the authority encodes each pair $(\mathbf{R}_{i,0}, \mathbf{R}_{i,1})$ for $i \in [\ell] \setminus \{j\}$ as OT messages before providing them to the key-recipient. By the security of an oblivious transfer protocol, the key-recipient can only recover the $\ell - 1$ components that correspond to the attribute bits $\{x_i\}_{i \in [\ell] \setminus \{j\}}$.

This demonstrates the main intuition for our MA-ABE. We note that in this construction a key-recipient must send a formal key-request (in the form of an OT message) to each of the key-generating authorities and therefore, the construction violates the syntactical requirements for an MA-ABE scheme in the GID model. However, in our construction, each of the key components are still generated independently by each of the key generating authorities without any form of interaction among them. Furthermore, the receiver privacy property of an oblivious transfer protocol guarantees that any irrelevant components of the attribute string $x_1, \dots, x_\ell$ remains hidden from each of the key-generating authorities and therefore, our construction satisfies the main conceptual requirements of an MA-ABE scheme. We model our construction in a new model that we call the OT model and prove correctness and security in this model.

3 Preliminaries

We begin by introducing some of the notations that we use in this work. For any two integers $n < m$, we write $[n, m]$ to denote the set of integers $\{n, n+1, \dots, m\}$. When $n = 1$, we simply write $[n]$ to denote the set of integers $\{1, \dots, n\}$. We write $\mathsf{Funs}[\mathcal{X}, \mathcal{Y}]$ to denote the set of all functions mapping from a domain $\mathcal{X}$ to a range $\mathcal{Y}$. Unless specified otherwise, we use λ to denote the security parameter. We say a function $f(\lambda)$ is negligible in λ, denoted by $\mathsf{negl}(\lambda)$, if $f(\lambda) = o(1/\lambda^c)$ for all $c \in \mathbb{N}$. We say that an event happens with overwhelming probability if its complement happens with negligible probability. We say an algorithm is efficient if it runs in probabilistic polynomial

time in the length of its input. We use $\mathsf{poly}(\lambda)$ to denote a quantity whose value is bounded by a fixed polynomial in λ. For our security experiments, we often write $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ to denote separate components of a single algorithm $\mathcal{A}$ that participate in different phases of an experiment. We assume that the components $\mathcal{A}_1$ and $\mathcal{A}_2$ share local state.

Vectors and matrices. We use bold lowercase letters (*e.g.*, $\mathbf{v}, \mathbf{w}$) to denote vectors and bold uppercase letter (*e.g.*, $\mathbf{A}, \mathbf{B}$) to denote matrices. Throughout this work, we always use the infinity norm for vectors and matrices. Namely, for a vector $\mathbf{x} \in \mathbb{Z}^n$, we write $\|\mathbf{x}\|$ to denote $\max_i |x_i|$. Similarly, for a matrix $\mathbf{A} = (A_{i,j}) \in \mathbb{Z}^{n \times m}$, we write $\|\mathbf{A}\|$ to denote $\max_{i,j} |A_{i,j}|$.

Modular rounding. For two integers $p \leq q$, we define the modular "rounding" function

$$\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p \text{ that maps } x \to \lfloor (p/q) \cdot x \rceil$$

and extend it coordinate-wise to matrices and vectors over $\mathbb{Z}_q$. Here, the operation $\lfloor \cdot \rceil$ is the rounding operation over the real numbers.

3.1 Statistical Distance

For two distributions X, Y over a finite domain Ω, the *statistical distance* between X and Y is defined by $\Delta(X, Y) = \frac{1}{2} \sum_{\omega \in \Omega} |X(\omega) - Y(\omega)|$. If X,Y are distribution ensembles parameterized by the security parameters, we say that X and Y are statistically indistinguishable denoted $X \overset{s}{\approx} Y$ if $\Delta(X, Y)$ is negligible. For a distribution X, we write $x \leftarrow X$ to denote the procedure of sampling x according to distribution X. For a finite domain Ω, we write $x \xleftarrow{R} \Omega$ to denote the procedure of sampling x uniformly from Ω. For a distribution ensemble $\chi = \chi(\lambda)$ over the integers, and integer bounds $B = B(\lambda)$, we say that χ is *B-bounded* if $\Pr_{x \leftarrow \chi(\lambda)} \left[|x| \leq B(\lambda) \right] = 1$.

We recall two statistical facts that we use throughout the analysis of our constructions. Since we do not require the most general formulations of these statements, we tailor them specifically for our needs.

Lemma 3.1 (Leftover Hash Lemma [ILL89, DORS08])**.** *Let n, m, q be positive integers with $m = \Theta(n \log q)$. Then, for $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n \times m}$, $\mathbf{r} \xleftarrow{R} \{0,1\}^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^n$, the following two distributions are statistically indistinguishable:*

$$(\mathbf{A}, \mathbf{A} \cdot \mathbf{r}) \qquad \overset{s}{\approx} \qquad (\mathbf{A}, \mathbf{u}).$$

Lemma 3.2 (Smudging Lemma [AJLA+12])**.** *Let $B_1 = B_1(\lambda)$, and $B_2 = B_2(\lambda)$ be positive integers and let $e_1 \in [-B_1, B_1]$ be a fixed integer. Let $e_2 \xleftarrow{R} [-B_2, B_2]$ be chosen uniformly at random. Then, the distribution of $e_1 + e_2$ is statistically indistinguishable from that of e_1 as long as $B_1/B_2 = \mathsf{negl}(\lambda)$.*

3.2 Lattice Preliminaries

In this section, we provide background on the learning with errors assumption and lattice trapdoors.

Learning with errors. The learning with errors (LWE) assumption was first introduced by Regev [Reg05]. In the same work, Regev showed that solving LWE in the *average case* is as hard as (quantumly) approximating several standard lattice problems in the *worst case*. We state the assumption below.

Definition 3.3 (Learning with Errors [Reg05])**.** Fix a security parameter λ and integers $n = n(\lambda)$, $m = m(\lambda)$, $q = q(\lambda)$, and an error distribution $\chi = \chi(\lambda)$ over the integers. Then, the (decisional) learning with errors (LWE) assumption $\mathsf{LWE}_{n,m,q,\chi}$ states that for $\mathbf{A} \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $\mathbf{s} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{e} \xleftarrow{R} \chi^m$, and $\mathbf{u} \xleftarrow{R} \mathbb{Z}_q^m$, the following two distributions are computationally indistinguishable:

$$(\mathbf{A}, \mathbf{s}^T\mathbf{A} + \mathbf{e}^T) \quad \text{and} \quad (\mathbf{A}, \mathbf{u}^T)$$

When the error distribution χ_B is a B-bounded discrete Gaussian distribution, then the $\mathsf{LWE}_{n,m,q,\chi_B}$ assumption is true assuming that various worst-case lattice problems such as GapSVP_γ and SIVP_γ on an n-dimensional lattice are hard for $\gamma = \tilde{O}(n \cdot q/B)$ by a quantum algorithm [Reg05]. Similar reductions of LWE to the *classical* hardness of approximating worst-case lattice problems are also known [Pei09, ACPS09, MM11, MP12, BLP+13].

The gadget matrix. We define the "gadget matrix" $\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_n \in \mathbb{Z}_q^{n\times n\cdot\lceil\log q\rceil}$ where $\mathbf{g} = (1, 2, 4, \ldots, 2^{\lceil\log q\rceil-1})$. We define the inverse function $\mathbf{G}^{-1} : \mathbb{Z}_q^{n\times m} \to \mathbb{Z}_q^{n\lceil\log q\rceil\times m}$ which expands each entry $x \in \mathbb{Z}_q$ in the input matrix into a column of size $\lceil\log q\rceil$ consisting of the bits of the binary representation of x. To simplify the notation, we always assume that $\mathbf{G}$ has width m (in our construction, $m = \Theta(n \log q)$). Note that this is without loss of generality since we can always extend $\mathbf{G}$ by appending all-zero columns. It is easy to see that for any matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, we have $\mathbf{G} \cdot \mathbf{G}^{-1}(\mathbf{A}) = \mathbf{A}$.

Lattice trapdoors. Although LWE is believed to be hard, the problem becomes easy with some auxiliary trapdoor information. Lattice trapdoors have been used in a wide variety of context and are studied extensively in the literature [Ajt99, GPV08, AP09, MP12, LW15]. Since the specific details of the trapdoor constructions are not necessary for this work, we highlight just the properties that we require in this work.

Theorem 3.4 (Lattice Trapdoors [Ajt99, GPV08, AP09, MP12, LW15])**.** *Let λ be the security parameter and n, m, q be a set of lattice parameters, and B be a norm bound such that that $m = \Omega(n \log q)$ and a bound $B = \Omega(\sqrt{n})$ Then, there exists a tuple of efficient algorithms* $(\mathsf{TrapGen}, \mathsf{Invert}, \mathsf{Sample})$ *with the following properties:*

- $\mathsf{TrapGen}(1^\lambda) \to (\mathbf{A}, \mathsf{td})$*: On input the security parameter λ, the trapdoor generation algorithm returns a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and a trapdoor* td*.*
- $\mathsf{Invert}(\mathbf{A}, \mathsf{td}, \mathbf{y}, \gamma) \to \mathbf{x}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, trapdoor* td*, target vector $\mathbf{y} \in \mathbb{Z}_q^n$, and norm bound $\gamma \in \mathbb{N}$, the inversion algorithm returns a vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\mathbf{A} \cdot \mathbf{x} = \mathbf{y}$ and $\|\mathbf{x}\| \le \gamma$.*
- $\mathsf{Sample}(1^\lambda, \gamma) \to \mathbf{x}$*: On input the security parameter λ and a norm bound $\gamma \in \mathbb{N}$, the sampling algorithm returns a vector $\mathbf{x} \in \mathbb{Z}^m$ such that $\|\mathbf{x}\| \le \gamma$.*

The algorithms above satisfy the following property. For any $\mathbf{y} \in \mathbb{Z}_q^n$ and $\gamma \ge B$, setting $(\mathbf{A}, \mathsf{td}) \leftarrow \mathsf{TrapGen}(1^\lambda)$, $\mathbf{y} \xleftarrow{R} \mathbb{Z}_q^n$, $\mathbf{x} \leftarrow \mathsf{Invert}(\mathbf{A}, \mathsf{td}, \mathbf{y}, \gamma)$, $\mathbf{A}' \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $\mathbf{x}' \leftarrow \mathsf{Sample}(1^\lambda, \gamma)$, and $\mathbf{y}' = \mathbf{A}' \cdot \mathbf{x}'$, we have have

$$(\mathbf{A}, \mathbf{x}, \mathbf{y}) \quad \overset{s}{\approx} \quad (\mathbf{A}', \mathbf{x}', \mathbf{y}').$$

Traditionally, lattice trapdoors for $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ consist of a set of short generating vectors of the lattice that is induced by $\mathbf{A}$. In this work, we make use of an alternative form of lattice trapdoors called a $\mathbf{G}$-trapdoor formalized in [MP12]. A $\mathbf{G}$-trapdoor of a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ consists of a full-rank, low-norm matrix $\mathbf{R} \in \mathbb{Z}^{m \times m}$ satisfying the relation $\mathbf{AR} = \mathbf{G}$. These types of trapdoor matrices have additional statistical properties that we use in our multi-authority attribute-based encryption constructions. We highlight the key properties in the following theorem.

Theorem 3.5 ([CHKP10, ABB10, MP12, BGG+14])**.** *Let λ be a security parameter and n, m, q be lattice parameters, and B be a noise bound such that $m = \Omega(n \log q)$ and $B = \Omega(\sqrt{n})$. Then, there exist a pair of algorithms* $(\mathsf{SampleLeft}, \mathsf{SampleRight})$ *with the following syntax:*

- $\mathsf{SampleLeft}(\mathbf{A}, \mathsf{td}_{\mathbf{A}}, \mathbf{B}, \mathbf{y}, \gamma) \to \mathbf{x}$*: On input a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$, trapdoor $\mathsf{td}_{\mathbf{A}}$, matrix $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$, target vector $\mathbf{y} \in \mathbb{Z}_q^n$, and a norm bound γ, the* $\mathsf{SampleLeft}$ *algorithm returns a vector $\mathbf{x} \in \mathbb{Z}^{2m}$.*

- $\mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{y}, \gamma) \to \mathbf{x}$*: On input matrices $\mathbf{A}, \mathbf{B} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{R} \in \mathbb{Z}^{m \times m}$, target vector $\mathbf{y} \in \mathbb{Z}_q^n$, and a norm bound γ, the* $\mathsf{SampleRight}$ *algorithm returns a vector $\mathbf{x} \in \mathbb{Z}^{2m}$.*

The algorithms above satisfy the following properties. For any $(\mathbf{A}, \mathsf{td}_{\mathbf{A}}) \leftarrow \mathsf{TrapGen}(1^\lambda)$, $\mathbf{B} \in \mathbb{Z}_q^{n \times m}$, $\mathbf{R} \in \mathbb{Z}^{m \times m}$ such that $\mathbf{A} \cdot \mathbf{R} + \mathbf{G} = \mathbf{B}$, and $B \leq \|\mathbf{R}\| \cdot \omega(m\sqrt{\log m}) \leq \gamma$, and setting $\mathbf{x} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathsf{td}_{\mathbf{A}}, \mathbf{B}, \mathbf{y}, \gamma)$ and $\mathbf{x}' \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{B}, \mathbf{R}, \mathbf{y}, \gamma)$, we have

1. *$[\mathbf{A}|\mathbf{B}] \cdot \mathbf{x} = [\mathbf{A}|\mathbf{B}] \cdot \mathbf{x}' = \mathbf{y}$ and $\|\mathbf{x}\|, \|\mathbf{x}'\| \leq \gamma$.*
2. *The vectors $\mathbf{x}$ and $\mathbf{x}'$ are statistically indistinguishable.*

3.3 Pseudorandom Functions

We review the basic definition of a *pseudorandom function*.

Definition 3.6 (Pseudorandom Function [GGM84])**.** Let $\mathcal{K}$, $\mathcal{X}$, and $\mathcal{Y}$ be sets. Then, an efficiently computable deterministic function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a secure *pseudorandom function* (PRF) if for all efficient adversaries $\mathcal{A}$, $k \xleftarrow{R} \mathcal{K}$, and $f \xleftarrow{R} \mathsf{Funs}[\mathcal{X}, \mathcal{Y}]$, we have

$$\left| \Pr[\mathcal{A}^{F(k,\cdot)}(1^\lambda) = 1] - \Pr[\mathcal{A}^{f(\cdot)}(1^\lambda) = 1] \right| = \mathsf{negl}(\lambda).$$

In this work, we use a special family of pseudorandom functions called *key-homomorphic PRFs* that satisfy additional algebraic properties. Specifically, the key space $\mathcal{K}$ and the range $\mathcal{Y}$ of the PRF exhibit certain group structures such that its evaluation on any fixed input $x \in \mathcal{X}$ is homomorphic with respect to these group structures. Formally, we define a key-homomorphic PRF as follows.

Definition 3.7 (Key-Homomorphic PRF [BLMR13])**.** Let $(\mathcal{K}, \oplus)$, $(\mathcal{Y}, \otimes)$ be groups. Then, an efficiently computable deterministic function $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ is a *key-homomorphic* PRF if

1. F is a secure PRF (Definition 3.6)
2. For every key $k_1, k_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$, we have $F(k_1, x) \otimes F(k_2, x) = F(k_1 \oplus k_2, x)$.

We note that the function $F_k(x) = F(k, x)$ is a group homomorphism from $\mathcal{K}$ to $\mathcal{Y}$. This implies that for the identity elements $0_{\mathcal{K}} \in \mathcal{K}$ and $0_{\mathcal{Y}} \in \mathcal{Y}$, we have $F(0_{\mathcal{K}}, x) = 0_{\mathcal{Y}}$ for any $x \in \mathcal{X}$.

Although key homomorphic PRFs are natural primitives, we currently do not know how to construct them without either relying on the random oracle heuristic or using strong cryptographic tools such as multi-linear maps [BLMR13]. Since the goal in this work is to instantiate our construction from LWE alone, we work with a relaxed notion called *almost* key-homomorphic PRFs, which we can currently construct from LWE.

Definition 3.8 (γ-Almost Key-Homomorphic PRF [BLMR13])**.** Let $(\mathcal{K}, \oplus)$ be a group and p be a positive integer. Then, an efficiently computable deterministic function $F : \mathcal{K} \times \mathcal{X} \to \mathbb{Z}_p$ is an *γ-almost key-homomorphic* PRF if

1. F is a secure PRF (Definition 3.6)
2. For every key $k_1, k_2 \in \mathcal{K}$ and every $x \in \mathcal{X}$, we have $F(k_1, x) + F(k_2, x) = F(k_1 \oplus k_2, x) + e$ for an error term $|e| \leq \gamma$.

3.4 (Leveled) Homomorphic Encryption

Following the presentation of [GVW15], we give a minimal definition of a (leveled) homomorphic encryption scheme that suffices for our construction. Note that a leveled homomorphic encryption scheme is one that only supports an *a priori* bounded number of homomorphic operations. This is to contrast it with the notion of a fully homomorphic encryption scheme (FHE) scheme, which supports an arbitrary number of homomorphic operations on ciphertexts

Definition 3.9 (Homomorphic Encryption)**.** A (leveled) *homomorphic* encryption scheme for the message space $\{0,1\}^*$ is a tuple of algorithms $\Pi_{\mathsf{HE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^d) \to \mathsf{sk}$: On input the security parameter λ and depth bound d, the key generation algorithm returns a secret key sk.
- $\mathsf{Encrypt}(\mathsf{sk}, \mu) \to \mathsf{ct}$: On input a secret key sk and a message $\mu \in \{0,1\}^*$, the encryption algorithm returns a ciphertext ct.
- $\mathsf{Eval}(C, \mathsf{ct}) \to \hat{\mathsf{ct}}$: On input a circuit $C : \{0,1\}^* \to \{0,1\}$ and a ciphertext ct, the evaluation algorithm returns a homomorphically evaluated ciphertext $\hat{\mathsf{ct}}$.
- $\mathsf{Decrypt}(\mathsf{sk}, \hat{\mathsf{ct}}) \to \mu$: On input a secret key sk and a ciphertext $\hat{\mathsf{ct}}$, the decryption algorithm returns a message $\mu \in \{0,1\}$.

A homomorphic encryption must satisfy the following compactness, correctness, and security properties.

Definition 3.10 (Compactness)**.** Let $\Pi_{\mathsf{HE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{Decrypt})$ be a homomorphic encryption scheme for the message space $\{0,1\}^*$. We say that Π_{HE} satisfies *compactness* if there exists a polynomial $\mathsf{poly}(\cdot,\cdot)$ such that for all $\lambda, d \in \mathbb{N}$, $\mu \in \{0,1\}^*$, $C : \{0,1\}^* \to \{0,1\}$, and setting $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mu)$, and $\hat{\mathsf{ct}} \leftarrow \mathsf{Eval}(C, \mathsf{ct})$, we have $|\hat{\mathsf{ct}}| \leq \mathsf{poly}(\lambda, d)$.

Definition 3.11 (Correctness)**.** Let $\Pi_{\mathsf{HE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{Decrypt})$ be a homomorphic encryption scheme for the message space $\{0,1\}^*$. We say that Π_{HE} satisfies *correctness* if for all $\lambda, d \in \mathbb{N}$, $\mu \in \{0,1\}^*$, $C : \{0,1\}^* \to \{0,1\}$, and setting $\mathsf{sk} \leftarrow \mathsf{KeyGen}(1^\lambda)$, $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mu)$, and $\hat{\mathsf{ct}} \leftarrow \mathsf{Eval}(C, \mathsf{ct})$, we have

$$\Pr[\mathsf{Decrypt}(\mathsf{sk}, \hat{\mathsf{ct}}) = C(\mu)] = 1.$$

Definition 3.12 (Security)**.** Let $\Pi_{\mathsf{HE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{Decrypt})$ be a homomorphic encryption scheme for the message space $\{0,1\}^*$. We say that Π_{HE} satisfies *security* if for all $\lambda, d \in \mathbb{N}$, efficient adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, and setting $(\mu_0, \mu_1) \leftarrow \mathcal{A}_1^{\mathsf{Encrypt}(\mathsf{sk},\cdot)}(1^\lambda)$ where $|\mu_0| = |\mu_1|$, and $\mathsf{ct}_\beta \leftarrow \mathsf{Encrypt}(\mathsf{sk}, \mu_\beta)$ for $\beta \in \{0,1\}$, we have

$$\left| \Pr[\mathcal{A}^{\mathsf{Encrypt}(\mathsf{sk},\cdot)}(\mathsf{ct}_0) = 1] - \Pr[\mathcal{A}^{\mathsf{Encrypt}(\mathsf{sk},\cdot)}(\mathsf{ct}_1) = 1] \right| = \mathsf{negl}(\lambda).$$

3.5 Oblivious Transfer

Finally, we recall the notion of an oblivious transfer protocol. Following the presentation of [PVW08, BD18], we define an oblivious transfer protocol with respect to the concrete algorithms that are run by each of the participants in the protocol. We label each of these algorithms with names that intuitively describes their algebraic instantiations from lattices. In particular, we refer to the algorithms that are run by the receiver as the $\mathsf{KeyGen}_{\mathsf{R}}$ and $\mathsf{Decrypt}_{\mathsf{R}}$ algorithms and the algorithm that is run by the sender as the $\mathsf{Encrypt}_{\mathsf{S}}$ algorithm.

Definition 3.13 (Oblivious Transfer)**.** An (1-out-of-2) *oblivious transfer* protocol Π_{OT} for a message space $\mathcal{M}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{OT}} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{R}}, \mathsf{Encrypt}_{\mathsf{S}}, \mathsf{Decrypt}_{\mathsf{R}})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$: On input the security parameter λ, the setup algorithm returns a common reference string crs and a trapdoor td.
- $\mathsf{KeyGen}_{\mathsf{R}}(\mathsf{crs}, \beta) \to (\mathsf{pk}, \mathsf{sk})$: On input a common reference string crs and a bit $\beta \in \{0,1\}$, the (receiver) key generation algorithm returns a public key pk and a secret key sk.
- $\mathsf{Encrypt}_{\mathsf{S}}(\mathsf{crs}, (\mu_0, \mu_1), \mathsf{pk}) \to \mathsf{ct}$: On input a common reference string crs, a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$, and a public key pk, the (sender) encryption algorithm returns a ciphertext ct.
- $\mathsf{Decrypt}_{\mathsf{R}}(\mathsf{crs}, \mathsf{sk}, \mathsf{ct}) \to \mu'$: On input a common reference string crs, a secret key sk, and a ciphertext ct, the decryption algorithm returns a message $\mu' \in \mathcal{M}$.

An oblivious transfer protocol must satisfy the following correctness property.

Definition 3.14 (Correctness)**.** Let $\Pi_{\mathsf{OT}} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{R}}, \mathsf{Encrypt}_{\mathsf{S}}, \mathsf{Decrypt}_{\mathsf{R}})$ be an oblivious transfer protocol for a message space $\mathcal{M}$. Then, we say that Π_{OT} satisfies *correctness* if for all $\lambda \in \mathbb{N}$, $\beta \in \{0,1\}$, setting $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}_{\mathsf{R}}(\mathsf{crs}, \beta)$, we have

$$\Pr\left[\mathsf{Decrypt}_{\mathsf{R}}(\mathsf{crs}, \mathsf{sk}, \mathsf{Encrypt}_{\mathsf{S}}(\mathsf{crs}, (\mu_0, \mu_1), \mathsf{pk})) = \mu_\beta\right] = 1.$$

There are many ways of defining security for oblivious transfer protocols. For instance, their security conditions can be most rigorously formulated in the *universally composable* (UC) framework [Can01]. In this work, we work with simpler (and weaker) privacy definitions for oblivious transfer protocols that we define as follows.

Definition 3.15 (Receiver Privacy)**.** Let $\Pi_{\mathsf{OT}} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{R}}, \mathsf{Encrypt}_{\mathsf{S}}, \mathsf{Decrypt}_{\mathsf{R}})$ be an oblivious transfer protocol for a message space $\mathcal{M}$. Then, we say that Π_{OT} satisfies *receiver privacy* if for all efficient adversaries $\mathcal{A}$, setting $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$ and $\mathsf{pk}_\beta \leftarrow \mathsf{KeyGen}_{\mathsf{R}}(\mathsf{crs}, \beta)$ for $\beta \in \{0,1\}$, we have

$$\left| \Pr[\mathcal{A}(\mathsf{crs}, \mathsf{pk}_0) = 1] - \Pr[\mathcal{A}(\mathsf{crs}, \mathsf{pk}_1) = 1] \right| = \mathsf{negl}(\lambda),$$

Definition 3.16 (Sender Security)**.** Let $\Pi_{\mathsf{OT}} = (\mathsf{Setup}, \mathsf{KeyGen}_{\mathsf{R}}, \mathsf{Encrypt}_{\mathsf{S}}, \mathsf{Decrypt}_{\mathsf{R}})$ be an oblivious transfer protocol for a message space $\mathcal{M}$. Then, we say that Π_{OT} satisfies (statistical) *sender security* if there exists an efficient extractor OTExt such that for any message $\mu_0, \mu_1 \in \mathcal{M}$ for which $|\mu_0| = |\mu_1|$, public key $\mathsf{pk} \in \{0,1\}^*$, and (unbounded) adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, setting $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, $\beta \leftarrow \mathsf{OTExt}(\mathsf{td}, \mathsf{pk})$, $\mathsf{ct}_{\mathsf{real}} \leftarrow \mathsf{Encrypt}_{\mathsf{S}}(\mathsf{pk}, \mu_0, \mu_1)$, $\mathsf{ct}^{(0)}_{\mathsf{ideal}} \leftarrow \mathsf{Encrypt}_{\mathsf{S}}(\mathsf{pk}, \mu_0, \bot)$, and $\mathsf{ct}^{(1)}_{\mathsf{ideal}} \leftarrow \mathsf{Encrypt}_{\mathsf{S}}(\mathsf{pk}, \bot, \mu_1)$, we have

$$\left| \Pr[\mathcal{A}_2(\mathsf{crs}, \mathsf{pk}, \mathsf{ct}_{\mathsf{real}}) = 1] - \Pr[\mathcal{A}_2(\mathsf{crs}, \mathsf{pk}, \mathsf{ct}^{(\beta)}_{\mathsf{ideal}}) = 1] \right| = \mathsf{negl}(\lambda).$$

We note that by a standard hybrid argument, the sender security property can be extended to hold over multiples pairs of messages $(\mu_{1,0}, \mu_{1,1}), \ldots, (\mu_{Q,0}, \mu_{Q,1})$ and multiple public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_Q$ in a straightforward way.

4 Universal Thresholdizers

One of the main building blocks of our multi-authority attribute-based encryption is a notion called a *universal thresholdizer*, which was introduced by Boneh et al. [BGG+18] as a general tool for threshold cryptography. They showed that existing homomorphic encryption schemes [BGV12, GSW13] based on the LWE assumption can be used to construct universal thresholdizer schemes for arbitrary message spaces and bounded depth circuits. Unfortunately, the security guarantee that their construction satisfies is not fully sufficient for our multi-authority attribute-based encryption scheme. In this section, we augment the construction of [BGG+18] using key-homomorphic PRFs to satisfy a stronger security definition that we need for our construction in Section 5. We begin by recalling the formal definitions of a universal thresholdizer scheme in Section 4.1. Then, in Section 4.2, we describe some of the general properties of the existing key-homomorphic PRFs and homomorphic encryption schemes that we use for our new thresholdizer construction. Finally, we present our universal thresholdizer construction in Section 4.3 and provide the formal proof of correctness and security in Appendix A.

4.1 Definitions

A universal thresholdizer (UT) scheme allows a user to divide a secret message $x \in \{0,1\}^*$ into a number of shares $\mathsf{s}_1, \ldots, \mathsf{s}_\ell$ such that independent parties can evaluate a function $C : \{0,1\}^* \to \{0,1\}$ on the shares $\mathsf{p}_i \leftarrow \mathsf{Eval}(C, \mathsf{s}_i)$ to derive new sets of shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ of the message $C(x)$. A key property that a universal thresholdizer scheme must satisfy is that the shares $\mathsf{s}_1, \ldots, \mathsf{s}_\ell$ are *reusable*. Namely, the evaluated shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ that are output by the evaluation algorithm $\mathsf{Eval}(C, \mathsf{s}_j)$ for $j \in [\ell]$ must not only hide information about the original message x, but also hide enough information about the original shares $\mathsf{s}_1, \ldots, \mathsf{s}_\ell$ such that the shares can be *reused* for an a priori unbounded number of evaluations.

Formally, we define the algorithms of a universal thresholdizer scheme as follows. We note that the original syntax of a universal thresholdizer in [BGG+18] is defined very generally with respect to arbitrary access structures. For our multi-authority ABE schemes in Section 5, we will only require a universal thresholdizer that supports ℓ-out-of-ℓ access structures. Therefore, for simplicity, we restrict our definition for the special case of ℓ-out-of-ℓ access structures.

Definition 4.1 (Universal Thresholdizer [BGG+18])**.** A *universal thresholdizer* (UT) scheme Π_{UT} with input space $\{0,1\}^*$ and output space $\{0,1\}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{UT}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\ell, 1^d, x) \to (\mathsf{s}_1, \ldots, \mathsf{s}_\ell)$: On input the security parameter λ, number of shares ℓ, depth bound d, an input $x \in \{0,1\}^*$, the setup algorithm returns a set of shares $\mathsf{s}_1, \ldots, \mathsf{s}_\ell$.
- $\mathsf{Eval}(C, \mathsf{s}_j) \to \mathsf{p}_j$: On input a circuit $C : \{0,1\}^* \to \{0,1\}$ and a share s_j, the evaluation algorithm returns an evaluation share p_j.
- $\mathsf{Combine}(\mathsf{p}_1, \ldots, \mathsf{p}_\ell) \to y$: On input a set of evaluation shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$, the combining algorithm returns a final evaluation $y \in \{0,1\}$.

We now define the correctness and security requirements for a universal thresholdizer scheme.

Definition 4.2 (Correctness)**.** Let $\Pi_{\mathsf{UT}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ be a universal thresholdizer scheme. Then, we say that Π_{UT} satisfies *correctness* if for all $\lambda, \ell, d \in \mathbb{N}$, input $x \in \{0,1\}^*$, circuit $C : \{0,1\}^* \to \{0,1\}$ of depth at most d, and setting $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^d, x)$, $\mathsf{p}_j \leftarrow \mathsf{Eval}(C, \mathsf{s}_j)$ for $j \in [\ell]$, we have

$$\Pr\left[\mathsf{Combine}(\{\mathsf{p}_j\}_{j\in[\ell]}) = C(x)\right] = 1 - \mathsf{negl}(\lambda).$$

For security, we require that the evaluation shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ do not leak any information about the original message x other than what can be derived from the final output output of the computation $C(x)$. Furthermore, we require that this property holds even when the shares $\mathsf{s}_1, \ldots, \mathsf{s}_\ell$ are reused for an *unbounded* number of evaluations. A natural way to formalize this property is to define an experiment between an adversary and a challenger where the adversary makes an unbounded number of (adaptive) evaluation queries consisting of circuits $C_1, \ldots, C_Q$. The challenger in the experiment must then respond to the adversary's queries with the evaluations $\mathsf{p}_{i,j} \leftarrow \mathsf{Eval}(C_i, \mathsf{s}_j)$ for $i = 1, \ldots, Q$. We say that a universal thresholdizer is secure if there exists a simulator that can simulate the partial evaluations $\mathsf{p}_{i,j}$ just given the final output $C_1(x), \ldots, C_Q(x)$ such that the adversary cannot distinguish whether these evaluation shares are generated by the honest evaluation algorithm or by the simulator. This is the standard universal thresholdizer security definition as formalized in [BGG+18].

Definition 4.3 (Security)**.** Let $\Pi_{\mathsf{UT}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ be a universal thresholdizer scheme. Then, we say that Π_{UT} satisfies *security* if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for all $\lambda, \ell, d \in \mathbb{N}$, and adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, setting $(x, S) \leftarrow \mathcal{A}_1(1^\lambda, 1^\ell)$ where $S \subsetneq [\ell]$, $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^d, x)$, $(\tilde{\mathsf{s}}_1, \ldots, \tilde{\mathsf{s}}_\ell) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^\ell, 1^d, S, 1^{|x|})$, we have

$$\left|\Pr\left[\mathcal{A}_2^{\mathcal{O}_{\mathsf{real}}(\cdot,\cdot)}(\{\mathsf{s}_j\}_{j\in S}) = 1\right] - \Pr\left[\mathcal{A}_2^{\mathcal{O}_{\mathsf{ideal}}(\cdot,\cdot)}(\{\tilde{\mathsf{s}}_j\}_{j\in S}) = 1\right]\right| = \mathsf{negl}(\lambda),$$

where the oracles $\mathcal{O}_{\mathsf{real}}$ and $\mathcal{O}_{\mathsf{ideal}}$ are defined as follows:

- $\mathcal{O}_{\mathsf{real}}(C, j)$: On input a circuit $C : \{0,1\}^* \to \{0,1\}$ and an index $j \in [\ell] \backslash S$, the oracle computes $\mathsf{p}_j \leftarrow \mathsf{Eval}(C, \mathsf{s}_j)$ and returns p_j.
- $\mathcal{O}_{\mathsf{ideal}}(C, j)$: On input a circuit $C : \{0,1\}^* \to \{0,1\}$ and an index $j \in [\ell] \backslash S$, the oracle invokes the simulator $\mathsf{p}_j \leftarrow \mathsf{Sim}_2\big(C, \{\tilde{\mathsf{s}}_j\}_{j\in[\ell]}, C(x)\big)$ and returns p_j.

For our multi-authority attribute-based encryption construction in Section 5, we require a universal thresholdizer scheme to satisfy a stronger security definition. Namely, we require that the evaluation shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ not only hide information about the message x, but they are computationally indistinguishable from *perfect* secret shares of the output $C(x)$. In other words, we require that, to an efficient adversary, the evaluation shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ are computationally indistinguishable from uniformly random elements $\tilde{\mathsf{p}}_1, \ldots, \tilde{\mathsf{p}}_\ell$ for which $\mathsf{Combine}(\tilde{\mathsf{p}}_1, \ldots, \tilde{\mathsf{p}}_\ell) = C(x)$. We refer to this property as *strong security* and formally define it as follows.

Definition 4.4 (Strong Security)**.** Let $\Pi_{\mathsf{UT}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ be a universal thresholdizer scheme. Then, we say that Π_{UT} satisfies *strong security* if there exists a simulator $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for all $\lambda, \ell, d \in \mathbb{N}$, and adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, setting $(x, S) \leftarrow \mathcal{A}_1(1^\lambda, 1^\ell, 1^d)$ where $S \subseteq [\ell]$, $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^d, S, |x|)$, we have

$$\left| \Pr\left[\mathcal{A}_2^{\mathcal{O}_{\mathsf{real}}(\cdot,\cdot)}(\{\mathsf{s}_j\}_{j\in S}) = 1\right] \Pr\left[\mathcal{A}_2^{\mathcal{O}_{\mathsf{ideal}}^{(\mathsf{strong})}(\cdot,\cdot)}(\{\tilde{\mathsf{s}}_j\}_{j\in S}) = 1\right] \right| = \mathsf{negl}(\lambda),$$

where the oracles $\mathcal{O}_{\mathsf{real}}$ and $\mathcal{O}_{\mathsf{ideal}}$ are defined as follows:

- $\mathcal{O}_{\mathsf{real}}(C, j)$: On input a circuit $C : \{0,1\}^* \to \{0,1\}$ and an index $j \in [N] \backslash S$, the oracle computes $\mathsf{p}_j \leftarrow \mathsf{Eval}(C, \mathsf{s}_j)$ and returns p_j.
- $\mathcal{O}_{\mathsf{ideal}}^{(\mathsf{strong})}(C, j)$: Throughout the experiment, the oracle keeps record of $\mathcal{A}$'s queries (C, j). For each query (C, j), the oracle invokes the simulator Sim_2 depending on $\mathcal{A}$'s previous queries.
  - If there exists an index $i \in [\ell]$ for which $\mathcal{A}$ did not previously make a query (C, i), then the oracle samples and returns $\mathsf{p}_j \xleftarrow{\mathrm{R}} \{0,1\}^t$ where t is the maximum bit length of an evaluation share.
  - Otherwise, if $\mathcal{A}$ had previously queried (C, i) for all $i \in [\ell] \backslash \{j\}$, then the oracle returns $\mathsf{p}_j \leftarrow \mathsf{Sim}_2\big(C, \{\tilde{\mathsf{s}}_j\}_{j\in[\ell]}, C(x), \{\mathsf{p}_i\}_{i\in[\ell]\backslash\{j\}}\big)$ where $\{\mathsf{p}_i\}_{i\in[\ell]\backslash\{j\}}$ denotes $\mathcal{O}_{\mathsf{ideal}}^{(\mathsf{strong})}$'s previous responses to $\mathcal{A}$'s oracle queries (C, i) for $i \in [\ell] \backslash \{j\}$.

Remark 4.5 (Standard vs. Strong Security)**.** The key distinction between the standard notion of security [BGG+18] and strong security that we define in Definition 4.4 is in the amount of information that is revealed by the evaluation shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$. The standard security definition guarantees that a single evaluation share p_i hides information about x, but not necessarily information about the output $C(x)$. Specifically, the simulator Sim_2 in Definition 4.3 requires the final message $C(x)$ as part of its input to simulate the shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$. The strong security definition, on the other hand, guarantees that a single evaluation share p_i does not reveal any information about the output $C(x)$. In fact, it guarantees that any strict subset of the shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ appear to be purely random strings. It is only when an adversary gets access to all of the evaluation shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ (and therefore, can reconstruct $C(x)$) that it learns the output $C(x)$. This property plays a key role in proving the security of our multi-authority attribute-based encryption scheme in Section 5 that must handle an adversary's adaptive key queries.

Remark 4.6 (Robustness)**.** In addition to the security requirement of Definition 4.4, we can also require that a universal thresholdizer scheme satisfies *robustness* [BGG+18]. The robustness property guarantees that any improperly (maliciously) generated evaluation share p_i can be efficiently detected by any user in the system. A universal thresholdizer scheme that satisfies robustness can be used to

provide a similar verifiability guarantee for our multi-authority attribute-based encryption scheme. Since the primary focus of this work is to construct multi-authority attributed-based encryption schemes that satisfies traditional ciphertext security, we leave the formal treatment of robustness and verifiability for universal thresholdizers and multi-authority attribute-based encryption schemes for future work.

Remark 4.7 (Arbitrary Message Space)**.** We note that in Definition 4.1, we restrict the evaluation algorithm Eval to take in as input a circuit $C : \{0,1\}^* \to \{0,1\}$. This is done purely for simplicity, and the definition can be straightforwardly extended to support circuits of arbitrary polynomial lengths $C : \{0,1\}^* \to \{0,1\}^*$. A universal thresholdizer scheme that supports circuits with range $\{0,1\}$ can generically support circuits of arbitrary lengths $C : \{0,1\}^* \to \{0,1\}^*$ via standard techniques (i.e., parallel repetition).

4.2 Key-Homomorphic PRFs and Homomorphic Encryption

To construct a universal thresholdizer scheme that satisfies our strong notion of security (Definition 4.4), we rely on existing key-homomorphic PRFs and homomorphic encryption schemes from LWE. As we do not require the full details of these constructions, we summarize the key properties that we need from them in the following theorem.

Theorem 4.8 ([BLMR13, BP14, BGV12, GSW13])**.** *Let λ be the security parameter, (n, m, q, χ_B) be a set of parameters for the learning with errors problem, and let p be a rounding modulus such that $m = O(n \log q)$ and $q = p \cdot n^{\omega(1)}$. Then, assuming the hardness of the $\mathsf{LWE}_{n,\mathsf{poly}(n,q),q,\chi}$ problem, there exists a 1-almost key-homomorphic PRF $F : \mathcal{K} \times \{0,1\}^* \to \mathcal{Y}$ with key space $\mathcal{K} = (\mathbb{Z}_q^n, +)$, and $\mathcal{Y} = (\mathbb{Z}_p, +)$, and a homomorphic encryption scheme $\Pi_{\mathsf{HE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{Decrypt})$ for the message space $\{0,1\}^*$ with the following properties:*

- *On input 1^λ and 1^d, the homomorphic encryption setup algorithm Setup outputs a secret key $\mathsf{sk} \in \mathbb{Z}_p^n$.*
- *On input a circuit $C : \{0,1\}^* \to \{0,1\}$ of depth d and a ciphertext ct, the homomorphic evaluation algorithm Eval outputs a ciphertext $\hat{\mathsf{ct}} \in \mathbb{Z}_p^n$.*
- *The decryption algorithm takes a ciphertext $\mathsf{ct} \in \mathbb{Z}_p^n$ and proceeds in two steps:*
  1. *It first computes the inner product $\langle \mathsf{sk}, \mathsf{ct} \rangle$ which produces $y = \lfloor p/2 \rceil \cdot \mu + e \in \mathbb{Z}_p$ for some noise term $e \in [-B_{\mathsf{HE}}, B_{\mathsf{HE}}]$ where $B_{\mathsf{HE}} = B \cdot m^{O(d)}$.*
  2. *It outputs 0 if $|y| \leq \lfloor q/4 \rceil$ and outputs 1 otherwise.*

4.3 Universal Thresholdizer Construction

In this section, we construct a universal thresholdizer scheme that satisfies strong security (Definition 4.4). The construction is a simple augmentation of the universal thresholdizer construction of [BGG+18] with key-homomorphic PRFs. In the construction of Boneh et al.., the setup algorithm generates a homomorphic encryption key $\mathsf{sk}^* \in \mathbb{Z}_p^n$ and breaks it into shares $\mathsf{sk}_1, \ldots, \mathsf{sk}_\ell$ such that $\mathsf{sk}_1 + \cdots + \mathsf{sk}_\ell = \mathsf{sk}^*$. Then, a single share s_j of a message $x \in \{0,1\}^*$ is defined to consist of a homomorphic encryption of x, $\mathsf{ct} \leftarrow \mathsf{HE.Encrypt}(\mathsf{sk}^*, x)$, as well as a share of the encryption key

sk_j. The evaluation of the shares on a circuit $C : \{0,1\}^* \to \{0,1\}$ then consists of homomorphically evaluating the ciphertext $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$ and computing a "noisy" partial decryption $\mathsf{p}_j \leftarrow \langle \mathsf{s}_j, \hat{\mathsf{ct}} \rangle + \mathsf{noise}$. Then, using the linearity of inner products, the combine algorithm can decrypt the message $C(x)$ by summing up the evaluation shares p_j for $j \in [\ell]$.

This construction does not satisfy the strong requirements of Definition 4.4 because the partial decryptions $\langle \mathsf{s}_j, \hat{\mathsf{ct}} \rangle + \mathsf{noise}$ are structured elements in $\mathbb{Z}_p$. To achieve strong security, we modify the construction such that the setup algorithm additionally generates a set of (almost) key-homomorphic PRF keys $k_1, \ldots, k_\ell \in \mathbb{Z}_q^n$ such that $k_1 + \cdots + k_\ell = 0$. We then define a share s_j of a message $x \in \{0,1\}^*$ to consist of the ciphertext ct, a decryption key share sk_j, and a PRF key k_j,

$$\mathsf{s}_j = (\mathsf{ct}, \mathsf{sk}_j, k_j) \qquad j \in [\ell].$$

Now, the evaluation algorithm remains unchanged except that it additionally *blinds* the partial decryptions $\langle \mathsf{s}_j, \hat{\mathsf{ct}} \rangle + \mathsf{noise}$ with PRF evaluations $\mathsf{z}_j \leftarrow F(k_j, \hat{\mathsf{ct}})$. By the key-homomorphic property, the elements $\mathsf{z}_1, \ldots, \mathsf{z}_\ell$ are shares of zero (i.e. $\mathsf{z}_1 + \cdots + \mathsf{z}_\ell \approx 0$) and therefore, do not interfere with the correct decryption of the ciphertext by the combining algorithm. Since F is a PRF, however, each partial decryption $\langle \mathsf{sk}_j, \hat{\mathsf{ct}} \rangle + \mathsf{noise} + \mathsf{z}_j$ is now computationally indistinguishable from uniformly random elements in $\mathbb{Z}_p$.

Construction 4.9. Let λ be the security parameter, (n, m, q, χ_B) be a set of parameters for the learning with errors problem, and let p be a rounding modulus such that Our universal thresholdizer scheme relies on the following primitives that satisfy the properties of Theorem 4.8:

- Let $F : \mathbb{Z}_q^n \times \{0,1\}^* \to \mathbb{Z}_p$ be a 1-almost key-homomorphic PRF (Definition 3.8).
- Let $\Pi_{\mathsf{HE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Eval}, \mathsf{Decrypt})$ be a homomorphic encryption scheme for the message space $\{0,1\}^*$ (Definition 3.9).

Using these algorithms, we construct a universal thresholdizer $\Pi_{\mathsf{UT}} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Combine})$ for the same message space $\{0,1\}^*$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell, 1^d, x)$: On input the security parameter λ, number of shares ℓ, depth bound d, and a message $x \in \{0,1\}^*$, the setup algorithm samples ℓ keys for the homomorphic encryption scheme Π_{HE} and ℓ keys for the PRF F as follows:
  - Sample Π_{HE} keys $\mathsf{sk}_j \leftarrow \mathsf{HE.KeyGen}(1^\lambda, 1^d)$ for $j \in [\ell]$.
  - Sample $k_1, \ldots, k_{\ell-1} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$ and set $k_\ell = -(k_1 + \ldots k_{\ell-1})$.

  It then combines the keys $\mathsf{sk}^* = \mathsf{sk}_1 + \ldots + \mathsf{sk}_\ell$ and encrypts the message x under the combined key $\mathsf{ct} \leftarrow \mathsf{HE.Encrypt}(\mathsf{sk}^*, x)$. Finally, it sets each share as $\mathsf{s}_j = (\mathsf{ct}, \mathsf{sk}_j, k_j)$ for $j \in [\ell]$.
- $\mathsf{Eval}(C, \mathsf{s}_j)$: On input a circuit $C : \{0,1\}^* \to \{0,1\}$ and a share s_j, the evaluation algorithm parses $\mathsf{s}_j = (\mathsf{ct}, \mathsf{sk}_j, k_j)$, and homomorphically evaluates $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$. It then generates a smudging error term $e_j^{(\mathsf{sm})} \xleftarrow{\mathrm{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, a zero share $\mathsf{z}_j \leftarrow F(k_j, \hat{\mathsf{ct}})$, and returns $\mathsf{p}_j = \langle \mathsf{sk}_j, \hat{\mathsf{ct}} \rangle + e_j^{(\mathsf{sm})} + \mathsf{z}_j$.
- $\mathsf{Combine}(\mathsf{p}_1, \ldots, \mathsf{p}_\ell)$: On input a set of evaluation shares $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$, the combining algorithm adds $\mathsf{p}^* = \mathsf{p}_1 + \ldots + \mathsf{p}_\ell$, and returns 0 if $|\mathsf{p}^*| \leq p/4$ and returns 1 otherwise.

We now state the correctness and security theorems for Construction 4.9.

Theorem 4.10 (Correctness)**.** *Suppose that* $B \cdot m^{O(d)} + \ell \cdot B_{\mathsf{sm}} + \ell < \lfloor p/4 \rceil$*, the PRF* $F : \mathbb{Z}_q^n \times \{0,1\}^* \to \mathbb{Z}_p$ *satisfies 1-almost key-homomorphism (Definition 3.7), and the homomorphic encryption scheme* Π_{HE} *satisfies correctness (Definition 3.11). Furthermore, suppose that* F *and* Π_{HE} *satisfy the properties in Theorem 4.8. Then, the universal thresholdizer scheme in Construction 4.9 satisfies correctness (Definition 4.2).*

Theorem 4.11 (Security)**.** *Suppose that* $(B \cdot m^{O(d)} + \ell)/B_{\mathsf{sm}} = \mathsf{negl}(\lambda)$*,* $F : \mathbb{Z}_q^n \times \{0,1\}^* \to \mathbb{Z}_p$ *is a secure 1-almost key-homomorphism (Definition 3.7), and* Π_{HE} *is a secure homomorphic encryption scheme (Definition 3.12). Furthermore, suppose that* F *and* Π_{HE} *satisfy the properties in Theorem 4.8. Then, the universal thresholdizer scheme in Construction 4.9 satisfies security (Definition 4.4).*

We provide the proof of Theorems 4.10 and 4.11 in Appendix A.

Parameter instantiation. The parameters for Construction 4.9 can be set to satisfy the conditions of Theorems 4.8, 4.10, and 4.11 in a straightforward way. One possible instantiation is as follows:

- $n = \mathsf{poly}(\lambda)$,
- $\ell, d = \mathsf{poly}(\lambda)$,
- $\chi_B = D_{\mathbb{Z},\sqrt{n}}$,
- $B_{\mathsf{sm}} = B \cdot 2^{\tilde{O}(d)}$,
- $p = B \cdot 2^{\tilde{O}(d)}$,
- $q = p \cdot n^{\omega(1)}$,
- $m = \Theta(n \log q)$,

where $D_{\mathbb{Z},\sqrt{n}}$ is the discrete Gaussian distribution over the integers. An (almost) key-homomorphic PRF can be based on the LWE assumption via the constructions of [BLMR13, BP14]. A homomorphic encryption scheme can be based on the LWE assumption via the constructions of [BGV12, GSW13].

5 Multi-Authority ABE in the OT Model

In this section, we define the notion of a multi-authority attributed-based encryption (MA-ABE) scheme. Recall that in a ciphertext-policy attribute-based encryption (ABE) scheme [SW05, GPSW06], a ciphertext ct_ϕ is generated with respect to a policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$, and a decryption key sk_x is bound to an attribute string $x \in \{0,1\}^\ell$. A key sk_x can be used to decrypt a ciphertext ct_ϕ if and only if $\phi(x) = 0$.[5] In a standard ABE scheme, a single authority who holds the master secret key msk generates the whole decryption key sk_x for each user.

In a multi-authority ABE scheme [SW05], the master secret key msk can be further divided into multiple master secret keys $\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell$ that are distributed among separate key-generating authorities. An authority who holds msk_j can generate a decryption key component sk_{j,x_j} for a single attribute bit $x_j \in \{0,1\}$ *without* interacting with any other key-generating authorities. The collection of these key components that are generated by each authorities $\mathsf{sk}_x = (\mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell})$ make up a single decryption key for the attribute string $x \in \{0,1\}^\ell$ and can be used to decrypt a ciphertext ct_ϕ if and only if $\phi(x) = 0$.

[5] Like in many previous papers that treat attribute-based encryption constructions from lattices, we use $\phi(x) = 0$ to specify the condition for correct decryption as it is more natural for our algebraic construction in Section 7. This is in contrast to the traditional convention, which uses $\phi(x) = 1$ as the condition for correct decryption.

If no communication is allowed among the authorities when generating the keys, there is no way for the authorities to even agree on the specific recipient of a decryption key. Therefore, a multi-authority ABE is generally defined in the *GID* model [Cha07, LW11] where each recipient of a decryption key is identified by some *global identity string* $\mathtt{gid} \in \{0,1\}^*$. For instance, a $\mathtt{gid}$ can be a user's driver's license, student ID, or voter registration number that the recipient provides to the key-generating authorities. A key-generating authority takes in the string $\mathtt{gid}$ and produces a decryption key component $\mathsf{sk}_{\mathtt{gid},j,x_j}$ that is bound under a single $\mathtt{gid}$ and an attribute bit $x_j \in \{0,1\}$. The complete key $\mathsf{sk}_{\mathtt{gid},x} = (\mathsf{sk}_{\mathtt{gid},1,x_1}, \ldots, \mathsf{sk}_{\mathtt{gid},\ell,x_\ell})$ bound under a single $\mathtt{gid}$ can then be used to decrypt a ciphertext ct_ϕ if and only if $\phi(x) = 0$.

In this work, we consider a new relaxed model called the *oblivious transfer* (OT) model. The OT model is similar to the traditional GID model where individual key components are bound under some string $\mathtt{gid} \in \{0,1\}^*$. However, in contrast to the GID model where a $\mathtt{gid}$ can be an *arbitrary* string, we allow a $\mathtt{gid}$ to be a *structured* string that is generated by the key-recipients. Specifically, to receive a key sk_x for an attribute string $x \in \{0,1\}^\ell$, each key-recipient generates a formal *key-request* $\mathsf{req}_x \in \{0,1\}^*$ to provide to each of the key-generating authorities. Upon request for a decryption key req_x, each authority with msk_j generates a key component $\mathsf{sk}_{\mathsf{req}_x,j,x_j}$ to provide to the corresponding recipient. The components that are generated for the same request string req can be combined to form a complete decryption key $\mathsf{sk}_{\mathsf{req},x} = (\mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell})$. An MA-ABE scheme in the OT model must guarantee *correctness* and *ciphertext security* meaning that a key $\mathsf{sk}_{\mathsf{req}_x}$ can decrypt a ciphertext ct_ϕ encrypted under $\phi : \{0,1\}^\ell \to \{0,1\}$ if and only if $\phi(x) = 0$. Furthermore, an MA-ABE scheme in the OT model must satisfy an additional *recipient privacy* condition, which guarantees that a key-request req_x does not reveal information about an attribute string $x \in \{0,1\}^\ell$ to the authorities. As the correctness and security requirements of an MA-ABE scheme in this model are largely analogous to an *oblivious transfer protocol*, we refer to this model as the OT model.

Formally, we define the algorithm syntax of a multi-authority attribute-based encryption scheme in the OT model as follows.

Definition 5.1 (MA-ABE)**.** A *multi-authority attribute-based encryption* (MA-ABE) scheme $\Pi_{\mathsf{MA\text{-}ABE}}$ for a message space $\mathcal{M}$ and a circuit class $\mathcal{C}$ consists of a tuple of algorithms $\Pi_{\mathsf{MA\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyRequest}, \mathsf{KeyGen}, \mathsf{KeyCombine}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp}, \mathsf{msk}_1, \ldots, \mathsf{msk}_\ell)$: On input the security parameter λ and the number of authorities in the system ℓ, the setup algorithm generates a set of public parameters pp and a set of master secret keys $\mathsf{msk}_1, \ldots, \mathsf{msk}_\ell$ for the authority.

- $\mathsf{KeyRequest}_{\mathsf{pp}}(x) \to (\mathsf{req}_x, \mathsf{st})$: On input an attribute string $x \in \{0,1\}^\ell$, the key-request algorithm returns a request string $\mathsf{req} \in \{0,1\}^*$ and a recipient state st.

- $\mathsf{KeyGen}_{\mathsf{pp}}(\mathsf{msk}_j, \mathsf{req}, b) \to \mathsf{sk}_{\mathsf{req},j,b}$: On input a master secret key msk_j, a key-request $\mathsf{req} \in \{0,1\}^*$, and an attribute bit $b \in \{0,1\}$, the key generation algorithm returns a key component $\mathsf{sk}_{\mathsf{req},j,b}$.

- $\mathsf{KeyCombine}_{\mathsf{pp}}(\mathsf{st}, \mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell}) \to \mathsf{sk}_x$: On input a recipient state st, and a set of secret key components $\mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell}$, the key combining algorithm returns a key sk_x.

- $\mathsf{Encrypt}_{\mathsf{pp}}(\phi, \mu) \to \mathsf{ct}$: On input a policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$ in $\mathcal{C}$ and a message $\mu \in \mathcal{M}$, the encryption algorithm returns a ciphertext ct.

- $\mathsf{Decrypt}_{\mathsf{pp}}(\mathsf{sk}_x, \mathsf{ct}) \to \mu$: On input a decryption key sk_x and a ciphertext ct, the decryption algorithm returns a message $\mu \in \mathcal{M} \cup \{\perp\}$.

To simplify notation in the paper, we remove the public parameters pp from the algorithm syntax and assume that all algorithms excluding Setup have access to pp as part of their input.

In the description of the algorithms above, the tuple $(\mathsf{Setup}, \mathsf{Encrypt}, \mathsf{Decrypt})$ specify the standard setup, encryption, and decryption algorithms of an attribute-based encryption scheme. The tuple of algorithms $(\mathsf{KeyRequest}, \mathsf{KeyGen}, \mathsf{KeyCombine})$ specify a distributed key-generation procedure. Specifically, a recipient of a decryption key for an attribute string $x \in \{0,1\}^\ell$ generates a formal key-request by invoking the algorithm $(\mathsf{req}_x, \mathsf{st}) \leftarrow \mathsf{KeyRequest}(x)$, and each key-generating authority invokes the algorithm $\mathsf{sk}_{\mathsf{req},j,x_j} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}_x, x_j)$ to generate a single key component. These individual components can then be combined by the key-recipient $\mathsf{sk}_x \leftarrow \mathsf{KeyCombine}(\mathsf{st}, \mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell})$ to produce a complete decryption key sk_x.

We note that even though a key-recipient generates a request string req_x for an attribute string $x \in \{0,1\}^\ell$, it is up to the authorities with msk_j for $j \in [\ell]$ to determine which key component $\mathsf{sk}_{\mathsf{req},j,0}$ or $\mathsf{sk}_{\mathsf{req},j,1}$ that the recipient receives. As is formally captured by the correctness and security definitions below, if a recipient produces a request string req_x for an attribute string $x \in \{0,1\}^\ell$, but an authority produces a key component $\mathsf{sk}_{\mathsf{req},j,1-x_j} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}_x, 1-x_j)$, then $\mathsf{sk}_{\mathsf{req},j,1-x_j}$ loses any functionality as a key component. This prevents a malicious key-recipient from "tricking" a key generating authority into providing a key for an unsuitable attribute for the user.

Correctness. We define the correctness condition for an MA-ABE scheme in the natural way. Namely, the correctness condition guarantees that for properly generated key components $\mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell}$ for an attribute string $x = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$ can be combined to form a complete key sk_x that can decrypt any ciphertext encrypted under a policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$ for which $\phi(x) = 0$.

Definition 5.2 (Correctness)**.** Let $\Pi_{\mathsf{MA\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyRequest}, \mathsf{KeyGen}, \mathsf{KeyCombine}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a multi-authority attributed-based encryption scheme for a message space $\mathcal{M}$ and circuit class $\mathcal{C}$. Then, we say that $\Pi_{\mathsf{MA\text{-}ABE}}$ satisfies correctness if for all $\lambda, \ell \in \mathbb{N}$, request strings $\mathsf{req} \in \{0,1\}^*$, messages $m \in \mathcal{M}$, policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$ in $\mathcal{C}$, attribute string $x \in \{0,1\}^\ell$ for which $\phi(x) = 0$, if we set $(\mathsf{pp}, \mathsf{msk}_1, \ldots, \mathsf{msk}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $(\mathsf{req}, \mathsf{st}) \leftarrow \mathsf{KeyRequest}(x)$, $\mathsf{sk}_{\mathsf{req},j,x_j} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}, x_j)$, $\mathsf{sk}_x \leftarrow \mathsf{KeyCombine}(\mathsf{st}, \mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell})$, we have

$$\Pr\left[\mathsf{Decrypt}\big(\mathsf{sk}_x, \mathsf{Encrypt}(\phi, \mu)\big) = \mu\right] = 1 - \mathsf{negl}(\lambda).$$

Security. The security of a standard attribute-based encryption scheme guarantees that any colluding set of users, who hold decryption keys sk_x for $\phi(x) = 1$, must not be able to learn any information from any ciphertexts ct_ϕ that are bound to ϕ. In an MA-ABE scheme, we additionally take into account any colluding set of corrupt authorities who hold master secret keys. To define the security requirement formally, we first define the MA-ABE ciphertext security experiment as follows.

Definition 5.3 (Ciphertext Security Experiment)**.** Let $\Pi_{\mathsf{MA\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyRequest}, \mathsf{KeyGen}, \mathsf{KeyCombine}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a multi-authority attribute-based encryption scheme for a message space $\mathcal{M}$ and circuit class $\mathcal{C}$. Then, for $\lambda, \ell \in \mathbb{N}$, adversary $\mathcal{A}$, and $\beta \in \{0,1\}$, we define the MA-ABE *ciphertext security* experiment $\mathsf{Expt}^{\mathsf{CS}}_{\Pi_{\mathsf{MA\text{-}ABE}}}(\lambda, \ell, \mathcal{A}, \beta)$ between an adversary $\mathcal{A}$ and a challenger as follows:

$\mathsf{Expt}^{\mathsf{CS}}_{\Pi_{\mathsf{MA\text{-}ABE}}}(\lambda, \ell, \mathcal{A}, \beta)$:

- **Setup phase**: At the start of the experiment, the adversary $\mathcal{A}$ specifies a set of corrupt authorities $S \subsetneq [\ell]$ to the challenger. The challenger then generates the public parameters and the master secret keys $(\mathsf{pp}, \mathsf{msk}_1, \ldots, \mathsf{msk}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, and provides pp and $\{\mathsf{msk}_j\}_{j \in S}$ to $\mathcal{A}$.
- **Query phase**: The adversary $\mathcal{A}$ can make two types of (adaptive) oracle queries to the challenger:
  - *Key queries*: The adversary $\mathcal{A}$ makes a polynomial number of (adaptive) key queries. Each query consists of a tuple (req, j, b) for a request string $\mathsf{req} \in \{0,1\}^*$, authority index $j \in [\ell]$, and attribute bit $b \in \{0,1\}$. For each query, the challenger generates $\mathsf{sk}_{\mathsf{req},j,b} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}, b)$ and provides $\mathsf{sk}_{\mathsf{req},j,b}$ to $\mathcal{A}$.
  - *Challenge query*: The adversary $\mathcal{A}$ makes a single challenge query of the form (ϕ, μ_0, μ_1) consisting of a policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$. The challenger computes $\mathsf{ct}_\beta \leftarrow \mathsf{Encrypt}(C, \mu_\beta)$ and provides ct_β to $\mathcal{A}$.
- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a guess $\beta' \in \{0,1\}$. The challenger outputs β' as the output of the experiment.

Since an adversary in the MA-ABE security experiment gets access to master secret keys, formally defining security requires some care. At the intuitive level, we require that an MA-ABE scheme satisfies "best possible" security against an adversary who holds a set of master secret keys and decryption key components. The functionality of an MA-ABE scheme inherently allows an adversary $\mathcal{A}$, who holds a master secret key msk_j, to generate both decryption key components $\mathsf{sk}_{\mathsf{req},j,0} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}, 0)$, and $\mathsf{sk}_{\mathsf{req},j,1} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}, 1)$ for any request string $\mathsf{req} \in \{0,1\}^*$. Therefore, we view an adversary $\mathcal{A}$ who holds a master secret key msk_j to have in possession (in addition to the key components that it receives from the key queries) an infinite set of decryption key components $\{\mathsf{sk}_{\mathsf{req},j,0}, \mathsf{sk}_{\mathsf{req},j,1}\}_{\mathsf{req} \in \{0,1\}^*}$. Then, we say that an MA-ABE scheme is secure if an adversary $\mathcal{A}$ cannot learn any information from a ciphertext ct_ϕ as long as $\mathcal{A}$ cannot form a complete set of decryption key components $\mathsf{sk}_{\mathsf{req},x} = (\mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell})$, for an attribute string $x = (x_1, \ldots, x_\ell)$ for which $\phi(x) = 0$, by combining all the decryption key components that it has in its possession: the components that it can derive using its own set of master secret keys and the components that it receives from the key queries.

Although this security requirement is natural and intuitive, defining it formally can be quite cumbersome. To simplify the notation, we use special *wildcard* notations that we define as follows.

Definition 5.4. Let $x \in \{0,1,\star\}^\ell$ be a string with wildcards and $\phi : \{0,1\}^\ell \to \{0,1\}$ be a circuit. Then, we say that x satisfies ϕ, denoted $\phi(x) = 0$, if there exists a string $\hat{x} \in \{0,1\}^\ell$ such that

1. $\phi(\hat{x}) = 0$,
2. $x_j = \hat{x}_j$ for all indices $j \in [\ell]$ where $x_j \neq \star$.

Using this notation, we define the ciphertext security requirement for an MA-ABE scheme as follows.

Definition 5.5 (Ciphertext Security)**.** Let $\Pi_{\mathsf{MA\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyRequest}, \mathsf{KeyGen}, \mathsf{KeyCombine}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a multi-authority attribute-based encryption scheme for a message space $\mathcal{M}$, let $\mathcal{A}$ be an adversary, and let $\mathsf{Expt}^{\mathsf{CS}}_{\Pi_{\mathsf{MA\text{-}ABE}}}(\lambda, \ell, \mathcal{A}, \beta)$ be the ciphertext security experiment as defined in Definition 5.3. Then, for each $\mathsf{req} \in \{0,1\}^\ell$, we assign an attribute vector with wildcards $x_{\mathsf{req}} \in \{0,1,\star\}^\ell \cup \{\perp\}$ according to $\mathcal{A}$'s queries during the query phase of the experiment:

- We set $x_{\mathsf{req}} = \perp$ if there exists an index $j \in [\ell]$ for which $j \notin S$, but $\mathcal{A}$ does not make any key query $(\mathsf{req}, j, 0)$ or $(\mathsf{req}, j, 1)$.
- Otherwise, we set each component of $x_{\mathsf{req}} = (x_{\mathsf{req},1}, \ldots, x_{\mathsf{req},\ell})$ as follows:
  - We set $x_{\mathsf{req},j} = 0$ if $j \notin S$ and $\mathcal{A}$ makes a key query $(\mathsf{req}, j, 0)$, but not $(\mathsf{req}, j, 1)$.
  - We set $x_{\mathsf{req},j} = 1$ if $j \notin S$ and $\mathcal{A}$ makes a key query $(\mathsf{req}, j, 1)$, but not $(\mathsf{req}, j, 0)$.
  - We set $x_{\mathsf{req},j} = \star$ if either $j \in S$ or $\mathcal{A}$ makes both key queries $(\mathsf{req}, j, 0)$ and $(\mathsf{req}, j, 1)$.

Then, we say that an adversary $\mathcal{A}$ is *admissible* if $\phi(x_{\mathsf{req}}) = 1$ for all $\mathsf{req} \in \{0,1\}^*$ for which $x_{\mathsf{req}} \neq \perp$. We say that a multi-authority attribute-based encryption scheme $\Pi_{\mathsf{MA\text{-}ABE}}$ satisfies *ciphertext security* if for all $\lambda, \ell \in \mathbb{N}$ and (efficient) admissible adversaries $\mathcal{A}$, we have

$$\left|\Pr\left[\mathsf{Expt}^{\mathsf{CS}}_{\Pi_{\mathsf{MA\text{-}ABE}}}(\lambda, \ell, \mathcal{A}, 0) = 1\right] - \Pr\left[\mathsf{Expt}^{\mathsf{CS}}_{\Pi_{\mathsf{MA\text{-}ABE}}}(\lambda, \ell, \mathcal{A}, 1) = 1\right]\right| = \mathsf{negl}(\lambda).$$

Remark 5.6 (Selective Security)**.** In the ciphertext security experiment of Definition 5.3, the adversary $\mathcal{A}$ submits the policy circuit $\phi \in \mathcal{C}$ for the challenge ciphertext *after* it receives the public parameters and the challenger's responses to its key queries. An MA-ABE scheme that satisfies ciphertext security with respect to this experiment is often referred to as satisfying *adaptive* ciphertext security. We can define a weaker (but still useful) ciphertext security experiment that is identical to Definition 5.3, but where the adversary $\mathcal{A}$ must declare the challenge policy circuit $\phi \in \mathcal{C}$ *before* the start of the experiment. An MA-ABE scheme that satisfies security with respect to this weaker experiment is said to satisfy *selective* ciphertext security. Our MA-ABE construction in Section 6 will be shown to satisfy selective security. An MA-ABE scheme that satisfies selective security can be shown to satisfy adaptive security via complexity leveraging [BB04] albeit with a subexponential loss in the reduction.

We note that one of the main motivations for a multi-authority ABE scheme is *privacy* of attributes. Namely, a key-recipient with an attribute $x_j \in \{0,1\}$ for some $j \in [\ell]$ must not necessarily reveal x_j to any other key-generating authority other than the authority who holds msk_j. In the GID model, each key-generating authority is only provided a global identity string $\mathtt{gid}$ of the user, which is completely independent of a user's attribute. Therefore, in the GID model, the privacy of a key-recipient's attribute string $x \in \{0,1\}^*$ is captured innately. In the OT model, however, we allow a user to submit a request string req_x that can encode the entire attribute string $x \in \{0,1\}^\ell$. Therefore, to guarantee the privacy of a user's attribute, we must specifically define a *receiver privacy* requirement. We formally define receiver privacy for MA-ABE in the OT model as follows.

Definition 5.7 (Receiver Privacy)**.** Let $\Pi_{\mathsf{MA\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyRequest}, \mathsf{KeyGen}, \mathsf{KeyCombine}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a multi-authority attribute-based encryption scheme in the OT model for a message space $\mathcal{M}$ and circuit class $\mathcal{C}$. Then, we say that $\Pi_{\mathsf{MA\text{-}ABE}}$ satisfies *receiver privacy* if for all efficient adversaries $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $(\mathsf{pp}, \mathsf{msk}_1, \ldots, \mathsf{msk}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $(x_0, x_1) \leftarrow \mathcal{A}_1(\mathsf{pp}, \mathsf{msk}_1, \ldots, \mathsf{msk}_\ell)$, and $(\mathsf{st}_\beta, \mathsf{req}_\beta) \leftarrow \mathsf{KeyRequest}(x_\beta)$ for $\beta \in \{0,1\}$, we have

$$\left|\Pr[\mathcal{A}_2(\mathsf{req}_0) = 1] - \Pr[\mathcal{A}_2(\mathsf{req}_1) = 1]\right| = \mathsf{negl}(\lambda).$$

Remark 5.8 (Comparison to GID Model)**.** We note that the OT model is a relaxation of the traditional GID model where each key-recipient is identified by a global string $\mathtt{gid} \in \{0,1\}^*$. Namely,

if we restrict the request algorithm $(\mathsf{req}, \mathsf{st}) \leftarrow \mathsf{KeyRequest}(x)$ to set $\mathsf{req} = \mathtt{gid}$ and $\mathsf{st} = \varepsilon$, then the syntax of the MA-ABE algorithms in the OT model precisely matches those of the MA-ABE algorithms in the GID model. The main conceptual difference between the two models is in the way the string $\mathsf{req} = \mathtt{gid}$ depends on an attribute. In the GID model, the request string $\mathsf{req} = \mathtt{gid}$ is required to be completely *independent* of any attribute string $x \in \{0,1\}^\ell$. In the OT model, we require the request string req_x to be *computationally* independent of the attribute string x (as guaranteed by the receiver privacy condition in Definition 5.7), but still allow req_x to depend on x at an information theoretic level. Since a request string req_x can depend now depend on x, it can be used as a key-recipient's binding commitment to a single attribute string x. It is this property that we take advantage of to construct our MA-ABE scheme in the OT model in Sections 6 and 7.

6 Decomposable ABE

In this section, we define the notion of a *decomposable* attribute-based encryption scheme. We use this notion as a stepping stone for constructing a full MA-ABE scheme. We first provide the formal definitions of a decomposable ABE scheme in Section 6.1. Then, in Section 6.2, we show how to combine a decomposable ABE scheme with a universal thresholdizer to construct an MA-ABE scheme in the OT model.

6.1 Definitions

A decomposable attribute-based encryption scheme can be viewed as a standard ABE scheme with an additional decomposability property on the decryption keys. The syntax of the algorithms for a decomposable ABE scheme largely remains identical to a standard ABE scheme. In particular, the master secret key msk is still a single object that is required for generating an entire decryption key sk_x for an attribute string $x = (x_1, \ldots, x_\ell) \in \{0,1\}^\ell$. The only additional property that we require from a decomposable ABE scheme is that the decryption keys sk_x are naturally decomposable into multiple components $\mathsf{sk}_0, \mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell}$. The components sk_{x_j} for $j \in [\ell]$ are part of the key sk_x that correspond to each bit of the attribute string $x_j \in \{0,1\}$. The component sk_0 is a part of the key that is independent of the attribute string $x \in \{0,1\}^\ell$ and its only functionality is to bind the attribute components $\mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell}$ together. We refer to the key component sk_0 as the *binding component* of the key and the key components $\mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell}$ as the *attribute components*.

To enforce this decomposability property in the definition, we define the (randomized) key generation algorithm to take in a master secret key msk and return a binding component sk_0 and *all possible* attribute components

$$\begin{pmatrix} \mathsf{sk}_{1,0} & \mathsf{sk}_{2,0} & \cdots & \mathsf{sk}_{\ell-1,0} & \mathsf{sk}_{\ell,0} \\ \mathsf{sk}_{1,1} & \mathsf{sk}_{2,1} & & \mathsf{sk}_{\ell-1,1} & \mathsf{sk}_{\ell,1} \end{pmatrix}.$$

An ABE decryption key for an attribute string $x \in \{0,1\}^\ell$ is then defined to be the collection of key components $\mathsf{sk}_x = (\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell})$.

Formally, we define the algorithms of a decomposable ABE scheme as follows.

Definition 6.1 (Decomposable Attribute-Based Encryption)**.** A *decomposable attributed-based encryption* (DABE) scheme Π_{DABE} for a message space $\mathcal{M}$ and a circuit class $\mathcal{C}$ consists of a tuple of algorithms $\Pi_{\mathsf{DABE}} = (\mathsf{DABE.Setup}, \mathsf{DABE.KeyGen}, \mathsf{DABE.Encrypt}, \mathsf{DABE.Decrypt})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, 1^\ell) \to (\mathsf{pp}, \mathsf{msk})$: On input the security parameter λ and the number of attributes ℓ, the setup algorithm generates a set of public parameters pp and a master secret key msk.
- $\mathsf{KeyGen}_{\mathsf{pp}}(\mathsf{msk}) \to \left(\mathsf{sk}_0, \{\mathsf{sk}_{j,b}\}_{j\in[\ell],b\in\{0,1\}}\right)$: On input a master secret key msk, the key generation algorithm returns a set of key components

$$\begin{pmatrix} \mathsf{sk}_0 & \mathsf{sk}_{1,0} & \mathsf{sk}_{2,0} & \dots & \mathsf{sk}_{\ell,0} \\ & \mathsf{sk}_{1,1} & \mathsf{sk}_{2,1} & & \mathsf{sk}_{\ell,1} \end{pmatrix}.$$

- $\mathsf{Encrypt}_{\mathsf{pp}}(\phi, \mu) \to \mathsf{ct}$: On input a policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$ in $\mathcal{C}$ and a message $\mu \in \mathcal{M}$, the encryption algorithm returns a ciphertext ct.
- $\mathsf{Decrypt}_{\mathsf{pp}}(\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \dots, \mathsf{sk}_{\ell,x_\ell}, \mathsf{ct}) \to \mu$: On input a set of secret key components $\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \dots, \mathsf{sk}_{\ell,x_\ell}$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M} \cup \{\perp\}$.

To simplify notation in the paper, we remove the public parameters pp from the algorithm syntax and assume that all algorithms excluding Setup have access to pp as part of their input.

Correctness. The correctness requirement for a DABE scheme is identical to the correctness requirement for a standard ABE scheme. Namely, for any properly generated set of key components $\left(\mathsf{sk}_0, \{\mathsf{sk}_{j,b}\}_{j\in[\ell],b\in\{0,1\}}\right) \leftarrow \mathsf{KeyGen}(\mathsf{msk})$, the collection of components $\mathsf{sk}_x = (\mathsf{sk}_0, \mathsf{sk}_{x_1}, \dots, \mathsf{sk}_{x_\ell})$ must decrypt ciphertexts ct_ϕ for any policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$ in $\mathcal{C}$ for which $\phi(x) = 0$.

Definition 6.2 (Correctness)**.** Let $\Pi_{\mathsf{DABE}} = (\mathsf{DABE.Setup}, \mathsf{DABE.KeyGen}, \mathsf{DABE.Encrypt}, \mathsf{DABE.Decrypt})$ be a decomposable attribute-based encryption scheme for a message space $\mathcal{M}$ and circuit class $\mathcal{C}$. Then, we say that Π_{DABE} satisfies correctness if for all $\lambda, \ell \in \mathbb{N}$, $\mu \in \mathcal{M}$, $\phi : \{0,1\}^\ell \to \{0,1\}$ in $\mathcal{C}$, and $x \in \{0,1\}^\ell$ for which $\phi(x) = 0$, if we set $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $\left(\mathsf{sk}_0, \{\mathsf{sk}_{j,b}\}_{j\in[\ell],b\in\{0,1\}}\right) \leftarrow \mathsf{KeyGen}_{\mathsf{pp}}(\mathsf{msk})$, and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\phi, \mu)$, we have

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \dots, \mathsf{sk}_{\ell,x_\ell}, \mathsf{ct}) = \mu\right] = 1 - \mathsf{negl}(\lambda).$$

Security. The security requirement for a DABE scheme is defined identically to the security requirement for a standard ABE scheme. As in the standard ABE security definition, an adversary who holds a decryption key $\mathsf{sk}_x = (\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \dots, \mathsf{sk}_{\ell,x_\ell})$ should not be able to learn any information from a ciphertext ct_ϕ for which $\phi(x) = 1$.

Definition 6.3 (Security Experiment)**.** Let $\Pi_{\mathsf{DABE}} = (\mathsf{DABE.Setup}, \mathsf{DABE.KeyGen}, \mathsf{DABE.Encrypt}, \mathsf{DABE.Decrypt})$ be a decomposable attribute-based encryption scheme for a message space $\mathcal{M}$ and circuit class $\mathcal{C}$. Then, for $\lambda, \ell \in \mathbb{N}$, adversary $\mathcal{A}$, and $\beta \in \{0,1\}$, we define the decomposable attribute-based encryption security experiment $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, \beta)$ between an adversary $\mathcal{A}$ and a challenger as follows:

$\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, \beta)$:

- **Setup phase**: At the start of the experiment, the challenger generates the public parameters and master secret key $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$ and provides pp to $\mathcal{A}$.
- **Query phase**: The adversary $\mathcal{A}$ can make two types of (adaptive) oracle queries to the challenger:

- *Key queries*: The adversary $\mathcal{A}$ makes a polynomial number of (adaptive) key queries. Each query consists of an attribute string $x \in \{0,1\}^\ell$. For each query, the challenger invokes $(\mathsf{sk}_0, \{\mathsf{sk}_{j,b}\}_{j\in[\ell],b\in\{0,1\}}) \leftarrow \mathsf{KeyGen}(\mathsf{msk})$ and provides $\mathsf{sk}_x = (\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell})$ to $\mathcal{A}$.
- *Challenge query*: The adversary $\mathcal{A}$ makes a single challenge query of the form (ϕ, μ_0, μ_1) consisting of a policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$, and a pair of messages $\mu_0, \mu_1 \in \mathcal{M}$. The challenger computes $\mathsf{ct}_\beta \leftarrow \mathsf{Encrypt}(C, \mu_\beta)$ and provides ct_β to $\mathcal{A}$.

- **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a guess $\beta' \in \{0,1\}$. The challenger outputs β' as the output of the experiment.

We define the admissibility condition of an adversary analogously to the standard ABE security definition.

Definition 6.4 (Security)**.** Let $\Pi_{\mathsf{DABE}} = (\mathsf{DABE.Setup}, \mathsf{DABE.KeyGen}, \mathsf{DABE.Encrypt}, \mathsf{DABE.Decrypt})$ be a decomposable attribute-based encryption scheme for a message space $\mathcal{M}$, let $\mathcal{A}$ be an adversary, and let $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, \beta)$ be the ciphertext security experiment as defined in Definition 6.3. We say that an adversary $\mathcal{A}$ is *admissible* if for any key query $x \in \{0,1\}^\ell$ and challenge query (ϕ, μ_0, μ_1) that it makes during the experiment, we have $\phi(x) = 1$. We say that a decomposable attribute-based encryption scheme Π_{DABE} satisfies security if for all $\lambda, \ell \in \mathbb{N}$ and (efficient) admissible adversaries $\mathcal{A}$, we have

$$\left|\Pr\left[\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 0) = 1\right] - \Pr\left[\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 1) = 1\right]\right| = \mathsf{negl}(\lambda).$$

Remark 6.5 (Selective Security)**.** As in the case of MA-ABE (Remark 5.6), we can define a *selective* variant of the DABE security experiment. Here, the adversary must declare a policy ϕ for the challenge ciphertext *before* the start of the experiment. Our DABE constructions in Section 7 will be shown to be secure in the weaker selective security. Any DABE scheme that satisfies selective security can be shown to satisfy full adaptive security via complexity leveraging [BB04] albeit with a subexponential loss in the reduction.

6.2 DABE to MA-ABE in the OT Model

In this section, we construct an MA-ABE scheme in the OT model by combining a DABE scheme with a universal thresholdizer scheme (Definition 4.1) and an oblivious transfer protocol (Definition 3.13). The high level intuition for the construction is simple. The encryption algorithm for the MA-ABE scheme remains identical to the DABE encryption algorithm. During setup, the key-generating authorities of the MA-ABE scheme are provided UT *shares* of a DABE master secret key $\mathsf{dabe.msk}$. The key request algorithm for $x \in \{0,1\}^\ell$ simply consists of ℓ OT (receiver) messages corresponding to each of the attribute bits $x = (x_1, \ldots, x_\ell)$. Upon a key-request, an authority with share s_j homomorphically evaluates the key generation algorithm to produce shares

$$\begin{pmatrix} \mathsf{p}_0^{(\mathsf{bind})} & \mathsf{p}_{1,0}^{(\mathsf{att})} & \mathsf{p}_{2,0}^{(\mathsf{att})} & \cdots & \mathsf{p}_{\ell-1,0}^{(\mathsf{att})} & \mathsf{p}_{\ell,0}^{(\mathsf{att})} \\ & \mathsf{p}_{1,1}^{(\mathsf{att})} & \mathsf{p}_{2,1}^{(\mathsf{att})} & & \mathsf{p}_{\ell-1,1}^{(\mathsf{att})} & \mathsf{p}_{\ell,1}^{(\mathsf{att})} \end{pmatrix},$$

where each shares correspond to the decryption key components

$$\begin{pmatrix} \mathsf{sk}_0 & \mathsf{sk}_{1,0} & \mathsf{sk}_{2,0} & \cdots & \mathsf{sk}_{\ell-1,0} & \mathsf{sk}_{\ell,0} \\ & \mathsf{sk}_{1,1} & \mathsf{sk}_{2,1} & & \mathsf{sk}_{\ell-1,1} & \mathsf{sk}_{\ell,1} \end{pmatrix}.$$

Then, it defines the key component $\mathsf{sk}_{\mathsf{req},j,b}$ to consist of the evaluation shares p_0 and $\mathsf{p}_{j,b}$, as well as the OT encodings of the pairs of evaluation shares $\{(\mathsf{p}_{i,0}, \mathsf{p}_{i,1})\}_{i\in[\ell]\setminus\{j\}}$. The key combining algorithm decodes the OT messages, combines the UT share evaluations to generate $(\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell})$. The security properties of the universal thresholdizer scheme and oblivious transfer protocol guarantees that a malicious key-recipient does not gain access to the attribute key components $\mathsf{sk}_{1,1-x_1}, \ldots, \mathsf{sk}_{\ell,1-x_\ell}$. The security then follows from DABE security.

Formally, we define our MA-ABE construction as follows.

Construction 6.6. Let λ be the security parameter. Our multi-authority attributed based encryption relies on the following algorithms:

- Let $\Pi_{\mathsf{DABE}} = (\mathsf{DABE.Setup}, \mathsf{DABE.KeyGen}, \mathsf{DABE.Encrypt}, \mathsf{DABE.Decrypt})$ be a decomposable attribute-based encryption scheme for a message space $\mathcal{M}$ and circuit class $\mathcal{C}$ (Definition 6.1). Furthermore, we define the following parameters:
  - Let d be the maximum depth of any circuit computing the key generation algorithm $\mathsf{DABE.KeyGen}(\cdot)$,
  - Let r be the maximum number of random bits that are required by $\mathsf{DABE.KeyGen}(\cdot)$.
  - Let t be the maximum bit length of a single decryption key component such that $|\mathsf{sk}_0|, |\mathsf{sk}_{j,b}| \leq t$ for any $j \in [\ell]$, $b \in \{0,1\}$.
- Let $\Pi_{\mathsf{UT}} = (\mathsf{UT.Setup}, \mathsf{UT.Eval}, \mathsf{UT.Combine})$ be a universal thresholdizer with input space $\{0,1\}^*$ and output space $\{0,1\}^k$ (Definition 4.1, Remark 4.7).
- Let $\Pi_{\mathsf{OT}} = (\mathsf{OT.Setup}, \mathsf{OT.KeyGen_R}, \mathsf{OT.Encrypt_S}, \mathsf{OT.Decrypt_R})$ be an oblivious transfer protocol for the message space $\mathcal{M} = \{0,1\}^*$.
- Let $F_0 : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^r$ be a PRF (Definition 3.6).
- Let $F_1 : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^{\ell t}$ be a PRF (Definition 3.6).

Using these algorithms, we construct a multi-authority attribute-based encryption scheme $\Pi_{\mathsf{MA\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyRequest}, \mathsf{KeyGen}, \mathsf{KeyCombine}, \mathsf{Encrypt}, \mathsf{Decrypt})$ in the OT model for the message space $\mathcal{M}$ and circuit class $\mathcal{C}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell)$: On input the security parameter λ, and the number of attributes ℓ, the setup algorithm samples a PRF key $\mathsf{prf.k} \xleftarrow{R} \mathcal{K}$ and also instantiates:
  - $(\mathsf{dabe.pp}, \mathsf{dabe.msk}) \leftarrow \mathsf{DABE.Setup}(1^\lambda, 1^\ell)$,
  - $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{UT.Setup}\big(1^\lambda, 1^\ell, 1^d, (\mathsf{dabe.msk}, \mathsf{prf.k})\big)$,
  - $(\mathsf{ot.crs}, \mathsf{td}) \leftarrow \mathsf{OT.Setup}(1^\lambda)$.

  It sets $\mathsf{pp} = (\mathsf{dabe.pp}, \mathsf{ot.crs})$ and $\mathsf{msk}_j = \mathsf{s}_j$ for $j \in [\ell]$.
- $\mathsf{KeyRequest}(x)$: On input an attribute string $x \in \{0,1\}^\ell$, the key request algorithm generates the OT keys $(\mathsf{ot.pk}_j, \mathsf{ot.sk}_j) \leftarrow \mathsf{OT.KeyGen_R}(\mathsf{ot.crs}, x_j)$ for all $j \in [\ell]$. It sets $\mathsf{req}_x = \{\mathsf{ot.pk}_j\}_{j\in[\ell]}$, $\mathsf{st} = \{\mathsf{ot.sk}_j\}_{j\in[\ell]}$, and returns $(\mathsf{req}_x, \mathsf{st})$.
- $\mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}, x_j)$: On input a master secret key $\mathsf{msk}_j = \{\mathsf{ot.pk}_i\}_{i\in[\ell]}$, a request req, and an attribute bit $x_j \in \{0,1\}$, the key generation algorithm first constructs $2\ell + 1$ circuits

$\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}(\mathsf{dabe.msk}, \mathsf{prf.k})$:

1. $\rho \leftarrow F_0(\mathsf{prf.k}, \mathsf{req})$.
2. $\big(\mathsf{dabe.sk}_0, \{\mathsf{dabe.sk}_{k,b}\}_{k\in[\ell],b\in\{0,1\}}\big) \leftarrow \mathsf{DABE.KeyGen}(\mathsf{dabe.msk}; \rho)$.
3. $(\alpha_{k,1}, \ldots, \alpha_{k,\ell}) \leftarrow F_1(\mathsf{prf.k}, (\mathsf{req}, k))$ for $k \in [\ell]$.
4. $\alpha_{k,0} \leftarrow \alpha_{k,1} \oplus \cdots \oplus \alpha_{k,\ell}$ for $k \in [\ell]$.
5. Return $\big(\mathsf{dabe.sk}_0 \oplus \alpha_{0,0}, \{\alpha_{k,0}\}_{k\in[\ell]}\big)$.

$\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}(\mathsf{dabe.msk}, \mathsf{prf.k})$:

1. $\rho \leftarrow F_0(\mathsf{prf.k}, \mathsf{req})$.
2. $\big(\mathsf{dabe.sk}_0, \{\mathsf{dabe.sk}_{k,b}\}_{k\in[\ell],b\in\{0,1\}}\big) \leftarrow \mathsf{DABE.KeyGen}(\mathsf{dabe.msk}; \rho)$.
3. $(\alpha_{k,1}, \ldots, \alpha_{k,\ell}) \leftarrow F_1(\mathsf{prf.k}, (\mathsf{req}, k))$ for $k \in [\ell]$.
4. $\alpha_{k,0} \leftarrow \alpha_{k,1} \oplus \cdots \oplus \alpha_{k,\ell}$ for $k \in [\ell]$.
5. Return $\big(\mathsf{dabe.sk}_{i,b} \oplus \alpha_{i,i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}}\big)$.

and evaluates each of these circuits on the share s_j:

- $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{UT.Eval}\big(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \mathsf{s}_j\big)$.
- $\mathsf{p}^{(\mathsf{att})}_{i,b,j} \leftarrow \mathsf{UT.Eval}\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}, \mathsf{s}_j\big)$ for $i \in [\ell]$, $b \in \{0,1\}$.

Then, it generates OT ciphertexts $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}\big(\mathsf{ot.crs}, \big(\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}\big), \mathsf{ot.pk}_i\big)$ for all $i \in [\ell]\setminus\{j\}$. It sets

$$\mathsf{sk}_{\mathsf{req},j,x_j} = \big(\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\setminus\{j\}}\big).$$

- $\mathsf{KeyCombine}(\mathsf{st}, \mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell})$: On input a recipient state st, and a set of secret key component $\mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell}$, the key combining algorithm first parses each key components $\mathsf{sk}_{\mathsf{req},j,x_j} = \big(\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\setminus\{j\}}\big)$ for $j \in [\ell]$. Then, it decrypts the OT messages $\mathsf{p}^{(\mathsf{att})}_{i,x_j,j} \leftarrow \mathsf{OT.Decrypt}_{\mathsf{R}}(\mathsf{ot.sk}_j, \mathsf{ot.ct}_{i,j})$ for all $i, j \in [\ell]$, and combines the UT evaluation shares

  - $\big(\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]}\big) \leftarrow \mathsf{UT.Combine}\big(\mathsf{p}^{(\mathsf{bind})}_{0,1}, \ldots, \mathsf{p}^{(\mathsf{bind})}_{0,\ell}\big)$.
  - $\big(\mathsf{dabe.sk}'_{i,x_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}}\big) \leftarrow \mathsf{UT.Combine}\big(\mathsf{p}^{(\mathsf{att})}_{i,x_i,1}, \ldots, \mathsf{p}^{(\mathsf{att})}_{i,x_i,\ell}\big)$ for $i \in [\ell]$,

  It sets

  - $\mathsf{dabe.sk}_0 = \mathsf{dabe.sk}'_0 \oplus \Big(\bigoplus_{k\in[\ell]} \alpha_{0,k}\Big)$,
  - $\mathsf{dabe.sk}_{i,x_i} = \mathsf{dabe.sk}'_{i,x_i} \oplus \Big(\bigoplus_{k\in[0,\ell]\setminus\{i\}} \alpha_{i,k}\Big)$ for $i \in [\ell]$,

  and returns $\mathsf{sk}_x = (\mathsf{dabe.sk}_0, \mathsf{dabe.sk}_{1,x_1}, \ldots, \mathsf{dabe.sk}_{\ell,x_\ell})$.

- $\mathsf{Encrypt}(\phi, \mu)$: On input a circuit $\phi : \{0,1\}^\ell \to \{0,1\}$ and a message $\mu \in \mathcal{M}$, the encryption algorithm computes $\mathsf{ct} \leftarrow \mathsf{DABE.Encrypt}(\phi, \mu)$, and returns ct.

- $\mathsf{Decrypt}(\mathsf{st}, \mathsf{sk}_x, \mathsf{ct})$: On input a state $\mathsf{st} = \{\mathsf{ot.sk}_j\}_{j\in[\ell]}$, secret key $\mathsf{sk}_x = (\mathsf{dabe.sk}_0, \mathsf{dabe.sk}_{1,x_1}, \ldots, \mathsf{dabe.sk}_{\ell,x_\ell})$, and ciphertext ct, the decryption algorithm decrypts the message $\mu \leftarrow \mathsf{DABE.Decrypt}\big(\mathsf{dabe.sk}_0, \mathsf{dabe.sk}_{1,x_1}, \ldots, \mathsf{dabe.sk}_{\ell,x_\ell}, \mathsf{ct}\big)$, and returns μ.

We now state the correctness and security theorems for Construction 6.6.

Theorem 6.7 (Correctness)**.** *Suppose that the decomposable attribute-based encryption scheme* Π_{DABE} *satisfies correctness (Definition 6.2), the universal thresholdizer* Π_{UT} *satisfies correctness (Definition 4.2), and the oblivious transfer protocol* Π_{OT} *satisfies correctness (Definition 3.14). Then the multi-authority attribute-based encryption scheme* $\Pi_{\mathsf{MA\text{-}ABE}}$ *in Construction 6.6 satisfies correctness (Definition 5.2).*

Theorem 6.8 (Ciphertext Security)**.** *Suppose that the decomposable attribute-based encryption scheme* Π_{DABE} *satisfies selective security (Definition 6.4), the universal thresholdizer* Π_{UT} *satisfies strong security (Definition 4.4), the oblivious transfer protocol* Π_{OT} *satisfies statistical sender privacy (Definition 3.16), and the PRFs* F_0 *and* F_1 *satisfy security (Definition 3.6). Then the multi-authority attribute-based encryption scheme* $\Pi_{\mathsf{MA\text{-}ABE}}$ *in Construction 6.6 satisfies selective ciphertext security in the OT model (Definition 5.5).*

Theorem 6.9 (Receiver Privacy)**.** *Suppose that the oblivious transfer protocol* Π_{OT} *satisfies receiver privacy (Definition 3.15). Then the multi-authority attribute-based encryption scheme* $\Pi_{\mathsf{MA\text{-}ABE}}$ *in Construction 6.6 satisfies receiver privacy (Definition 5.7).*

Instantiation. The UT scheme Π_{UT} and the DABE scheme Π_{DABE} that are required for Construction 6.6 can be instantiated with our constructions in Section 4 and 7. The PRF F can be instantiated from any existing lattice-based PRF (i.e. [GGM84]). The oblivious transfer protocol can be instantiated with the construction of Peikert et al. [PVW08] whose security can be based on LWE. We note that for security in the UC framework, the Peikert et al. OT construction can only be used to carry out a bounded number of OT exchanges per CRS. However, for the privacy definitions that we work with as outline in Section 3.5, the construction in [PVW08] can support an unbounded number of OT exchanges. Alternatively, we can also use the OT protocol in the recent work of Brakerski and Döttling [BD18] with complexity leveraging. The Brakerski-Döttling OT construction satisfies all of our security requirements in Section 3.5 except that the extractor in Definition 3.16 must run in super-polynomial time (hence the need for complexity-leveraging).

7 DABE for Circuits

In this section, we construct a decomposable attribute-based encryption scheme for the class of bounded depth circuits. The construction is an adaptation of the standard attribute-based encryption scheme of [BGG+14]. We first recall the matrix embeddings technique of [BGG+14] that we use in our construction description in Section 7.1. We then provide our DABE construction in Section 7.2 and prove correctness and security in Appendix C.

7.1 Matrix Embeddings

The matrix embeddings technique was first introduced by Boneh et al. [BGG+14]. A matrix embedding scheme allows one to embed a sequence of input bits $x_1, \ldots, x_\rho \in \{0,1\}$ into matrices $\mathbf{A}_1, \ldots, \mathbf{A}_\rho \in \mathbb{Z}_q^{n \times m}$ such that any circuit (of bounded depth) can be homomorphic evaluated on the encoded input bits. Since the specific implementation of the homomorphic operations are not essential to this work, we provide the basic syntax of the algorithms that we need in the theorem below. We refer to [BGG+14] for the complete details.

Theorem 7.1 (Matrix Embeddings [BGG+14])**.** *Fix a security parameter λ, and parameters n, m, q such that $m = \Omega(n \log q)$. Then, there exist a tuple of efficiently computable algorithms $(\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}, \mathsf{Eval}_{\mathsf{Sim}})$ with the following syntax:*

- $\mathsf{Eval}_{\mathsf{pk}}\big(C, (\mathbf{A}_i)_{i\in[\rho]}\big) \to \mathbf{A}_C$*: On input a circuit $C : \{0,1\}^\rho \to \{0,1\}$, and a set of matrices $(\mathbf{A}_i)_{i\in[\rho]}$ in $\mathbb{Z}_q^{n\times m}$, the $\mathsf{Eval}_{\mathsf{pk}}$ algorithm outputs a matrix $\mathbf{A}_C \in \mathbb{Z}_q^{n\times m}$.*
- $\mathsf{Eval}_{\mathsf{ct}}\big(C, x, (\mathbf{A}_i)_{i\in[\rho]}, (\mathbf{a}_i)_{i\in[\rho]}\big) \to \mathbf{a}_C$*: On input a circuit $C : \{0,1\}^\rho \to \{0,1\}$, an input $x \in \{0,1\}^\rho$, a set of matrices $(\mathbf{A}_i)_{i\in[\rho]}$ in $\mathbb{Z}_q^{n\times m}$, and a set of vectors $(\mathbf{a}_i)_{i\in[\rho]}$, the $\mathsf{Eval}_{\mathsf{ct}}$ algorithm outputs a vector $\mathbf{a}_C \in \mathbb{Z}_q^m$.*
- $\mathsf{Eval}_{\mathsf{Sim}}\big(C, \mathbf{x}, (\mathbf{A}_i)_{i\in[\rho]}, (\mathbf{R}_i)_{i\in[\rho]}\big) \to \mathbf{R}_C$*: On input a circuit $C : \{0,1\}^\rho \to \{0,1\}$, an input $x \in \{0,1\}^\rho$, a set of matrices $(\mathbf{A}_i)_{i\in[\rho]}$ in $\mathbb{Z}_q^{n\times m}$, and a set of matrices $(\mathbf{R}_i)_{i\in[\rho]}$ in $\mathbb{Z}_q^{m\times m}$, the $\mathsf{Eval}_{\mathsf{Sim}}$ algorithm outputs a vector $\mathbf{R}_C \in \mathbb{Z}_q^{m\times m}$.*

Then, for any set of matrices $(\mathbf{A}_i)_{i\in[\rho]}$ in $\mathbb{Z}_q^{n\times m}$, any input $x \in \{0,1\}^\rho$, and any boolean circuit $C : \{0,1\}^\rho \to \{0,1\}$ of depth d, the algorithms $(\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}, \mathsf{Eval}_{\mathsf{Sim}})$ satisfies the following correctness conditions.

1. *Let $(\mathbf{a}_i)_{i\in[\rho]}$ be any set of vectors of the form*

$$\mathbf{a}_i = \mathbf{s}^T(\mathbf{A}_i + x_i \cdot \mathbf{G}) + \mathbf{e}_i^T \quad \forall i \in [\rho],$$

   for some vector $\mathbf{s} \in \mathbb{Z}_q^n$ and where $\|\mathbf{e}_i\| \le B$ for all $i \in [\rho]$. Then, if we compute the matrix $\mathbf{A}_C \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(C, (\mathbf{A}_i)_{i\in[\rho]}\big)$ and the vectors $\mathbf{a}_{C,x} \leftarrow \mathsf{Eval}_{\mathsf{ct}}\big(C, x, (\mathbf{A}_i)_{i\in[\rho]}, (\mathbf{a}_i)_{i\in[\rho]}\big)$, we have that

$$\mathbf{a}_{C,x} = \mathbf{s}^T(\mathbf{A}_C + C(x) \cdot \mathbf{G}) + \mathbf{e}_{C,x}^T,$$

   for some error vector $\|\mathbf{e}_{C,x}\| \le B \cdot m^{O(d)}$.

2. *Let $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$, $(\mathbf{R}_i)_{i\in[\rho]}$ be any set of matrices such that $\mathbf{A} \cdot \mathbf{R}_i - x_i \cdot \mathbf{G} = \mathbf{A}_i$ for all $i \in [\rho]$ where $\|\mathbf{R}_i\| \le B$. Then, if we compute the matrix $\mathbf{A}_C \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(C, (\mathbf{A}_i)_{i\in[\rho]}\big)$ and $\mathbf{R}_C \leftarrow \mathsf{Eval}_{\mathsf{Sim}}\big(C, \mathbf{x}, (\mathbf{A}_i)_{i\in[\rho]}, (\mathbf{R}_i)_{i\in[\rho]}\big)$, we have that*

$$\mathbf{A} \cdot \mathbf{R}_C - C(x) \cdot \mathbf{G} = \mathbf{A}_C,$$

   and $\|\mathbf{R}_C\| \le B \cdot m^{O(d)}$.

7.2 Construction

In this section, we present our decomposable ABE scheme that supports bounded depth circuits. The construction is an adaptation of the standard attribute-based encryption scheme of [BGG+14]. Although the construction in [BGG+14] is a key-policy ABE scheme in its original form, it can be translated to a ciphertext-policy ABE scheme using universal circuits. In the translated construction, the ABE ciphertext contains a matrix-embedding of the constraint function $\phi : \{0,1\}^\ell \to \{0,1\}$ for $|\phi| = N$,

$$\big\{\mathbf{s}^T(\mathbf{A}_i + \phi_i \cdot \mathbf{G}) + \mathsf{noise}\big\}_{i\in[N]},$$

where $\mathbf{A}_1, \ldots, \mathbf{A}_N \in \mathbb{Z}_q^{n\times m}$ are public matrices and $\mathbf{s} \in \mathbb{Z}_q^n$ is a random secret vector. The decryption key $\mathsf{sk}_{\mathcal{U}_x}$ is then bound to a universal circuit $\mathcal{U}_x$, which takes in a bit representation of a circuit ϕ

and evaluates it on a hard-coded input $\mathcal{U}_x(\phi) = \phi(x)$. Algebraically, the key $\mathsf{sk}_{\mathcal{U}_x}$ is defined to be a short vector in $\mathbb{Z}_q^{2m}$ that allows decryption if and only if the homomorphic computation of the universal circuit $\mathcal{U}_x$ on the matrix embeddings of ϕ is done correctly.

In its natural algebraic form, $\mathsf{sk}_{\mathcal{U}_x}$ is not decomposable. Therefore, to construct a decomposable ABE, we include an additional set of vectors

$$\left\{\mathbf{s}^T\mathbf{B}_j + \mathsf{noise}\right\}_{j\in[\ell]}$$

in the ciphertext where $\mathbf{B}_1, \ldots, \mathbf{B}_\ell \in \mathbb{Z}_q^{n\times m}$ are another set of public matrices. Conceptually, these vectors represent the matrix encodings of yet "undetermined" set of variables in $\{0,1\}^\ell$. We then define an attribute key sk_{j,x_j} for a bit $x_j \in \{0,1\}$ to be a short matrix $\mathbf{R}_{j,x_j} \in \mathbb{Z}^{m\times m}$ that *translates* the undetermined encoding vectors $\mathbf{s}^T\mathbf{B}_j + \mathsf{noise}$ to an encoding of x_j as follows

$$\left(\mathbf{s}^T\mathbf{B}_j + \mathsf{noise}\right)\cdot\mathbf{R}_{j,x_j} \approx \mathbf{s}^T(\mathbf{C}_j + x_j\cdot\mathbf{G}) + \mathsf{noise}.$$

Finally, we define a binding key sk_0 to consist of the old ABE decryption key $\mathsf{sk}_{\mathcal{U}}$ for the universal circuit $\mathcal{U}(\phi, x) = \phi(x)$, which allows for decryption if and only if the correct homomorphic computation of the universal circuit $\mathcal{U}$ is done correctly on the encodings

$$\left\{\mathbf{s}^T(\mathbf{A}_i + \phi_i\cdot\mathbf{G}) + \mathsf{noise}\right\}_{i\in[N]} \quad \bigcup \quad \left\{\mathbf{s}^T(\mathbf{C}_j + x_j\cdot\mathbf{G}) + \mathsf{noise}\right\}_{j\in[\ell]}.$$

The decryption algorithm now works by first translating a ciphertext to valid encodings of the tuple $(\phi, x) \in \{0,1\}^N \times \{0,1\}^\ell$ using the attribute keys. It then decrypts the message using $\mathsf{sk}_{\mathcal{U}}$ as in the original ciphertext-policy attribute-based encryption scheme.

Construction 7.2. Let λ be the security parameter, and (n, m, q, χ) be a set of LWE parameters. Our decomposable attribute-based encryption scheme relies on the following set of algorithms:

- Let $(\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}}, \mathsf{Eval}_{\mathsf{Sim}})$ be the set of matrix embeddings algorithms as in Theorem 7.1.
- Let $(\mathsf{TrapGen}, \mathsf{Invert}, \mathsf{Sample})$ and $(\mathsf{SampleLeft}, \mathsf{SampleRight})$ be the trapdoor algorithms from Theorems 3.4 and 3.5.

Using these algorithms, we construct a decomposable ABE scheme $\Pi_{\mathsf{DABE}} = (\mathsf{DABE.Setup}, \mathsf{DABE.KeyGen}, \mathsf{DABE.Encrypt}, \mathsf{DABE.Decrypt})$ for the message space $\mathcal{M} = \{0,1\}$ and the class of bounded depth circuits $\mathcal{C}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^\ell, 1^d)$: On input the security parameter λ, number of attributes ℓ, and a depth bound d, the setup algorithm first samples a set of trapdoor matrices $(\mathbf{A}, \mathbf{A}^{-1}) \leftarrow \mathsf{TrapGen}(1^\lambda)$, and $(\mathbf{B}_j, \mathbf{B}_j^{-1}) \leftarrow \mathsf{TrapGen}(1^\lambda)$ for $j \in [\ell]$. It additionally samples a set of uniformly random matrices and vectors $\mathbf{A}_1, \ldots \mathbf{A}_N \xleftarrow{R} \mathbb{Z}_q^{n\times m}$, $\mathbf{d} \xleftarrow{R} \mathbb{Z}_q^n$. Finally, sets

$$\mathsf{pp} = \left(\mathbf{A}, \{\mathbf{A}_i\}_{i\in[N]}, \{\mathbf{B}_j\}_{j\in[\ell]}, \mathbf{d}\right), \quad \mathsf{msk} = \left(\mathbf{A}^{-1}, \{\mathbf{B}_j^{-1}\}_{j\in[\ell]}\right).$$

- $\mathsf{KeyGen}(\mathsf{msk})$: On input a master secret key $\mathsf{msk} = \mathbf{A}^{-1}, \{\mathbf{B}_j^{-1}\}_{j\in[\ell]}$, the key generation algorithm samples random matrices $\mathbf{C}_1, \ldots, \mathbf{C}_\ell \xleftarrow{R} \mathbb{Z}_q^{n\times m}$. Then, it uses the trapdoor information $\mathbf{B}_1^{-1}, \ldots, \mathbf{B}_\ell^{-1}$ to sample short preimages $\mathbf{R}_{j,b} \leftarrow \mathsf{Invert}(\mathbf{B}_j, \mathbf{B}_j^{-1}, \mathbf{C}_j + b\cdot\mathbf{G})$ for all $j \in [\ell]$, $b \in \{0,1\}$ such that

$$\mathbf{B}_j\cdot\mathbf{R}_{j,b} = \mathbf{C}_j + b\cdot\mathbf{G} \qquad \forall\ j \in [\ell], b \in \{0,1\},$$

and sets $\mathsf{sk}_{j,b} = (\mathbf{C}_j, \mathbf{R}_{j,b})$ for $j \in [\ell]$, $b \in \{0,1\}$. Then, the key generation algorithm computes $\mathbf{A}_{\mathcal{U}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(\mathcal{U}, \big((\mathbf{A}_i)_{i\in[N]}, (\mathbf{C}_j)_{j\in[\ell]}\big)\big)$, and samples a short preimage $\mathbf{u} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}^{-1}, \mathbf{A}_{\mathcal{U}}, \mathbf{d}, \gamma)$ for $\gamma = Bm^{O(d)}$. Finally, it sets $\mathsf{sk}_0 = \mathbf{u}$ and returns $(\mathsf{sk}_0, \{\mathsf{sk}_{j,b}\}_{j\in[\ell],b\in\{0,1\}})$.

- $\mathsf{Encrypt}(\phi, \mu)$: On input a policy circuit $\phi : \{0,1\}^\ell \to \{0,1\}$ of size $|\phi| = N$, and a message $\mu \in \mathcal{M}$, the encryption algorithm first samples a random vector $\mathbf{s} \xleftarrow{\mathrm{R}} \mathbb{Z}_q^n$, error vectors $\mathbf{e}_{\mathbf{A}} \leftarrow \chi^m$, $e_{\mathbf{d}} \leftarrow \chi$, and short matrices $\mathbf{S}_1, \ldots, \mathbf{S}_N \xleftarrow{\mathrm{R}} \{0,1\}^{m\times m}$, $\widetilde{\mathbf{R}}_1, \ldots, \widetilde{\mathbf{R}}_\ell \xleftarrow{\mathrm{R}} \{0,1\}^{m\times m}$. It then computes the ciphertext vectors

  - $\mathbf{a} = \mathbf{s}^T\mathbf{A} + \mathbf{e}_{\mathbf{A}}^T$,
  - $\mathbf{a}_i = \mathbf{s}^T(\mathbf{A}_i + \phi_i \cdot \mathbf{G}) + \mathbf{e}_{\mathbf{A}}^T \cdot \mathbf{S}_i$ for $i \in [N]$,
  - $\mathbf{b}_j = \mathbf{s}^T\mathbf{B}_j + \mathbf{e}_{\mathbf{A}}^T \cdot \widetilde{\mathbf{R}}_j$ for $j \in [\ell]$,
  - $d = \mathbf{s}^T\mathbf{d} + e_{\mathbf{d}} + \lfloor q/2 \rceil \cdot \mu$.

  and sets $\mathsf{ct} = \big(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d, \phi\big)$.

- $\mathsf{Decrypt}(\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell}, \mathsf{ct})$: On input a set of secret keys $\mathsf{sk}_0 = \mathbf{u}$, $\mathsf{sk}_{1,x_1} = (\mathbf{C}_1, \mathbf{R}_1), \ldots, \mathsf{sk}_{\ell,x_\ell} = (\mathbf{C}_\ell, \mathbf{R}_\ell)$, and a ciphertext $\mathsf{ct} = \big(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d, \phi\big)$, the decryption algorithm first computes $\mathbf{c}_j = \mathbf{b}_j^T\mathbf{R}_j$ for all $j \in [\ell]$. Then, it evaluates $\mathbf{a}_{\mathcal{U}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}\big(\mathcal{U}, (\phi, x), \big((\mathbf{a}_i)_{i\in[N]}, (\mathbf{c}_j)_{j\in[\ell]}\big)\big)$, and $\hat{\mu} \leftarrow d - (\mathbf{a}^T \| \mathbf{a}_{\mathcal{U}}^T) \cdot \mathbf{u}$. Finally, the decryption algorithm returns 0 if $|\hat{\mu}| \le \lfloor q/4 \rceil$ and it returns 1 otherwise.

We now state the correctness and security statements for the decomposable ABE scheme of Construction 7.2.

Theorem 7.3 (Correctness)**.** *Let λ be the security parameter and (n, m, q, χ_B) be a set of LWE parameters such that $B^2 m^{O(d)} < q$. Then, the decomposable attribute-based encryption scheme Π_{DABE} in Construction 7.2 satisfies correctness.*

Theorem 7.4 (Security)**.** *Let λ be the security parameter and (n, m, q, χ_B) be a set of LWE parameters such that $m = \Omega(n \log q)$ and $q = B \cdot m^{O(d)}$. Then, assuming the hardness of the $\mathsf{LWE}_{n,m+1,q,\chi}$ problem, the decomposable attribute-based encryption scheme Π_{DABE} from Construction 7.2 satisfies selective, weak DABE security (Definition 6.4).*

We provide the proofs for Theorems 7.3 and 7.4 in Appendix C.

Parameter instantiation. The parameters for Construction 7.2 can be set to satisfy the requirements of Theorems 7.1, 7.3, and 7.4 similarly to [BGG+14]. One way of instantiating the parameters is as follows:

- $n = \mathsf{poly}(\lambda)$,
- $\ell, d = \mathsf{poly}(\lambda)$,
- $\chi_B = D_{\mathbb{Z},\sqrt{n}}$,
- $q = B^2 \cdot 2^{\tilde{O}(d)}$,
- $m = \Theta(n \log q)$,

where $D_{\mathbb{Z},\sqrt{n}}$ is the discrete Gaussian distribution over the integers.

Acknowledgments

This work was funded by NSF, DARPA, a grant from ONR, and the Simons Foundation. Opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA.

References

[ABB10] Shweta Agrawal, Dan Boneh, and Xavier Boyen. Efficient lattice (H)IBE in the standard model. In *EUROCRYPT*, 2010.

[ACPS09] Benny Applebaum, David Cash, Chris Peikert, and Amit Sahai. Fast cryptographic primitives and circular-secure encryption based on hard learning problems. In *CRYPTO*, 2009.

[AJLA+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *EUROCRYPT*, 2012.

[Ajt99] Miklós Ajtai. Generating hard instances of the short basis problem. In *ICALP*, 1999.

[AP09] Joël Alwen and Chris Peikert. Generating shorter bases for hard random lattices. In *STACS*, 2009.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, 2004.

[BD18] Zvika Brakerski and Nico Döttling. Two-message statistically sender-private ot from lwe. In *TCC*, 2018.

[BF01] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, 2001.

[BGG+14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, 2014.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter MR Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In *CRYPTO*, 2018.

[BGV12] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *ITCS*, 2012.

[BLMR13] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *CRYPTO*. 2013.

[BLP+13] Zvika Brakerski, Adeline Langlois, Chris Peikert, Oded Regev, and Damien Stehlé. Classical hardness of learning with errors. In *STOC*, 2013.

[Boy13] Xavier Boyen. Attribute-based functional encryption on lattices. In *TCC*. 2013.

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO*, 2014.

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *TCC*, 2011.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *ITCS*, 2014.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, 2001.

[CC09] Melissa Chase and Sherman SM Chow. Improving privacy and security in multi-authority attribute-based encryption. In *CCS*, 2009.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, 2007.

[CHKP10] David Cash, Dennis Hofheinz, Eike Kiltz, and Chris Peikert. Bonsai trees, or how to delegate a lattice basis. In *EUROCRYPT*, 2010.

[Coc01] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA International Conference on Cryptography and Coding*, 2001.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM journal on computing*, 38(1):97–139, 2008.

[GGM84] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions (extended abstract). In *FOCS*, 1984.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *CCS*, 2006.

[GPV08] Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In *STOC*, 2008.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *CRYPTO*. 2013.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, 2013.

[GVW15] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Predicate encryption for circuits from LWE. In *CRYPTO*, 2015.

[HW13] Susan Hohenberger and Brent Waters. Attribute-based encryption with fast decryption. In *PKC*. 2013.

[ILL89] Russell Impagliazzo, Leonid A Levin, and Michael Luby. Pseudo-random generation from one-way functions. In *STOC*, 1989.

[LOS+10] Allison Lewko, Tatsuaki Okamoto, Amit Sahai, Katsuyuki Takashima, and Brent Waters. Fully secure functional encryption: Attribute-based encryption and (hierarchical) inner product encryption. In *EUROCRYPT*, 2010.

[LW11] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, 2011.

[LW15] Vadim Lyubashevsky and Daniel Wichs. Simple lattice trapdoor sampling from a broad class of distributions. In *PKC*, 2015.

[MM11] Daniele Micciancio and Petros Mol. Pseudorandom knapsacks and the sample complexity of LWE search-to-decision reductions. In *CRYPTO*, 2011.

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In *EUROCRYPT*, 2012.

[OT10] Tatsuaki Okamoto and Katsuyuki Takashima. Fully secure functional encryption with general relations from the decisional linear assumption. In *CRYPTO*, 2010.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, 2009.

[PVW08] Chris Peikert, Vinod Vaikuntanathan, and Brent Waters. A framework for efficient and composable oblivious transfer. In *CRYPTO*, 2008.

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*, 1984.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, 2005.

[Wat12] Brent Waters. Functional encryption for regular languages. In *CRYPTO*. 2012.

[WFL19] Zhedong Wang, Xiong Fan, and Feng-Hao Liu. Fe for inner products and its application to decentralized abe. In *PKC*, 2019.

A Proofs in Section 4

A.1 Proof of Theorem 4.10

Fix $\lambda, \ell, d \in \mathbb{N}$, input $x \in \{0,1\}^*$, and a circuit $C : \{0,1\}^* \to \{0,1\}$. We must show that for $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell, 1^d, x)$, and $\mathsf{p}_j \leftarrow \mathsf{Eval}(C, \mathsf{s}_j)$ for all $j \in [N]$, we have

$$\Pr\left[\mathsf{Combine}(\{\mathsf{p}_j\}_{j\in[N]}) = C(x)\right] = 1 - \mathsf{negl}(\lambda).$$

By the specification of the combining algorithm $\mathsf{Combine}$, it is sufficient to show that $\left|\sum_{j\in[\ell]} \mathsf{p}_j\right| \leq \lfloor p/4 \rceil$ if and only if $C(x) = 0$.

Let $\mathsf{s}_j = (\mathsf{ct}, \mathsf{sk}_j, k_j)$ for $j \in [\ell]$ where $\mathsf{ct} \leftarrow \mathsf{HE.Encrypt}(\mathsf{sk}_1 + \ldots + \mathsf{sk}_\ell, x)$, and $k_1 + \ldots, k_\ell = 0 \in \mathbb{Z}_q^n$. Then, the evaluation algorithm $\mathsf{Eval}(C, \mathsf{s}_j)$ sets $\mathsf{p}_j = \left\langle \mathsf{sk}_j, \hat{\mathsf{ct}} \right\rangle + e_j^{(\mathsf{sm})} + \mathsf{z}_j$ where $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$, $e_j^{(\mathsf{sm})} \xleftarrow{\mathrm{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, and $\mathsf{z}_j \leftarrow F(k_j, \hat{\mathsf{ct}})$. We first note that by the 1-almost key-homomorphic property of F (Definition 3.8), the sum of the outputs of the PRF is short $|\mathsf{z}_1 + \ldots, +\mathsf{z}_\ell| \leq \ell$. Furthermore, by the triangle inequality, the sum of the smudging error terms are also short $|e_1^{(\mathsf{sm})}, \ldots, e_\ell^{(\mathsf{sm})}| \leq \ell \cdot B_{\mathsf{sm}}$. Finally, by the correctness of Π_{HE} (Definition 3.11), the inner product produces a noisy evaluation $\left\langle \mathsf{sk}_1 + \mathsf{sk}_\ell, \mathsf{ct} \right\rangle = \lfloor q/2 \rceil \cdot C(x) + e_{\mathsf{HE}}$ where $|e_{\mathsf{HE}}| \leq B \cdot m^{O(d)}$. Therefore,

$$\begin{aligned}\left|\sum_{j\in[\ell]} \mathsf{p}_j\right| &= \left|\left\langle \mathsf{sk}_1 + \ldots, \mathsf{sk}_\ell, \mathsf{ct} \right\rangle + (e_1^{(\mathsf{sm})} + \ldots + e_\ell^{(\mathsf{sm})}) + (\mathsf{z}_1 + \ldots + \mathsf{z}_\ell)\right| \\ &\leq \left|\lfloor q/2 \rceil \cdot C(x)\right| + B \cdot m^{O(d)} + \ell \cdot B_{\mathsf{sm}} + \ell\end{aligned}$$

By assumption, we have $B \cdot m^{O(d)} + \ell \cdot B_{\mathsf{sm}} + \ell < p/4$. Hence, we have $\left|\sum_{j\in[\ell]} \mathsf{p}_j\right| \leq \lfloor p/4 \rceil$ if and only if $C(x) = 0$ and the theorem follows.

A.2 Proof of Theorem 4.11

We proceed via a hybrid argument. The high-level intuition for the sequence of hybrid experiments is quite simple. Namely, we define a sequence of ℓ hybrid experiments to argue (using PRF security) that the PRF evaluations $\mathsf{z}_j \leftarrow F(k_j, \hat{\mathsf{ct}})$ for $j \in [\ell]$ are computationally indistinguishable from uniformly random elements in $\mathbb{Z}_p$ under the condition that their sum results in a correct decryption $\lfloor p/2 \rceil \cdot C(x) + e_{\mathsf{HE}}$ for some error term e_{HE}. Once we make this switch, it is easy to simulate the partial decryptions $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ as they are blinded by uniformly random elements in $\mathbb{Z}_p$. Therefore, we can use semantic security of the homomorphic encryption scheme to change the ciphertext ct that is generated by the setup algorithm from an encryption of x to an encryption of an all-zero string $0^{|x|}$.

Turning this intuition into a formal proof, however, requires some care. This is mainly due to the fact that the adversary $\mathcal{A}$ can ask for a set of shares $S \subsetneq [\ell]$ at the beginning of each experiment, which can be different for each hybrid experiment. In particular, for any index $j \in S \subsetneq [\ell]$, we cannot switch the PRF evaluations $F(k_j, \hat{\mathsf{ct}})$ to uniformly random elements in $\mathbb{Z}_p$ since $\mathcal{A}$ has access to the PRF key k_j. Hence, in each of the hybrid experiments, we must specify a challenger that takes into account $\mathcal{A}$'s choice of the set $S \subsetneq [\ell]$. This makes the description of the hybrid experiments somewhat cumbersome, but the underlying intuition remains unchanged from before.

To simplify the proof, we make a simplifying assumption on the adversary $\mathcal{A}$. Namely, we assume that each of $\mathcal{A}$'s queries (C, j) to the challenger is unique ($\mathcal{A}$ never makes the same query (C, j) twice). This is without loss of generality as the challenger can always use a PRF to derandomize the evaluation algorithm. With this simplification, we formally specify our hybrid experiments $\mathrm{HYB}_0, \mathrm{HYB}_{1,0}, \mathrm{HYB}_{1,1} \ldots, \mathrm{HYB}_{1,\ell-1}, \mathrm{HYB}_2$, and HYB_3 as follows.

- HYB_0: This is the real universal thresholdizer (adaptive) security experiment in Definition 4.4. Specifically, we divide the experiment into three phases as follows:
  - **Setup phase**: At the start of the experiment, the adversary $\mathcal{A}$ commits to a secret input $x \in \{0,1\}^*$ and a set of corrupt authorities $S \subsetneq [\ell]$. The challenger samples homomorphic

encryption keys $\mathsf{sk}_j \leftarrow \mathsf{HE.KeyGen}(1^\lambda, 1^d)$ for $j \in [\ell]$, and PRF keys $k_1, \ldots, k_{\ell-1} \xleftarrow{R} \mathbb{Z}_q^n$, and sets $k_\ell = -(k_1 + \cdots + k_{\ell-1})$ as in the real Setup algorithm. It then encrypts $\mathsf{ct} \leftarrow \mathsf{HE.Encrypt}(\mathsf{sk}_1 + \cdots + \mathsf{sk}_\ell, x)$, sets $\mathsf{s}_j = (\mathsf{ct}, \mathsf{sk}_j, k_j)$ for $j \in [\ell]$, and provides $\{\mathsf{s}_j\}_{j \in S}$ to $\mathcal{A}$.

- **Query phase**: For each query (C, j) that $\mathcal{A}$ makes, the challenger computes $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$. It then generates an error term $e_j^{(\mathsf{sm})} \xleftarrow{R} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, zero shares $\mathsf{z}_j \leftarrow F(k_j, \hat{\mathsf{ct}})$, and returns $\mathsf{p}_j = \langle \mathsf{sk}_j, \mathsf{ct} \rangle + e_j^{(\mathsf{sm})} + \mathsf{z}_j$ to $\mathcal{A}$.
- **Output phase**: At the end of the experiment, adversary $\mathcal{A}$ outputs a bit β, which is the output of the experiment.

- $\text{HYB}_{1,0}$: This experiment is the same as HYB_0 except for the way the challenger answers $\mathcal{A}$'s evaluation queries during the query phase of the experiment. Namely, at the start of the experiment, when $\mathcal{A}$ commits to a set of corrupt users $S \subsetneq [\ell]$, the challenger sets $i_{\mathsf{max}} = \max_{i \in [\ell] \setminus S} i$, and instantiates a look-up table $H : \{0,1\}^* \to \mathbb{Z}_p^\ell$ that maps circuits to ℓ scalar elements in $\mathbb{Z}_p$. When $\mathcal{A}$ submits a query (C, j) during the query phase of the experiment, the challenger first checks if C is already contained in the table H. If this is the case, then it looks up $C \mapsto (\mathsf{p}_1, \ldots, \mathsf{p}_\ell)$ and provides p_j to $\mathcal{A}$. Otherwise, the challenger generates $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ as follows:

  - For all $i \neq i_{\mathsf{max}}$, the challenger computes $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$, $e_i^{(\mathsf{sm})} \xleftarrow{R} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, and $\mathsf{z}_i \leftarrow F(k_i, \hat{\mathsf{ct}})$. It then sets $\mathsf{p}_i = \langle \mathsf{sk}_i, \mathsf{ct} \rangle + e_i^{(\mathsf{sm})} + \mathsf{z}_i$.
  - For $i = i_{\mathsf{max}}$, the challenger samples $e_i^{(\mathsf{sm})} \xleftarrow{R} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, computes $\mathsf{z}_\iota \leftarrow F(k_\iota, \hat{\mathsf{ct}})$ for $\iota \in [\ell] \setminus \{i_{\mathsf{max}}\}$, and sets $\mathsf{p}_i = \lfloor p/2 \rceil \cdot C(x) - \sum_{\iota \in [\ell] \setminus \{i_{\mathsf{max}}\}} \left(\langle \mathsf{sk}_\iota, \hat{\mathsf{ct}} \rangle + \mathsf{z}_\iota \right) + e_i^{(\mathsf{sm})}$.

  The challenger provides p_j to $\mathcal{A}$ and stores $C \mapsto (\mathsf{p}_1, \ldots, \mathsf{p}_\ell)$ in the table H. The rest of the experiment remains unchanged.

The sequence of hybrid experiments $\text{HYB}_{1,j}$ for $j = 1, \ldots, \ell - 1$ are defined as follows:

- $\text{HYB}_{1,j}$: This experiment is the same as $\text{HYB}_{1,j-1}$ except that the challenger replaces the PRF evaluation $F(k_j, \cdot)$ with a random function $f(\cdot)$. Specifically, if user j is a corrupt user $j \in S$ or $j = \max_{i \in [\ell]} i$, then the challenger proceeds exactly as in $\text{HYB}_{j,j-1}$. If this is not the case, then whenever the challenger requires the evaluation of $F(k, \cdot)$, it uses a truly random function $f(\cdot)$. The rest of the experiment remains unchanged.

The hybrid experiments HYB_2 and HYB_3 are defined as follows:

- HYB_2: This experiment is the same as $\text{HYB}_{1,\ell-1}$, but we change the way the challenger answers $\mathcal{A}$'s evaluation queries during the query phase. Specifically, when the adversary $\mathcal{A}$ makes a query (C, j), the challenger checks if $\mathcal{A}$ previously queried (C, j) for all $i \in [\ell] \setminus (S \cup \{j\})$. If this is not the case, then the challenger sets $\mathsf{p}_j \xleftarrow{R} \mathbb{Z}_p$.

  If $\mathcal{A}$ previously queried (C, i) for all $i \in [\ell] \setminus (S \cup \{j\})$, then it looks up all of its previous responses $\{\mathsf{p}_i\}_{i \in [\ell] \setminus S \cup \{j\}}$, samples $e_j^{(\mathsf{sm})} \xleftarrow{R} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, computes $\mathsf{z}_{i'} \leftarrow F(k_{i'}, \hat{\mathsf{ct}})$ for all $i' \in S$, and then sets

$$\mathsf{p}_j = \lfloor p/2 \rceil \cdot C(x) + \sum_{i \in [\ell] \setminus S \cup \{j\}} \mathsf{p}_i + \sum_{i' \in S} \langle \mathsf{sk}_{i'}, \hat{\mathsf{ct}} \rangle + e_j^{(\mathsf{sm})}.$$

- HYB_3: This experiment is the same as HYB_2 except for the way the challenger generates the homomorphic encryption ciphertext. Namely, during the setup phase of the protocol, instead of encrypting $\mathsf{ct} \leftarrow \mathsf{HE.Encrypt}(\mathsf{sk}_1 + \cdots + \mathsf{sk}_\ell, x)$, it encrypts $\mathsf{ct} \leftarrow \mathsf{HE.Encrypt}(\mathsf{sk}_1 + \cdots + \mathsf{sk}_\ell, 0^{|x|})$. The rest of the experiment remains unchanged.

  This experiment corresponds to the ideal universal thresholdizer (adaptive) security experiment in Definition 4.4. In particular, the description of the challenger in the setup phase of the experiment corresponds to the simulator Sim_1, and the description of the challenger during the query phase of the experiment corresponds to the simulator Sim_2.

We now show that each consecutive hybrid experiments are indistinguishable to an adversary. Below, we write $\text{HYB}(\mathcal{A})$ to denote the random variable that represents the output of the experiment HYB with respect to an adversary $\mathcal{A}$.

Lemma A.1. *Suppose that $(B \cdot m^{O(d)} + \ell)/B_{\mathsf{sm}} = \mathsf{negl}(\lambda)$, the homomorphic encryption Π_{HE} satisfies correctness, and the PRF $F : \mathbb{Z}_q^n \times \{0,1\}^* \to \mathbb{Z}_p$ satisfies 1-almost key-homomorphism (Definition 3.7). Then, for any (unbounded) adversary $\mathcal{A}$,*

$$\left| \Pr[\text{HYB}_0(\mathcal{A}) = 1] - \Pr[\text{HYB}_{1,0}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. The only difference between the hybrid experiments HYB_0 and $\text{HYB}_{1,0}$ is in the way the challenger simulates the partial evaluations during the query phase of the experiment. Specifically, for each query (C, i_{max}) that $\mathcal{A}$ makes, each challenger in HYB_0 and $\text{HYB}_{1,0}$ defines $\mathsf{p}_{i_{\mathsf{max}}}$ as follows:

- In HYB_0, the challenger samples $e_{i_{\mathsf{max}}}^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, computes $\mathsf{z}_{i_{\mathsf{max}}} \leftarrow F(k_j, \hat{\mathsf{ct}})$, and returns $\mathsf{p}_{i_{\mathsf{max}}} = \langle \mathsf{sk}_{i_{\mathsf{max}}}, \mathsf{ct} \rangle + e_{i_{\mathsf{max}}}^{(\mathsf{sm})} + \mathsf{z}_{i_{\mathsf{max}}}$ to the adversary $\mathcal{A}$.
- In $\text{HYB}_{1,0}$, the challenger samples error terms $e_{i_{\mathsf{max}}}^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, and sets

$$\mathsf{p}_{i_{\mathsf{max}}} = \lfloor p/2 \rceil \cdot C(x) - \sum_{\iota \in [\ell] \setminus \{i_{\mathsf{max}}\}} \left(\langle \mathsf{sk}_\iota, \hat{\mathsf{ct}} \rangle + \mathsf{z}_\iota \right) + e_{i_{\mathsf{max}}}^{(\mathsf{sm})},$$

  where $\mathsf{z}_\iota \leftarrow F(k_\iota, \hat{\mathsf{ct}})$ for $\iota \in [\ell] \setminus \{i_{\mathsf{max}}\}$. In particular, $\mathsf{p}_{i_{\mathsf{max}}}$ satisfies the following equality:

$$\begin{aligned}
\mathsf{p}_{i_{\mathsf{max}}} &= \lfloor p/2 \rceil \cdot C(x) - \sum_{\iota \in [\ell] \setminus \{i_{\mathsf{max}}\}} \left(\langle \mathsf{sk}_i, \hat{\mathsf{ct}} \rangle + \mathsf{z}_\iota \right) + e_{i_{\mathsf{max}}}^{(\mathsf{sm})} \\
&= \Big\langle \sum_{j \in [\ell]} \mathsf{sk}_j, \hat{\mathsf{ct}} \Big\rangle + e_{\mathsf{HE}} - \sum_{\iota \in [\ell] \setminus \{i_{\mathsf{max}}\}} \left(\langle \mathsf{sk}_\iota, \hat{\mathsf{ct}} \rangle + \mathsf{z}_\iota \right) + e_{i_{\mathsf{max}}}^{(\mathsf{sm})} \\
&= \langle \mathsf{sk}_{i_{\mathsf{max}}}, \hat{\mathsf{ct}} \rangle + e_{\mathsf{HE}} - \sum_{\iota \in [\ell] \setminus \{i_{\mathsf{max}}\}} \mathsf{z}_\iota + e_{i_{\mathsf{max}}}^{(\mathsf{sm})} \\
&= \langle \mathsf{sk}_{i_{\mathsf{max}}}, \hat{\mathsf{ct}} \rangle + \big(\underbrace{e_{\mathsf{HE}} + e_{\mathsf{kh}}}_{e^*} \big) + e_{i_{\mathsf{max}}}^{(\mathsf{sm})} + \mathsf{z}_{i_{\mathsf{max}}},
\end{aligned}$$

  where e_{HE} denote the decryption error and e_{kh} denotes the error from the 1-almost key-homomorphism. By the correctness of Π_{HE} and the 1-almost key-homomorphism of F, we have $|e_{\mathsf{dec}}| \leq B \cdot m^{O(d)}$, $|e_{\mathsf{kh}}| \leq \ell$.

Therefore, from the perspective of the adversary $\mathcal{A}$, the only difference between the hybrid experiments HYB_0 and $\text{HYB}_{1,0}$ is the additional error term e^* for each of its queries (C, j_{max}). Since $|e^*| \leq B \cdot m^{O(d)} + \ell$ and by assumption, $(B \cdot m^{O(d)} + \ell)/B_{\mathsf{sm}} = \mathsf{negl}(\lambda)$, $\mathcal{A}$'s views of the two hybrid experiments are statistically indistinguishable by the smudging lemma (Lemma 3.2). □

Lemma A.2. *Suppose that the key-homomorphic PRF $F : \mathbb{Z}_q^n \times \{0,1\}^* \to \mathbb{Z}_p$ is secure (Definition 3.6). Then, for any efficient adversary $\mathcal{A}$ and $j \in [\ell - 1]$,*

$$\left| \Pr[\text{HYB}_{1,j}(\mathcal{A}) = 1] - \Pr[\text{HYB}_{1,j-1}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes the experiments $\text{HYB}_{1,j-1}$ and $\text{HYB}_{1,j}$. Then, we construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the PRF security of F. By definition, the experiments $\text{HYB}_{1,j}$ and $\text{HYB}_{1,j-1}$ are equivalent if $j \in S$ or $j = i_{\mathsf{max}}$, and therefore, assume that $j \neq j_{\mathsf{max}}$. The algorithm $\mathcal{B}$ proceeds as follows:

- *Setup phase*: At the start of the experiment, when $\mathcal{A}$ commits to $x \in \{0,1\}^*$ and a set of corrupt authorities $S \subsetneq [\ell]$, algorithm $\mathcal{B}$ samples homomorphic encryption keys $\mathsf{sk}_i \leftarrow \mathsf{HE.KeyGen}(1^\lambda, 1^d)$ for $i \in [\ell]$ and encrypts $\mathsf{ct} \leftarrow \mathsf{HE.Encrypt}(\mathsf{sk}_1 + \cdots + \mathsf{sk}_\ell, x)$ just as in the two hybrid experiments. Then, for users $i \in S \cup [j, \ell] \backslash \{i_{\mathsf{max}}\}$, the challenger samples PRF keys $k_i \xleftarrow{\text{R}} \mathbb{Z}_q^n$, sets $\mathsf{s}_i = (\mathsf{ct}, \mathsf{sk}_i, k_i)$, and provides $\{\mathsf{s}_i\}_{i \in S}$ to $\mathcal{A}$. Finally, it instantiates a look-up table $H : \{0,1\}^* \to \mathbb{Z}_p^\ell$.

- *Query phase*: When $\mathcal{A}$ submits a query (C, i'), algorithm $\mathcal{B}$ checks if C is already contained in the table H. If this is the case, then it looks up $C \mapsto (\mathsf{p}_1, \ldots, \mathsf{p}_\ell)$ and provides $\mathsf{p}_{i'}$ to $\mathcal{A}$. Otherwise, the challenger generates $\mathsf{p}_1, \ldots, \mathsf{p}_\ell$ as follows:

  - For $i \notin S$ and $i \leq j - 1$, the challenger computes $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$, samples $e_i^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, $\mathsf{z}_i \xleftarrow{\text{R}} \mathbb{Z}_p$, and sets $\mathsf{p}_i = \langle \mathsf{sk}_i, \hat{\mathsf{ct}} \rangle + e_i^{(\mathsf{sm})} + \mathsf{z}_i$ as specified in $\text{HYB}_{1,j-1}$ and $\text{HYB}_{1,j}$.
  - For $i = j$, algorithm $\mathcal{B}$ computes $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$ and samples $e_i^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$. Then, it queries $\hat{\mathsf{ct}}$ to the PRF challenger to get back z_i. Finally, it sets $\mathsf{p}_i = \langle \mathsf{sk}_i, \mathsf{ct} \rangle + e_i^{(\mathsf{sm})} + \mathsf{z}_i$.
  - For $i \in S \cup [j, \ell] \backslash \{i_{\mathsf{max}}\}$, the challenger evaluates $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$ and sets $\mathsf{p}_i = \langle \mathsf{sk}_i, \mathsf{ct} \rangle + e_i^{(\mathsf{sm})} + \mathsf{z}_i$ for $e_i^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$, and $\mathsf{z}_i \leftarrow F(k_i, \hat{\mathsf{ct}})$ as specified in $\text{HYB}_{1,j-1}$ and $\text{HYB}_{1,j}$.
  - For $i = i_{\mathsf{max}}$, the challenger sets $\mathsf{p}_i = \lfloor p/2 \rceil \cdot C(x) - \sum_{\iota \in [\ell - 1]} \left(\langle \mathsf{sk}_\iota, \hat{\mathsf{ct}} \rangle + \mathsf{z}_\iota \right) + e_i^{(\mathsf{sm})}$ for $e_i^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$ just as in $\text{HYB}_{1,j-1}$ and $\text{HYB}_{1,j}$.

  Algorithm $\mathcal{B}$ then provides $\mathsf{p}_{i'}$ to $\mathcal{A}$ and stores $C \mapsto (\mathsf{p}_1, \ldots, \mathsf{p}_\ell)$ in the table H.

- *Output phase*: When $\mathcal{A}$ outputs a bit β, algorithm $\mathcal{B}$ outputs the same bit β.

Correctness of the simulation. By definition, the view that $\mathcal{B}$ provides for $\mathcal{A}$ is exactly as in the experiments $\text{HYB}_{1,j-1}$ and $\text{HYB}_{1,j}$ except for the way it defines the partial evaluations p_j during the query phase of the protocol.

- If $\mathcal{B}$ is interacting with the real PRF challenger, then the zero shares z_j corresponds to the real PRF evaluation $F(k_j, \hat{\mathsf{ct}})$. Therefore, the partial evaluation $\mathsf{p}_j = \langle \mathsf{sk}_j, \hat{\mathsf{ct}} \rangle + e_j^{(\mathsf{sm})} + \mathsf{z}_j$ is distributed exactly as in $\text{HYB}_{1,j-1}$.
- If $\mathcal{B}$ is interacting with the ideal PRF challenger, then the error term z_j is a uniformly random element $\mathsf{z}_j \xleftarrow{\text{R}} \mathbb{Z}_p$. Therefore, the partial evaluation $\mathsf{p}_j = \langle \mathsf{sk}_j, \hat{\mathsf{ct}} \rangle + e_j^{(\mathsf{sm})} + \mathsf{z}_j$ is distributed exactly as in $\text{HYB}_{1,j}$.

Therefore, depending on whether it is interacting with the real or ideal PRF challenger, algorithm $\mathcal{B}$ simulates the view of $\text{HYB}_{1,j-1}$ or $\text{HYB}_{1,j}$. Hence, with the same distinguishing advantage as $\mathcal{A}$ on the experiments $\text{HYB}_{1,j-1}$ and $\text{HYB}_{1,j}$, algorithm $\mathcal{B}$ breaks the PRF security of F. Then, assuming that F is a secure PRF, the distinguishing advantage of $\mathcal{A}$ must be negligible and the lemma follows. □

Lemma A.3. *For any (unbounded) adversary $\mathcal{A}$,*

$$\left| \Pr[\text{HYB}_{1,\ell-1}(\mathcal{A}) = 1] - \Pr[\text{HYB}_2(\mathcal{A}) = 1] \right| = 0.$$

Proof. The difference between the hybrid experiments $\text{HYB}_{1,\ell-1}$ and HYB_2 are purely syntactical. In $\text{HYB}_{1,\ell-1}$, for each query (C, j) that $\mathcal{A}$ makes, the challenger sets $\mathsf{p}_i = \langle \mathsf{sk}_i, \hat{\mathsf{ct}} \rangle + e_i^{(\mathsf{sm})} + \mathsf{z}_i$ where $\mathsf{z}_i \xleftarrow{\text{R}} \mathbb{Z}_p$ for all $i \in [\ell] \backslash (S \cup \{i_{\mathsf{max}}\})$ and $\mathsf{p}_{i_{\mathsf{max}}}$ is set to satisfy the relation $\sum_{i \in [\ell] \backslash S} \mathsf{p}_i + \sum_{i' \in S} \langle \mathsf{sk}_{i'}, \hat{\mathsf{ct}} \rangle + \mathsf{z}_{i'} = e_{i_{\mathsf{max}}}^{(\mathsf{sm})}$ for some error terms $e_{i_{\mathsf{max}}}^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$. Since $\mathsf{z}^{(i)}$ for $i \in [\ell] \backslash (S \cup \{i_{\mathsf{max}}\})$ are sampled uniformly at random from $\mathbb{Z}_p$, the partial evaluations p_i for $i \in [\ell] \backslash S$ are distributed uniformly at random in $\mathbb{Z}_p$ under the condition that $\sum_{i \in [\ell] \backslash S} \mathsf{p}_i + \sum_{i' \in S} \langle \mathsf{sk}_{i'}, \hat{\mathsf{ct}} \rangle + \mathsf{z}_{i'} = e_{i_{\mathsf{max}}}^{(\mathsf{sm})}$ for $e_{i_{\mathsf{max}}}^{(\mathsf{sm})} \xleftarrow{\text{R}} [-B_{\mathsf{sm}}, B_{\mathsf{sm}}]$. By specification, this is exactly the distribution of the partial evaluations in HYB_2 and the lemma follows. □

Lemma A.4. *Suppose that the homomorphic encryption scheme Π_{HE} is secure (Definition 3.12). Then, for any efficient adversary $\mathcal{A}$,*

$$\left| \Pr[\text{HYB}_2(\mathcal{A}) = 1] - \Pr[\text{HYB}_3(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes the experiments HYB_2 and HYB_3. Then, we construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the security of Π_{HE}. The algorithm $\mathcal{B}$ proceeds as follows:

- *Setup phase*: At the start of the experiment, when $\mathcal{A}$ commits to a secret input $x \in \{0,1\}^*$ and a set of corrupt authorities $S \subsetneq [\ell]$, algorithm $\mathcal{B}$ samples homomorphic encryption keys $\mathsf{sk}_i \leftarrow \mathsf{HE.KeyGen}(1^\lambda, 1^d)$ and PRF keys $k_i \xleftarrow{\text{R}} \mathbb{Z}_q^n$ for $i \in S$. Then, it submits the two messages $(x, 0^{|x|})$ to the encryption challenger and gets back a ciphertext ct. It sets $\mathsf{s}_i = (\mathsf{ct}, \mathsf{sk}_i, k_i)$ and provides $\{\mathsf{s}_i\}_{i \in S}$ to $\mathcal{A}$.
- *Query phase*: The query phase of the experiment in HYB_2 and HYB_3 are identical by definition. Furthermore, for each query (C, j) that $\mathcal{A}$ makes, the partial evaluations p_j's are either sampled uniformly at random from $\mathbb{Z}_p$ or simulatable given $\{\mathsf{sk}_i\}_{i \in S}$ and $\hat{\mathsf{ct}} \leftarrow \mathsf{HE.Eval}(C, \mathsf{ct})$. Therefore, $\mathcal{B}$ simulates the view of $\mathcal{A}$ exactly as in HYB_2 and HYB_3.
- *Output phase*: When $\mathcal{A}$ outputs a bit β, algorithm $\mathcal{B}$ outputs the same bit β.

Correctness of the simulation. Except for the ciphertext ct, algorithm $\mathcal{B}$ exactly simulates the view of $\mathcal{A}$ in HYB_2 and HYB_3 simply by definition. Now, if $\mathsf{ct} = \mathsf{HE.Encrypt}(\mathsf{sk}^*, x)$ for some encryption key sk^*, then ct is distributed exactly as in HYB_2. If $\mathsf{ct} = \mathsf{HE.Encrypt}(\mathsf{sk}^*, 0^{|x|})$, then ct is distributed exactly as in HYB_3. Therefore, with the same distinguishing advantage of $\mathcal{A}$, algorithm $\mathcal{B}$ breaks the security of the homomorphic encryption scheme Π_{HE}. Then, assuming that Π_{HE} is a secure encryption scheme, the distinguishing advantage of $\mathcal{A}$ must be negligible. The lemma follows. □

B Proofs in Section 6

B.1 Proof of Theorem 6.7

Fix the security parameter λ and the number of attributes ℓ. Let $\mu \in \mathcal{M}$ be any message, $\phi : \{0,1\}^\ell \to \{0,1\}$ be any policy circuit in $\mathcal{C}$, and $x \in \{0,1\}^\ell$ be an attribute string such that $\phi(x) = 0$. We must show that for $(\mathsf{pp}, \mathsf{msk}_1, \ldots, \mathsf{msk}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $(\mathsf{req}, \mathsf{st}) \leftarrow \mathsf{KeyRequest}(x)$, $\mathsf{sk}_{\mathsf{req},j,x_j} \leftarrow \mathsf{KeyGen}(\mathsf{msk}_j, \mathsf{req}, x_j)$ for $j \in [\ell]$, and $\mathsf{sk}_x \leftarrow \mathsf{KeyCombine}(\mathsf{st}, \mathsf{sk}_{\mathsf{req},1,x_1}, \ldots, \mathsf{sk}_{\mathsf{req},\ell,x_\ell})$, we have

$$\Pr\left[\mathsf{Decrypt}\big(\mathsf{sk}_x, \mathsf{Encrypt}(\phi, \mu)\big) = \mu\right] = 1 - \mathsf{negl}(\lambda).$$

By specification, the $\mathsf{KeySend}(\mathsf{msk}_j, x_j, \mathsf{req})$ algorithm sets $\mathsf{sk}_{\mathsf{req},j,x_j} = \big(\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\setminus\{j\}}\big)$ where

- $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \mathsf{s}_j)$,
- $\mathsf{p}^{(\mathsf{att})}_{i,b,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}, \mathsf{s}_j)$ for $i \in [\ell]$, $b \in \{0,1\}$,

and $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}\big(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}), \mathsf{ot.pk}_i\big)$ for all $i \in [\ell]\setminus\{j\}$. The key combining algorithm decrypts the OT messages $\tilde{\mathsf{p}}^{(\mathsf{att})}_{i,x_j,j} \leftarrow \mathsf{OT.Decrypt}_{\mathsf{R}}(\mathsf{ot.sk}_j, \mathsf{ot.ct}_{i,j})$ for all $i, j \in [\ell]$, combines the UT evaluation shares

- $(\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]}) \leftarrow \mathsf{UT.Combine}\big(\tilde{\mathsf{p}}^{(\mathsf{bind})}_{0,1}, \ldots, \tilde{\mathsf{p}}^{(\mathsf{bind})}_{0,\ell}\big)$,
- $(\mathsf{dabe.sk}'_{i,x_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}}) \leftarrow \mathsf{UT.Combine}\big(\tilde{\mathsf{p}}^{(\mathsf{att})}_{i,x_i,1}, \ldots, \tilde{\mathsf{p}}^{(\mathsf{att})}_{i,x_i,\ell}\big)$ for $i \in [\ell]$,

and sets

- $\mathsf{dabe.sk}_0 = \mathsf{dabe.sk}'_0 \oplus \big(\bigoplus_{k\in[\ell]} \alpha_{0,k}\big)$,
- $\mathsf{dabe.sk}_{i,x_i} = \mathsf{dabe.sk}'_{i,x_i} \oplus \big(\bigoplus_{k\in[0,\ell]\setminus\{i\}} \alpha_{i,k}\big)$ for $i \in [\ell]$.

The decryption algorithm returns $\mu' \leftarrow \mathsf{DABE.Decrypt}\big(\mathsf{dabe.sk}'_0, \mathsf{dabe.sk}'_{1,x_1}, \ldots, \mathsf{dabe.sk}'_{\ell,x_\ell}\big)$. By the correctness of Π_{OT}, we have $\tilde{\mathsf{p}}^{(\mathsf{bind})}_{0,j} = \mathsf{p}^{(\mathsf{bind})}_{0,j}$ for all $j \in [\ell]$ and $\tilde{\mathsf{p}}^{(\mathsf{att})}_{i,x_j,j} = \mathsf{p}^{(\mathsf{att})}_{i,x_j,x_j}$ for all $i, j \in [\ell]$. Therefore, by the correctness condition of Π_{UT}, and the specifications of the circuits $\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}$ and $\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}$, the key components $\mathsf{dabe.sk}_0, \mathsf{dabe.sk}_{1,x_1}, \ldots, \mathsf{dabe.sk}_{\ell,x_\ell}$ correspond to properly generated key components of $\mathsf{DABE.KeyGen}(\mathsf{dabe.msk})$. Since $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\phi, \mu) = \mathsf{DABE.Encrypt}(\phi, \mu)$, we have $\mu' = \mathsf{DABE.Decrypt}\big(\mathsf{dabe.sk}_0, \mathsf{dabe.sk}_{1,x_1}, \ldots, \mathsf{dabe.sk}_{\ell,x_\ell}\big) = \mu$. The correctness now follows.

B.2 Proof of Theorem 6.8

We proceed via a hybrid argument. In our hybrid description below, we use $\mathsf{Sim} = (\mathsf{Sim}_1, \mathsf{Sim}_2)$ to denote the universal thresholdizer simulator in Definition 4.4 and OTExt to denote the OT extractor in Definition 3.16. For simplicity in the presentation of the proof, we make two assumptions on an adversary $\mathcal{A}$:

- We first assume that all key queries (req, j, x_j) that $\mathcal{A}$ makes during the query phase of the experiment are unique. This is without loss of generality as the challenger can always answer repeated queries consistently.
- Let S be the set of corrupt authorities that $\mathcal{A}$ commits to at the beginning of the MA-ABE security experiment. Then we assume that for all key queries (req, j, b) that $\mathcal{A}$ makes during the query phase of the experiment, we have $j \notin S$. This is without loss of generality as $\mathcal{A}$ already holds master secret keys $\{\mathsf{msk}_j\}_{j\in S}$ and therefore, it can answer the key queries by itself.

With these simplifying assumptions, we define our sequence of hybrid experiments as follows:

- HYB_0: This is the ciphertext security experiment $\mathsf{Expt}^{(\mathsf{CS})}_{\Pi_{\mathsf{MA\text{-}ABE}}}(\lambda, \ell, \mathcal{A}, 0)$ of Definition 5.5. Specifically, the challenger proceeds in each phases of the experiment as follows:
  - **Setup phase**: At the start of the experiment, $\mathcal{A}$ specifies a set of corrupt authorities $S \subseteq [\ell]$ and a policy circuit $\phi \in \mathcal{C}$ for the challenge ciphertext. The challenger samples a PRF key $\mathsf{prf.k} \xleftarrow{\text{R}} \mathcal{K}$, computes $(\mathsf{dabe.pp}, \mathsf{dabe.msk}) \leftarrow \mathsf{DABE.Setup}(1^\lambda)$, $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{UT.Setup}(1^\lambda, 1^\ell, 1^d, (\mathsf{dabe.msk}, \mathsf{prf.k}))$, and $(\mathsf{ot.crs}, \mathsf{td}) \leftarrow \mathsf{OT.Setup}(1^\lambda)$ as in the real setup algorithm. It provides $\mathsf{pp} = (\mathsf{dabe.pp}, \mathsf{ot.crs})$ and $\{\mathsf{s}_j\}_{j\in S}$ to $\mathcal{A}$.
  - **Query phase**: The challenger responds to each of $\mathcal{A}$'s queries as follows:
    - *Key queries*: For each query (req, j, b) that $\mathcal{A}$ makes, the challenger parses $\mathsf{req} = \{\mathsf{ot.pk}_i\}_{i\in[\ell]}$, computes
      - $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \mathsf{s}_j)$,
      - $\mathsf{p}^{(\mathsf{att})}_{i,b,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}, \mathsf{s}_j)$ for $i \in [\ell]$, $b \in \{0,1\}$,

      and sets $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}\big(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}), \mathsf{ot.pk}_i\big)$ for all $i \in [\ell]\backslash\{j\}$. It provides $\mathsf{sk}_{\mathsf{req},j,x_j} = \big(\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\backslash\{j\}}\big)$ to $\mathcal{A}$.
    - *Challenge query*: When $\mathcal{A}$ makes a challenge query (μ_0, μ_1), the challenger computes $\mathsf{ct} \leftarrow \mathsf{DABE.Encrypt}(\phi, \mu_0)$ and returns ct.
  - **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, which becomes the output of the experiment.
- HYB_1: In this experiment, we change the way the challenger responds to $\mathcal{A}$'s key queries. Namely, for each key query (req, j, b) for $\mathsf{req} = \{\mathsf{pk}_i\}_{i\in[\ell]}$ that $\mathcal{A}$ submits during the query phase of the protocol, the challenger extracts $x'_i \leftarrow \mathsf{OTExt}(\mathsf{td}, \mathsf{pk}_i)$ for all $i \in [\ell]$ and sets
  - $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \mathsf{s}_j)$,
  - $\mathsf{p}^{(\mathsf{att})}_{j,b,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,b}, \mathsf{s}_j)$,

– $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}, \mathsf{s}_j)$ for $i \in [\ell]\backslash\{j\}$,
– $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j} \leftarrow \perp$ for $i \in [\ell]\backslash\{j\}$.

It then encrypts $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}\big(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}), \mathsf{ot.pk}_j\big)$ for all $i \in [\ell]\backslash\{j\}$ and provides $\mathsf{sk}_{\mathsf{req},j,x_j} = (\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,b,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\backslash\{j\}})$ to $\mathcal{A}$.

The rest of the experiment remains identical to HYB_0.

- HYB_2: In this experiment, we change the way the challenger responds to $\mathcal{A}$'s key queries. Namely, during the setup phase, instead of running the real universal thresholdizer setup, the challenger invokes the simulator $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^\ell, 1^d, S, 1^\kappa)$ where κ is the maximum bit length of any $(\mathsf{dabe.msk}, \mathsf{prf.k})$. It then provides $\mathsf{pp} = (\mathsf{dabe.pp}, \mathsf{ot.crs})$ and $\{\mathsf{s}_i\}_{j\in S}$ to $\mathcal{A}$.

  During the query phase of the experiment, the challenger maintains a lookup table $H : \{0,1\}^* \times [0,\ell] \times [\ell]\backslash S \to \{0,1\}^\rho$ that maps request-index-index tuples to evaluation shares $(\mathsf{req}, i, k) \mapsto \mathsf{p}$ where $|\mathsf{p}| = \rho$. Now, when $\mathcal{A}$ makes a key query (req, j, x_j) during the query phase (by assumption, $j \notin S$), the challenger still extracts $x'_i \leftarrow \mathsf{OTExt}(\mathsf{td}, \mathsf{pk}_i)$ for $i \in [\ell]$. However, instead of evaluating $\mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \mathsf{s}_j)$ and $\mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}, \mathsf{s}_j)$ for $i \in [\ell]$, $b \in \{0,1\}$, the challenger invokes the simulator Sim_2 for the evaluation shares depending on $\mathcal{A}$'s previous key queries. Specifically, on (req, j, x_j), the challenger sets the evaluation shares as follows:

  – To set $\mathsf{p}^{(\mathsf{bind})}_{0,j}$, the challenger checks if $\mathcal{A}$ had previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for each $k \in [\ell]\backslash S$. If this is the case, then it looks up the mappings $(\mathsf{req}, 0, k) \mapsto \mathsf{p}^{(\mathsf{bind})}_{0,k}$ for $k \in [\ell]\backslash S$ in H. Then, it invokes the simulator $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]}), \{\mathsf{p}^{(\mathsf{bind})}_{0,k}\}_{k\in[\ell]\backslash S}\big)$.
    Otherwise, if $\mathcal{A}$ had not previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for all $k \in [\ell]\backslash S$, then it samples $\mathsf{p}^{(\mathsf{bind})}_{0,j} \xleftarrow{\text{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, 0, j) \mapsto \mathsf{p}^{(\mathsf{bind})}_{0,j}$ to H.

  – To set $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$, the challenger first checks if $x'_j = x_j$. If this is not the case, then it samples $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \xleftarrow{\text{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, j, j) \mapsto \mathsf{p}^{(\mathsf{bind})}_{j,x_j,j}$ to H.
    Otherwise, if $x'_j = x_j$, then the challenger checks if $\mathcal{A}$ had previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for each $k \in [\ell]\backslash(S \cup \{j\})$. If this is the case, then it looks up the mappings $(\mathsf{req}, j, k) \mapsto \mathsf{p}^{(\mathsf{att})}_{j,x_j,k}$ for $k \in [\ell]\backslash(S \cup \{j\})$ in H. Then, it invokes the simulator $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,x_j}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\backslash\{j\}}), \{\mathsf{p}^{(\mathsf{att})}_{j,x_j,k}\}_{k\in[0,\ell]\backslash(S\cup\{j\})}\big)$.
    Otherwise, if $\mathcal{A}$ had not previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for all $k \in [\ell]\backslash(S \cup \{j\})$, then it samples $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \xleftarrow{\text{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, j, j) \mapsto \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$ to H.

  – To set $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j}$ for $i \in [\ell]\backslash\{j\}$, the challenger checks if $\mathcal{A}$ had previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for each $k \in [\ell]\backslash(S \cup \{i\})$. If this is the case, then it looks up the mappings $(\mathsf{req}, i, k) \mapsto \mathsf{p}^{(\mathsf{att})}_{i,x'_i,k}$ for $k \in [\ell]\backslash(S \cup \{i\})$ in H. Then, it invokes the simulator $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_{i,x'_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\backslash\{i\}}), \{\mathsf{p}^{(\mathsf{att})}_{i,x'_i,k}\}_{k\in[0,\ell]\backslash(S\cup\{i\})}\big)$.
    Otherwise, if $\mathcal{A}$ had not previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for all $k \in [\ell]\backslash(S \cup \{i\})$, then it samples $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j} \xleftarrow{\text{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, i, j) \mapsto \mathsf{p}^{(\mathsf{bind})}_{i,x'_i,j}$ to H.

– For the rest of the evaluation shares $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j}$ for $i \in [\ell]\backslash\{j\}$, it sets each of them $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j} \leftarrow \bot$.

The challenger then encrypts $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}), \mathsf{ot.pk}_j)$ for all $i \in [\ell]\backslash\{j\}$ and provides $\mathsf{sk}_{\mathsf{req},j,x_j} = (\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\backslash\{j\}})$ to $\mathcal{A}$.

The rest of the experiment remains unchanged.

- HYB_3: This experiment is the same as HYB_2 except that the challenger replaces the PRFs $F_0(\mathsf{prf.k}, \cdot)$ and $F_1(\mathsf{prf.k}, \cdot)$ with random functions. Specifically, the challenger in HYB_3 emulates the challenger in HYB_2, but whenever it must evaluate $F_0(\mathsf{prf.k}, \cdot)$ or $F_1(\mathsf{prf.k}, \cdot)$, it uses truly random functions $f_0(\cdot) \xleftarrow{\text{R}} \mathsf{Funs}[\{0,1\}^*, \{0,1\}^r]$ and $f_1(\cdot) \xleftarrow{\text{R}} \mathsf{Funs}[\{0,1\}^*, \{0,1\}^{t\ell}]$. The rest of the experiment remains unchanged.

- HYB_4: This experiment is the same as HYB_2 except that during the query phase, when the adversary $\mathcal{A}$ makes the single challenge query (μ_0, μ_1), the challenger encrypts $\mathsf{ct} \leftarrow \mathsf{DABE.Encrypt}(\phi, \mu_1)$ instead of encrypting μ_0.

- HYB_5: Starting from this experiment, we revert the changes that we made from HYB_0. In this experiment, instead of using truly random functions $f_0 \xleftarrow{\text{R}} \mathsf{Funs}[\{0,1\}^*, \{0,1\}^r]$ and $f_1 \xleftarrow{\text{R}} \mathsf{Funs}[\{0,1\}^*, \{0,1\}^{\ell t}]$, the challenger uses the real PRFs $F_0(\mathsf{prf.k}, \cdot)$ and $F_1(\mathsf{prf.k}, \cdot)$.

- HYB_6: In this experiment, the challenger reverts back to using the real universal thresholdizer setup and evaluation algorithms as in HYB_1.

- HYB_7: In this experiment, the challenger reverts back to the way it responds to $\mathcal{A}$'s key queries in HYB_0. Namely, for each key query (req, j, b) for $\mathsf{req} = \{\mathsf{pk}_i\}_{i\in[\ell]}$ that $\mathcal{A}$ submits during the query phase of the protocol, instead of invoking the OT extractor, the challenger encrypts both evaluation shares $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}\big(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}), \mathsf{ot.pk}_j\big)$ for all $i \in [\ell]\backslash\{j\}$. This experiment corresponds to the ciphertext security experiment $\mathsf{Expt}^{(\mathsf{CS})}_{\Pi_{\mathsf{MA\text{-}ABE}}}(1^\lambda, 1^\ell, \mathcal{A}, 1)$

We now show that each consecutive hybrid experiments are either statistically or computationally indistinguishable. For the lemma statements below, we write $\text{HYB}(\mathcal{A})$ to denote the random variable that represents the output of HYB with respect to an adversary $\mathcal{A}$.

Lemma B.1. *Suppose that the oblivious transfer protocol Π_{OT} satisfies sender security (Definition 3.16). Then, for any efficient adversary $\mathcal{A}$,*

$$\big|\Pr[\text{HYB}_0(\mathcal{A}) = 1] - \Pr[\text{HYB}_1(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. The lemma follows immediately from the definition of sender security of Π_{OT} (Definition 3.16). □

Lemma B.2. *Suppose that the universal thresholdizer Π_{UT} satisfies strong security (Definition 4.4). Then, for any efficient adversary $\mathcal{A}$,*

$$\big|\Pr[\text{HYB}_1(\mathcal{A}) = 1] - \Pr[\text{HYB}_2(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes the experiments HYB_1 and HYB_2. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the universal thresholdizer security of Π_{UT}. Interacting with a UT challenger (Definition 4.4), algorithm $\mathcal{B}$ simulates each phases of the experiment as follows:

- **Setup phase**: At the start of the experiment, adversary $\mathcal{A}$ specifies a set of corrupt authorities $S \subsetneq [\ell]$ and a policy circuit $\phi \in \mathcal{C}$ for the challenge ciphertext. Algorithm $\mathcal{B}$ samples $\mathsf{prf.k} \xleftarrow{R} \mathcal{K}$, $(\mathsf{dabe.pp}, \mathsf{dabe.msk}) \leftarrow \mathsf{DABE.Setup}(1^\lambda)$, and submits $\big((\mathsf{dabe.msk}, \mathsf{prf.k}), S\big)$ to the UT challenger to receive back $\{\mathsf{s}_j\}_{j\in S}$. It then generates $(\mathsf{ot.crs}, \mathsf{td}) \leftarrow \mathsf{OT.Setup}(1^\lambda)$ and provides $\mathsf{pp} = (\mathsf{dabe.pp}, \mathsf{ot.crs})$ and $\{\mathsf{s}_j\}_{j\in S}$ to $\mathcal{A}$.
- **Query phase**: Algorithm $\mathcal{B}$ simulates the responses to $\mathcal{A}$'s queries as follows:
  - *Key queries*: For each query (req, j, x_j) that $\mathcal{A}$ makes, algorithm $\mathcal{B}$ parses $\mathsf{req} = \{\mathsf{pk}_i\}_{i\in[\ell]}$, and extracts $x'_i \leftarrow \mathsf{OTExt}(\mathsf{td}, \mathsf{pk}_i)$ for $i \in [\ell]$. Then, it computes $(\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]}) \leftarrow \mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}(\mathsf{dabe.msk}, \mathsf{prf.k})$, $(\mathsf{dabe.sk}'_{i,b}, \{\alpha_{k,i}\}) \leftarrow \mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}(\mathsf{dabe.msk}, \mathsf{prf.k})$ for $i \in [\ell]$, $b \in \{0,1\}$, and submits the following circuits and their evaluations to the UT challenger under index $j \in [\ell]$:
    - $\mathcal{B}$ submits $\big(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]})\big)$ to receive $\mathsf{p}^{(\mathsf{bind})}_{0,j}$,
    - $\mathcal{B}$ submits $\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,x_j}, (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\setminus\{j\}})\big)$ to receive $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$,
    - $\mathcal{B}$ submits $\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}, (\mathsf{dabe.sk}'_{i,x'_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})\big)$ for $i \in [\ell]\setminus\{j\}$ to receive $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j}$ for $i \in [\ell]\setminus\{j\}$,

    and also sets $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j} \leftarrow \perp$ for $i \in [\ell]\setminus\{j\}$. Algorithm $\mathcal{B}$ then encrypts $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}, \mathsf{ot.pk}_j)$ for all $i \in [\ell]\setminus\{j\}$ and provides $\mathsf{sk}_{\mathsf{req},j,x_j} = (\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\setminus\{j\}})$ to $\mathcal{A}$.
  - *Challenge query*: For $\mathcal{A}$'s challenge query (ϕ, μ_0, μ_1), algorithm $\mathcal{B}$ encrypts $\mathsf{ct} \leftarrow \mathsf{DABE.Encrypt}(\mathsf{dabe.pp}, \phi, \mu_0)$ and provides ct to $\mathcal{A}$.
- **Output phase**: When $\mathcal{A}$ outputs a bit β', algorithm $\mathcal{B}$ also outputs β'.

Correctness of the simulation. During the setup phase, when $\mathcal{B}$ submits $\big((\mathsf{dabe.msk}, \mathsf{prf.k}), S\big)$ to the UT challenger, it receives back a set of shares $\{\mathsf{s}_j\}_{j\in S}$ that are either properly generated shares $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{UT.Setup}(1^\lambda, 1^\ell, 1^d, (\mathsf{dabe.msk}, \mathsf{prf.k}))$ or simulated shares $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^\ell, 1^d, S, 1^{|\mathsf{dabe.msk}|+|\mathsf{prf.k}|})$. Therefore, depending on whether $\mathcal{B}$ is interacting with a real or ideal UT challenger, it correctly simulates the setup phase of either HYB_1 or HYB_2.

Let us now analyze the way algorithm $\mathcal{B}$ simulates the query phase. It is easy to see from definition that algorithm $\mathcal{B}$ perfectly simulates the response to $\mathcal{A}$'s challenger query as in HYB_1 and HYB_2. Therefore, let us focus on the way $\mathcal{B}$ simulates $\mathcal{A}$'s key queries. On a key query (req, j, x_j) that $\mathcal{A}$ makes, algorithm $\mathcal{B}$ first extracts $x'_i \leftarrow \mathsf{OTExt}(\mathsf{td}, \mathsf{pk}_i)$ for $i \in [\ell]$ and computes $(\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]}) \leftarrow \mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}(\mathsf{dabe.msk}, \mathsf{prf.k})$, $(\mathsf{dabe.sk}'_{i,b}, \{\alpha_{k,i}\}) \leftarrow \mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}(\mathsf{dabe.msk}, \mathsf{prf.k})$ for $i \in [\ell]$, $b \in \{0,1\}$. Then, it submits the circuits (and their evaluations)

- $\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}$ and $(\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]})$,
- $\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,x_j}$ and $(\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\setminus\{j\}})$,

- $\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}$ and $(\mathsf{dabe.sk}'_{i,x'_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})$ for $i \in [\ell]\setminus\{j\}$

to receive the evaluation shares $\mathsf{p}^{(\mathsf{bind})}_{0,j}$, $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$, and $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j}$ for $i \in [\ell]\setminus\{j\}$. By definition, if $\mathcal{B}$ is interacting with a real UT challenger, it receives the shares

- $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \mathsf{s}_j)$,
- $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,x_j}, \mathsf{s}_j)$,
- $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j} \leftarrow \mathsf{UT.Eval}(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}, \mathsf{s}_j)$ for $i \in [\ell]\setminus\{j\}$,

Hence, these evaluation shares are distributed identically as in HYB_1. If $\mathcal{B}$ is interacting with an ideal UT challenger, then the partial evaluations $\mathsf{p}^{(\mathsf{bind})}_{0,j}$, $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$, and $\{\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j}\}_{i\in[\ell]\setminus\{j\}}$ are distributed uniformly at random in $\{0,1\}^\rho$ or

- $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]}), \{\mathsf{p}^{(\mathsf{bind})}_{0,k}\}_{k\in[\ell]\setminus S}\big)$,
- $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,x_j}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\setminus\{j\}}), \{\mathsf{p}^{(\mathsf{att})}_{j,x_j,k}\}_{k\in[0,\ell]\setminus(S\cup\{j\})}\big)$,
- $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_{i,x'_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})\{\mathsf{p}^{(\mathsf{att})}_{i,x'_i,k}\}_{k\in[0,\ell]\setminus(S\cup\{i\})}\big)$ for $i \in [\ell]\setminus\{j\}$,

depending on whether it previously submitted the circuits $\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}$, $\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},k,x_k}$, and $\{\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}\}_{i\in[\ell]\setminus\{j\}}$ for all $k \in [\ell]\setminus\{j\}$. This is precisely the distribution of the evaluation shares that are generated by the challenger in HYB_2 by design. As algorithm $\mathcal{B}$ sets $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j} \leftarrow \perp$ for $i \in [\ell]\setminus\{j\}$, $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}), \mathsf{ot.pk}_j)$ for $i \in [\ell]\setminus\{j\}$, and $\mathsf{sk}_{\mathsf{req},j,x_j} = (\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,b,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\setminus\{j\}})$, algorithm $\mathcal{B}$ perfectly simulates HYB_1 or HYB_2 depending on whether it is interacting with the real or the ideal UT challenger.

As $\mathcal{B}$ correctly simulates both the setup and query phase of either HYB_1 or HYB_2 to $\mathcal{A}$, it can break the security of the universal thresholdizer scheme with the same distinguishing advantage as $\mathcal{A}$. Therefore, assuming that Π_{HE} is a secure universal thresholdizer scheme, $\mathcal{A}$'s distinguishing advantage of HYB_1 and HYB_2 must be negligible. The lemma follows. □

Lemma B.3. *Suppose that the PRFs $F_0 : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^r$ and $F_1 : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^{\ell t}$ satisfy security (Definition 3.6). Then, for any efficient adversary $\mathcal{A}$,*

$$\big|\Pr[\text{HYB}_2(\mathcal{A}) = 1] - \Pr[\text{HYB}_3(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. The lemma follows immediately from the definition of PRF security (Definition 3.6). □

Lemma B.4. *Suppose that the decomposable attribute-based encryption scheme Π_{DABE} satisfies security (Definition 6.4). Then, for any efficient adversary $\mathcal{A}$,*

$$\big|\Pr[\text{HYB}_3(\mathcal{A}) = 1] - \Pr[\text{HYB}_4(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an (admissible) adversary that distinguishes experiments HYB_3 and HYB_4. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to break the DABE security of Π_{DABE}. Interacting with the DABE challenger, algorithm $\mathcal{B}$ simulates each phases of experiment as follows:

- *Setup phase*: At the start of the experiment, adversary $\mathcal{A}$ specifies a set of corrupt authorities $S \subsetneq [\ell]$ and a circuit $\phi \in \mathcal{C}$ for the challenge ciphertext. Algorithm $\mathcal{B}$ submits ϕ to the decentralized ABE challenger to receive back $\mathsf{dabe.pp}$. Then, it samples $(\mathsf{ot.crs}, \mathsf{td}) \leftarrow \mathsf{OT.Setup}(1^\lambda)$, and instantiates the shares $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{Sim}_1(1^\lambda, 1^\ell, 1^d, S, 1^\kappa)$ where κ is the maximum bit length of any $\mathsf{dabe.msk}$ and $\mathsf{prf.k}$. It provides $\mathsf{pp} = (\mathsf{dabe.pp}, \mathsf{ot.crs})$ and $\{\mathsf{s}_j\}_{j \in S}$ to $\mathcal{A}$.

- *Query phase*: Algorithm $\mathcal{B}$ responds to $\mathcal{A}$'s queries as follows:

  - *Key queries*: Throughout the query phase, algorithm $\mathcal{B}$ maintains the following lookup tables:

    * The table $H : \{0,1\}^* \times [0, \ell] \times [\ell] \backslash S \to \{0,1\}^\rho$ as specified in HYB_2 (and therefore in HYB_3 and HYB_4),
    * A table $H^{(\mathsf{bind})} : \{0,1\}^* \to \{0,1\}^{\ell t}$ that maps request strings to the output values of $\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}$ as $\mathsf{req} \mapsto (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k \in [\ell]})$,
    * A table $H^{(\mathsf{att})} : \{0,1\}^* \times [\ell] \times \{0,1\}^{\ell t}$ that maps request-index pairs to the output values of $\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,b}$ as $(\mathsf{req}, i) \mapsto (\mathsf{dabe.sk}'_{i,x'_i}, \{\alpha_{k,i}\}_{k \in [0,\ell] \backslash \{i\}})$.

    When $\mathcal{A}$ makes a key query (req, j, x_j), algorithm $\mathcal{B}$ first parses $\mathsf{req} = \{\mathsf{pk}_i\}_{i \in [\ell]}$ and extracts $x'_i \leftarrow \mathsf{OTExt}(\mathsf{pk}_i)$ for $i \in [\ell]$. Then, it simulates the evaluation shares depending on the following conditions:

    1. $x_j = x'_j$,
    2. $\mathcal{A}$ had previously made queries (req, i, x'_i) for all $i \in [\ell] \backslash (S \cup \{j\})$.

    If any of these conditions are not satisfied, then algorithm $\mathcal{B}$ sets the evaluation shares as follows:

    * To set $\mathsf{p}^{(\mathsf{bind})}_{0,j}$, the challenger checks if $\mathcal{A}$ had previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for each $k \in [\ell] \backslash S$. If this is the case, it first looks up the mappings $(\mathsf{req}, 0, k) \mapsto \mathsf{p}^{(\mathsf{att})}_{0,k}$ for $k \in [\ell] \backslash S$ in H. Then, it checks whether there exists a mapping $\mathsf{req} \mapsto (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k \in [\ell]})$ in $H^{(\mathsf{bind})}$. If a mapping does not exist, then it samples $(\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k \in [\ell]}) \xleftarrow{\text{R}} \{0,1\}^{\ell t}$ and adds the mapping $\mathsf{req} \mapsto (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k \in [\ell]})$ to $H^{(\mathsf{bind})}$. Finally, it invokes the simulator $\mathsf{p}^{(\mathsf{bind})}_{0,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{bind})}_{\mathsf{req}}, \{\mathsf{s}_j\}_{j \in [\ell]}, (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k \in [\ell]}), \{\mathsf{p}^{(\mathsf{bind})}_{0,k}\}_{k \in [\ell]}\big)$.
      Otherwise, if $\mathcal{A}$ had not previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for all $k \in [\ell]$, then it simply samples $\mathsf{p}^{(\mathsf{bind})}_{0,j} \xleftarrow{\text{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, 0, j) \mapsto \mathsf{p}^{(\mathsf{bind})}_{0,j}$ to H.
    * To set $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$, the challenger first checks if $x'_j = x_j$. If this is not the case, then it samples $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \xleftarrow{\text{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, j, j) \mapsto \mathsf{p}^{(\mathsf{bind})}_{j,x_j,j}$ to H.
      Otherwise, if $x'_j = x_j$, then the challenger checks if $\mathcal{A}$ had previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for each $k \in [\ell] \backslash \{j\}$. If this is the case, it first looks up the mappings $(\mathsf{req}, j, k) \mapsto \mathsf{p}^{(\mathsf{att})}_{j,x_j,k}$ for $k \in [\ell] \backslash (S \cup \{i\})$ in H. Then, it checks whether there exists a mapping $(\mathsf{req}, j) \mapsto (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k \in [0,\ell] \backslash \{j\}})$ in $H^{(\mathsf{att})}$. If a mapping does not exist, then it samples $(\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k \in [0,\ell] \backslash \{j\}}) \xleftarrow{\text{R}} \{0,1\}^{\ell t}$ and adds the mapping $(\mathsf{req}, j) \mapsto (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k \in [0,\ell] \backslash \{j\}})$ to $H^{(\mathsf{att})}$. Finally, it invokes the simulator $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,x_j}, \{\mathsf{s}_j\}_{j \in [\ell]}, (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k \in [0,\ell] \backslash \{j\}}), \{\mathsf{p}^{(\mathsf{att})}_{j,x_j,k}\}_{k \in [0,\ell] \backslash \{j\}}\big)$.

Algorithm $\mathcal{B}$ also addes the mapping $(\mathsf{req}, j) \mapsto (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\setminus(S\cup\{j\})})$ to $H^{(\mathsf{att})}$.

Otherwise, if $\mathcal{A}$ had not previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for all $k \in [\ell]\setminus(S \cup \{j\})$, then it samples $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, j, j) \mapsto \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$ to H.

* To set $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j}$ the challenger proceeds for each $i \in [\ell]\setminus\{j\}$ as follows. It first checks if $\mathcal{A}$ had previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for each $k \in [\ell]\setminus\{i\}$. If this is the case, it first looks up the mappings $(\mathsf{req}, i, k) \mapsto \mathsf{p}^{(\mathsf{att})}_{i,x'_i,k}$ for $k \in [\ell]\setminus\{i\}$ in H. Then, it checks if there exists a mapping $(\mathsf{req}, i) \mapsto (\mathsf{dabe.sk}'_{i,x_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})$ in $H^{(\mathsf{att})}$. If a mapping does not exist, then it samples $(\mathsf{dabe.sk}'_{i,x_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}}) \xleftarrow{\mathrm{R}} \{0,1\}^{\ell t}$ and adds $(\mathsf{req}, i) \mapsto (\mathsf{dabe.sk}'_{i,x_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})$ to $H^{(\mathsf{att})}$. Finally, it invokes the simulator $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},i,x'_i}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_{i,x'_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}}), \{\mathsf{p}^{(\mathsf{att})}_{i,x'_i,k}\}_{k\in[0,\ell]\setminus(S\cup\{i\})}\big)$. Algorithm $\mathcal{B}$ also adds the mapping $(\mathsf{req}, i) \mapsto (\mathsf{dabe.sk}'_{i,x'_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})$ to $H^{(\mathsf{att})}$. If $\mathcal{A}$ had not previously made queries $(\mathsf{req}, k, 0)$ or $(\mathsf{req}, k, 1)$ for all $k \in [\ell]\setminus\{i\}$, then it samples $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and adds $(\mathsf{req}, i, j) \mapsto \mathsf{p}^{(\mathsf{bind})}_{i,x'_i,j}$ to H.
* For the rest of the evaluation shares $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j}$ for $i \in [\ell]\setminus\{j\}$, it sets each of them $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j} \leftarrow \perp$.

Otherwise, if $x_j = x'_j$ and $\mathcal{A}$ had previously made queries (req, i, x'_i) for all $i \in [\ell]$, then algorithm $\mathcal{B}$ submits the attribute string $(x'_1, \ldots, x'_\ell)$ to its DABE challenger to receive back the key components $\mathsf{dabe.sk}_0, \mathsf{dabe.sk}_{1,x'_1}, \ldots, \mathsf{dabe.sk}_{\ell,x'_\ell}$. It then sets the evaluation shares as follows:

* To set $\mathsf{p}^{(\mathsf{bind})}_{0,j}$, the challenger looks up the mapping $(\mathsf{req}, 0, j) \mapsto \mathsf{p}^{(\mathsf{att})}_{0,j}$ in H.
* To set $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j}$, the challenger first looks up the mappings $\mathsf{req} \mapsto (\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]})$ in $H^{(\mathsf{bind})}$ and $(\mathsf{req}, i) \mapsto (\mathsf{dabe.sk}'_{i,x_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})$ for $i \in [\ell]\setminus\{j\}$. Then, it sets $(\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\setminus\{j\}})$ as follows:
  - $\mathsf{dabe.sk}'_{j,x_j} = \mathsf{dabe.sk}_{j,x_j} \oplus \Big(\bigoplus_{k\in[0,\ell]\setminus\{j\}} \alpha_{j,k}\Big)$,
  - $\alpha_{k,j} = \mathsf{dabe.sk}_{i,x'_i} \oplus \mathsf{dabe.sk}'_{i,x'_i} \oplus \Big(\bigoplus_{k\in[0,\ell]\setminus\{i,j\}} \alpha_{i,k}\Big)$ for $i \in [\ell]$.

  Then, it invokes the simulator to generate the evaluation shares $\mathsf{p}^{(\mathsf{att})}_{j,x_j,j} \leftarrow \mathsf{Sim}_2\big(\mathsf{C}^{(\mathsf{att})}_{\mathsf{req},j,x_j}, \{\mathsf{s}_j\}_{j\in[\ell]}, (\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\setminus\{j\}}), \{\mathsf{p}^{(\mathsf{att})}_{j,x_j,k}\}_{k\in[0,\ell]\setminus\{j\}}\big)$.
* To set $\mathsf{p}^{(\mathsf{att})}_{i,x'_i,j}$ for $i \in [\ell]\setminus\{j\}$, the challenger looks up the mappings $(\mathsf{req}, i, k) \mapsto \mathsf{p}^{(\mathsf{att})}_{i,x'_i,k}$ for $k \in [\ell]\setminus\{i\}$ in H.
* For the rest of the evaluation shares $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j}$ for $i \in [\ell]\setminus\{j\}$, it sets each of them $\mathsf{p}^{(\mathsf{att})}_{i,1-x'_i,j} \leftarrow \perp$.

Finally, algorithm $\mathcal{B}$ encrypts $\mathsf{ot.ct}_{i,j} \leftarrow \mathsf{OT.Encrypt}_{\mathsf{S}}(\mathsf{ot.crs}, (\mathsf{p}^{(\mathsf{att})}_{i,0,j}, \mathsf{p}^{(\mathsf{att})}_{i,1,j}), \mathsf{ot.pk}_j)$ for all $i \in [\ell]\setminus\{j\}$ and provides $\mathsf{sk}_{\mathsf{req},j,x_j} = (\mathsf{p}^{(\mathsf{bind})}_{0,j}, \mathsf{p}^{(\mathsf{att})}_{j,x_j,j}, \{\mathsf{ot.ct}_{i,j}\}_{i\in[\ell]\setminus\{j\}})$ to $\mathcal{A}$.

– *Challenge query*: When $\mathcal{A}$ makes a challenge query (μ_0, μ_1), algorithm $\mathcal{B}$ submits (μ_0, μ_1) to the DABE challenger and receives back ct. It relays ct to $\mathcal{A}$.

- *Output phase*: When $\mathcal{A}$ outputs a bit β', algorithm $\mathcal{B}$ also outputs β'.

Correctness of the simulation. Algorithm $\mathcal{B}$ correctly simulates the setup phase of HYB_3 and HYB_4 simply by definition. It is also easy to see that depending on whether $\mathcal{B}$ is interacting in $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 0)$ or $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 1)$, it perfectly simulates the response to $\mathcal{A}$'s challenge query as in HYB_3 or HYB_4.

Therefore, the only component to verify is whether algorithm $\mathcal{B}$ perfectly simulates the responses to $\mathcal{A}$'s key queries. However, this also follows by construction. Namely, during the simulation, when $\mathcal{A}$ makes a key query (req, j, x_j), algorithm $\mathcal{B}$ samples the components $(\mathsf{dabe.sk}'_0, \{\alpha_{k,0}\}_{k\in[\ell]})$, $(\mathsf{dabe.sk}'_{j,x_j}, \{\alpha_{k,j}\}_{k\in[0,\ell]\setminus\{j\}})$, and $(\mathsf{dabe.sk}'_{i,x_i}, \{\alpha_{k,i}\}_{k\in[0,\ell]\setminus\{i\}})$ uniformly at random in $\{0,1\}^{\ell t}$ under the condition that

- $\mathsf{dabe.sk}_0 = \mathsf{dabe.sk}'_0 \oplus \left(\bigoplus_{k\in[\ell]} \alpha_{0,k}\right)$,
- $\mathsf{dabe.sk}_{i,x_i} = \mathsf{dabe.sk}'_{i,x_i} \oplus \left(\bigoplus_{k\in[0,\ell]\setminus\{i\}} \alpha_{i,k}\right)$ for $i \in [\ell]$.

This is exactly the way the challenger generates these components in the specification of the experiments HYB_3 and HYB_4.

Finally, we must determine whether algorithm $\mathcal{B}$ is an admissible adversary for the DABE security experiment. Let $\mathsf{req} = \{\mathsf{pk}_j\}_{j\in[\ell]}$ be any request string that $\mathcal{A}$ submits as a key query during the simulation and let $x'_j \leftarrow \mathsf{OTExt}(\mathsf{td}, \mathsf{pk}_j)$ for $j \in [\ell]$. Then, by construction, algorithm $\mathcal{B}$ ever makes a key query on the attribute string $(x'_1, \ldots, x'_\ell) \in \{0,1\}^\ell$ to the DABE challenger if and only if $\mathcal{A}$ submits the set of queries $\{(\mathsf{req}, j, x'_j)\}_{j\in[\ell]\setminus S}$ during the simulation. Since $\mathcal{A}$ itself is an admissible adversary for the MA-ABE security game, we have $\phi(x'_1, \ldots, x'_\ell) = 1$. Therefore, algorithm $\mathcal{B}$ is also an admissible adversary for the DABE security experiment.

As $\mathcal{B}$ correctly simulates $\mathcal{A}$'s view of the experiments HYB_3 and HYB_4 depending whether it is interacting in $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 0)$ or $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 1)$, it can distinguish the two experiments with the same advantage as $\mathcal{A}$. Therefore, assuming that Π_{DABE} is a secure DABE, $\mathcal{A}$'s distinguishing advantage of HYB_3 and HYB_4 must be negligible. The lemma follows.

□

Lemma B.5. *Suppose that the PRF $F_0 : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^r$ and $F_1 : \mathcal{K} \times \{0,1\}^* \to \{0,1\}^{\ell t}$ satisfy security (Definition 3.6). Then, for any efficient adversary $\mathcal{A}$,*

$$\left|\Pr[\text{HYB}_4(\mathcal{A}) = 1] - \Pr[\text{HYB}_5(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. The proof is identical to the proof of Lemma B.3. □

Lemma B.6. *Suppose that the universal thresholdizer Π_{UT} satisfies strong security (Definition 4.4). Then, for any efficient adversary $\mathcal{A}$,*

$$\left|\Pr[\text{HYB}_5(\mathcal{A}) = 1] - \Pr[\text{HYB}_6(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. The proof is identical to the proof of Lemma B.2. □

Lemma B.7. *Suppose that the oblivious transfer protocol Π_{OT} satisfies sender security (Definition 3.16). Then, for any efficient adversary $\mathcal{A}$,*

$$\left|\Pr[\text{HYB}_6(\mathcal{A}) = 1] - \Pr[\text{HYB}_7(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. The proof is identical to the proof of Lemma B.1. □

B.3 Proof of Theorem 6.9

The theorem follows from the definition of receiver privacy (Definition 3.15) in a straightforward way. Let $\mathcal{A}$ be an adversary in the receiver privacy experiment (Definition 5.7). We construct an algorithm $\mathcal{B}$ that uses $\mathcal{B}$ to break the receiver privacy of Π_{OT}. Algorithm $\mathcal{B}$ proceeds as follows:

- At the start of the experiment, algorithm $\mathcal{B}$ generates $(\mathsf{dabe.pp}, \mathsf{dabe.msk}) \leftarrow \mathsf{DABE.Setup}(1^\lambda, 1^\ell)$ and $(\mathsf{s}_1, \ldots, \mathsf{s}_\ell) \leftarrow \mathsf{UT.Setup}(1^\lambda, 1^\ell, 1^d, \mathsf{dabe.msk})$ as in the real setup algorithm. For the common reference string $\mathsf{ot.crs}$, algorithm $\mathcal{B}$ receives it from the OT challenger. It provides $\mathsf{pp} = (\mathsf{dabe.pp}, \mathsf{ot.crs})$ and $\{\mathsf{msk}_j = \mathsf{s}_j\}_{j\in[\ell]}$ to $\mathcal{A}$.
- When $\mathcal{A}$ returns a pair of attribute strings $x_0, x_1 \in \{0,1\}^\ell$, algorithm $\mathcal{B}$ first parses $x_0 = (x_{0,1}, \ldots, x_{0,\ell})$ and $x_1 = (x_{1,1}, \ldots, x_{1,\ell})$. Then, it submits $(x_{0,j}, x_{1,j})$ for all $j \in [\ell]$ to its OT challenger to receive back OT public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_\ell$. It sets $\mathsf{req} = \{\mathsf{ot.pk}_j\}_{j\in[\ell]}$ and provides req to $\mathcal{A}$.
- When $\mathcal{A}$ returns a bit β', algorithm $\mathcal{B}$ returns the same bit β'.

It is easy to see that $\mathcal{B}$'s simulation of the public parameters $\mathsf{pp} = (\mathsf{dabe.pp}, \mathsf{ot.crs})$ and the master secret keys $\{\mathsf{msk}_j = \mathsf{s}_j\}_{j\in[\ell]}$ are distributed exactly as in the real setup algorithm. When $\mathcal{A}$ returns the strings $x_0, x_1 \in \{0,1\}^\ell$, algorithm $\mathcal{B}$ submits the pairs $(x_{0,j}, x_{1,j})$ to the OT challenger. By definition, the OT challenger returns the set of public keys $\mathsf{ot.pk}_j \leftarrow \mathsf{OT.KeyGen}_{\mathsf{R}}(\mathsf{ot.crs}, x_{0,j})$ or $\mathsf{ot.pk}_j \leftarrow \mathsf{OT.KeyGen}_{\mathsf{R}}(\mathsf{ot.crs}, x_{1,j})$ for $j \in [\ell]$, which are the outputs of the real key request algorithm $\mathsf{KeyRequest}(x_0)$ and $\mathsf{KeyRequest}(x_1)$ respectively.

Since $\mathcal{B}$ perfectly simulates the challenge $\mathsf{req}_\beta \leftarrow \mathsf{KeyRequest}(x_\beta)$ for $\beta \in \{0,1\}$, it distinguishes req_0 and req_1 with the same advantage as $\mathcal{A}$. Therefore, assuming that Π_{OT} satisfies receiver privacy, Construction 6.6 must also satisfy receiver privacy. The theorem follows.

C Proofs in Section 7

C.1 Proof of Theorem 7.3

Fix $\lambda, \ell \in \mathbb{N}$, $\mu \in \mathcal{M}$, $\phi : \{0,1\}^\ell \to \{0,1\}$, and $x \in \{0,1\}^\ell$ for which $\phi(x) = 0$. We must show that for $(\mathsf{pp}, \mathsf{msk}) \leftarrow \mathsf{Setup}(1^\lambda, 1^\ell)$, $\left(\mathsf{sk}_0, \{\mathsf{sk}_{j,b}\}_{j\in[\ell],b\in\{0,1\}}\right) \leftarrow \mathsf{KeyGen}(\mathsf{msk})$, and $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\phi, \mu)$, we have

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_0, \mathsf{sk}_{1,x_1}, \ldots, \mathsf{sk}_{\ell,x_\ell}, \mathsf{ct}) = \mu\right] = 1 - \mathsf{negl}(\lambda).$$

Let $\mathsf{ct} = \left(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d, \phi\right)$ be the ciphertext that is output by the encryption algorithm $\mathsf{Encrypt}(\phi, \mu)$ and let $\mathsf{sk}_{j,x_j} = (\mathbf{C}_j, \mathbf{R}_{j,x_j})$ for $j \in [\ell]$ be the attribute key components that are output by the attribute key generation algorithm $\mathsf{KeyGen}(\mathsf{msk})$. Then, since $\mathbf{B}_j \cdot \mathbf{R}_{j,x_j} = \mathbf{C}_j + x_j \cdot \mathbf{G}$ for $j \in [\ell]$, we have

- $\mathbf{a} = \mathbf{s}^T\mathbf{A} + \mathbf{e}_{\mathbf{A}}^T$,
- $\mathbf{a}_i = \mathbf{s}^T(\mathbf{A}_i + \phi_i \cdot \mathbf{G}) + \mathbf{e}_{\mathbf{A},i}^T$ for all $i \in [N]$,
- $\mathbf{c}_j = \mathbf{b}_j^T\mathbf{R}_{j,x_j} = \mathbf{s}^T(\mathbf{C}_j + x_j \cdot \mathbf{G}) + \mathbf{e}_{\mathbf{C},j}^T$ for all $j \in [\ell]$,
- $d = \mathbf{s}^T\mathbf{d} + e_d + \lfloor q/2 \rceil \cdot \mu$,

for some vector $\mathbf{s} \in \mathbb{Z}_q^n$ and a set of error vectors $\mathbf{e}_{\mathbf{A}}, \{\mathbf{e}_{\mathbf{A},i}\}, \{\mathbf{e}_{\mathbf{C},j}\}_{j\in[N]}, e_{\mathbf{d}}$ with norm at most $m \cdot B$. Therefore, by the property of the algorithms $(\mathsf{Eval}_{\mathsf{pk}}, \mathsf{Eval}_{\mathsf{ct}})$ of Theorem 7.1, the vector $\mathbf{a}_{\mathcal{U}} \leftarrow \mathsf{Eval}_{\mathsf{ct}}\big(\mathcal{U}, (\phi, x), \big((\mathbf{a}_i)_{i\in[N]}, (\mathbf{c}_j)_{j\in[\ell]}\big)\big)$ satisfies

$$\mathbf{a}_{\mathcal{U}} = \mathbf{s}^T(\mathbf{A}_{\mathcal{U}} + \mathcal{U}(\phi, x) \cdot \mathbf{G}) + \mathbf{e}_{\mathcal{U}}^T = \mathbf{s}^T\mathbf{A}_{\mathcal{U}} + \mathbf{e}_{\mathcal{U}}^T,$$

for some error vector $\mathbf{e}_{\mathcal{U}} \in \mathbb{Z}^m$ such that $\|\mathbf{e}_{\mathcal{U}}\| \leq B \cdot m^{O(d)}$. By construction, the decryption algorithm takes $\mathsf{sk}_0 = \mathbf{u}$ and returns 0 if and only if

$$\begin{aligned}
\big|d - (\mathbf{a}^T \| \mathbf{a}_{\mathcal{U}}^T) \cdot \mathbf{u}\big| &= \big|\mathbf{s}^T\mathbf{d} + e_d + \lfloor q/2 \cdot \mu \rceil - \mathbf{s}^T(\mathbf{a}^T \| \mathbf{a}_{\mathcal{U}}^T) \cdot \mathbf{u} - (\mathbf{e}_{\mathbf{A}}^T \| \mathbf{e}_{\mathcal{U}}^T) \cdot \mathbf{u}\big| \\
&= \big|\mathbf{s}^T\mathbf{d} + e_{\mathbf{d}} + \lfloor q/2 \cdot \mu \rceil - \mathbf{s}^T\mathbf{d} - (\mathbf{e}_{\mathbf{A}}^T \| \mathbf{e}_{\mathcal{U}}^T) \cdot \mathbf{u}\big| \\
&= \big|\lfloor q/2 \cdot \mu \rceil + e_{\mathbf{d}} - (\mathbf{e}_{\mathbf{A}}^T \| \mathbf{e}_{\mathcal{U}}^T) \cdot \mathbf{u}\big| \\
&\leq \lfloor q/4 \rceil.
\end{aligned}$$

Since $|e_{\mathbf{d}}|, \|\mathbf{e}_{\mathbf{A}}\|, \|\mathbf{u}\| \leq B$ and $\|\mathbf{e}_{\mathcal{U}}\| \leq B \cdot m^{O(d)}$, we have $\big|e_{\mathbf{d}} - (\mathbf{e}_{\mathbf{A}}^T \| \mathbf{e}_{\mathcal{U}}^T) \cdot \mathbf{u}\big| \leq B + 2B^2 m^{d+2}$. The correctness now follows from the fact that $q > B^2 m^{O(d)}$.

C.2 Proof of Theorem 7.4

We proceed via a sequence of hybrid experiments that are defined as follows:

- HYB_0: This experiment corresponds to the security experiment $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 0)$ in Definition 6.3. Specifically, the challenger proceeds in each phases of the experiment as follows:
  - **Setup phase**: At the start of the experiment, the challenger receives a commitment to a constraint function $\phi \in \mathcal{C}$ from $\mathcal{A}$. The challenger generates:
    * $(\mathbf{A}, \mathbf{A}^{-1}) \leftarrow \mathsf{TrapGen}(1^\lambda)$,
    * $\mathbf{A}_i \xleftarrow{\text{R}} \mathbb{Z}_q^{n\times m}$ for $i \in [N]$
    * $(\mathbf{B}_j, \mathbf{B}_j^{-1}) \leftarrow \mathsf{TrapGen}(1^\lambda)$ for $j \in [\ell]$,
    * $\mathbf{d} \leftarrow \mathbb{Z}_q^n$,

    as in the real setup algorithm. It provides $\mathsf{pp} = \big(\mathbf{A}, \{\mathbf{A}_i\}_{i\in[N]}, \{\mathbf{B}_j\}_{j\in[\ell]}, \mathbf{d}\big)$ to $\mathcal{A}$.
  - **Query phase**: The challenger responds to $\mathcal{A}$'s queries as follows:
    * *Key queries*: For each query $x \in \{0,1\}^\ell$, the challenger samples random matrices $\mathbf{C}_1, \ldots, \mathbf{C}_\ell \xleftarrow{\text{R}} \mathbb{Z}_q^{n\times m}$. Then, it uses the trapdoor $\mathbf{B}_1^{-1}, \ldots, \mathbf{B}_\ell^{-1}$ to sample short preimages $\mathbf{R}_{j,b} \leftarrow \mathsf{Invert}(\mathbf{B}_j, \mathbf{B}_j^{-1}, \mathbf{C}_j + b\cdot\mathbf{G})$ and sets $\mathsf{sk}_{j,b} = (\mathbf{C}_j, \mathbf{R}_{j,b})$ for all $j \in [\ell]$, $b \in \{0,1\}$. Then, the challenger computes $\mathbf{A}_{\mathcal{U}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(\mathcal{U}, \big((\mathbf{A}_i)_{i\in[N]}, (\mathbf{C}_j)_{j\in[\ell]}\big)\big)$ and samples a short preimage $\mathbf{u} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}^{-1}, \mathbf{A}_{\mathcal{U}}, \mathbf{d}, \gamma)$ for $\gamma = Bm^{O(d)}$. It sets $\mathsf{sk}_0 = \mathbf{u}$ and provides $(\mathsf{sk}_0, \mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell})$ to $\mathcal{A}$.
    * *Challenge query*: For the challenge query (ϕ, μ_0, μ_1), the challenger samples a random vector $\mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_q^n$, and error vectors $\mathbf{e}_{\mathbf{A}} \leftarrow \chi_B^m$, $e_{\mathbf{d}} \leftarrow \chi_B$. It computes the ciphertext vectors
      · $\mathbf{a} = \mathbf{s}^T\mathbf{A} + \mathbf{e}_{\mathbf{A}}^T$,
      · $\mathbf{a}_i = \mathbf{s}^T(\mathbf{A}_i + \phi_i \cdot \mathbf{G}) + \mathbf{e}_{\mathbf{A}}^T \cdot \mathbf{S}_i$ for $i \in [N]$,
      · $\mathbf{b}_j = \mathbf{s}^T\mathbf{B}_j + \mathbf{e}_{\mathbf{A}}^T \cdot \widetilde{\mathbf{R}}_j$ for $j \in [\ell]$,

$\cdot\ d = \mathbf{s}^T\mathbf{d} + e_{\mathbf{d}} + \lfloor q/2 \rceil \cdot \mu_0$,

as in the real encryption algorithm and provides $\mathsf{ct} = \big(\mathbf{a}_0, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d, \phi\big)$ to $\mathcal{A}$.

– **Output phase**: At the end of the experiment, the adversary $\mathcal{A}$ outputs a guess $\beta' \in \{0,1\}$. The challenger echoes β' as the output of the experiment.

- HYB_1: This experiment is identical to HYB_0 except for the way the challenger generates the public matrices. Namely, we make the following modifications to challenger:

  1. **Setup phase**: The challenger samples the matrices $\mathbf{A}$ uniformly at random from $\mathbb{Z}_q^{n\times m}$ as opposed to invoking the trapdoor generation algorithm $\mathsf{TrapGen}$. Then, instead of sampling the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_N \xleftarrow{\text{R}} \mathbb{Z}_q^{n\times m}$ and $(\mathbf{B}_j, \mathbf{B}_j^{-1}) \leftarrow \mathsf{TrapGen}(1^\lambda)$, the challenger samples a set of uniform matrices $\mathbf{S}_1, \ldots, \mathbf{S}_N \xleftarrow{\text{R}} \{0,1\}^{m\times m}$, $\widetilde{\mathbf{R}}_1, \ldots, \widetilde{\mathbf{R}}_\ell \xleftarrow{\text{R}} \{0,1\}^{m\times m}$, and sets $\mathbf{A}_i = \mathbf{A}\cdot\mathbf{S}_i - \phi_i \cdot \mathbf{G}$ for $i \in [N]$, and $\mathbf{B}_j = \mathbf{A}\cdot\widetilde{\mathbf{R}}_j$ for $j \in [\ell]$. The rest of the setup phase remains unchanged.
  2. **Query phase**: The challenger responds to $\mathcal{A}$'s queries as follows:
     – *Key queries*: For each query $x \in \{0,1\}^\ell$, the challenger samples matrices $\mathbf{R}_{1,x_1}, \ldots, \mathbf{R}_{\ell,x_\ell} \xleftarrow{\text{R}} \chi_B^{m\times m}$ and sets $\mathbf{C}_j = \mathbf{B}\cdot\mathbf{R}_{j,x_j} - x_j\cdot\mathbf{G}$ for $j \in [\ell]$. It sets $\mathsf{sk}_{j,x_j} = (\mathbf{C}_j, \mathbf{R}_{j,x_j})$. Then, the challenger takes the matrices $\mathbf{R}_{1,x_1}, \ldots, \mathbf{R}_{\ell,x_\ell}$ and computes $\mathbf{R}_{\mathcal{U}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}\big(\mathcal{U}, (\phi, x), \big((\mathbf{A}_i)_{i\in[N]}, (\mathbf{B}_j)_{j\in[\ell]}\big), \big((\mathbf{S}_i)_{i\in[N]}, (\widetilde{\mathbf{R}}_j\cdot\mathbf{R}_j)_{j\in[\ell]}\big)\big)$. It then samples $\mathbf{u} \leftarrow \mathsf{SampleRight}(\mathbf{A}, \mathbf{A}_{\mathcal{U}}, \mathbf{R}_{\mathcal{U}}, \mathbf{d}, \gamma)$ for $\gamma = Bm^{O(d)}$, sets $\mathsf{sk}_0 = \mathbf{u}$, and provides $(\mathsf{sk}_0, \mathsf{sk}_{x_1}, \ldots, \mathsf{sk}_{x_\ell})$ to $\mathcal{A}$.
     – *Challenge query*: The challenger responds to $\mathcal{A}$'s challenge query identically as in HYB_0.

- HYB_2: This experiment is identical to HYB_1 except for the way the challenger generates the challenge ciphertext. Namely, when $\mathcal{A}$ makes a challenge query (ϕ, μ_0, μ_1) during the query phase of the experiment, the challenger samples uniformly random vectors $\mathbf{a} \xleftarrow{\text{R}} \mathbb{Z}_q^m$, $\mathbf{a}_1, \ldots, \mathbf{a}_N \xleftarrow{\text{R}} \mathbb{Z}_q^m$, $\mathbf{b}_1, \ldots, \mathbf{b}_\ell \xleftarrow{\text{R}} \mathbb{Z}_q^m$, $d \xleftarrow{\text{R}} \mathbb{Z}_q$, and provides $\mathsf{ct} = \big(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d, \phi\big)$ to $\mathcal{A}$.

- HYB_3: Starting from this experiment, we "revert" back the changes that we made in the previous hybrids. This experiment is identical to HYB_2 except for the way the challenge ciphertext. When $\mathcal{A}$ makes a challenge query (ϕ, μ_0, μ_1) during the query phase of the experiment, the challenger samples a random vector $\mathbf{s} \xleftarrow{\text{R}} \mathbb{Z}_q^n$, and error vectors $\mathbf{e}_{\mathbf{A}} \leftarrow \chi_B^m$, $e_{\mathbf{d}} \leftarrow \chi_B$. It computes the ciphertext vectors

  – $\mathbf{a} = \mathbf{s}^T\mathbf{A} + \mathbf{e}_{\mathbf{A}}^T$,
  – $\mathbf{a}_i = \mathbf{s}^T(\mathbf{A}_i + \phi_i\cdot\mathbf{G}) + \mathbf{e}_{\mathbf{A}}^T\cdot\mathbf{S}_i$ for $i \in [N]$,
  – $\mathbf{b}_j = \mathbf{s}^T\mathbf{B}_j + \mathbf{e}_{\mathbf{A}}^T\cdot\widetilde{\mathbf{R}}_j$ for $j \in [\ell]$,
  – $d = \mathbf{s}^T d + e_{\mathbf{d}} + \lfloor q/2 \rceil \cdot \mu_1$,

  and provides $\mathsf{ct} = \big(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d, \phi\big)$ to $\mathcal{A}$.

- HYB_4: This experiment is defined identically to HYB_0 except for the way the challenger generates the challenge ciphertext. Namely, instead of encrypting the message μ_0, the challenger

encrypts μ_1. This experiment corresponds to the security experiment $\mathsf{Expt}_{\Pi_{\mathsf{DABE}}}(\lambda, \ell, \mathcal{A}, 1)$ in Definition 6.3.

We now show that each consecutive hybrid experiments are indistinguishable to an adversary. Below, we write $\text{HYB}(\mathcal{A})$ to denote the random variable that represents the output of experiment HYB with respect to an adversary $\mathcal{A}$.

Lemma C.1. *Suppose that $m = \Omega(n \log q)$. Then, for any (unbounded) adversary $\mathcal{A}$,*

$$\left| \Pr[\text{HYB}_0(\mathcal{A}) = 1] - \Pr[\text{HYB}_1(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be any distinguisher for HYB_0 and HYB_1. We show that $\mathcal{A}$'s views of HYB_0 and HYB_1 for each phases of the protocol is statistically indistinguishable.

- **Setup phase**: We show that each components of the public parameters pp in the two experiments are statistically indistinguishable.
  - In HYB_0, the challenger generates the matrix $\mathbf{A}$ from the trapdoor generation algorithm $(\mathbf{A}, \mathbf{A}^{-1}) \leftarrow \mathsf{TrapGen}(1^\lambda)$ while in HYB_1, the challenger generates the matrix $\mathbf{A}' \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$. By the property of $\mathsf{TrapGen}$ (Theorem 3.4), the matrices $\mathbf{A}$ and $\mathbf{A}'$ are statistically indistinguishable.
  - In HYB_0, the challenger samples the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_N \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$ uniformly at random. In HYB_1, the challenger samples a set of matrices $\mathbf{S}_1, \ldots, \mathbf{S}_N \xleftarrow{\text{R}} \{0,1\}^{m \times m}$ and sets the matrices $\mathbf{A}'_i = \mathbf{A} \cdot \mathbf{S}_i - \phi \cdot \mathbf{G}$. By the leftover hash lemma (Theorem 3.1), the matrices $\mathbf{A} \cdot \mathbf{S}_i$ for $i \in [\ell]$ are statistically indistinguishable from uniformly random matrices in $\mathbb{Z}_q^{n \times m}$. Therefore, the matrices $\mathbf{A}_1, \ldots, \mathbf{A}_N$ in HYB_0 and $\mathbf{A}'_1, \ldots, \mathbf{A}'_N$ in HYB_1 are statistically indistinguishable.
  - In HYB_0, the challenger samples the matrices $\mathbf{B}_1, \ldots, \mathbf{B}_\ell \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$ uniformly at random. In HYB_1, the challenger samples a set of matrices $\widetilde{\mathbf{R}}_1, \ldots, \widetilde{\mathbf{R}}_\ell \xleftarrow{\text{R}} \{0,1\}^{m \times m}$ and sets the matrices $\mathbf{B}'_j = \mathbf{A} \cdot \widetilde{\mathbf{R}}_j$ for $j \in [\ell]$. By the leftover hash lemma (Lemma 3.1), the matrices $\mathbf{B}_1, \ldots, \mathbf{B}_\ell$ in HYB_0 and $\mathbf{B}'_1, \ldots, \mathbf{B}'_\ell$ in HYB_1 are statistically indistinguishable.
  - The rest of the public parameters pp are generated identically by construction.
- **Query phase**: By construction, the challenge ciphertext is generated identically in the two experiment. We show that $\mathcal{B}$'s responses to $\mathcal{A}$'s key queries are statistically indistinguishable. Let $x \in \{0,1\}^\ell$ be a key query made by $\mathcal{A}$:
  - *Attribute components*: In HYB_0, the challenger samples random matrices $\mathbf{C}_1, \ldots, \mathbf{C}_\ell \xleftarrow{\text{R}} \mathbb{Z}_q^{n \times m}$. Then, it samples preimages $\mathbf{R}_{j,x_j} \leftarrow \mathsf{Invert}(\mathbf{B}_j, \mathbf{B}_j^{-1}, \mathbf{C}_j + x_j \cdot \mathbf{G})$, and sets $\mathsf{sk}_{j,x_j} = (\mathbf{C}_j, \mathbf{R}_{j,x_j})$.

    In HYB_1, the challenger samples random matrices $\mathbf{R}'_1, \ldots, \mathbf{R}'_\ell \xleftarrow{\text{R}} \chi_B^{m \times m}$, and sets $\mathbf{C}'_j = \mathbf{B}'_j \cdot \mathbf{R}'_j - x_j \cdot \mathbf{G}$. It sets $\mathsf{sk}_{j,x_j} = (\mathbf{C}'_j, \mathbf{R}'_{j,x_j})$. By the property of the trapdoor generation algorithm (Theorem 3.4) and the leftover hash lemma (Lemma 3.1), the matrices $(\mathbf{C}_j, \mathbf{R}_j)$ in HYB_0 and $(\mathbf{C}'_j, \mathbf{R}'_j)$ in HYB_1 are statistically indistinguishable.
  - *Binding components*: In HYB_0, the challenger evaluates $\mathbf{A}_{\mathcal{U}} \leftarrow \mathsf{Eval}_{\mathsf{pk}}\big(\mathcal{U}, \big((\mathbf{A}_i)_{i \in [N]}, (\mathbf{C}_j)_{j \in [\ell]}\big)\big)$, and samples a preimage $\mathbf{u} \leftarrow \mathsf{SampleLeft}(\mathbf{A}, \mathbf{A}^{-1}, \mathbf{A}_{\mathcal{U}}, \mathbf{d}, \gamma)$. It sets $\mathsf{sk}_0 = \mathbf{u}$.

In HYB_1, the challenger evaluates $\mathbf{R}'_{\mathcal{U}} \leftarrow \mathsf{Eval}_{\mathsf{Sim}}\big(\mathcal{U}, (\phi, x), \big((\mathbf{A}'_i)_{i\in[N]}, (\mathbf{B}'_j)_{j\in[\ell]}\big), \big((\mathbf{S}'_i)_{i\in[N]}, (\widetilde{\mathbf{R}}'_j \cdot \mathbf{R}'_j)_{j\in[\ell]}\big)\big)$. It then samples a preimage $\mathbf{u}' \leftarrow \mathsf{SampleRight}(\mathbf{A}', \mathbf{A}'_{\mathcal{U}}, \mathbf{R}'_{\mathcal{U}}, \mathbf{d}', \gamma)$ for $\gamma = Bm^{O(d)}$, and provides $\mathbf{u}$ to $\mathcal{A}$. By the property of $\mathsf{Eval}_{\mathsf{Sim}}$ (Theorem 7.1), we have $\|\mathbf{R}'_{\mathcal{U}}\| \le Bm^{O(d)}$. Furthermore, by the admissibility condition of $\mathcal{A}$, the matrix $\mathbf{R}'_{\mathcal{U}}$ satisfies $\mathbf{A}\cdot\mathbf{R}'_{\mathcal{U}} - \mathcal{U}(\phi,x)\cdot\mathbf{G} = \mathbf{A}\cdot\mathbf{R}'_{\mathcal{U}} - \mathbf{G}$. Therefore, by the property of the $\mathsf{SampleLeft}$ and $\mathsf{SampleRight}$ algorithms (Theorem 3.5), the vectors $\mathbf{u}$ in HYB_1 and $\mathbf{u}'$ in HYB_2 are statistically indistinguishable.

- **Output phase**: The two experiments HYB_0 and HYB_1 are identical by definition.

We have shown that $\mathcal{A}$'s view of the experiments HYB_0 and HYB_1 are statistically indistinguishable for each phases of the experiment. It follows that any (unbounded) adversary $\mathcal{A}$ can distinguish the experiments HYB_0 and HYB_1 with at most negligible advantage. □

Lemma C.2. *Suppose that the* $\mathsf{LWE}_{n,m+1,q,\chi_B}$ *problem is hard. Then, for any efficient adversary* $\mathcal{A}$*,*

$$\big|\Pr[\text{HYB}_1(\mathcal{A}) = 1] - \Pr[\text{HYB}_2(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. Let $\mathcal{A}$ be an adversary that distinguishes HYB_1 and HYB_2. We construct an algorithm $\mathcal{B}$ that uses $\mathcal{A}$ to solve the $\mathsf{LWE}_{n,m+1,q,\chi_B}$ problem. Algorithm $\mathcal{B}$ simulates the view of $\mathcal{A}$ as follows:

- **Setup phase**: In the beginning of the experiment, algorithm $\mathcal{B}$ receives a commitment to a constraint function ϕ from $\mathcal{A}$. It then recieves $\mathsf{LWE}_{n,m+1,q,\chi_B}$ challenge vectors
  - $(\mathbf{A}, \mathbf{a}) \in \mathbb{Z}_q^{n\times m} \times \mathbb{Z}_q^m$,
  - $(\mathbf{d}, \hat{d}) \in \mathbb{Z}_q^n \times \mathbb{Z}_q$,

  and uses the matrix $\mathbf{A}$ to generate the rest of the public parameters as specified in the experiments HYB_1 and HYB_2. Namely, it sets
  - $\mathbf{A}_i = \mathbf{A}\cdot\mathbf{S}_i - \phi_i\cdot\mathbf{G}$ for $i \in [N]$ where $\mathbf{S}_i \xleftarrow{\text{R}} \{0,1\}^{m\times m}$.
  - $\mathbf{B}_j = \mathbf{A}\cdot\widetilde{\mathbf{R}}_j$ for $j \in [\ell]$ where $\widetilde{\mathbf{R}}_j \xleftarrow{\text{R}} \{0,1\}^{m\times m}$.

  It provides $\mathsf{pp} = \big(\mathbf{A}, (\mathbf{A}_i)_{i\in[N]}, (\mathbf{B}_j)_{j\in[\ell]}, \mathbf{d}\big)$ and provides pp to $\mathcal{A}$.
- **Query phase**: Algorithm $\mathcal{B}$ responds to $\mathcal{A}$'s key queries according to the specifications of HYB_1 and HYB_2 (which are identical). For the challenge query (ϕ, μ_0, μ_1), it defines the vectors
  - $\mathbf{a}_i^T = \mathbf{a}^T\cdot\mathbf{S}_i$ for $i \in [N]$,
  - $\mathbf{b}_j^T = \mathbf{a}^T\cdot\widetilde{\mathbf{R}}_j$ for $j \in [\ell]$.
  - $d = \hat{d} + \lfloor q/2 \rceil \cdot \mu_0$.

  It sets the ciphertext $\mathsf{ct} = \big(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d\big)$ and provides ct to $\mathcal{A}$.
- **Output phase**: When $\mathcal{A}$ outputs a bit $\beta' \in \{0,1\}$, algorithm $\mathcal{B}$ also outputs β'.

We now show that depending on whether $\mathcal{B}$ receives real $\mathsf{LWE}_{n,m+1,q,\chi_B}$ samples or uniformly random samples, it perfectly simulates either HYB_1 or HYB_2 for $\mathcal{A}$.

- By definition, the matrix $\mathbf{A} \in \mathbb{Z}_q^{n\times m}$ and vector $\mathbf{d} \in \mathbb{Z}_q^n$ are uniformly random. Therefore, algorithm $\mathcal{B}$ perfectly simulates the public parameters pp for HYB_1 and HYB_2.

- Algorithm $\mathcal{B}$ perfectly simulates the responses for the key generation queries for HYB_1 and HYB_2 simply by definition.
- Now, consider the challenge query that is generated by $\mathcal{B}$.
  - If $(\mathbf{a} \parallel \hat{d}) = (\mathbf{s}^T\mathbf{A} + \mathbf{e}_{\mathbf{A}}^T \parallel \mathbf{s}^T\mathbf{d} + e_{\mathbf{d}}) \in \mathbb{Z}_q^{m+1}$ for some secret vector $\mathbf{s} \in \mathbb{Z}_q^n$ and error vectors $\mathbf{e}_{\mathbf{A}}, e_{\mathbf{d}} \xleftarrow{\text{R}} \chi_B^{m+1}$, then we have
    - $\mathbf{a}_i^T = \mathbf{a}^T \cdot \mathbf{S}_i = \mathbf{s}^T\mathbf{A} \cdot \mathbf{S}_i + \mathbf{e}_{\mathbf{A}}^T \cdot \mathbf{S}_i = \mathbf{s}^T(\mathbf{A}_i + \phi_i \cdot \mathbf{G}) + \mathbf{e}_{\mathbf{A}}^T \cdot \mathbf{S}_i$ for $i \in [N]$,
    - $\mathbf{b}_j^T = \mathbf{a}^T \cdot \widetilde{\mathbf{R}}_j = \mathbf{s}^T\mathbf{A} \cdot \widetilde{\mathbf{R}}_j + \mathbf{e}_{\mathbf{A}}^T \cdot \widetilde{\mathbf{R}}_j = \mathbf{s}^T\mathbf{B}_j + \mathbf{e}_{\mathbf{A}}^T \cdot \widetilde{\mathbf{R}}_j$ for $j \in [\ell]$,
    - $d = \hat{d} + \lfloor q/2 \rceil \cdot \mu_0 = \mathbf{s}^T\mathbf{d} + e_{\mathbf{d}} + \lfloor q/2 \rceil \cdot \mu_0$.

    Therefore, the ciphertext $\mathsf{ct} = \big(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d\big)$ is distributed exactly as in HYB_1.
  - If $(\mathbf{a} \parallel \hat{d}) \xleftarrow{\text{R}} \mathbb{Z}_q^{m+1}$, then the components
    - $\mathbf{a}_i^T = \mathbf{a}^T\mathbf{S}_i$ for $i \in [N]$,
    - $\mathbf{b}_j^T = \mathbf{a}^T\widetilde{\mathbf{R}}_j$ for $j \in [N]$,
    - $d = \hat{d} + \lfloor q/2 \rceil \cdot \mu_0$,

    are all uniform by the leftover hash lemma (Lemma 3.1). Therefore, the ciphertext $\mathsf{ct} = \big(\mathbf{a}, (\mathbf{a}_i)_{i\in[N]}, (\mathbf{b}_j)_{j\in[\ell]}, d\big)$ is distributed as in HYB_3.

We have shown that depending on whether $\mathcal{B}$ receives real $\mathsf{LWE}_{n,m+1,q,\chi_B}$ samples or uniformly random samples, it perfectly simulates $\mathcal{A}$'s views of either HYB_1 and HYB_2. This means that with the same distinguishing advantage of $\mathcal{A}$, algorithm $\mathcal{B}$ solves the $\mathsf{LWE}_{n,m+1,q,\chi_B}$ problem. Therefore, assuming that the $\mathsf{LWE}_{n,m+1,q,\chi_B}$ problem is hard, $\mathcal{A}$'s distinguishing advantage of the experiments HYB_1 and HYB_2 is negligible. The lemma follows. □

Lemma C.3. *Suppose that the $\mathsf{LWE}_{n,m+1,q,\chi_B}$ problem is hard. Then, for any efficient adversary $\mathcal{A}$,*

$$\big|\Pr[\text{HYB}_2(\mathcal{A}) = 1] - \Pr[\text{HYB}_3(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. The proof is identical to the proof of Lemma C.2 □

Lemma C.4. *Suppose that $m = \Omega(n \log q)$. Then, for any (unbounded) adversary $\mathcal{A}$,*

$$\big|\Pr[\text{HYB}_3(\mathcal{A}) = 1] - \Pr[\text{HYB}_4(\mathcal{A}) = 1]\big| = \mathsf{negl}(\lambda).$$

Proof. The proof is identical to the proof of Lemma C.1. □