Multi-Authority Registered Attribute-Based Encryption

George Lu
UT Austin
gclu@cs.utexas.edu

Brent Waters
UT Austin and NTT Research
bwaters@cs.utexas.edu

David J. Wu
UT Austin
dwu4@cs.utexas.edu

Abstract

Registered attribute-based encryption (ABE) enables fine-grained access control to encrypted data without a trusted authority. In this model, users generate their own public keys and *register* their public key along with a set of attributes with a key curator. The key curator aggregates the public keys into a short master public key that functions as the public key for an ABE scheme.

A limitation of ABE (registered or centralized) is the assumption that a single entity manages all of the attributes in a system. In many settings, the attributes belong to different organizations, making it unrealistic to expect that a single entity manage all of them. In the centralized setting, this motivated the notion of multi-authority ABE, where multiple independent authorities control their individual set of attributes. Access policies are then defined over attributes across multiple authorities.

In this work, we introduce *multi-authority registered ABE*, where multiple (independent) key curators each manage their individual sets of attributes. Users can register their public keys with any key curator, and access policies can be defined over attributes from multiple key curators. Multi-authority registered ABE combines the trustless nature of registered ABE with the decentralized nature of multi-authority ABE.

We start by constructing a multi-authority registered ABE scheme from composite-order pairing groups. This scheme supports an a priori bounded number of users and access policies that can be represented by a linear secret sharing scheme (which includes monotone Boolean formulas). Our construction relies on a careful integration of ideas from pairing-based registered ABE and multi-authority ABE schemes. We also construct a multi-authority registered ABE scheme that supports an unbounded number of users and arbitrary monotone policies using indistinguishability obfuscation (and function-binding hash functions).

1 Introduction

Attribute-based encryption (ABE) [SW05, GPSW06] is a generalization of public-key encryption that enables fine-grained access control to encrypted data. In a (ciphertext-policy) ABE scheme, each user's key is associated with a set of attributes (e.g., "faculty in the math department" and "resident in California") while ciphertexts are associated with policies (e.g., "residents in Georgia" AND "faculty in Computer Science"). Only users whose attributes satisfy the ciphertext policy are able to decrypt and learn the message. While ABE augments public-key encryption with expressive fine-grained decryption capabilities, it also radically changes the trust model. Instead of users generating their own individual keys, ABE assumes there is a central *trusted* authority who is responsible for generating the decryption keys for each user. The central authority possesses a long-term secret used for key generation, and if this long-term key is ever compromised or exfiltrated, the attacker immediately gains the ability to decrypt *all* ciphertexts in the system.

Registration-based cryptography. The work of Garg, Hajiabadi, Mahmoody, and Rahimi [GHMR18] introduces the concept of registration-based encryption as an alternative approach to augment public-key encryption with fine-grained decryption capabilities *without* needing to rely on a trusted party. Specifically, [GHMR18] consider the setting of identity-based encryption (where secret keys and ciphertexts are associated with identities and decryption succeeds as long as the identities associated with the ciphertext and the secret key match). Instead of a trusted authority generating decryption keys, in registration-based encryption, users generate their own public/private key-pairs (just as in traditional public-key encryption). Then, the user registers their *public* key (along with their identity) with a key curator. The key curator's job is to *aggregate* all of the individual users' keys into a single short public key that functions as a public key for an identity-based encryption scheme. While the key curator is a central authority, it does *not* possess any secrets. The entire key aggregation process is a deterministic and transparent procedure. Thus, anyone can audit the state of the key curator and verify that it is behavior correctly.

Since the work of [GHMR18], many works have explored constructions of registration-based encryption from different cryptographic assumptions as well as new designs with better concrete efficiency [GHM+19, GV20, CES21, GKMR23, DKL+23, FKdP23]. The concept of registration-based cryptography has also been extended to the setting of attribute-based encryption [HLWW23, FWW23, ZZGQ23, GLWW24, AT24, CHW25, ZZC+25, WW25], functional encryption [FFM+23, BLM+24, DPY24, PS25], broadcast encryption [BZ14, FWW23, GLWW23, KMW23, CW24, GKPW24, CHW25, WW25], and traitor tracing [BLM+24].

Registered ABE. In this work, we focus on registered ABE, a notion first introduced in the work of Hohenberger, Lu, Waters, and Wu [HLWW23]. In this setting, users generate their own public/private keys and then register their public key together with a set of attributes with the key curator. Typically, we consider the key curator to be an administrator that controls a set of attributes for an organization or a domain. When a user registers their public key, the key curator would determine (possibly in collaboration with the user) which set of attributes are most appropriate. As a concrete example, a company might run a registered ABE scheme to manage keys for their employees. When an employee registers their public key with the system, the system administrator would assign the employee a set of attributes according to their role and access control permissions within the company. Using registered ABE in place of vanilla ABE ensures that there is no central point of failure within the system (e.g., the system administrator does not have any secrets of its own).

The challenge: cross-domain policies. Standard registered ABE works well when there is a single key curator responsible for a collection of attributes within their particular domain. However, in many applications, access policies might cross domain boundaries. For a concrete use case, one can imagine a collaborative research project between multiple universities and government agencies. Here, access to sensitive data could be dictated by a combination of institutional affiliation and role within each organization (e.g., faculty at a university with a government clearance). To support such policies with standard registered ABE, we would require a single key curator that is responsible for managing attributes for *both* institutions, which would in turn require coordination between the two. A more natural solution would be to allow the two entities to individually manage their own attributes, but still support encryption to policies that involve attributes across multiple domains.

Multi-authority ABE. A sequence of works [Cha07, LCLS08, MKE08, CC09, LW11] have shown how to extend vanilla ABE to the *multi-authority* setting. In a multi-authority ABE scheme, there are many key-issuing authorities and each authority is only responsible for issuing keys for the attributes in their domain. Moreover, during encryption, the user can encrypt to policies defined over attributes from different authorities. Notably, the fully decentralized scheme of Lewko and Waters [LW11] allows anyone to be an authority and does not require any interaction between authorities. However, much like standard ABE, each authority still retains a long-term master secret key in order to issue keys.

Registered multi-authority ABE. In this work, we introduce a generalization of registered ABE to the multi-authority setting. In this model, anyone can become a key curator. Each key curator has a set of attributes that they manage, and users can register their public key with any number of key curators. We do not require any coordination between key curators. During encryption, one can specify an access policy over any collection of attributes, which can involve attributes from multiple independent key curators. As usual, decryption is successful only if the user possesses a set of attributes that satisfy the access policy. Multi-authority registered ABE thus combines the trustless nature of registered ABE with the support for cross-domain policies in multi-authority ABE.

1.1 Our Contributions

In this work, we formally initiate the study of multi-authority registered ABE.

Definitions and construction blueprint. Our first contribution is a formal definition that combines the features of multi-authority ABE [LW11] with registered ABE [HLWW23]. We then describe a general template for constructing multi-authority registered ABE by starting from a "slotted" version of the primitive (similar to [GHMR18, HLWW23]) that does *not* support dynamic registrations. In this model, we assume that all of the users register at the same time and there is a key-aggregation algorithm that combines their public keys into the public key for a particular authority. In the single-authority setting, it is straightforward to lift from the slotted scheme to the standard variant where users can dynamically register using a power-of-two approach. The analogous transformation is much more complicated in the multi-authority setting because each user can register for a different subset of authorities, and we do not allow any coordination between authorities. Nonetheless, we show in Section 6 that using cover-free sets, we can generically transform any slotted multi-authority registered ABE scheme into a full multi-authority registered ABE scheme. This allows us to focus on constructing the simpler slotted notion of the primitive in the rest of the paper.

Slotted multi-authority registered ABE from pairings. Next, we show how to construct a slotted multi-authority registered ABE scheme using composite-order pairing groups. Our scheme supports an a priori bounded number of users (as is the case for all pairing-based registered ABE schemes [HLWW23, ZZGQ23, GLWW24, AT24]) and access policies that can be described by a monotone Boolean formula (or more generally, any policy that admits a linear secret sharing scheme). Our construction leverages ideas from the original registered ABE scheme from [HLWW23] together with the multi-authority ABE scheme from [LW11]. A notable feature of our (slotted) multi-authority registered ABE is that we do *not* need to rely on the random oracle model (in contrast to pairing-based multi-authority ABE schemes such as [LW11]). We refer to Section 2 for a more detailed technical overview of our construction as well as a discussion of how the semantics of multi-authority registered ABE allows us to achieve security in the plain model (*without* random oracles). Note that multi-authority registered ABE does not imply multi-authority ABE (just as registered ABE does not imply standard ABE), so our results do *not* imply a multi-authority centralized ABE scheme without random oracles.

Multi-authority registered ABE from obfuscation. Our pairing-based construction inherits several of the limitations that apply to all pairing-based registered ABE schemes: (1) it needs a large structured common reference string; (2) it supports a restricted set of policies (e.g., monotone Boolean formulas); and (3) it imposes an a priori bound on the total number of users. To show that it is feasible to overcome all of these limitations, we also construct a multi-authority registered ABE scheme using indistinguishability obfuscation and function-binding hash functions [FWW23]; the latter can be built from the learning with errors (LWE) assumption. Our obfuscation-based construction has a transparent (i.e., public-coin) setup process and supports arbitrary policies and an arbitrary number of users.

2 Technical Overview

In this work, we define the notion of *multi-authority* registered ABE and give constructions from bilinear groups and indistinguishability obfuscation. Similar to multi-authority ABE [LW11], we assume that each user has a unique (global) user identifier gid. The user identifier will be used to tie the decryption keys (across different authorities) together. We also assume that each authority has a unique authority identifier aid. For ease of exposition, we focus on the setting where each authority manages just one attribute. This way, we can view policies as functions over authorities. We begin with the syntax of a multi-authority registered ABE scheme (and refer to Definition 4.1 for the full definition):

- $\mathsf{GlobalSetup}(1^\lambda) \to (\mathsf{gpp}, \mathsf{gep})$: The global setup algorithm takes the security parameter λ and outputs the global public parameters gpp and a (compact) set of global encryption parameters gep.

- $\mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid}) \to (\mathsf{pk}_{\mathsf{gid}}, \mathsf{sk}_{\mathsf{gid}})$: The key-generation algorithm uses the global public parameters gpp and the user identifier gid to generate a public/secret key-pair.
- $\mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}, \mathsf{pk}_{\mathsf{gid}}) \to (\mathsf{mpk}', \mathsf{aux}')$: The registration algorithm uses the global public parameters gpp, the current state of the key curator aux, the user identifier gid, and the associated public key $\mathsf{pk}_{\mathsf{gid}}$, and outputs a new master public key mpk' and curator state aux'. We require this algorithm to be deterministic (so that it is possible to audit the key curator).
- $\mathsf{UpdateKey}(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}) \to \mathsf{hsk}_{\mathsf{gid}}$: The update algorithm takes the global public parameters gpp, the curator state aux, and the identifier gid for the user requesting an update, and outputs a helper decryption key $\mathsf{hsk}_{\mathsf{gid}}$.
- $\mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, m) \to \mathsf{ct}$: The encryption algorithm take the global encryption parameters gep, a set of authority identifiers S_{enc}, a policy function φ defined over those authorities, the master public keys $\mathsf{mpk}_{\mathsf{aid}}$ associated with the authorities $\mathsf{aid} \in S_{\mathsf{enc}}$, and a message m, and outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{gpp}, S, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}})\}_{\mathsf{aid} \in S}, \mathsf{ct}) \to m$: The decryption algorithm takes in the global public parameters gpp, a set of authority identifiers S, and the secret keys $\mathsf{sk}_{\mathsf{aid},\mathsf{gid}}$ and helper decryption keys $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}}$, and a ciphertext ct, and outputs the message m.

The correctness requirement is that any user registered with a set of authorities that satisfy the access policy associated with a ciphertext can successfully recover the message. Security says that any set of users who individually do not satisfy the access policy cannot learn anything about the encrypted message.

Slotted multi-authority registered ABE. Much like in [HLWW23], our constructions of multi-authority registered ABE will proceed in two steps: (1) we first construct a "slotted" version of the primitive; and (2) we show how to generically upgrade the slotted primitive to the full primitive. The slotted multi-authority registered ABE scheme makes the following simplifying assumptions:

- The slotted scheme supports a fixed number of users L (which is provided as an explicit parameter to the setup algorithm $\mathsf{GlobalSetup}$).
- The scheme supports one-shot aggregation rather than incremental updates to the public key. Instead of the RegPK, $\mathsf{UpdateKey}$ algorithms, the slotted scheme has a single $\mathsf{Aggregate}$ algorithm that takes as input the global parameters gpp, and a set of L public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ for user identifiers $\mathsf{gid}_1, \ldots, \mathsf{gid}_L$. The aggregate algorithm outputs a succinct master public key mpk and helper decryption keys hsk_i for each of the users.
- Finally, the key-generation algorithm is associated with a specific slot index $i \in [L]$. Namely, in the slotted multi-authority registered ABE scheme, the KeyGen algorithm takes a slot index $i \in [L]$ as input instead of the user identifier (the user identifier is now specified during aggregation).

The correctness requirement for the slotted scheme states that decryption is successful whenever a user is registered with a set of authorities that satisfy the ciphertext's access policy, and moreover, the associated public keys are all registered to the *same* slot (across all authorities). The security requirement is that the adversary cannot learn anything about an encrypted message if there is no single gid^* *and* slot i^* for which the adversary possesses an authorized set of secret keys. In Section 6, we show how to compile any slotted multi-authority registered ABE scheme into a standard multi-authority registered ABE scheme. We also provide an overview of this transformation later in this section.

Multi-authority registered ABE from pairings. Our first construction of a slotted multi-authority registered ABE will use composite-order pairing groups. In this setting, we have a cyclic group $\mathbb{G}$ of composite order $N = p_1 p_2 p_3 p_4$ (where p_1, p_2, p_3, p_4 are distinct primes). By the Chinese Remainder Theorem, this group decomposes into a product of groups $\mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_3 \times \mathbb{G}_4$ where $|\mathbb{G}_i| = p_i$. This group will also be equipped with an efficiently-computable non-degenerate bilinear map $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$. We refer to Section 3.1 for a formal definition.

We start with a simplified description of the slotted registered ABE scheme from [HLWW23]. This construction will only support an AND policy on two attributes a_0 and a_1 (i.e., the policy $\mathsf{a}_0 \wedge \mathsf{a}_1$). While this is a toy example, it will suffice for illustrating some of the technical challenges involved in generalizing this to the multi-authority setting. The [HLWW23] construction technically works over composite-order pairing groups, but for ease of exposition, we just focus on the action in the $\mathbb{G}_1$ subgroup and omit the extra randomizing factors from the other subgroups (the extra components are only needed for security, not correctness). Below, we write g to be the generator of $\mathbb{G}_1$.

- **Common reference string:** The common reference string includes a description of the group description along with the following components.
  - **General components:** The CRS contains random group elements $h \xleftarrow{R} \mathbb{G}$ and $Z = e(g, g)^\alpha$ where $\alpha \xleftarrow{R} \mathbb{Z}_N$.
  - **Slot components:** For each slot $i \in [L]$, the CRS contains elements
  $$A_i = g^{t_i} \quad , \quad B_i = g^\alpha h^{t_i} \quad , \quad U_i = g^{u_i}$$
  where $t_i, u_i \xleftarrow{R} \mathbb{Z}_N$. These terms are used in both encryption and decryption. Having independent elements for each slot is necessary to defend against collusion attacks between users registered to different slots.
  - **Cross terms:** Finally, for each pair of slots $i \neq j \in [L]$, the CRS includes cross-terms $W_{j,i} = g^{t_i u_j}$, which are used for decryption.
- **User keys:** To sample a public key for slot i, the user samples $r_i \xleftarrow{R} \mathbb{Z}_N$, and computes $T_i = g^{r_i}$ and $V_{j,i} = A_i^{r_j}$ for $j \neq i \in [L]$. The public and secret keys are then
$$\mathsf{pk} = \left(T_i\,,\, \{V_{j,i}\}_{j \neq i}\right) \quad , \quad \mathsf{sk} = r_i.$$
- **Aggregation:** To aggregate a collection of L public keys $\mathsf{pk}_i = (T_i, \{V_{j,i}\}_{j \neq i})$ together with their associated attributes $S_i \subseteq \{\mathsf{a}_0, \mathsf{a}_1\}$ into a single master public key, the key curator computes an aggregated public key $\hat{T}$ and for each $b \in \{0, 1\}$, an attribute-specific encryption key $\hat{U}_b$ (associated with the attribute a_b):
$$\hat{T} = \prod_{i \in [L]} T_i \quad , \quad \hat{U}_b = \prod_{i \in [L] : \mathsf{a}_b \notin S_i} U_i.$$
Finally, the key-curator computes a set of "cross terms" which will be important for decryption:
$$\hat{V}_i = \prod_{j \neq i} V_{j,i} \quad , \quad \hat{W}_{i,b} = \prod_{j \neq i : \mathsf{a}_b \notin S_j} W_{j,i}$$
The final master public key and helper decryption keys consists of the group elements
$$\mathsf{mpk} = \left(\hat{T}, \{\hat{U}_b\}_{b \in \{0,1\}}\right) \quad , \quad \mathsf{hsk}_i = \left(\hat{V}_i, \{\hat{W}_{i,b}\}_{b \in \{0,1\}}\right).$$
- **Encryption:** To encrypt a message μ, the encryptor samples randomness $s \xleftarrow{R} \mathbb{Z}_N$, and two group elements h_0, h_1 such that $h_0 h_1 = h$. Then, the encrypter constructs the following components:
  - **Message-embedding components:** $C_1 = \mu \cdot Z^s$ and $C_2 = g^s$.
  - **Slot-specific blinding:** $E_{\mathsf{slot}} = \hat{T}^{-s}$.
  - **Attribute-specific blinding:** For each $b \in \{0, 1\}$, $E_{\mathsf{attrib},b} = \hat{U}_b^{-s}$.

  The final ciphertext is then $(C_1, C_2, C_{3,0}, C_{3,1})$ where[1]
  $$C_{3,0} = h_0^s \cdot E_{\mathsf{slot}} \cdot E_{\mathsf{attrib},0} \quad , \quad C_{3,1} = h_1^s \cdot E_{\mathsf{slot}} \cdot E_{\mathsf{attrib},1}.$$
  Recall that we are considering the conjunction policy $\mathsf{a}_0 \wedge \mathsf{a}_1$.

[1]Here, we slightly diverge from the construction of [HLWW23] by bundling the slot-specific and the attribute-specific components together (in $C_{3,0}$ and $C_{3,1}$). The work of [HLWW23] separate these into separate terms.

- **Decryption:** For a user in slot $i \in [L]$ to decrypt, they should satisfy two requirements: (1) they know the secret key for the public key for slot i, and (2) slot i is registered to both attributes. These correspond to being able to remove the slot-specific and attribute-specific blinding components respectively. To decrypt, the user first computes
$$\frac{C_1}{e(C_2, B_i)} = \frac{\mu \cdot e(g,g)^{\alpha s}}{e(g,g)^{\alpha s} e(g,h)^{st_i}} = \frac{\mu}{e(g,h)^{st_i}}.$$
This yields the message blinded by $e(g,h)^{-st_i}$. To cancel out the $e(g,h)^{-st_i}$ term, we observe that $C_{3,0}, C_{3,1}$ essentially contain an additive secret sharing of h^s. Pairing with A_i yields $e(g,h)^{st_i}$. However, pairing $C_{3,0}, C_{3,1}$ with A_i also result in additional terms $e(A_i, E_{\mathsf{slot}})$ and $e(A_i, E_{\mathsf{attrib},b})$, respectively, which need to be removed as follows:

  - **Slot check:** First, we have
$$e(A_i, E_{\mathsf{slot}}) = e\left(g^{t_i}, \prod_{j \in [L]} g^{-sr_j}\right) = e(g,g)^{-st_i \sum_{j \in [L]} r_j}.$$
We can equivalently write this as
$$e(A_i, E_{\mathsf{slot}}) = e(g,g)^{-st_i \sum_{j \neq i} r_j} \cdot e(g,g)^{-st_i r_i},$$
which we can express as follows:
$$e(C_2, \hat{V}_i) = e\left(g^s, \prod_{j \neq i} g^{t_i r_j}\right) = e(g,g)^{st_i \sum_{j \neq i} r_j}$$
$$e(C_2, A_i^{r_i}) = e(g^s, g^{t_i r_i}) = e(g,g)^{st_i r_i}$$
This means
$$e(A_i, E_{\mathsf{slot}}) \cdot e(C_2, \hat{V}_i) \cdot e(C_2, A_i^{r_i}) = 1.$$

  - **Policy check:** Next, we have
$$e(A_i, E_{\mathsf{attrib},b}) = e\left(g^{t_i}, \prod_{j \in [L] : \mathsf{a}_b \notin S_j} g^{-su_j}\right) = e(g,g)^{-st_i \sum_{j \in [L] : \mathsf{a}_b \notin S_j} u_j}.$$
If the user (for slot i) is authorized (for the policy $\mathsf{a}_0 \wedge \mathsf{a}_1$), then they possess attributes a_0 and a_1. In this case, for all $b \in \{0,1\}$, $\mathsf{a}_b \in S_i$. This means $\{j \in [L] : \mathsf{a}_b \notin S_j\} = \{j \neq i : \mathsf{a}_b \notin S_j\}$. Now, by definition of $\hat{W}_{i,b}$, we have
$$e(C_2, \hat{W}_{i,b}) = e\left(g^s, \prod_{j \neq i : \mathsf{a}_b \notin S_j} g^{t_i u_j}\right) = e(g,g)^{st_i \sum_{j \in [L] : \mathsf{a}_b \notin S_j} u_j}.$$
This means
$$e(A_i, E_{\mathsf{attrib},b}) \cdot e(C_2, \hat{W}_{i,b}) = 1.$$

  Putting everything together, for $b \in \{0,1\}$, the decrypter first computes
$$\begin{aligned} D_{\mathsf{a}_b} &= e(C_{3,b}, A_i) \cdot e(C_2, \hat{V}_i A_i^{r_i} \hat{W}_{i,b}) \\ &= e(h_b^s, A_i) e(E_{\mathsf{slot}}, A_i) e(E_{\mathsf{attrib},b}, A_i) e(C_2, \hat{V}_i) e(C_2, A_i^{r_i}) e(C_2, \hat{W}_{i,b}) \\ &= e(h_b^s, A_i). \end{aligned}$$
Finally, the decrypter outputs
$$\frac{C_1}{e(C_2, B_i)} \cdot D_{\mathsf{a}_0} \cdot D_{\mathsf{a}_1} = \frac{\mu}{e(g,h)^{st_i}} \cdot e(h^s, A_i) = \frac{\mu}{e(g,h)^{st_i}} \cdot e(g,h)^{st_i} = \mu.$$

Independent aggregation. First, we observe that the aggregation algorithm in the above scheme already is naturally decomposable. An authority (with associated attribute a) can take a collection of public keys $\mathsf{pk}_{\mathsf{a},i} = (T_{\mathsf{a},i}, \{V_{\mathsf{a},j,i}\}_{j \neq i})$ and compute the aggregated terms $\hat{T}_{\mathsf{a}}, \hat{U}_{\mathsf{a}}, \hat{V}_{\mathsf{a},i}, \hat{W}_{\mathsf{a},i}$ as in the single-authority scheme. When it comes time to encrypt, the encrypter can compute the ciphertext components C_1, C_2 exactly as above. For the slot-specific and attribute-specific components E_{slot} and $E_{\mathsf{attrib},b}$ that comprise $C_{3,b}$, the encrypter would then use the aggregated components $\hat{T}_{\mathsf{a}_b}, \hat{U}_{\mathsf{a}_b}$ from the relevant authority a_b.

Preventing user collusion. The main security challenge in the multi-authority setting is preventing the adversary from combining key components for different user identifiers gid across multiple authorities to decrypt an unauthorized ciphertext. Of course, for correctness, the user should be able to combine key components for a common user identifier. In multi-authority ABE, the standard approach to prevent collusion is to use a hash $H(\mathsf{gid})$ of the user identifier as a source of common randomness to tie together the secret keys issued by different authorities for the same gid [LW11, RW15, DKW21, DKW23, WWW22]. With the exception of [WWW22], the hash function is modeled as a random oracle in the security analysis.

In the registration-based setting, the secret keys are chosen by the user rather than a central authority. Thus, to bind the user keys to a particular gid, the key curator uses $H(\mathsf{gid})$ to derive the common randomness for each gid at aggregation time. In particular, the key curator now computes the attribute-specific encryption key $\hat{U}_{\mathsf{a}}$ as

$$\hat{U}_{\mathsf{a}} = \prod_{i \in [L]} H(\mathsf{gid}_{\mathsf{a},i}),$$

where $H(\cdot)$ outputs a group element in $\mathbb{G}_1$ and $\mathsf{gid}_{\mathsf{a},i}$ is the user identifier registered to authority a on slot i. An important distinction between the single-authority scheme sketched above and the multi-authority scheme is the fact that in the attribute-specific encryption key, *every* slot index $i \in [L]$ must be associated with some term $H(\mathsf{gid}_{\mathsf{aid},i})$. This will be important to prevent collusion attacks (where a user tries to mix and match components associated with different authorities). We contrast this with the single-authority setting where it was sufficient for the key curator to omit U_i for any slot i associated with an user that does not possess the associated attribute. Finally, a straightforward way in the multi-authority setting for ensuring every slot is associated with some $\mathsf{gid}_{\mathsf{a},i}$ is to have the key curator associate a dummy key with any unused slot.

Returning now to the construction, in the ciphertext, to ensure $E_{\mathsf{attrib},b}$ *only* cancels out when the decrypter uses the *same* user identifier gid, the encryptor additionally samples $s_0 \xleftarrow{\text{R}} \mathbb{Z}_N$ and computes

$$E_{\mathsf{attrib},0} = \hat{U}_{\mathsf{a}_0}^{s_0} \quad , \quad E_{\mathsf{attrib},1} = \hat{U}_{\mathsf{a}_1}^{-s_0}.$$

The key observation is that $H(\mathsf{gid}_{\mathsf{a}_0,i})^{s_0}$ and $H(\mathsf{gid}_{\mathsf{a}_1,i})^{-s_0}$ cancel out only when the same gid is associated with slot i.

Removing the random oracle. While the above change prevents collusion between users with different user identifiers, it introduces additional obstacles for correctness. This is because in addition to computing the attribute-specific encryption key $\hat{U}_{\mathsf{a}}$, each key curator also needs to compute the (aggregated) cross terms $\hat{W}_{\mathsf{a},i,b}$ to allow users to efficiently cancel out the attribute-specific blinding terms during decryption. In the single-authority scheme, the key curator computes $\hat{W}_{i,b}$ from the cross-terms $W_{j,i} = g^{t_i u_j}$ in the CRS. In our multi-authority scheme, we have replaced the attribute-specific encryption key $\hat{U} = \prod_{i \in [L] : \mathsf{a} \notin S_i} U_i$ with $\hat{U}_{\mathsf{a}} = \prod_{i \in [L]} H(\mathsf{gid}_{\mathsf{a},i})$, so it is no longer clear which cross terms we would include in the CRS to ensure correctness. Indeed, the choice of cross terms seemingly depends on the set of potential user identifiers.

However, we observe that in the (slotted) registration-based setting, our requirements on the hash function H are in fact much weaker than that in the non-registered setting. This is because in regular multi-authority ABE, authorities have the ability to issue secret-key components for any number of user identifiers, and the ability to find any nontrivial algebraic relationship between the hashes of any pair of distinct identifiers could compromise security.

In the slotted registered setting, the slot-specific components already prevent keys registered to different slots from being combined in a way that enables decryption. Suppose we hone in on a single slot. In this example where the policy is a conjunction of two attributes, the decrypter can essentially use at most two different gid's to try and decrypt (one for each authority/attribute appearing in the challenge ciphertext). Thus, it suffices to rule out algebraic relations between only two gid's. As such, we can instantiate the hash function H with a (keyed) 2-universal hash function

$$H(\vec{U}_i, \mathsf{gid}) = U_{i,0} \cdot U_{i,1}^{\mathsf{gid}}$$

where $\vec{U}_i = (U_{i,0}, U_{i,1})$ and we sample $\vec{U}_i \xleftarrow{R} \mathbb{G}^2$ for each slot i. This way, we can simultaneously give out $\vec{K}_{i,j} = (U_{i,0}^{t_j}, U_{i,1}^{t_j})$ and compute the cross terms as

$$\hat{W}_i = \prod_{j \neq i} H(\vec{K}_{j,i}, \mathsf{gid}_i) = \prod_{j \neq i} H(\vec{U}_j, \mathsf{gid}_i)^{t_i}.$$

More generally, for policies involving P attributes, we would use a P-universal hash function. Putting everything together, our slotted multi-authority registered ABE scheme (for policies involving up to 2 attributes) is defined as follows:

- **Common reference string:** The common reference string includes a description of the group description along with the following components:
  - **General components:** The CRS contains random group elements $h \xleftarrow{R} \mathbb{G}$ and $Z = e(g, g)^{\alpha}$ where $\alpha \xleftarrow{R} \mathbb{Z}_N$.
  - **Slot components:** For each slot $i \in [L]$, the CRS contains elements

    $$A_i = g^{t_i} \quad , \quad B_i = g^{\alpha} h^{t_i} \quad , \quad \vec{U}_i = (g^{u_{i,0}}, g^{u_{i,1}})$$

    where $t_i, u_{i,0}, u_{i,1} \xleftarrow{R} \mathbb{Z}_N$. The slot components A_i, B_i serve the same purpose as in the single-authority scheme, and $\vec{U}_i$ is the hash key used to protect against collusion attacks involving multiple gid's.
  - **Cross terms:** Finally, for each pair of slots $i \neq j$, the CRS includes $\vec{K}_{j,i} = (g^{t_i u_{j,0}}, g^{t_i u_{j,1}})$, which are used for decryption.
- **User keys:** User keys are computed exactly as in the single authority scheme, where the user samples $r_i \xleftarrow{R} \mathbb{Z}_N$ and computes $T_i = g^{r_i}$ and $V_{j,i} = A_i^{r_j}$ for $j \neq i$. The public and secret keys are then

  $$\mathsf{pk} = \left(T_i\,,\, \{V_{j,i}\}_{j \neq i}\right) \quad , \quad \mathsf{sk} = r_i.$$

- **Aggregation:** To aggregate a collection of public keys $\mathsf{pk}_{\mathsf{a},i} = \left(T_i, \{V_{j,i}\}_{j \neq i}\right)$ along with their associated user identifiers $\mathsf{gid}_{\mathsf{a},i}$, the key curator (associated with the attribute a) first computes $\hat{T}_{\mathsf{a}} = \prod_{i \in [L]} T_i$. To encode the gid's associated with each slot, the key curator computes

  $$\hat{U}_{\mathsf{a}} = \prod_{i \in [L]} U_{i,0} \cdot U_{i,1}^{\mathsf{gid}_{\mathsf{a},i}}$$

  Finally, the key curator also computes the cross terms

  $$\hat{V}_{\mathsf{a},i} = \prod_{j \neq i} V_{j,i} \quad , \quad \hat{W}_{\mathsf{a},i} = \prod_{j \neq i} K_{j,i,0} K_{j,i,1}^{\mathsf{gid}_{\mathsf{a},i}}$$

  The final master public key and helper decryption keys consists of group elements

  $$\mathsf{mpk}_{\mathsf{a}} = \hat{T}_{\mathsf{a}}, \hat{U}_{\mathsf{a}} \quad , \quad \mathsf{hsk}_{\mathsf{a},i} = \hat{V}_{\mathsf{a},i}, \hat{W}_{\mathsf{a},i}$$

- **Encryption:** To encrypt a message μ under the access policy $\mathsf{a}_0 \wedge \mathsf{a}_1$, the encryptor first parses the two attribute master public keys

  $$\mathsf{mpk}_{\mathsf{a}_b} = \hat{T}_{\mathsf{a}_b}, \hat{U}_{\mathsf{a}_b}$$

  and samples randomness $s, s_0 \xleftarrow{R} \mathbb{Z}_N$, along with two group elements h_0, h_1 such that $h_0 h_1 = h$. It the constructs the following components:

- **Message-embedding components:** $C_1 = \mu \cdot Z^s$ and $C_2 = g^s$.
- **Slot-specific blinding:** $E_{\text{slot},0} = \hat{T}_{\mathsf{a}_0}^{-s}$ and $E_{\text{slot},1} = \hat{T}_{\mathsf{a}_1}^{-s}$.
- **User-identifier-specific blinding:** $E_{\text{gid},0} = \hat{U}_{\mathsf{a}_0}^{s_0}$ and $E_{\text{gid},1} = \hat{U}_{\mathsf{a}_1}^{-s_0}$.

The final ciphertext is then $(C_1, C_2, C_{3,0}, C_{3,1}, C_4)$ where

$$C_{3,0} = h_0^s \cdot E_{\text{slot}} \cdot E_{\text{gid},0} \quad , \quad C_{3,1} = h_1^s \cdot E_{\text{slot}} \cdot E_{\text{gid},1} \quad , \quad C_4 = g^{s_0}.$$

- **Decryption:** To decrypt, a user in slot i starts by computing

$$\frac{C_1}{e(C_2, B_i)} = \frac{\mu \cdot e(g,g)^{\alpha s}}{e(g,g)^{\alpha s} e(g,h)^{s t_i}} = \frac{\mu}{e(g,h)^{s t_i}}.$$

As in the single authority scheme, the user can compute $e(A_i, C_{3,0}C_{3,1})$ to obtain $e(g,h)^{st_i}$ with extraneous terms of the form $e(A_i, E_{\text{slot}})$ and $e(A_i, E_{\text{gid},b})$. Similar to the single-authority scheme, these terms can be removed via the following procedures:

  - **Slot check:** This process proceeds almost exactly as the single-authority version, where we use the fact that
$$e(A_i, E_{\text{slot,b}}) \cdot e(C_2, \hat{V}_{\mathsf{a}_b,i}) \cdot e(C_2, A_i^{r_i}) = 1.$$
Note that computing $e(C_2, A_i^{r_i})$ requires knowledge of the secret key r_i.
  - **Policy check:** Although the definition of E_{gid} is different from E_{attrib} in the single-authority setting, we can still write

$$\begin{aligned} e(A_i, E_{\text{gid},0}) &= e\left(g^{t_i}, \prod_{j\in[L]} g^{s_0(u_{j,0}+u_{j,1}\mathsf{gid}_{\mathsf{a}_0,j})}\right) \\ &= e(g,g)^{s_0 t_i \sum_{j\in[L]}(u_{j,0}+u_{j,1}\mathsf{gid}_{\mathsf{a}_0,j})} \\ e(A_i, E_{\text{gid},1}) &= e\left(g^{t_i}, \prod_{j\in[L]} g^{-s_0(u_{j,0}+u_{j,1}\mathsf{gid}_{\mathsf{a}_1,j})}\right) \\ &= e(g,g)^{-s_0 t_i \sum_{j\in[L]}(u_{j,0}+u_{j,1}\mathsf{gid}_{\mathsf{a}_1,j})} \end{aligned}$$

In addition, for $b \in \{0,1\}$,
$$e(C_4, \hat{W}_{\mathsf{a}_b,i}) = e(g,g)^{s_0 t_i \sum_{j\neq i}(u_{j,0}+u_{j,1}\mathsf{gid}_{\mathsf{a}_b,j})}.$$
This means

$$\begin{aligned} e(A_i, E_{\text{gid},0}) \cdot e(C_4, \hat{W}_{\mathsf{a}_0,i})^{-1} &= e(g,g)^{s_0 t_i (u_{i,0}+u_{i,1}\mathsf{gid}_{\mathsf{a}_0,i})} \\ e(A_i, E_{\text{gid},1}) \cdot e(C_4, \hat{W}_{\mathsf{a}_1,i}) &= e(g,g)^{-s_0 t_i (u_{i,0}+u_{i,1}\mathsf{gid}_{\mathsf{a}_1,i})} \end{aligned}$$

When $\mathsf{gid}_{\mathsf{a}_0,i} = \mathsf{gid}_{\mathsf{a}_1,i}$, then

$$e(A_i, E_{\text{gid},0}) \cdot e(C_4, \hat{W}_{\mathsf{a}_0,i})^{-1} \cdot e(A_i, E_{\text{gid},1}) \cdot e(C_4, \hat{W}_{\mathsf{a}_1,i}) = 1.$$

Taken together, the decrypter first computes

$$\begin{aligned} D_{\mathsf{a}_0} &= e(C_{3,0}, A_i) \cdot e(C_2, \hat{V}_{\mathsf{a}_0,i} A_i^{r_i}) \cdot e(C_4, \hat{W}_{\mathsf{a}_0,i})^{-1}, \\ D_{\mathsf{a}_1} &= e(C_{3,1}, A_i) \cdot e(C_2, \hat{V}_{\mathsf{a}_1,i} A_i^{r_i}) \cdot e(C_4, \hat{W}_{\mathsf{a}_1,i}). \end{aligned}$$

Then it outputs
$$\frac{C_1}{e(C_2, B_i)} \cdot D_{\mathsf{a}_0} \cdot D_{\mathsf{a}_1} = \mu.$$

Slotted to standard multi-authority registered ABE. To lift a slotted multi-authority registered ABE scheme to a normal multi-authority ABE scheme (which supports dynamic user registration), we apply a similar "powers-of-two transformation" as in [GHMR18, HLWW23]. However, the lack of coordination between authorities introduces several additional complications. Recall the notion of slotted (multi-authority) registered ABE imposes a few important limitations compared to the standard primitive:

- **Handling dynamic registrations.** The scheme supports one-shot aggregation rather than incremental updates to the public key. In the single-authority setting, this is handled by instantiating $k = \log L$ schemes where the i^{th} scheme supports 2^i users. By updating the master public keys associated with each scheme only when a multiple of 2^i keys are registered, we ensure that the number of key updates for any individual user is logarithmic. At encryption time, the encrypter encrypts to all k such schemes under the same access policy. In the multi-authority setting, users may register their attributes with different authorities in an arbitrary order and moreover, a single user's keys might end up in different sub-schemes across different authorities (e.g., as part of the i^{th} master public key for one authority, and the j^{th} master public key for a different authority). Thus, rather than having k independent master public keys, we additionally need our schemes to support a policy over all of the k sub-schemes maintained by each authority. The challenge now is that our slotted approach assumes the number of slots is the same across authorities, but the powers-of-two approach assumes authorities maintain schemes with differing number of slots.

  We solve this problem by padding (and virtualization). Namely, each authority in our scheme functions as k "authorities" for the underlying slotted scheme (that supports L users). At any point in time, the i^{th} scheme contains a maximum of 2^i users, and the remaining slots will be filled by "dummy keys" (included as part of the public parameters). This mimics the powers-of-two approach (which ensures users only need to retrieve polylogarithmically-many decryption updates), but has the additional property that there is only a single slotted scheme at all times; the authority is just managing $(1 + \log L)$ virtual authorities. When encrypting to an access policy φ, we replace it with an augmented policy φ' where each instance of the authority aid is replaced with the tuple $\bigvee_{i \in [k]} (\mathsf{aid}, i)$, where (aid, i) is the identifier for the i^{th} virtual authority maintained by authority aid.

- **Synchronizing slots across authorities.** The key-generation algorithm for the slotted scheme is tied to a specific slot $i \in [L]$. In the single-authority setting, the user first uses the CRS components to compute the message blinded with the slot-specific randomness (specifically $e(g, h)^{st_i}$), before using their secret key to remove this term. In the multi-authority setting, without any additional coordination, users may be assigned different "slots" for each authority and consequently, generate incompatible key components that are insufficient for decryption. At the same time, since each slot for any given authority can only map to a single user, we cannot hope to map arbitrary user identifiers into a polynomial number of slots without collisions. Our solution is to use a cover-free family of sets (i.e., families of sets where no set is contained within the union of N other sets within the family) [KS64, EFF85]. Specifically, each gid is associated with a set F_{gid} in the cover-free set system. When generating a key, the user would generate $|F_{\mathsf{gid}}|$ copies, one for each slot $i \in F_{\mathsf{gid}}$ and register all $|F_{\mathsf{gid}}|$ copies with the key curator (if another user has already registered their key in slot i, then the key curator skips the key). The cover-free property ensures that for any gid^* appearing in a given ciphertext, there always exists some slot i^* where gid^* is the *only* gid mapped to that slot. This ensures decryption correctness and only incurs polynomial overhead (in the total number of slots).

Security analysis. We prove security of our construction using the dual system methodology [Wat09, LW10], where we step through a hybrid sequence and gradually replace the challenge ciphertext and the slot parameters (in the CRS) with semi-functional versions. The invariant is that keys associated with semi-functional slots can decrypt normal ciphertexts and keys associated with normal slots can decrypt semi-functional ciphertexts. However, keys registered to semi-functional slots cannot decrypt semi-functional ciphertexts. Once the ciphertext and all of the slots are in the semi-functional mode, we have removed the adversary's ability to decrypt the challenge ciphertext, and can directly argue semantic security. While this high-level blueprint has been successfully used to realize many forms of ABE [Wat09, LW10, LOS+10, LW11, HLWW23, ZZGQ23, GLWW24], we highlight some of the technical challenges unique to our setting:

- **Enabling post-challenge corruptions:** In registered ABE, the adversary can corrupt registered users at any time and obtain their secret key. This can happen before as well as after seeing the challenge ciphertext. In prior works on single-authority registered ABE, there was no need to consider post-challenge corruptions. This is because once the adversary requests the challenge ciphertext, the adversary has fully determined which set of registered users can and cannot decrypt. The work of [HLWW23] thus shows how to generically convert any adversary that makes post-challenge corruption queries into one that corrupts *all* unauthorized users before requesting the challenge ciphertext. Thus, without loss of generality, in normal registered ABE, there is no need to separately handle post-challenge corruption queries. In multi-authority registered ABE, this approach no longer applies and security against adversaries only making pre-challenge corruption queries does not appear to generically imply security against adversaries that can make post-challenge corruption queries (see Remark 4.7).

  Fortunately, the bulk of the dual system argument is compatible with these post-challenge corruption queries. The main exception to this is handling a "proof of well-formedness" attached to the public keys in our scheme to ensure an adversary does not submit public keys outside the support of KeyGen. In [HLWW23], these proofs are built using an explicit algebraic validity-check mechanism, while in [ZZGQ23], they attach a quasi-adaptive non-interactive zero-knowledge (NIZK) proof to each public key. In both cases, these proofs of well-formedness are tied to the particular construction and need to be carefully integrated into the security analysis. While it seems plausible that these approaches generalize to the multi-authority registered ABE setting and still allow us to prove security against post-challenge corruption queries, we opt for a simpler and more modular approach. Namely, we first consider security against "semi-malicious" adversaries that are constrained to only registering keys that are in the support of the honest key-generation algorithm (i.e., in the security game, the adversary provides the key-generation randomness used to sample any key it chooses). We then show that our dual-system proof strategy is sufficient to prove semi-malicious security even against adversaries that can make post-challenge corruption queries. Finally, we show that any semi-maliciously-secure scheme can be generically lifted to full security using standard (simulation-sound) non-interactive zero-knowledge (NIZK) proofs of knowledge [Sah99, DDO+01].

- **Delayed** gid **programming.** When executing a dual-system proof strategy for multi-authority ABE (c.f., [LW11]), one typically switches the decryption keys from normal mode to semi-functional mode through a sequence of hybrids. Each hybrid focuses on switching the keys associated with a single gid^*. For instance, in the [LW11] analysis, to switch the keys associated with gid^* from normal to semi-functional, their strategy is to program $H(\mathsf{gid}^*)$ to output an element in $\mathbb{G}_1 \times \mathbb{G}_2$ (i.e., *introduce* a $\mathbb{G}_2$ component into $H(\mathsf{gid}^*)$). This introduces an additional blinding factor into the secret keys associated with gid^*, which allows the reduction to replace a normal key for gid^* with a semi-functional one. Before moving on to the next $\mathsf{gid}^{**} \neq \mathsf{gid}^*$, they need to first restore $H(\mathsf{gid}^*)$ to outputting an element of $\mathbb{G}_1$. Otherwise, they cannot switch keys associated with gid^{**} to semi-functional ones. In our construction, we replace the random oracle H with a keyed hash function. This means the public parameters is already committing to the value of $H(\mathsf{gid})$ *before* knowing exactly which user identifiers gid the adversary will query in the security proof. Thus, executing a similar proof strategy [LW11] would require the adversary to pre-commit to the gid's for which it will require keys. This leads to a statically-secure scheme.

  To prove adaptive security in this work, we instead rely on a *delayed programming* strategy. Like [LW11], we inject additional randomness into the hash key $\vec{U}_i$ (for slot i). We then use this randomness *in conjunction* with the encryption randomness to program the user-identifier-specific blinding factors $E_{\mathsf{gid},b}$ that appear in the challenge ciphertext. The key observation is that the challenge ciphertext is constructed *after* the adversary has selected the user identifier gid_i associated with each slot i. We refer to Sections 5.2.4 and 5.2.5 for more details.

An unbounded scheme via obfuscation. To demonstrate the feasibility of multi-authority registered ABE that can support an unbounded polynomial number of users as well as general circuit access policies, we construct such a scheme from indistinguishability obfuscation ($i\mathcal{O}$). Our construction shares a similar high-level structure with the obfuscation-based registered ABE scheme in [HLWW23]. Namely, the master public key mpk_a for each authority (with attribute a) is a hash of the $(\mathsf{gid}, \mathsf{pk})$ pairs registered to that authority while the ciphertext is an obfuscated program with the relevant authority master public keys mpk_a, the access policy C, and the message μ embedded inside. The obfuscated program takes the following inputs:

- a set of attributes S that satisfies the policy (i.e., $C(S) = 1$);
- a user identifier gid^*;
- a public key pk_a^* for each $a \in S$ together with the associated secret key sk_a^*;
- local openings of $(\mathsf{gid}^*, \mathsf{pk}_a^*)$ to mpk_a for all $a \in S$

The obfuscated program checks that the policy is satisfied, that the secret keys are valid, and that the local openings are valid. If all of the conditions pass, then the program outputs the message μ. We give the formal details along with the security proof in Section 7. Notably, we leverage function-binding hash functions [FWW23] to avoid a subexponential security loss in the reduction (i.e., which would be incurred by a proof strategy that iterates over every possible user identifier gid). Our proof technique here relies on function-binding hash functions for a new class of predicates which we call unique-block-selection predicates (see Definition 7.3). We show how to construct function-binding hash functions for this family of predicates using a leveled homomorphic encryption scheme in Appendix C; our approach follows the same structure as earlier constructions of somewhere statistically binding hash functions [HW15] or function-binding hash functions for disjunctions [FWW23]. In addition, to support *post-challenge* corruption queries, we replace the public-key encryption scheme in the above sketch with a non-committing encryption scheme.

3 Preliminaries

We begin by introducing the notation we use in this work. Many of our conventions are taken directly from [HLWW23]. We write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$, $[0, n]$ to denote the set $\{0, \ldots, n\}$, and $\mathbb{Z}_n$ to denote the integers modulo n. For a set S, we write 2^S to denote the power set of S. For a finite set S, we write $x \xleftarrow{\mathrm{R}} S$ to denote sampling an element uniformly at random from S. We use bold uppercase letters (e.g., $\mathbf{A}, \mathbf{B}$) to denote matrices and bold lowercase letters (e.g., $\mathbf{u}, \mathbf{v}$) to denote vectors. We use non-boldface letters to refer to their components (e.g., $\mathbf{v} = [v_1, \ldots, v_n]^{\mathsf{T}}$). We write $\mathsf{poly}(\lambda)$ to denote a function that is bounded by a fixed polynomial in λ and $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all $c \in \mathbb{N}$. We say an event occurs with overwhelming probability if its complement occurs with negligible probability. We say an algorithm is efficient if it runs in probabilistic polynomial time in its input length.

Access structures and linear secret sharing. We recall the definition of monotone access structures and linear secret sharing which we use in this work. Our presentation is taken verbatim from that of [HLWW23].

Definition 3.1 (Access Structure [Bei96])**.** Let S be a set and let 2^S denote the power set of S (i.e., the set of all subsets of S). An access structure on S is a set $\mathbb{A} \subseteq 2^S \setminus \varnothing$ of non-empty subsets of S. We refer to the elements of $\mathbb{A}$ as the *authorized* sets and those not in $\mathbb{A}$ as the *unauthorized* sets. We say an access structure is *monotone* if for all sets $S_1, S_2 \in 2^S$, if $S_1 \in \mathbb{A}$ and $S_1 \subseteq S_2$, then $S_2 \in \mathbb{A}$.

Definition 3.2 (Linear Secret Sharing Scheme [Bei96])**.** Let Φ be a set of parties. A linear secret sharing scheme over a ring $\mathbb{Z}_N$ for Φ is a pair $(\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_N^{\ell \times n}$ is a "share-generating" matrix and $\rho \colon [\ell] \to \Phi$ is a "row-labeling" function. The pair $(\mathbf{M}, \rho)$ satisfy the following properties:

- **Share generation:** To share a value $s \in \mathbb{Z}_N$, sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and define the vector $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, $\mathbf{u} = \mathbf{M}\mathbf{v}$ is the vector of shares where $u_i \in \mathbb{Z}_N$ belongs to party $\rho(i)$ for each $i \in [\ell]$.
- **Share reconstruction:** Let $S \subseteq \Phi$ be a set of parties and let $I_S = \{i \in [\ell] : \rho(i) \in S\}$ be the row indices associated with S. Let $\mathbf{M}_S \in \mathbb{Z}_N^{|I_S| \times n}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ that are indexed by I_S. If S is an authorized set of parties, then there exists a vector $\boldsymbol{\omega}_S \in \mathbb{Z}_N^{|I_S|}$ such that $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$, where $\mathbf{e}_1^{\mathsf{T}} = [1, 0, \ldots, 0]$ denotes the first elementary basis vector. Conversely, if $S \subseteq \Phi$ is an *unauthorized* set of parties,

then $\mathbf{e}_1^{\mathsf{T}}$ is not in the row-span of $\mathbf{M}$ (i.e., there does not exist $\boldsymbol{\omega}_S \in \mathbb{Z}_N^{|S|}$ where $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$). Equivalently, when S is unauthorized, there exists a vector $\mathbf{v}^* \in \mathbb{Z}^n$ where the first component $v_1^* = 1$ such that $\mathbf{M}_S \mathbf{v}^* = \mathbf{0}$ (i.e., the vector $\mathbf{v}^*$ is orthogonal to the rows of $\mathbf{M}$ associated with the attributes in S).

3.1 Composite-Order Pairing Groups

We now recall the notion of a composite-order pairing group along with the primary cryptographic assumptions we use in this work.

Definition 3.3 (Four-Prime Composite-Order Bilinear Group [BGN05]). A (symmetric) four-prime composite-order bilinear group generator is an efficient algorithm CompGroupGen that takes as input the security parameter λ and outputs a description $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e)$ of a bilinear group where $p_1, p_2, p_3, p_4 > 2^\lambda$ are distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order $N = p_1 p_2 p_3 p_4$, g is a generator of $\mathbb{G}$, and $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map (called the "pairing"). We require that the group operation in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the pairing operation be efficiently computable.

Notation. Let $\mathbb{G}$ be a cyclic group with order $N = p_1 p_2 p_3 p_4$ and generator g. In the following, we will write $\mathbb{G}_1 = \langle g^{p_2 p_3 p_4} \rangle$ to denote the subgroup of $\mathbb{G}$ of order p_1. We define $\mathbb{G}_2, \mathbb{G}_3, \mathbb{G}_4$ analogously. By the Chinese Remainder Theorem, if g_1, g_2, g_3, g_4 are generators of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, \mathbb{G}_4$, respectively, then $g_1 g_2 g_3 g_4 \in \mathbb{G}$ is a generator of $\mathbb{G}$, and moreover, every element $h \in \mathbb{G}$ can be *uniquely* written as $g_1^{x_1} g_2^{x_2} g_3^{x_3} g_4^{x_4}$ where $x_1 \in \mathbb{Z}_{p_1}$, $x_2 \in \mathbb{Z}_{p_2}$, $x_3 \in \mathbb{Z}_{p_3}$, and $x_4 \in \mathbb{Z}_{p_4}$. More generally, for $i, j \in \{1,2,3,4\}$, we write $\mathbb{G}_{i,j} = \langle g^{N/p_i p_j} \rangle$ to be the subgroup of $\mathbb{G}$ of order $p_i p_j$. We define $\mathbb{G}_{i,j,k}$ analogously. In the following description, we will say $h \in \mathbb{G}$ has a non-trivial component in the $\mathbb{G}_i$ subgroup if $x_i \neq 0$.

Generalized subgroup assumptions. Security of our construction relies on several variants of the subgroup decision assumptions introduced by Lewko and Waters [LW10] for constructing adaptively-secure (hierarchical) identity-based encryption, and subsequently by Lewko et al. [LOS⁺10] for constructing adaptively-secure attribute-based encryption. The first four assumptions are special cases of the generalized subgroup decision assumption from Bellare et al. [BWY11]. Lewko and Waters previously showed that all of the assumptions hold in the generic bilinear group model. The specific assumption we require are a subset of the assumptions from [GLWW23, Assumption 5.2].

Assumption 3.4 (Subgroup Decision Assumptions [LW10, GLWW23]). Let CompGroupGen be a four-prime composite-order bilinear group generator. Let $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$, $N = p_1 p_2 p_3 p_4$, $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, and $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_2 \xleftarrow{R} \mathbb{G}_2$, $g_3 \xleftarrow{R} \mathbb{G}_3$, and $g_4 \xleftarrow{R} \mathbb{G}_4$. We now define several pairs of assumptions $\mathcal{D}_0, \mathcal{D}_1$ where each distribution $\mathcal{D}_b = (D, T_b)$ consists of a set of common components D together with a challenge element T_b. We say that such an assumption holds with respect to CompGroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(D, T_0) = 1] - \Pr[\mathcal{A}(D, T_1) = 1]| = \mathsf{negl}(\lambda),$$

where the probability is taken over the choice of the common components D, the challenge element T_b, and the adversary's randomness.

Assumption 3.4a: Sample $r \xleftarrow{R} \mathbb{Z}_N$, and let

$$D = (\mathcal{G}, g_1, g_3, g_4), \qquad T_0 = g_1^r, \qquad T_1 = (g_1 g_2)^r.$$

Assumption 3.4b: Sample $s_{12}, s_{23}, r \xleftarrow{R} \mathbb{Z}_N$, and let

$$D = (\mathcal{G}, g_1, g_3, g_4, (g_1 g_2)^{s_{12}}, (g_2 g_3)^{s_{23}}), \qquad T_0 = (g_1 g_3)^r, \qquad T_1 = (g_1 g_2 g_3)^r.$$

Assumption 3.4c: Sample $s_{12}, s_{24}, r \xleftarrow{R} \mathbb{Z}_N$, and let

$$D = (\mathcal{G}, g_1, g_3, g_4, (g_1 g_2)^{s_{12}}, (g_2 g_4)^{s_{24}}), \qquad T_0 = (g_1 g_4)^r, \qquad T_1 = (g_1 g_2 g_4)^r.$$

Assumption 3.4d: Sample $\alpha, s, \zeta_1, \zeta_2, r \xleftarrow{R} \mathbb{Z}_N$, and let

$$D = \left(\mathcal{G}, g_1, g_2, g_3, g_4, g_1^\alpha g_2^{\zeta_1}, g_1^s g_2^{\zeta_2}\right), \qquad T_0 = e(g_1, g_1)^{\alpha s}, \qquad T_1 = e(g, g)^r.$$

Remark 3.5 (Equivalence of Assumption 3.4b and Assumption 3.4c)**.** Observe that Assumption 3.4b and Assumption 3.4c are equivalent up to relabeling of the primes p_3, p_4. For clarity in our reductions, we describe these as separate assumptions.

4 Multi-Authority Registered ABE

In this section, we formally introduce the concept of multi-authority registered ABE. Our definition combines the trustless feature of registered ABE [HLWW23], where individual users choose their own public/private keys, together with the decentralized nature of multi-authority ABE [Cha07, LW11], where different and independent authorities manage the attributes and public keys associated with their respective domains. For ease of exposition, we focus on the setting where each authority has one attribute. This readily extends to the setting where we implement an authority with K attributes using K single-attribute authorities. In our model, we associate each authority with a unique authority identifier aid and each user with a unique user identifier gid. By associating a single attribute with each authority, we can view an access policy as a mapping from a set of authority identifiers onto a Boolean value (that indicates whether the policy is satisfied or not). We now give the formal definition:

Definition 4.1 (Multi-Authority Registered ABE)**.** Let λ be a security parameter. Let $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of authority identifiers, $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of global user identifiers, and $\Phi = \{\Phi_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of access policies. We model each access policy $(S, \varphi) \in \Phi_\lambda$ as a Boolean-valued function $\varphi\colon 2^S \to \{0,1\}$ over a set of authority identifiers $S \subseteq \mathcal{AU}_\lambda$. A multi-authority registered ABE scheme $\Pi_{\mathsf{MA\text{-}RABE}}$ with authority identifiers $\mathcal{AU}$, user identifiers $\mathcal{GID}$, and policy family Φ is a tuple of efficient algorithms $\Pi_{\mathsf{MA\text{-}RABE}}$ = (GlobalSetup, KeyGen, RegPK, UpdateKey, Encrypt, Decrypt) with the following syntax:

- $\mathsf{GlobalSetup}(1^\lambda) \to (\mathsf{gpp}, \mathsf{gep})$: On input the security parameter λ, the global setup algorithm outputs a global set of public parameters gpp and a set of succinct global encryption parameters gep. We assume the global public parameters gpp and global encryption parameters gep implicitly contain a description of the security parameter 1^λ and the message space $\mathcal{M}$ (where $|\mathcal{M}| \geq 2$).
- $\mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid}) \to (\mathsf{pk}_{\mathsf{gid}}, \mathsf{sk}_{\mathsf{gid}})$: On input the global public parameters gpp and the user identifier gid, the key-generation algorithm outputs a public/secret key-pair $\mathsf{pk}_{\mathsf{gid}}$ and $\mathsf{sk}_{\mathsf{gid}}$.
- $\mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}, \mathsf{pk}_{\mathsf{gid}}) \to (\mathsf{mpk}', \mathsf{aux}')$: On input the global public parameters gpp, a (possibly empty) state aux, a user identifier gid, and a public key $\mathsf{pk}_{\mathsf{gid}}$, the registration algorithm *deterministically* outputs an updated master public key mpk′ and updated state aux′.
- $\mathsf{UpdateKey}(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}) \to \mathsf{hsk}_{\mathsf{gid}}$: On input the global parameters gpp, a state aux, and a user identifier gid, the update algorithm *deterministically* outputs a helper decryption key $\mathsf{hsk}_{\mathsf{gid}}$.
- $\mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid}\in S_{\mathsf{enc}}}, m) \to \mathsf{ct}$: On input the global encryption parameters gep, an access policy $(S_{\mathsf{enc}}, \varphi) \in \Phi_\lambda$ consisting of a set of authority identifiers $S_{\mathsf{enc}} \subseteq \mathcal{AU}_\lambda$ and a predicate $\varphi\colon 2^{S_{\mathsf{enc}}} \to \{0,1\}$, the corresponding authority master public keys $\{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid}\in S_{\mathsf{enc}}}$, and a message $\mu \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{gpp}, S, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}})\}_{\mathsf{aid}\in S}, \mathsf{ct}) \to \mu$: On input the global public parameters gpp, a set of authorities $S \subseteq \mathcal{AU}_\lambda$, the corresponding set of secret keys $\mathsf{sk}_{\mathsf{aid},\mathsf{gid}}$, helper decryption keys $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}}$ for the authorities in S, and a ciphertext ct, the decryption algorithm either outputs a message μ, where $\mu \in \mathcal{M} \cup \{\perp\}$ is either a message or a special symbol $\perp$ to indicate a decryption failure.

Remark 4.2 (Bounded Setup and Restricted Policy Families)**.** We say a multi-authority registered ABE scheme supports an *a priori* bounded number of users if $\mathsf{GlobalSetup}$ additionally takes a bound 1^L and we only require correctness where there is a maximum of L registered users for each authority. Additionally, $\mathsf{GlobalSetup}$ can also optionally take in an explicit policy index describing the class of access policies that the scheme supports. When unspecified, this is assumed to be a function of the security parameter.

Correctness. Correctness for a multi-authority registered ABE scheme requires that any user who has registered with a set of authorities $S \subseteq \mathcal{AU}_\lambda$ that satisfies a ciphertext's access policy φ (i.e., $\varphi(S) = 1$) should be able to decrypt. This is irrespective of the behavior of malicious users. Much like in multi-authority ABE, to distinguish between different users and prevent collusion across users, we assume that each user has a unique identifier gid, and we only support decryption with groups of keys issued under the *same* gid.

Update efficiency. Like other registration-based primitives [GHMR18, HLWW23, FFM⁺23, DPY24], when new users join the system, existing users will periodically need to request (and update) helper decryption keys from different authorities. Over the lifetime of the system, the number of updates each user needs to request should be small (as a function of the total number of users). While previous approaches modeled this by requiring that the total number of updates a user needs to make be polylogarithmic in the total number of registered users, we consider a more general approach in this work. For example, our approach can capture fine-grained policies such as a user should only have to update their key if at least T users joined the system since their last update, and moreover, the total number of updates each user has to make remain polylogarithmic in the total number of registered users.

To capture this more general notion of update efficiency, we introduce an $\mathsf{IsReady}\colon \mathbb{N}^3 \to \{0,1\}$ function that determines whether a particular decryption key $(\mathsf{sk}_{\mathsf{aid},\mathsf{gid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}})$ can be used to decrypt a given ciphertext ct (encrypted with respect to the master public key $\mathsf{mpk}_{\mathsf{aid}}$) or if an update is necessary. Specifically, the $\mathsf{IsReady}$ function takes as input three parameters (e, d, h) defined as follows:

- The e parameter is associated with the master public key $\mathsf{mpk}_{\mathsf{aid}}$ and corresponds to the total number of users registered with authority aid at the time $\mathsf{mpk}_{\mathsf{aid}}$ was generated.
- The d parameter corresponds to the number of users registered with authority aid immediately after user gid registers with authority aid.
- The h parameter corresponds to the number of users registered with authority aid at the time the last update was generated.

The $\mathsf{IsValid}$ function outputs 1 if and only if a helper decryption key is "up to date" relative to a particular master public key (i.e., can decrypt correctly). We require that the $\mathsf{IsReady}$ predicate satisfy the following basic invariant:

$$\forall e, d, h \in \mathbb{N} : h \geq e \geq d \implies \mathsf{IsReady}(e, d, h) = 1. \tag{4.1}$$

This invariant captures the following two properties:

- $e \geq d$ means the ciphertext was encrypted with respect to a master public key $\mathsf{mpk}_{\mathsf{aid}}$ constructed *after* user gid registers with authority aid.
- $h \geq e$ says that the helper decryption key $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}}$ is at least as recent as $\mathsf{mpk}_{\mathsf{aid}}$ (i.e., helper decryption keys should be backwards compatible and be able to decrypt ciphertexts associated with older versions of the master public key).

We remark that these are *minimal* requirements on the $\mathsf{IsReady}$ function, and we can impose more stringent requirements. For example, in Remark 4.4, we describe an $\mathsf{IsReady}$ function that corresponds to the standard requirement that the *total* number of updates a user has to request from a given authority scale polylogarithmically with the number of users registered with the particular authority. In contrast, an $\mathsf{IsReady}$ predicate that only satisfies the minimal requirement from Eq. (4.1) could yield a scheme where users must retrieve an update from an authority whenever a new user joins the system. Other choices of $\mathsf{IsReady}$ in turn capture different update efficiency requirements.

Definition 4.3 (Correctness and Compactness). Let $\Pi_{\text{MA-RABE}} = (\mathsf{GlobalSetup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{UpdateKey}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a multi-authority registered ABE scheme with authority identifiers $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda\in\mathbb{N}}$, user identifiers $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda\in\mathbb{N}}$, and policy family $\Phi = \{\Phi_\lambda\}_{\lambda\in\mathbb{N}}$. For a security parameter λ, an adversary $\mathcal{A}$, and an $\mathsf{IsReady}$ predicate satisfying Eq. (4.1), we define the correctness game between $\mathcal{A}$ and the challenger:

- **Setup phase:** The challenger starts by sampling the global parameters $(\mathsf{gpp}, \mathsf{gep}) \leftarrow \mathsf{GlobalSetup}(1^\lambda)$ and gives $(\mathsf{gpp}, \mathsf{gep})$ to $\mathcal{A}$. The challenger also initializes a dictionary D that maps authority identifiers aid to tuples $(c, k, j, (\mathsf{mpk}_0, \dots, \mathsf{mpk}_c), \mathsf{aux})$, where the components are defined as follows:
  - c represents the total number of users registered to this authority.
  - $k = (d, \mathsf{pk}_{\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{gid}^*})$ is a tuple representing the target user identifier $\mathsf{gid}^* \in \mathcal{GID}_\lambda$. Here, $d \in \mathbb{N}$ denotes the number of users registered with authority aid immediately after gid^* is registered, and $(\mathsf{pk}_{\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{gid}^*})$ are the public/secret keys associated with gid^* and authority aid (as sampled by the challenger in the security game). If the adversary has not yet registered a target user, then $k = \perp$.
  - $j = (h, \mathsf{hsk}_{\mathsf{gid}^*})$ is a tuple containing the counter value $h \in \mathbb{N}$ when the helper decryption key $\mathsf{hsk}_{\mathsf{gid}^*}$ for the target user gid^* was last requested (for authority aid). If the adversary has not yet registered a target user or requested an update, then $j = \perp$.
  - $(\mathsf{mpk}_0, \dots, \mathsf{mpk}_c)$ is the list of master public keys associated with the authority after each registration. In other words, mpk_0 is the initial master public key associated with authority aid (before any registrations) and mpk_i is the i^{th} master public key after the first i users have registered with aid.
  - aux is the auxiliary state of authority aid.

  Initially, for every authority $\mathsf{aid} \in \mathcal{AU}_\lambda$, the challenger initializes $c = 0$, $k = \perp$, $j = \perp$, $\mathsf{mpk}_0 = \perp$, and $\mathsf{aux} = \perp$ and sets $\mathsf{D}[\mathsf{aid}] \coloneqq (c, k, j, (\mathsf{mpk}_0), \mathsf{aux})$.

- **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following queries:
  - **Non-target registration query:** In a non-target registration query, the adversary $\mathcal{A}$ specifies a user identifier $\mathsf{gid} \in \mathcal{GID}_\lambda$, a public key $\mathsf{pk}_{\mathsf{gid}}$, and an authority $\mathsf{aid} \in \mathcal{AU}_\lambda$. The challenger responds as follows:
    1. The challenger retrieves $(c, k, j, (\mathsf{mpk}_0, \dots, \mathsf{mpk}_c), \mathsf{aux}) \leftarrow \mathsf{D}[\mathsf{aid}]$.
    2. The challenger registers the key by computing $(\mathsf{mpk}_{c+1}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}, \mathsf{pk}_{\mathsf{gid}})$.
    3. The challenger updates its dictionary by setting $\mathsf{D}[\mathsf{aid}] \coloneqq (c+1, k, j, (\mathsf{mpk}_0, \dots, \mathsf{mpk}_{c+1}), \mathsf{aux}')$ and replies to $\mathcal{A}$ with $(\mathsf{mpk}_{c+1}, \mathsf{aux}')$.
  - **Target registration query:** In a target registration query, the adversary specifies a user identifier $\mathsf{gid}^* \in \mathcal{GID}_\lambda$ and an authority $\mathsf{aid} \in \mathcal{AU}_\lambda$. The challenger responds as follows:
    1. If the adversary previously made a target registration query on any user identifier $\mathsf{gid} \neq \mathsf{gid}^*$, then the challenger halts with output 0. If the adversary previously made a non-target registration query for gid^* to authority aid, then the challenger also halts with output 0.
    2. Otherwise, the challenger retrieves $(c, k, j, (\mathsf{mpk}_0, \dots, \mathsf{mpk}_c), \mathsf{aux}) \leftarrow \mathsf{D}[\mathsf{aid}]$.
    3. The challenger samples a public/secret key-pair $(\mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*}) \leftarrow \mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid}^*)$ and then registers $\mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}$ by computing $(\mathsf{mpk}_{c+1}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}^*, \mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*})$.
    4. The challenger sets $k = (c+1, \mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*})$ and updates its dictionary by setting $\mathsf{D}[\mathsf{aid}] \coloneqq (c+1, k, j, (\mathsf{mpk}_0, \dots, \mathsf{mpk}_{c+1}), \mathsf{aux}')$ and replies to $\mathcal{A}$ with $(\mathsf{mpk}_{c+1}, \mathsf{aux}', \mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*})$.
  - **Request update query:** In a request-update query, the adversary specifies an authority identifier aid. The challenger looks up $(c, k, j, (\mathsf{mpk}_0, \dots, \mathsf{mpk}_c), \mathsf{aux}) \leftarrow \mathsf{D}[\mathsf{aid}]$. If $k = \perp$, then the challenger halts with output 0. Otherwise, the challenger parses $k = (d, \mathsf{pk}_{\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{gid}^*})$ and computes $\mathsf{hsk}' \leftarrow \mathsf{UpdateKey}(\mathsf{gpp}, \mathsf{aux}, \mathsf{pk}_{\mathsf{gid}^*})$. It sets $j' = (c, \mathsf{hsk}')$ and updates its dictionary by setting

$$\mathsf{D}[\mathsf{aid}] \coloneqq (c, k, j', (\mathsf{mpk}_0, \dots, \mathsf{mpk}_c), \mathsf{aux}).$$

- **Challenge phase:** During the challenge phase, the adversary $\mathcal{A}$ specifies an access policy $(S_{\text{enc}}, \varphi)$, a list of indices $\{e_{\text{aid}}\}_{\text{aid} \in S_{\text{enc}}}$ indicating which public key to use for each authority, and a message $\mu \in \mathcal{M}$. The challenger responds with a challenge ciphertext constructed as follows:
  1. For each aid $\in S_{\text{enc}}$, the challenger looks up $(c_{\text{aid}}, k_{\text{aid}}, j_{\text{aid}}, (\mathsf{mpk}_{\text{aid},0}, \ldots, \mathsf{mpk}_{\text{aid},c_{\text{aid}}}), \mathsf{aux}_{\text{aid}}) = \mathsf{D}[\text{aid}]$. If $k_{\text{aid}} = \bot$, then the challenger halts with output 0. Otherwise, it parses $k_{\text{aid}} = (d_{\text{aid}}, \mathsf{pk}_{\text{aid},\text{gid}^*}, \mathsf{sk}_{\text{aid},\text{gid}^*})$. If $e_{\text{aid}} > c_{\text{aid}}$ for any aid $\in S_{\text{enc}}$, then the challenger halts with outputs 0.
  2. Otherwise, the challenger computes $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{gep}, (S_{\text{enc}}, \varphi), \{(\text{aid}, \mathsf{mpk}_{\text{aid},e_{\text{aid}}})\}_{\text{aid} \in S_{\text{enc}}}, m)$, which it sends to the adversary.

  The adversary then sends a set $S \subseteq S_{\text{enc}}$ of authorities. Then, for each authority aid $\in S$, the challenger looks up $(c_{\text{aid}}, k_{\text{aid}}, j_{\text{aid}}, (\mathsf{mpk}_{\text{aid},0}, \ldots, \mathsf{mpk}_{\text{aid},c_{\text{aid}}}), \mathsf{aux}_{\text{aid}}) = \mathsf{D}[\text{aid}]$. The challenger then computes the output as follows:
  - If $k_{\text{aid}} = \bot$ or $j_{\text{aid}} = \bot$ for any aid $\in S$, the challenger outputs 0.
  - For each aid $\in S$, the challenger parses $k_{\text{aid}} = (d_{\text{aid}}, \mathsf{pk}_{\text{aid},\text{gid}^*}, \mathsf{sk}_{\text{aid},\text{gid}^*})$ and $j_{\text{aid}} = (h_{\text{aid}}, \mathsf{hsk}_{\text{aid},\text{gid}^*})$. The challenger checks that $\mathsf{IsReady}(e_{\text{aid}}, d_{\text{aid}}, h_{\text{aid}}) = 1$ and outputs 0 if not.
  - The challenger computes $\mu' \leftarrow \mathsf{Decrypt}(\mathsf{gpp}, S, \{(\text{aid}, \mathsf{sk}_{\text{aid}}, \mathsf{hsk}_{\text{aid}})\}_{\text{aid} \in S}, \mathsf{ct})$. If $\varphi(S) = 1$ and $\mu' \neq \mu$, then the challenger outputs 1. Otherwise the challenger outputs 0.

Perfect correctness. We say that $\Pi_{\text{MA-RABE}}$ is *perfectly correct* if for all (possibly unbounded) adversaries $\mathcal{A}$, $\Pr[b = 1] = 0$ in the correctness game defined above.

Compactness. We say that $\Pi_{\text{MA-RABE}}$ is *compact* if there exists a universal polynomial $\mathsf{poly}(\cdot, \cdot)$ such that the length of global encryption parameters gep and the length of the master public keys mpk in the above correctness game are bounded by $\mathsf{poly}(\lambda, \log L)$, where L is an upper bound on the number of registration queries algorithm $\mathcal{A}$ makes to any single authority.

Remark 4.4 (Polylogarithmic Update Efficiency)**.** Previous notions of registration-based cryptographic primitives (e.g., [GHMR18, HLWW23, FFM+23, DPY24]) require that the the number of updates a user has to make over the lifetime of a system scales polylogarithmically with the total number of registered users. We can capture this property in our framework by imposing a suitable combinatoric requirement on the IsReady function:

- There exists a family of intervals $\{[s_i, t_i]\}_{i \in \mathbb{N}}$ and a universal polynomial $\mathsf{poly}(\cdot)$ such that for all $L \in \mathbb{N}$,
  - $[L] \subseteq \bigcup_{i \in [\ell]} [s_i, t_i]$ where $\ell = \mathsf{poly}(\log L)$.
  - If $h \in [s_i, t_i]$ for some $i \in [\ell]$, then for all $d \leq e \leq t_i$, $\mathsf{IsReady}(e, d, h) = 1$.

  In other words, any sequence (of registrations) can be covered by a polylogarithmic number of intervals. Moreover, updating at any point in an interval ensures that the helper decryption key remains up-to-date until the number of keys exceeds the endpoint of the interval. Since only polylogarithmically-many intervals are needed to cover the set $[L]$, each user only needs to update their key a polylogarithmic number of times.

This examples illustrates the flexibility of our approach for modeling the update efficiency requirement for a registration-based cryptographic scheme. There are alternative efficiency requirements that we can also impose (and capture) using our formalism.

Remark 4.5 (Succinct Decryption Keys)**.** We say a multi-authority registered ABE scheme has *succinct decryption keys* if the following properties hold:

- The size of the secret key $\mathsf{sk}_{\text{aid},\text{gid}}$ and the helper decryption keys $\mathsf{hsk}_{\text{aid},\text{gid}}$ in the system have bounded size $\mathsf{poly}(\lambda, \log L)$, where L is the number of registered users.
- The Decrypt algorithm does *not* take the global parameters gpp as input.

We note that single-authority registered ABE schemes [HLWW23, FWW23, ZZGQ23, GLWW23, CHW25] all support succinct decryption keys. The *slotted* multi-authority registered ABE scheme we construct (Sections 4.1 and 5) also satisfies this requirement. However, our transformation from the slotted scheme to the full scheme in Section 6 does not preserve this property in the multi-authority setting.

Multi-authority registered ABE security. The security definition of multi-authority registered ABE is similar to that of plain registered ABE, where the adversary is permitted to register both honest and adversarially-chosen keys with different authorities. When registering a public key, the adversary specifies a user identifier gid. We then require semantic security for a challenge ciphertexts as long as the adversary does not possess a set of secret keys under a common user identifier gid which satisfies the challenge policy. We now give the formal definition:

Definition 4.6 (Security)**.** Let $\Pi_{\mathsf{MA\text{-}RABE}} = (\mathsf{GlobalSetup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{UpdateKey}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a multi-authority registered ABE scheme with authority space $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda\in\mathbb{N}}$, user identifiers $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda\in\mathbb{N}}$, and policy space $\Phi = \{\Phi_\lambda\}_{\lambda\in\mathbb{N}}$. For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0,1\}$, we define the following game between $\mathcal{A}$ and the challenger:

- **Setup phase:** The challenger runs $\mathsf{gpp} \leftarrow \mathsf{GlobalSetup}(1^\lambda)$ and gives gpp to the adversary. The challenger also initializes a list of honestly-generated keys $\mathcal{K} = \varnothing$ and a dictionary $\mathsf{D} = \varnothing$ that maps authority identifiers $\mathsf{aid} \in \mathcal{AU}_\lambda$ to the master public key and auxiliary state associated with the authority.
- **Query phase:** The adversary can now make the following queries:
  - **Register honest key:** In an honest-key-registration query, the adversary specifies an authority identifier $\mathsf{aid} \in \mathcal{AU}_\lambda$ and a user identifier $\mathsf{gid} \in \mathcal{GID}_\lambda$. The challenger looks up $(\mathsf{mpk}_{\mathsf{aid}}, \mathsf{aux}_{\mathsf{aid}}) = \mathsf{D}[\mathsf{aid}]$ and samples a key $(\mathsf{pk}_{\mathsf{aid},\mathsf{gid}}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}}) \leftarrow \mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid})$. Then, the challenger registers $(\mathsf{mpk}'_{\mathsf{aid}}, \mathsf{aux}'_{\mathsf{aid}}) \leftarrow \mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}_{\mathsf{aid}}, \mathsf{gid}, \mathsf{pk}_{\mathsf{aid},\mathsf{gid}})$ and updates its public key $\mathsf{D}[\mathsf{aid}] \coloneqq (\mathsf{mpk}'_{\mathsf{aid}}, \mathsf{aux}'_{\mathsf{aid}})$. The challenger replies to $\mathcal{A}$ with $(\mathsf{mpk}'_{\mathsf{aid}}, \mathsf{aux}'_{\mathsf{aid}}, \mathsf{pk}_{\mathsf{aid},\mathsf{gid}})$ and adds the entry $(\mathsf{aid}, \mathsf{gid}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}})$ to $\mathcal{K}$.
  - **Register corrupted key:** In a corrupt-key-registration query, the adversary specifies an authority identifier $\mathsf{aid} \in \mathcal{AU}_\lambda$, a user identifier gid, and a public key $\mathsf{pk}_{\mathsf{gid}}$. The challenger looks up $(\mathsf{mpk}_{\mathsf{aid}}, \mathsf{aux}_{\mathsf{aid}}) = \mathsf{D}[\mathsf{aid}]$ and computes $(\mathsf{mpk}'_{\mathsf{aid}}, \mathsf{aux}'_{\mathsf{aid}}) \leftarrow \mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}_{\mathsf{aid}}, \mathsf{gid}, \mathsf{pk}_{\mathsf{aid},\mathsf{gid}})$. The challenger updates the public key $\mathsf{D}[\mathsf{aid}] \coloneqq (\mathsf{mpk}'_{\mathsf{aid}}, \mathsf{aux}'_{\mathsf{aid}})$ and replies to $\mathcal{A}$ with $(\mathsf{mpk}'_{\mathsf{aid}}, \mathsf{aux}'_{\mathsf{aid}})$.
  - **Corrupt honest key:** In a corrupt-key-query, the adversary specifies an authority identifier $\mathsf{aid} \in \mathcal{AU}_\lambda$ and a user identifier $\mathsf{gid} \in \mathcal{GID}_\lambda$. If there exists an entry of the form $(\mathsf{aid}, \mathsf{gid}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}})$ in $\mathcal{K}$ for some $\mathsf{sk}_{\mathsf{aid},\mathsf{gid}}$, then the challenger replies with $\mathsf{sk}_{\mathsf{aid},\mathsf{gid}}$. Otherwise, it replies with $\perp$.
  - **Challenge:** In a challenge query, the adversary chooses two messages $(\mu_0^*, \mu_1^*) \in \mathcal{M}$ and a policy $(S_{\mathsf{enc}}, \varphi^*)$. The challenger replies with $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi^*), \{\mathsf{mpk}_{\mathsf{aid}}\}_{\mathsf{aid}\in S_{\mathsf{enc}}}, \mu_b^*)$. The adversary can make at most one challenge query.

After the adversary finishes making queries, it outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

For a user identifier $\mathsf{gid} \in \mathcal{GID}_\lambda$, let $S_{\mathsf{gid}} \subseteq S_{\mathsf{enc}}$ to be the set of authority identifiers $\mathsf{aid} \in S_{\mathsf{enc}}$ where the adversary either made a corrupt-key-query or a corrupt-key-registration query on $(\mathsf{aid}, \mathsf{gid})$. We say that an adversary $\mathcal{A}$ is admissible if $\varphi^*(S_{\mathsf{gid}}) = 0$ for all $\mathsf{gid} \in \mathcal{GID}_\lambda$. We say that a multi-authority registered ABE scheme is secure if for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the above security game.

Remark 4.7 (Post-Challenge Corruption Queries)**.** Definition 4.6 allows the adversary to make additional corruption queries *after* it makes the challenge query. One may recall that in the setting of (single-authority) registered ABE [HLWW23], it suffices to consider security *without* post-challenge corruption queries. As shown in [HLWW23, Lemma 4.10], security without post-corruption corruption queries generically implies security with post-corruption queries. This is because the adversary can always issue corruption queries on all of the keys registered to attribute sets S that do not satisfy its challenge policy right before it chooses the challenge ciphertext (since the policy associated with the challenge ciphertext completely determines whether a particular attribute set is corruptible or not). However, in the multi-authority setting, the adversary has the additional flexibility of corrupting *individual* attributes (registered to different authorities). As a result, we cannot rely on this argument to conclude that security without post-challenge corruption queries implies security with post-challenge corruption queries.

As a simple example, suppose an adversary registers honest keys for user gid to two attributes a_1 and a_2 and chooses $(a_1 \wedge a_2)$ as its challenge policy. In the multi-authority setting, the adversary is allowed to corrupt either attribute a_1 *or* attribute a_2. As long as it does not corrupt both of them, then the set of keys it possesses remains insufficient to decrypt the challenge ciphertext. Importantly, the adaptive adversary could choose to corrupt a_1 or a_2 *after* it sees the challenge ciphertext. The analogous notion in the single-authority setting means that the adversary can either choose to corrupt the user (and learn the key for a_1 *and* a_2) or not corrupt the user (and learn neither of the keys). The ability to selective corrupt individual attribute keys is a key difference between the single-authority and the multi-authority settings, and this distinction is the reason we explicitly consider and analyze post-challenge corruption queries throughout this work.

4.1 Slotted Multi-Authority Registered ABE

To construct multi-authority registered ABE, we follow the [HLWW23] template of first defining a *slotted* version where each authority supports a bounded number of users L. In this model, users generate their public keys for a particular slot $i \in [L]$. Then, given L public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ (one for each slot), the authority can aggregate them into a master public key mpk. Note that aggregation takes all L keys at once. Unlike Definition 4.1, there is no notion of users dynamically joining the system in a slotted multi-authority registered ABE scheme. We show in Section 6 how to generically transform a slotted multi-authority registered ABE scheme into a standard multi-authority registered ABE scheme.

Definition 4.8 (Slotted Multi-Authority Registered ABE)**.** Let λ be a security parameter. Let $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of authority identifiers, $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of global user identifiers, and $\Phi = \{\Phi_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of access policies. We model each access policy $(S, \varphi) \in \Phi_\lambda$ as a Boolean-valued function $\varphi \colon 2^S \to \{0,1\}$ over a set of authority identifiers $S \subseteq \mathcal{AU}_\lambda$. A slotted multi-authority registered ABE scheme $\Pi_{\mathsf{sMA\text{-}RABE}}$ with authority identifiers $\mathcal{AU}$, user identifiers $\mathcal{GID}$, and policy family Φ is a tuple of efficient algorithms $\Pi_{\mathsf{sMA\text{-}RABE}} = (\mathsf{GlobalSetup}$, KeyGen, $\mathsf{IsValid}$, $\mathsf{Aggregate}$, $\mathsf{Encrypt}$, $\mathsf{Decrypt})$ with the following syntax:

- $\mathsf{GlobalSetup}(1^\lambda, 1^L) \to (\mathsf{gpp}, \mathsf{gep})$: On input the security parameter λ and the number of slots L, the global setup algorithm outputs a global set of public parameters gpp and a global encryption parameters gep. We assume gep is implicitly included in gpp and moreover, that gep contain an implicit description of the security parameter 1^λ and the message space $\mathcal{M}$ (where $|\mathcal{M}| \geq 2$).

- $\mathsf{KeyGen}(\mathsf{gpp}, j) \to (\mathsf{pk}_j, \mathsf{sk}_j)$: On input the public parameters gpp and a target slot $j \in [L]$, the key-generation algorithm outputs a public/secret key-pair pk_j and sk_j.

- $\mathsf{IsValid}(\mathsf{gpp}, j, \mathsf{pk}_j) \to b$: On input the public parameters gpp, a slot index j, and a public key pk_j, the validity-checking algorithm outputs a bit $b \in \{0,1\}$.

- $\mathsf{Aggregate}(\mathsf{gpp}, \{(j, \mathsf{gid}_j, \mathsf{pk}_j)\}_{j \in [L]}) \to (\mathsf{mpk}, \{(j, \mathsf{hsk}_j)\}_{j \in [L]})$: On input the public parameters gpp together with L global user identifiers gid_j and public keys pk_j, the aggregation algorithm *deterministically* outputs a master public key mpk and a set of helper decryption keys $\{(j, \mathsf{hsk}_j)\}_{j \in [L]}$.

- $\mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu) \to \mathsf{ct}$: Given the global encryption parameters gep, an access policy $(S_{\mathsf{enc}}, \varphi) \in \Phi_\lambda$ consisting of a set of authority identifiers $S_{\mathsf{enc}} \subseteq \mathcal{AU}_\lambda$ and a predicate $\varphi \colon 2^{S_{\mathsf{enc}}} \to \{0,1\}$, a collection of authority master public keys $\{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}$, and a message μ, the encryption algorithm outputs a ciphertext ct.

- $\mathsf{Decrypt}(\mathsf{gpp}, S, j, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S}, \mathsf{ct}) \to \mu$: Given the global public parameters gpp, a set of authorities $S \subseteq \mathcal{AU}_\lambda$, a decryption slot j, a set of secret keys and helper decryption keys $\{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S}$, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M}$.

Moreover, the above algorithms should satisfy the following properties:

- **Completeness:** For all $\lambda, L \in \mathbb{N}$ and all indices $i \in [L]$,

$$\Pr\left[\mathsf{IsValid}(\mathsf{gpp}, j, \mathsf{pk}_j) = 1 : \begin{array}{c} \mathsf{gpp} \leftarrow \mathsf{GlobalSetup}(1^\lambda, 1^L) \\ (\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{KeyGen}(\mathsf{gpp}, j) \end{array}\right] = 1.$$

- **Correctness:** Take any $\lambda, L \in \mathbb{N}$, any $(\mathsf{gpp}, \mathsf{gep})$ in the support of $\mathsf{GlobalSetup}(1^\lambda, 1^L)$, any policy $(S_{\mathsf{enc}}, \varphi) \in \Phi_\lambda$, any decryption index $j^* \in [L]$, any target user identifier $\mathsf{gid}^* \in \mathcal{GID}_\lambda$, any decryption set $S^* \subseteq S_{\mathsf{enc}}$ where $\varphi(S^*) = 1$, any collection of key-pairs $(\mathsf{pk}_{j^*,\mathsf{aid}}, \mathsf{sk}_{j^*,\mathsf{aid}})$ in the support of $\mathsf{KeyGen}(1^\lambda, j^*)$ associated with authorities $\mathsf{aid} \in S^*$, any collection of user identifiers $\{(j, \mathsf{aid}, \mathsf{gid}_{j,\mathsf{aid}})\}_{(j,\mathsf{aid}) \in [L] \setminus \{j^*\} \times S_{\mathsf{enc}}}$, any collection of public keys $\{(j, \mathsf{aid}, \mathsf{pk}_{j,\mathsf{aid}})\}_{(j,\mathsf{aid}) \in [L] \setminus \{j^*\} \times S_{\mathsf{enc}}}$ where $\mathsf{IsValid}(\mathsf{gpp}, j, \mathsf{pk}_{j,\mathsf{aid}}) = 1$, and any message $\mu \in \mathcal{M}$. Then, define the following quantities:
  - For all $\mathsf{aid} \in S_{\mathsf{enc}}$, let $(\mathsf{mpk}_{\mathsf{aid}}, \mathsf{hsk}_{1,\mathsf{aid}}, \ldots, \mathsf{hsk}_{L,\mathsf{aid}}) = \mathsf{Aggregate}(\mathsf{gpp}, \{(j, \mathsf{gid}_{j,\mathsf{aid}}, \mathsf{pk}_{j,\mathsf{aid}})\}_{j \in [L]})$ where $\mathsf{gid}_{j^*,\mathsf{aid}} = \mathsf{gid}^*$.
  - Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu)$.

  Then $\Pi_{\mathsf{sMA\text{-}RABE}}$ is correct if

  $$\Pr[\mathsf{Decrypt}(\mathsf{gpp}, S^*, j^*, \{(\mathsf{aid}, \mathsf{sk}_{j^*,\mathsf{aid}}, \mathsf{hsk}_{j^*,\mathsf{aid}})\}_{\mathsf{aid} \in S^*}, \mathsf{ct}) = \mu] = 1.$$

- **Compactness:** There exists a universal polynomial $\mathsf{poly}(\cdot, \cdot)$ such that the length of the global encryption parameter gep output by $\mathsf{Setup}(1^\lambda, 1^L)$ and the length of the master public keys $|\mathsf{mpk}_{\mathsf{aid}}|$ output by $\mathsf{Aggregate}(\mathsf{gpp}, \cdot)$ satisfy $|\mathsf{gep}|, |\mathsf{mpk}_{\mathsf{aid}}| = \mathsf{poly}(\lambda, \log L)$.

- **Security:** Let $b \in \{0, 1\}$ be a bit. For an adversary $\mathcal{A}$, define the following security game between $\mathcal{A}$ and a challenger:
  - **Setup phase:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ begins by outputting the number of slots 1^L. The challenger then samples $(\mathsf{gpp}, \mathsf{gep}) \leftarrow \mathsf{GlobalSetup}(1^\lambda, 1^L)$ and gives gpp and gep to $\mathcal{A}$. The challenger also initializes a counter $\mathsf{ctr} = 0$, an (empty) dictionary $\mathsf{D} = \varnothing$, and an (empty) set $C = \varnothing$.
  - **Pre-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
    * **Key-generation query:** In a key-generation query, the adversary specifies a slot index $i \in [L]$. The challenger responds by incrementing the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$, sampling $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{gpp}, i)$ and replies with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$ to $\mathcal{A}$. The challenger adds $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary D.
    * **Key-corruption query:** In a key-corruption query, the adversary specifies an index $1 \le c \le \mathsf{ctr}$. In response, the challenger looks up the tuple $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[c]$ and replies to $\mathcal{A}$ with sk'.
  - **Challenge phase:** In the challenge phase, the adversary first specifies a policy $(S_{\mathsf{enc}}, \varphi) \in \Phi_\lambda$. For each authority $\mathsf{aid} \in S_{\mathsf{enc}}$ and each slot $i \in [L]$, the adversary specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}^*_{i,\mathsf{aid}})$ where either $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$ to reference a challenger-generated key or $c_{i,\mathsf{aid}} = \bot$ to reference a key outside this set. We require that every honestly-generated key $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$ appear at most once. The adversary also specifies two messages $\mu^*_0, \mu^*_1 \in \mathcal{M}$. The challenger responds by constructing $\mathsf{pk}_{i,\mathsf{aid}}$ as follows:
    * If $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$, then the challenger looks up $\mathsf{D}[c_{i,\mathsf{aid}}] = (i', \mathsf{pk}', \mathsf{sk}')$ and sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$. The challenger adds (i, aid) to C if the adversary issued a key-corruption query on index $c_{i,\mathsf{aid}}$.
    * If $c_{i,\mathsf{aid}} = \bot$, then the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}^*_{i,\mathsf{aid}}$ and adds (i, aid) to C.
    * Finally, the challenger checks $\mathsf{IsValid}(\mathsf{gpp}, i, \mathsf{pk}_{i,\mathsf{aid}}) = 1$. If not, the challenger halts with output 0>

    The challenger computes $(\mathsf{mpk}_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid},1}, \ldots, \mathsf{hsk}_{\mathsf{aid},L}) = \mathsf{Aggregate}(\mathsf{gpp}, \{(\mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}_{i,\mathsf{aid}})\}_{i \in [L]})$ for all $\mathsf{aid} \in S_{\mathsf{enc}}$ and replies with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi^*), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu^*_b)$. Since $\mathsf{Aggregate}$ is *deterministic* and can be run by $\mathcal{A}$ itself, there is no need to provide $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L)$ to $\mathcal{A}$. Similarly, there is no advantage to allowing the adversary to select the challenge policy and messages *after* seeing the aggregated key.
  - **Post-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
    * **Key-corruption query:** These are handled as in the pre-challenge query phase. If the index $c = c_{i,\mathsf{aid}}$ corresponds to a counter value that the adversary provided to the challenger in the challenge phase, then the challenger adds (i, aid) to the corruption set C.

   – **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  We say an adversary $\mathcal{A}$ is admissible if there does not exist a global id gid^* and slot $i^* \in [L]$ such that the set of attributes $\{\mathsf{aid} : (i^*, \mathsf{aid}) \in C \wedge \mathsf{gid}_{i,\mathsf{aid}} = \mathsf{gid}^*\}$ satisfies policy φ^*. Finally, we say that $\Pi_{\mathsf{sMA\text{-}RABE}}$ is secure if for all polynomials $L = L(\lambda)$ and all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$ in the above security experiment.

Semi-malicious correctness and security. The correctness and security requirements for a slotted multi-authority registered ABE scheme considers adversarially-chosen public keys. We also consider a weaker correctness and security requirement where the corresponding properties only hold for public keys that are in the support of the *honest* key-generation algorithm. These correspond to "semi-malicious" variants of the original properties. In Appendix A, we show that any semi-maliciously-secure scheme can be generically transformed into one that satisfies Definition 4.8 by attaching a NIZK proof of knowledge of the encryption randomness to each public key.

Definition 4.9 (Semi-Malicious Correctness and Security)**.** We say a slotted multi-authority registered ABE scheme $\Pi_{\mathsf{sMA\text{-}RABE}}$ is semi-maliciously secure if in the challenge phase of the security game from Definition 4.8, the adversary is additionally required to provide the randomness for any public key that it samples itself. In other words, in the challenge phase:

- **Challenge phase:** Specifically, if the adversary specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}^*_{i,\mathsf{aid}})$ during the challenge phase where $c_{i,\mathsf{aid}} = \perp$, the adversary must also provide randomness $r_{i,\mathsf{aid}}$. where $\mathsf{pk}^*_{i,\mathsf{aid}} = \mathsf{KeyGen}(\mathsf{gpp}, i; r_{i,\mathsf{aid}})$. If the adversary provides an incorrect or malformed $r_{i,\mathsf{aid}}$, then the challenger halts with output 0.

Along similar lines, we say that $\Pi_{\mathsf{sMA\text{-}RABE}}$ satisfies semi-malicious correctness if the correctness requirement is relaxed to only consider collection of public keys $\{\mathsf{pk}_i\}_{i \in [L]}$ where pk_i is in the support of $\mathsf{KeyGen}(\mathsf{gpp}, i)$. Since neither semi-malicious correctness nor semi-malicious security references an explicit $\mathsf{IsValid}$ function, we can omit the $\mathsf{IsValid}$ function and completeness requirements when considering a semi-maliciously secure scheme.

5 Constructing Slotted Multi-Authority Registered ABE from Pairings

In this section, we show how to construct a slotted multi-authority registered ABE scheme that supports access structures that possess a linear secret sharing scheme (Definition 3.2). Our construction relies on a composite-order pairing group. For simplicity, we impose a one-use restriction on the policy where we assume that each attribute is used at most once in the policy. For a fixed value k, we can transform any one-use scheme into a k-use scheme via the [LOS⁺10] approach of introducing k duplicate copies of each attribute. We then show that our scheme is *semi-maliciously* secure (Definition 4.9), which we can boost to full security using NIZK proofs (see Appendix A).

Construction 5.1 (Slotted Multi-Authority Attribute-Based Registration-Based Encryption.)**.** Let $\mathsf{CompGroupGen}$ be a four-prime composite-order bilinear group generator. Let $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda \in \mathbb{N}}$ be a universe of authority identifiers and $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda \in \mathbb{N}}$ be the set of user identifiers where $\mathcal{GID}_\lambda = \{0,1\}^\lambda$. Let $\Phi = \{\Phi_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of access policies where Φ_λ consists of access policies $(S_{\mathsf{enc}}, (\mathbf{M}, \rho))$ that can be described by a single-use[2] linear secret sharing scheme $(\mathbf{M}, \rho)$ over a maximum of $P = P(\lambda)$ attributes. We index the rows of $\mathbf{M}$ with the elements of S_{enc} (under a canonical ordering). Our construction will rely on the following P-wise independent hash function mapping a user identifier $\mathsf{gid} \in [2^\lambda]$ to a group element:

- $H(\vec{U}, \mathsf{gid})$: On input a vector of group elements $\vec{U} = (U_1, \ldots, U_P)$ and a user identifier $\mathsf{gid} \in [2^\lambda]$, output

$$H(\vec{U}, \mathsf{gid}) \coloneqq \prod_{i \in [P]} U_i^{\mathsf{gid}^{i-1}}.$$

  In other words, $H(\vec{U}, \mathsf{gid})$ can be viewed as evaluating a polynomial whose coefficients are determined by $\vec{U}$ at the point gid.

[2]Specifically, this means the row-labeling function ρ is an *injective* mapping.

We construct a slotted multi-authority registered ABE scheme $\Pi_{\mathsf{sMA\text{-}RABE}} = (\mathsf{GlobalSetup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with authority identifiers $\mathcal{AU}$, user identifiers $\mathcal{GID}$, and policy family Φ as follows:[3]

- $\mathsf{GlobalSetup}(1^\lambda, 1^L)$: On input the security parameter λ and the number of slots L, the setup algorithm starts by sampling $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. The setup algorithm now constructs the following quantities:
  - Let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g_1, g_3, g_4, e)$ to be the (public) group description, where $N = p_1p_2p_3p_4$.
  - Sample random exponents $\alpha, \beta \xleftarrow{R} \mathbb{Z}_N$ and set group elements $h = g_1^\beta$ and $Z = e(g_1, g_1)^\alpha$.
  - For each slot index $i \in [L]$, construct the following elements.
    * Sample exponents $t_i, \tau_i \xleftarrow{R} \mathbb{Z}_N$.
    * For each $a \in [P]$, sample $u_{i,a} \xleftarrow{R} \mathbb{Z}_N$ and set $U_{i,a} = (g_1g_4)^{u_{i,a}}$. Let $\vec{K}_i = (U_{i,1}, \dots, U_{i,P})$
    * For each $i, j \in [L]$ where $i \neq j$ and $a \in [P]$, sample $\gamma_{j,i,a} \xleftarrow{R} \mathbb{Z}_N$.
    * Define the group elements $A_i = (g_1g_3)^{t_i}$ and $B_i = g_1^\alpha h^{t_i}(g_3g_4)^{\tau_i}$.
    * For $j \neq i \in [L]$ and $a \in [P]$, let $J_{j,i,a} = U_{j,a}^{t_i}(g_3g_4)^{\gamma_{j,i,a}}$ and let $\vec{J}_{j,i} = (J_{j,i,1}, \dots, J_{j,i,P})$.
  - Output the global public parameters
$$\mathsf{gpp} = \Big(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, \vec{U}_i\}_{i\in[L]}, \{\vec{J}_{j,i}\}_{j\neq i\in[L]}\Big) \tag{5.1}$$
    and global encryption parameters $\mathsf{gep} = (\mathcal{G}, Z, g, h)$. The message space for the scheme is $\mathbb{G}_T$.

- $\mathsf{KeyGen}(\mathsf{gpp}, i)$: On input the global parameters gpp (with components given by Eq. (5.1)) and a slot index $i \in [L]$, the key-generation algorithm samples $r_i \xleftarrow{R} \mathbb{Z}_N$ and computes $T_i = g_1^{r_i}$. Then for each $j \neq i$, it computes the cross terms $V_{j,i} \leftarrow A_j^{r_i}$. Finally, it outputs the public key pk_i and the secret key sk_i defined as follows:
$$\mathsf{pk}_{i,\mathsf{aid}} = (T_i, \{V_{j,i}\}_{j\neq i\in[L]}) \quad \text{and} \quad \mathsf{sk}_i = r_i.$$
  Note that key generation does not tie keys to specific attributes or user identifiers.

- $\mathsf{Aggregate}(\mathsf{gpp}, (\mathsf{gid}_1, \mathsf{pk}_1), \dots, (\mathsf{gid}_L, \mathsf{pk}_L))$: On input the global parameters gpp (with components given by Eq. (5.1)), a collection of L global identifiers gid_i and associated public keys $\mathsf{pk}_i = (T_i, \{V_{j,i}\}_{j\neq i\in[L]})$, the aggregation algorithm computes the following components:
$$\hat{T} = \prod_{j\in[L]} T_j \quad , \quad \hat{V}_i = \prod_{j\neq i} V_{i,j} \quad , \quad Q = \prod_{j\in[L]} H(\vec{U}_j, \mathsf{gid}_j) \quad , \quad R_i = \prod_{j\neq i} H(\vec{J}_{j,i}, \mathsf{gid}_j).$$
  Finally, it outputs the master public key mpk and the slot-specific helper decryption keys hsk_i where
$$\mathsf{mpk} = (\hat{T}, Q) \quad \text{and} \quad \mathsf{hsk}_i = (\mathsf{mpk}, i, \hat{V}_i, R_i).$$

- $\mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, (\mathbf{M}, \rho)), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid}\in S_{\mathsf{enc}}}, \mu)$: On input the global master public key $\mathsf{gep} = (\mathcal{G}, Z, g, h)$, an access policy $(S_{\mathsf{enc}}, (\mathbf{M}, \rho))$, authority master public keys $\mathsf{mpk}_{\mathsf{aid}} = (\hat{T}_{\mathsf{aid}}, Q_{\mathsf{aid}})$, and a message $\mu \in \mathbb{G}_T$, the encryption algorithm starts by sampling a secret exponent $s \xleftarrow{R} \mathbb{Z}_N$. Then it constructs the ciphertext components as follows:
  - **Message-embedding components:** First, let $C_1 = \mu \cdot Z^s$ and $C_2 = g_1^s$.
  - **Policy-specific components:** Sample $\mathbf{v} \xleftarrow{R} \mathbb{Z}_N^n$ with $v_1 = s$ and $\mathbf{w} \xleftarrow{R} \mathbb{Z}_N^n$ with $w_1 = 0$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, sample $\eta_{\mathsf{aid}} \xleftarrow{R} \mathbb{Z}_p$ and let
$$C_{3,\mathsf{aid}} \leftarrow h^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{v}} \hat{T}_{\mathsf{aid}}^{-s} Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}} g_4^{\eta_{\mathsf{aid}}} \quad \text{and} \quad C_{4,\mathsf{aid}} \leftarrow g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}},$$
    where $\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \in \mathbb{Z}_N^n$ denotes the aid^th row of $\mathbf{M}$. It then outputs the ciphertext
$$\mathsf{ct} = \big((S_{\mathsf{enc}}, (\mathbf{M}, \rho)), C_1, C_2, \{(\mathsf{aid}, C_{3,\mathsf{aid}}, C_{4,\mathsf{aid}})\}_{\mathsf{aid}\in S_{\mathsf{enc}}}\big).$$

[3]As noted in Definition 4.9, we do not require an IsValid function if we are only considering semi-malicious correctness and security.

- $\mathsf{Decrypt}(\mathsf{gpp}, S, i, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{i,\mathsf{aid}})\}_{\mathsf{aid} \in S}, \mathsf{ct})$: On input the global public parameters gpp (with components given by Eq. (5.1)), a set of authorities $S \subseteq \mathcal{AU}_\lambda$, a slot $i \in [L]$, a set of secret keys $\mathsf{sk}_{\mathsf{aid}} = r_{i,\mathsf{aid}}$, helper keys $\mathsf{hsk}_{i,\mathsf{aid}} = \left(\mathsf{mpk}_{\mathsf{aid}}, i, \hat{V}_{i,\mathsf{aid}}, R_{i,\mathsf{aid}}\right)$, where $\mathsf{mpk}_{\mathsf{aid}} = \left(\hat{T}_{\mathsf{aid}}, Q_{\mathsf{aid}}\right)$, and the ciphertext

$$\mathsf{ct} = \left((S_{\mathsf{enc}}, (\mathbf{M}, \rho)), C_1, C_2, \{(\mathsf{aid}, C_{3,\mathsf{aid}}, C_{4,\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}\right)$$

  where $\mathbf{M} \in \mathbb{Z}_N^{P' \times n}$, the decryption algorithm proceeds as follows:

  - If S does not satisfy the policy $(S_{\mathsf{enc}}, (\mathbf{M}, \rho))$, then the decryption algorithm outputs $\perp$.
  - Otherwise, parse $S = \{\mathsf{aid}_1, \dots, \mathsf{aid}_{|S|}\}$. Let $\mathbf{M}_S$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by the image of elements of S under ρ. Since S is authorized, let $\boldsymbol{\omega}_S \in \mathbb{Z}_N^{|S|}$ be a vector such that $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$.
  - For each row $j \in [|S|]$ of $\mathbf{M}_S$, compute the associated row component

$$D_j = e(C_{3,\mathsf{aid}_j}, A_i) \cdot e\left(C_2, \hat{V}_{i,\mathsf{aid}_j} \cdot A_i^{r_{i,\mathsf{aid}_j}}\right) \cdot e(C_{4,\mathsf{aid}_j}, R_{i,\mathsf{aid}_j}).$$

  Finally, output

$$\frac{C_1}{e(C_2, B_i)} \cdot \prod_{j \in [|S|]} D_j^{\omega_{S,j}}. \tag{5.2}$$

5.1 Correctness and Compactness

Theorem 5.2 (Correctness). *Construction 5.1 satisfies semi-malicious correctness (Definition 4.9).*

Proof. Take any security parameter $\lambda \in \mathbb{N}$, slot parameter $L \in \mathbb{N}$, and index $i \in [L]$. Consider the following components in the correctness experiment:

- Let $(\mathsf{gpp}, \mathsf{gep}) \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^P)$ where

$$\mathsf{gpp} = \left(\mathcal{G}, Z, g, h, \{(A_i, B_i, \vec{U}_i\}_{i \in [L]}, \{\vec{J}_{j,i}\}_{j \neq i \in [L]}\right) \quad \text{and} \quad \mathsf{gep} = (\mathcal{G}, Z, g, h).$$

  By construction, the slot components can be written as $A_i = (g_1 g_3)^{t_i}$ and $B_i = g_1^\alpha h^{t_i} (g_3 g_4)^{\tau_i}$. The hash key can be written as $\vec{K}_i = (U_{i,1}, \dots, U_{i,P})$ where $U_{i,a} = (g_1 g_4)^{u_{i,a}}$ and $\vec{J}_{j,i} = (J_{j,i,1}, \dots, J_{j,i,P})$ where $J_{j,i,a} = U_{j,a}^{t_i} (g_3 g_4)^{\gamma_{j,i,a}}$.

- Take any $(\mathsf{pk}_i, \mathsf{sk}_i)$ in the support of $\mathsf{KeyGen}(\mathsf{gpp}, i)$. Then, we can write $\mathsf{sk}_i = r_i$ and $\mathsf{pk}_i = \left(T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i}\right)$ where

$$T_i = g_1^{r_i} \quad , \quad V_{j,i} = A_j^{r_i} = (g_1 g_3)^{t_j r_i}. \tag{5.3}$$

- Take any $j^* \in [L]$ and any set of public keys $\{\mathsf{pk}_{j,\mathsf{aid}}\}_{j \neq j^* \in [L]}$ in the support of $\mathsf{KeyGen}(\mathsf{gpp}, j)$. We can write $\mathsf{pk}_{j,\mathsf{aid}} = \left(T_{j,\mathsf{aid}}, \{V_{j,i,\mathsf{aid}}\}_{i \in [L] \setminus \{j\}}\right)$ where $T_{j,\mathsf{aid}} = g_1^{r_{i,\mathsf{aid}}}$ and $V_{j,i,\mathsf{aid}} = A_j^{r_{i,\mathsf{aid}}}$.

- For each aid, let $(\mathsf{mpk}_{\mathsf{aid}}, \mathsf{hsk}_{1,\mathsf{aid}}, \dots, \mathsf{hsk}_{L,\mathsf{aid}}) = \mathsf{Aggregate}(\mathsf{gpp}, (\mathsf{gid}_{1,\mathsf{aid}}, \mathsf{pk}_{1,\mathsf{aid}}), \dots, (\mathsf{gid}_{L,\mathsf{aid}}, \mathsf{pk}_{L,\mathsf{aid}}))$. The master public key $\mathsf{mpk}_{\mathsf{aid}}$ and the ith slot-specific helper decryption key $\mathsf{hsk}_{i,\mathsf{aid}}$ can be written as follows:

$$\mathsf{mpk}_{\mathsf{aid}} = \left(\hat{T}_{\mathsf{aid}}, Q_{\mathsf{aid}}\right) \quad \text{and} \quad \mathsf{hsk}_{i,\mathsf{aid}} = \left(\mathsf{mpk}_{\mathsf{aid}}, i, \hat{V}_{i,\mathsf{aid}}, R_{i,\mathsf{aid}}\right),$$

  where

$$\hat{T}_{\mathsf{aid}} = \prod_{j \in [L]} T_{j,\mathsf{aid}} \quad , \quad \hat{V}_{i,\mathsf{aid}} = \prod_{j \neq i} V_{i,j,\mathsf{aid}} \quad , \quad Q_{\mathsf{aid}} = \prod_{j \in [L]} H(\vec{K}_j, \mathsf{gid}_j) \quad , \quad R_{i,\mathsf{aid}} = \prod_{j \neq i} H(\vec{J}_{j,i}, \mathsf{gid}_j).$$

- Take any message $\mu \in \mathbb{G}_T$, policy $(S_{\mathrm{enc}}, (\mathbf{M}, \rho)) \in \Phi_\lambda$, and set of public keys $\{\mathsf{mpk}_{\mathsf{aid}}\}_{\mathsf{aid} \in S_{\mathrm{enc}}}$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{gep}, (S_{\mathrm{enc}}, (\mathbf{M}, \rho)), \{\mathsf{mpk}_{\mathsf{aid}}\}_{\mathsf{aid} \in S_{\mathrm{enc}}}, \mu)$. Then,
$$\mathsf{ct} = \big((S_{\mathrm{enc}}, (\mathbf{M}, \rho)), C_1, C_2, \{(\mathsf{aid}, C_{3,\mathsf{aid}}, C_{4,\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathrm{enc}}}\big),$$
where
$$C_1 = \mu \cdot Z^s, \quad C_2 = g_1^s, \quad C_{3,\mathsf{aid}} = h^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}} \hat{T}_{\mathsf{aid}}^{-s} Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} g_4^{\eta_{\mathsf{aid}}}, \quad C_{4,\mathsf{aid}} = g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} u}.$$

We now show that $\mathsf{Decrypt}(\mathsf{gpp}, \{\mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{i,\mathsf{aid}}\}_{\mathsf{aid} \in S}, S, i, \mathsf{ct})$ outputs μ. Denote the elements as $S = \{\mathsf{aid}_1, \ldots, \mathsf{aid}_{|S|}\}$. Let $\mathbf{M}_S$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by S. By assumption, we know that S satisfies the policy, so let $\boldsymbol{\omega}_S \in \mathbb{Z}_N^{|S|}$ be a vector such that $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$. We break up the decryption relation (Eq. (5.2)) into several pieces and analyze them individually:

- For each row $j \in [|S|]$, we have
$$D_j = \big(e(C_{3,\mathsf{aid}_j}, A_i) \cdot e(C_2, \hat{V}_{i,\mathsf{aid}_j} \cdot A_i^{r_{i,\mathsf{aid}_j}}) \cdot e(C_{4,\mathsf{aid}_j}, R_{i,\mathsf{aid}_j})\big).$$
We consider each of the individual components:
  - First consider $e(C_{3,\mathsf{aid}_j}, A_i)$. Since $h_2 \in \mathbb{G}_1$ and $\hat{T}, Q_{\mathsf{aid}} \in \mathbb{G}_{1,4}$, this means $C_{3,\mathsf{aid}_j} \in \mathbb{G}_{1,4}$. Since $A_i \in \mathbb{G}_{1,3}$, the pairing $e(C_{3,\mathsf{aid}_j}, A_i)$ only contains components in the order-p_1 subgroup of the target group. Thus, we can write
$$\begin{aligned} e(C_{3,\mathsf{aid}_j}, A_i) &= e\left(h^{\mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{v}} \hat{T}_{\mathsf{aid}_j}^{-s} Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{w}} g_4^{\eta_{\mathsf{aid}_j}},\ (g_1 g_3)^{t_i}\right) \\ &= e\left(h^{\mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{v}} \prod_{\ell \in [L]} T_{\ell,\mathsf{aid}_j}^{-s} \prod_{\ell \in [L]} H(\vec{K}_\ell, \mathsf{gid}_j)^{\mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{w}},\ g_1^{t_i}\right) \\ &= e(g_1, h)^{t_i \mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{v}} \cdot e(g_1, g_1)^{-s t_i \cdot \sum_\ell r_{j,\mathsf{aid}_j}} \cdot \prod_{\ell \in [L]} e\left(g, H(\vec{K}_\ell, \mathsf{gid}_j)\right)^{t_i \mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{w}}. \end{aligned}$$
  - Next, we compute $e(C_2, \hat{V}_{i,\mathsf{aid}_j} \cdot A_i^{r_{i,\mathsf{aid}_j}})$. Since $C_2 \in \mathbb{G}_1$, this term only contains components in the order-p_1 subgroup of the target group:
$$\begin{aligned} e(C_2, \hat{V}_{i,\mathsf{aid}_j} \cdot A_i^{r_{i,\mathsf{aid}_j}}) &= e\left(g_1^s,\ A_i^{r_{i,\mathsf{aid}_j}} \cdot \prod_{\ell \neq i} A_i^{r_{\ell,\mathsf{aid}_j}}\right) \\ &= e(g_1, A_i)^{s \sum_{\ell \in [L]} r_{\ell,\mathsf{aid}_j}} \\ &= e(g_1, g_1)^{s t_i \sum_{\ell \in [L]} r_{\ell,\mathsf{aid}_j}}. \end{aligned}$$
  - Finally, we compute $e(C_{4,\mathsf{aid}_j}, R_{i,\mathsf{aid}_j})$. Since $C_{4,\mathsf{aid}_j} \in \mathbb{G}_1$, this term only contains components in the order-p_1 subgroup of the target group:
$$\begin{aligned} e(C_{4,\mathsf{aid}_j}, R_{i,\mathsf{aid}_j}) &= e\left(g_1^{-\mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{w}},\ \prod_{\ell \neq i} H(\vec{J}_{\ell,i}, \mathsf{gid}_j)\right) \\ &= \prod_{\ell \neq i} e(g_1, H(\vec{J}_{\ell,i}, \mathsf{gid}_j))^{-\mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{w}} \\ &= \prod_{\ell \neq i} e(g_1, H(\vec{J}_{\ell,i}, \mathsf{gid}_j))^{-\mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{w}}. \end{aligned}$$

  Combining these three equations, and using the fact that $H(\vec{J}_{\ell,i}, \mathsf{gid}) = H(\vec{K}_\ell, \mathsf{gid})^{t_i}$, we can write
$$D_j = e(g_1, h)^{t_i \mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{v}} \cdot e(g_1, H(\vec{K}_i, \mathsf{gid}_i))^{t_i \mathbf{m}_{\mathsf{aid}_j}^{\mathsf{T}} \mathbf{w}}. \tag{5.4}$$

- Using Eq. (5.4), we can write

$$\prod_{j\in[|S|]} D_j^{\omega_{S,j}} = e(g_1,h)^{t_i\boldsymbol{\omega}_S^{\mathsf{T}}\mathbf{M}_S^{\mathsf{T}}\mathbf{v}} \cdot e(g_1, H(\vec{K}_i, \mathsf{gid}_i))^{t_i\boldsymbol{\omega}_S^{\mathsf{T}}\mathbf{M}_S^{\mathsf{T}}\mathbf{w}}.$$

  Since $\boldsymbol{\omega}_S^{\mathsf{T}}\mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$, $v_1 = s$, and $w_1 = 0$, this expression is equal to $e(g_1,h)^{st_i}$. This means

$$\frac{C_1}{e(C_2,B_i)} \cdot \prod_{j\in[|S|]} D_j^{\omega_{S,j}} = \frac{\mu\cdot e(g_1,g_1)^{\alpha s}}{e(g_1,g_1)^{\alpha s}e(g_1,h)^{st_i}} \cdot e(g_1,h)^{st_i} = \mu. \qquad \square$$

Theorem 5.3 (Compactness)**.** *Construction 5.1 is compact.*

Proof. This follow by inspection. Both the global encryption parameters gpp output by GlobalSetup and the authority master public keys output by Aggregate consist of a *constant* number of group elements. □

Theorem 5.4. *Suppose Assumption 3.4 holds with respect to* CompGroupGen. *Then Construction 5.1 satisfies semi-malicious security (Definition 4.9).*

We give the proof of Theorem 5.4 in the next section (Section 5.2).

5.2 Security Analysis of Construction 5.1 (Proof of Theorem 5.4)

Similar to many previous pairing-based registered ABE schemes [HLWW23, ZZGQ23, GLWW24, AT24], our security analysis follows the classic dual-system methodology [Wat09, LW10]. In this setting, we define a notion of semi-functional slots and semi-functional ciphertexts. Generally speaking, the semi-functional components have additional blinding factors in the $\mathbb{G}_2$ subgroups. When a key registered to a semi-functional slot is used to decrypt a normal ciphertext or vice versa, the $\mathbb{G}_2$ component is only present in the semi-functional key but not in the ciphertext (or vice versa), and decryption works as usual. However, when a key associated with a semi-functional slot is used to decrypt a semi-functional ciphertext, then there are random $\mathbb{G}_2$ components in both the key and in the ciphertext. In this case, decryption fails, and in fact, we can argue that the randomness in the $\mathbb{G}_2$ component suffices to blind the message. More precisely, the semi-functional ciphertexts in our scheme have the following form:

$$\mathsf{ct} = \Big((\mathbf{M},\rho)\,,\, C_1\,,\, C_2 g_2^{\zeta_2}\,,\, \big\{(\mathsf{aid}, C_{3,k}g_2^{\zeta_{3,k}}\,,\, C_{4,k})\big\}_{k\in[K]}\Big),$$

where $\zeta_2, \zeta_{3,k} \in \mathbb{Z}_N$. Analogously, the components associated with a semi-functional slot in our scheme will have the following structure:

$$A_i = g_1^{t_i}g_3^{\tau_i} \quad , \quad B_i = g_1^{\alpha}h^{t_i}(g_2g_3g_4)^{\tau_i'}.$$

5.2.1 Main Hybrids

We now give the formal hybrid structure for our security analysis. Each experiment is parameterized implicitly by a security parameter λ, a bit $b\in\{0,1\}$, and an adversary $\mathcal{A}$. We begin by describing the main hybrids we use in our security analysis:

- $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$: This is the semi-malicious security game where the challenger encrypts message μ_b^*. We recall the main steps here:
  - **Setup phase:** In the setup phase, the adversary $\mathcal{A}$ sends the slot count 1^L and the authority bound 1^P to the challenger. The challenger starts by initializing a counter $\mathsf{ctr} \leftarrow 0$ and an (empty) dictionary D to keep track of the key-generation queries. The challenger then samples $(\mathsf{gpp},\mathsf{gep}) \leftarrow \mathsf{GlobalSetup}(1^\lambda, 1^L)$ as follows:
    * Let $(\mathbb{G},\mathbb{G}_T,p_1,p_2,p_3,p_4,g,e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $N = p_1p_2p_3p_4$ and $\mathcal{G} = (\mathbb{G},\mathbb{G}_T,N,g,e)$ be the group description.

* Sample generators $g_1 \xleftarrow{\text{R}} \mathbb{G}_1$, $g_3 \xleftarrow{\text{R}} \mathbb{G}_3$, $g_4 \xleftarrow{\text{R}} \mathbb{G}_4$, exponents α, β, and let $h = g_1^\beta$.
* For each $i \in [L]$, $a \in [P]$, and $j \neq i \in [L]$, sample $t_i, \tau_i, u_{i,a}, \gamma_{j,i,a} \xleftarrow{\text{R}} \mathbb{Z}_N$. Then, define

$$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad \vec{U}_i = (U_{i,1}, \ldots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$

For $i \in [L]$ and $j \neq i \in [L]$, compute the hash-function cross terms

$$\vec{J}_{j,i} = (J_{j,i,1}, \ldots, J_{j,i,P}) \text{ where } J_{j,i,a} = U_{j,a}^{t_i} (g_3 g_4)^{\gamma_{j,i,a}}.$$

* Finally, compute $Z = e(g_1, g_1)^\alpha$ and set

$$\mathsf{gpp} = \Big(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, \vec{U}_i\}_{i \in [L]}, \{\vec{J}_{j,i}\}_{j \neq i \in [L]}\Big) \tag{5.5}$$

and the global encryption parameters $\mathsf{gep} = (\mathcal{G}, Z, g, h)$.

– **Query phase:** The challenger responds to the adversary's queries as follows:

* **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query on a slot i, the challenger starts by incrementing the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and samples $r_{\mathsf{ctr}} \xleftarrow{\text{R}} \mathbb{Z}_N$. It then computes $T_i \leftarrow g_1^{r_{\mathsf{ctr}}}$ and $V_{j,i} \leftarrow A_j^{r_{\mathsf{ctr}}}$ for $j \neq i$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, \{V_{j,i}\}_{j \neq i \in [L]})$ and responds with $(\mathsf{ctr}, i, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_{\mathsf{ctr}}$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary D.
* **Corruption query:** If the adversary makes a corruption query on an index $1 \leq \mathsf{ctr}^* \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[\mathsf{ctr}^*]$ and replies to $\mathcal{A}$ with sk'.

– **Challenge phase:** In the challenge phase, the adversary specifies a set of authorities $S_{\mathsf{enc}} \subseteq \mathcal{AU}$, an associated challenge policy $\varphi^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{P' \times n}$ and $\rho \colon S_{\mathsf{enc}} \to [P']$, and two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$. In addition, for each slot $i \in [L]$ and attribute $\mathsf{aid} \in S_{\mathsf{enc}}$, the adversary $\mathcal{A}$ specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}_{i,\mathsf{aid}}^*)$ to the challenger. The challenger sets up the public keys $\mathsf{pk}_{i,\mathsf{aid}}$ as follows:

* If $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_{i,\mathsf{aid}}] = (i', \mathsf{aid}', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$ and $\mathsf{aid} = \mathsf{aid}'$, the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$ and $r_{i,\mathsf{aid}} = \mathsf{sk}'$. Otherwise, the challenger aborts with output 0.
* If $c_{i,\mathsf{aid}} = (\perp, r_{i,\mathsf{aid}})$, then the challenger checks that $\mathsf{pk}_{i,\mathsf{aid}}^* = \mathsf{KeyGen}(\mathsf{gpp}, i; r_{i,\mathsf{aid}})$ outputs 1. If not, the challenger aborts with output 0. Otherwise, it sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}_{i,\mathsf{aid}}^*$.

For each public key $\mathsf{pk}_{i,\mathsf{aid}}$, the challenger parses it as $\mathsf{pk}_{i,\mathsf{aid}} = (T_{i,\mathsf{aid}}, \{V_{j,i,\mathsf{aid}}\}_{j \neq i \in [L]})$. Next, the challenger computes the aggregated components:

$$\hat{T}_{\mathsf{aid}} = \prod_{j \in [L]} T_{j,\mathsf{aid}} \quad , \quad Q_{\mathsf{aid}} = \prod_{j \in [L]} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}).$$

The challenger samples $s \xleftarrow{\text{R}} \mathbb{Z}_N$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, the challenger samples $\eta_{\mathsf{aid}} \xleftarrow{\text{R}} \mathbb{Z}_N$ and constructs the challenge ciphertext components as follows:

* **Message-embedding components:** First, let $C_1 = \mu_b^* \cdot Z^s$ and $C_2 = g_1^s$.
* **Policy-specific components:** Sample $v_2, \ldots, v_n, w_2, \ldots w_n \xleftarrow{\text{R}} \mathbb{Z}_N$ for the linear secret sharing scheme and let $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$ and $\mathbf{w} = [0, w_2, \ldots w_n]^{\mathsf{T}}$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, sample a blinding factor $\eta_{\mathsf{aid}} \xleftarrow{\text{R}} \mathbb{Z}_N$. Let

$$C_{3,\mathsf{aid}} = h^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}} \hat{T}_{\mathsf{aid}}^{-s} Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} g_4^{\eta_{\mathsf{aid}}} \quad \text{and} \quad C_{4,\mathsf{aid}} = g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}},$$

where $\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \in \mathbb{Z}_N^n$ denotes the $\rho(\mathsf{aid})$th row of $\mathbf{M}$.

The challenger replies with

$$\mathsf{ct}^* = \big((S_{\mathsf{enc}}, (\mathbf{M}, \rho)), C_1, C_2, \{(\mathsf{aid}, C_{3,\mathsf{aid}}, C_{4,\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}\big).$$

- **Post-challenge query phase**: The challenger responds to queries using the same procedure as in the pre-challenge query phase described above.
- **Output phase:** At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

- $\mathsf{Hyb}_0^{(b)}$: Same as $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$ except for the following (primarily syntactic) changes:
  - **Challenge phase:** The challenger constructs the ciphertext components as described in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$ except it modifies the distribution of the policy-specific components:
    - **Policy-specific components:** Sample $v_2,\ldots,v_n,w_2,\ldots w_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$ and $\mathbf{w} = [0, w_2, \ldots w_n]^{\mathsf{T}}$. Then, for each aid $\in S_{\mathsf{enc}}$, sample a blinding factor $\eta_{\mathsf{aid}} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and set
$$C_{3,\mathsf{aid}} = (g_1^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} g_4^{\eta_{\mathsf{aid}}} \quad \text{and} \quad C_{4,\mathsf{aid}} = g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}}.$$
- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$, except the challenge ciphertext is replaced by a semi-functional ciphertext. Specifically, in the challenge phase, the challenger proceeds as follows:
  - **Challenge phase:** The challenger then constructs the challenge ciphertext components as follows:
    - **Message-embedding components:** First, let $C_1 = \mu_b^* \cdot Z^s$ and $C_2 = (g_1 g_2)^s$.
    - **Policy-specific components:** Sample $v_2,\ldots,v_n,w_2,\ldots w_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$ and $\mathbf{w} = [0, w_2, \ldots w_n]^{\mathsf{T}}$. Then, for each aid $\in S_{\mathsf{enc}}$, sample a blinding factor $\eta_{\mathsf{aid}} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and set
$$C_{3,\mathsf{aid}} = ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} g_4^{\eta_{\mathsf{aid}}} \quad \text{and} \quad C_{4,\mathsf{aid}} = g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}}.$$
- $\mathsf{Hyb}_{2,\ell}^{(b)}$ for $\ell \in [0, L]$: Same as $\mathsf{Hyb}_2^{(b)}$, except the challenger change the first ℓ slots to semi-functional. Specifically, during the setup phase, for all $i \leq \ell$, the challenger sets the slot component B_i as
$$B_i = g_1^{\alpha} A_i^{\beta} (g_2 g_3 g_4)^{\tau_i}.$$
The rest of the public parameters (as well as the challenge ciphertext), are generated as in $\mathsf{Hyb}_1^{(b)}$.
- $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$: Same as $\mathsf{Hyb}_{2,L}^{(b)}$ except when constructing the challenge ciphertext, the challenger samples $C_1 \xleftarrow{\mathrm{R}} \mathbb{G}_T$. Importantly, this distribution is *independent* of the message.

We write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the random variable corresponding to the output of an execution of $\mathsf{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$ (and the implicit security parameter λ).

5.2.2 Proofs of Main Hybrids

In this section, we show that the output distributions of each adjacent pair of adjacent hybrid experiments in Section 5.2.1 are indistinguishable.

Lemma 5.5. *For all adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*
$$\Pr[\mathsf{Hyb}_{\mathsf{real}}^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1]$$

Proof. The only difference between $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$ and $\mathsf{Hyb}_0^{(b)}$ is that in $\mathsf{Hyb}_0^{(b)}$, the challenger replaces $\hat{T}_{\mathsf{aid}}^{-s}$ with $(g_1^s)^{-\sum_{i \in [L]} r_{i,\mathsf{aid}}}$ in $C_{3,\mathsf{aid}}$. By construction, in both experiments, for every $i \in [L]$ and attribute aid $\in S_{\mathsf{enc}}$, we have that $T_{i,\mathsf{aid}} = g_1^{r_{i,\mathsf{aid}}}$. Otherwise, the challenger in both experiments abort. In this case,
$$\hat{T}_{\mathsf{aid}}^{-s} = \left(\prod_{i \in [L]} g_1^{r_{i,\mathsf{aid}}} \right)^{-s} = g_1^{\sum_{i \in [L]} r_{i,\mathsf{aid}} \cdot (-s)} = (g_1^s)^{-\sum_{i \in [L]} r_{i,\mathsf{aid}}}.$$
Thus, these experiments are identical. □

Lemma 5.6. *Suppose Assumption 3.4a holds with respect to* CompGroupGen. *Then, for all efficient adversaries $\mathcal{A}$ and $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that distinguishes between $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$ with non-negligible advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 3.4a with the same advantage:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_3, g_4, T)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_3 \in \mathbb{G}_3$, $g_4 \in \mathbb{G}_4$, and either $T = g_1^s$ or $T = (g_1 g_2)^s$ for some $s \xleftarrow{\text{R}} \mathbb{Z}_N$. The components that depend on the challenge element T is colored for clarity.

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the slot bound 1^L as well as the authority bound 1^P.

3. Algorithm $\mathcal{B}$ samples $\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_N$. It sets $Z \leftarrow e(g_1, g_1)^\alpha$, and $h \leftarrow g_1^\beta$.

4. For each $i \in [L]$, $a \in [P]$, and $j \neq i \in [L]$, sample $t_i, \tau_i, u_{i,a}, \gamma_{j,i,a} \xleftarrow{\text{R}} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ constructs the slot components as follows:

$$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad \vec{U}_i = (U_{i,1}, \ldots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$

   For $i \in [L]$ and $j \neq i \in [L]$, it computes the hash-function cross terms

$$\vec{J}_{j,i} = (J_{j,i,1}, \ldots, J_{j,i,P}) \text{ where } J_{j,i,a} = U_{j,a}^{t_i} (g_3 g_4)^{\gamma_{j,i,a}}.$$

   Algorithm $\mathcal{B}$ gives the global parameters

$$\mathsf{gpp} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, \vec{U}_i\}_{i \in [L]}, \{\vec{J}_{j,i}\}_{j \neq i \in [L]}\right)$$

   and the global encryption parameters $\mathsf{gep} = (\mathcal{G}, Z, g, h)$. to the adversary $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary D to keep track of the key-generation queries.

5. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{\text{R}} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, and $V_{j,i} = A_j^{r_i}$ for $j \neq i$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, \{V_{j,i}\}_{j \neq i \in [L]})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary D. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \leq \mathsf{ctr}' \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[\mathsf{ctr}']$ and replies to $\mathcal{A}$ with sk'.

6. In the challenge phase, after $\mathcal{A}$ specifies a set of authorities $S_{\mathsf{enc}} \subseteq \mathcal{AU}$, an associated challenge policy $\varphi^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{P' \times n}$ and $\rho \colon S_{\mathsf{enc}} \to [P']$, and two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$. In addition, for each each slot $i \in [L]$ and attribute $\mathsf{aid} \in S_{\mathsf{enc}}$, it also specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}_{i,\mathsf{aid}}^*)$. Algorithm $\mathcal{B}$ sets up the public keys $\mathsf{pk}_{i,\mathsf{aid}}$ as in $\mathsf{Hyb}_0^{(b)}, \mathsf{Hyb}_1^{(b)}$:

   - If $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_{i,\mathsf{aid}}] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$ and $r_{i,\mathsf{aid}} = \mathsf{sk}'$. Otherwise, the challenger aborts with output 0.
   - If $c_{i,\mathsf{aid}} = (\perp, r_{i,\mathsf{aid}})$, then the challenger checks that $\mathsf{pk}_{i,\mathsf{aid}}^* = \mathsf{KeyGen}(\mathsf{gpp}, i; r_{i,\mathsf{aid}})$. If not, the challenger aborts with output 0. Otherwise, it sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}_{i,\mathsf{aid}}^*$.

   Finally, for each index $i \in [L]$ and $\mathsf{aid} \in S_{\mathsf{enc}}$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_{i,\mathsf{aid}} = (T_{i,\mathsf{aid}}, \{V_{j,i,\mathsf{aid}}\}_{j \neq i \in [L]})$.

7. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:

   - **Message-embedding components:** Set $C_1 = \mu_b^* \cdot e(g_1, T)^\alpha$ and $C_2 = T$.

- **Policy-specific components:** Sample $v_2, \ldots, v_n, w_2, \ldots w_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$ and $\mathbf{w} = [0, w_2, \ldots w_n]^{\mathsf{T}}$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, sample blinding factor $\eta_{\mathsf{aid}} \xleftarrow{R} \mathbb{Z}_N$ and set

$$C_{3,\mathsf{aid}} = T^{\beta \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} g_4^{\eta_{\mathsf{aid}}} \quad \text{and} \quad C_{4,k} = g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}}.$$

8. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

Observe that $e(g_1, T)^{\alpha} = e(g_1, g_1)^{\alpha s}$ regardless of whether $T = g_1^s$ or $T = (g_1 g_2)^s$. If $T = g_1^s$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_0^{(b)}$. Alternatively, when $T = (g_1 g_2)^s$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_1^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks Assumption 3.4a with the same advantage ε. □

Lemma 5.7. *For all adversaries $\mathcal{A}$, $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]\right| = \left|\Pr[\mathsf{Hyb}_{2,0}^{(b)}(\mathcal{A}) = 1]\right|.$$

Proof. Experiments $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_{2,0}^{(b)}$ are identical by definition. □

Lemma 5.8. *Suppose Assumption 3.4 holds with respect to* $\mathsf{CompGroupGen}$. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_{2,\ell-1}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

To prove Lemma 5.8, we will define a sequence of intermediate hybrids. We defer this to Section 5.2.3.

Lemma 5.9. *Suppose Assumption 3.4d holds with respect to* $\mathsf{CompGroupGen}$. *Then for all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_{2,L}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}_{2,L}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(b)}(\mathcal{A}) = 1]\right| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 3.4d with the same advantage:

1. First, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_2, g_3, g_4, X, Y, T)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, $g_3 \in \mathbb{G}_3$, $g_4 \in \mathbb{G}_4$, $X = g_1^{\alpha} g_2^{\zeta_1}$, $Y = g_1^s g_2^{\zeta_2}$ for some $\alpha, s, \zeta_1, \zeta_2 \xleftarrow{R} \mathbb{Z}_N$, and either $T = e(g_1, g_1)^{\alpha s}$ or $T = e(g, g)^r$, where $r \xleftarrow{R} \mathbb{Z}_N$. The components that depend on the challenge elements X, Y, T are colored for clarity.

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the slot bound 1^L as well as the authority bound 1^P.

3. Algorithm $\mathcal{B}$ samples $\beta \xleftarrow{R} \mathbb{Z}_N$. It sets $Z = e(g_1, X)$, and $h = g_1^{\beta}$.

4. For each $i \in [L]$, $a \in [P]$, and $j \neq i \in [L]$, algorithm $\mathcal{B}$ samples $t_i, \tau_i, u_{i,a}, \gamma_{j,i,a} \xleftarrow{R} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ constructs the (semi-functional) slot components as follows:

$$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = X A_i^{\beta} (g_2 g_3 g_4)^{\tau_i} \quad , \quad \vec{U}_i = (U_{i,1}, \ldots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$

For $j \neq i \in [L]$, it computes the hash-function cross terms

$$\vec{J}_{j,i} = (J_{j,i,1}, \ldots, J_{j,i,P}) \text{ where } J_{j,i,a} = U_{j,a}^{t_i} (g_3 g_4)^{\gamma_{j,i,a}}.$$

Algorithm $\mathcal{B}$ gives the global parameters

$$\mathsf{gpp} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, \vec{U}_i\}_{i \in [L]}, \{\vec{J}_{j,i}\}_{j \neq i \in [L]}\right)$$

and the global encryption parameters $\mathsf{gep} = (\mathcal{G}, Z, g, h)$. to the adversary $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary D to keep track of the key-generation queries.

5. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{Hyb}_{2,L}^{(b)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{R} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, and $V_{j,i} = A_j^{r_i}$ for $j \neq i$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, \{V_{j,i}\}_{j \neq i \in [L]})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary D. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \leq \mathsf{ctr}' \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[\mathsf{ctr}']$ and replies to $\mathcal{A}$ with sk'.
6. In the challenge phase, after $\mathcal{A}$ specifies a set of authorities $S_{\mathsf{enc}} \subseteq \mathcal{AU}$, an associated challenge policy $\varphi^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{P' \times n}$ and $\rho \colon S_{\mathsf{enc}} \to [P']$, and two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$. In addition, for each each slot $i \in [L]$ and attribute $\mathsf{aid} \in S_{\mathsf{enc}}$, it also specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}_{i,\mathsf{aid}}^*)$. Algorithm $\mathcal{B}$ sets up the public keys $\mathsf{pk}_{i,\mathsf{aid}}$ as in $\mathsf{Hyb}_{2,L}^{(b)}, \mathsf{Hyb}_{\mathsf{rand}}^{(b)}$:
   - If $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_{i,\mathsf{aid}}] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$ and $r_{i,\mathsf{aid}} = \mathsf{sk}'$. Otherwise, the challenger aborts with output 0.
   - If $c_{i,\mathsf{aid}} = (\perp, r_{i,\mathsf{aid}})$, then the challenger checks that $\mathsf{pk}_{i,\mathsf{aid}}^* = \mathsf{KeyGen}(\mathsf{gpp}, i; r_{i,\mathsf{aid}})$. If not, the challenger aborts with output 0. Otherwise, it sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}_{i,\mathsf{aid}}^*$.

   Finally, for each index $i \in [L]$ and $\mathsf{aid} \in S_{\mathsf{enc}}$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_{i,\mathsf{aid}} = (T_{i,\mathsf{aid}}, \{V_{j,i,\mathsf{aid}}\}_{j \neq i \in [L]})$.
7. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:
   - **Message-embedding components:** Set $C_1 = \mu_b^* \cdot T$ and $C_2 = Y$.
   - **Policy-specific components:** Sample $v_2, \ldots, v_n, w_2, \ldots w_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$ and $\mathbf{w} = [0, w_2, \ldots w_n]^{\mathsf{T}}$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, sample blinding factor $\eta_{\mathsf{aid}} \xleftarrow{R} \mathbb{Z}_N$ and set

$$C_{3,\mathsf{aid}} = Y^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} g_4^{\eta_{\mathsf{aid}}} \quad , \quad C_{4,\mathsf{aid}} = g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}}.$$

8. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

We now show that depending on the challenge T, algorithm $\mathcal{B}$ either simulates an execution of $\mathsf{Hyb}_{2,L}^{(b)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$ where $\zeta_2 \bmod p_2$ plays the role of $s \bmod p_2$:

- First, $Z = e(g_1, X) = e(g_1, g_1^{\alpha} g_2^{\zeta_1}) = e(g_1, g_1)^{\alpha}$.
- Consider the components of the global public parameters gpp:

$$B_i = XA_i^{\beta}(g_2 g_3 g_4)^{\tau_i} = g_1^{\alpha} g_2^{\zeta_1} A_i^{\beta} (g_2 g_3)^{\tau_i} = g_1^{\alpha} A_i^{\beta} g_2^{\zeta_1 + \tau_i} (g_3 g_4)^{\tau_i}$$
$$P_{i,\mathsf{aid}} = (Y g_3)^{\delta_{i,\mathsf{aid}}} = ((g_1^s g_2^{\zeta_2}) g_3)^{\delta_{i,\mathsf{aid}}},$$

  where $\tau_i, \delta_{i,\mathsf{aid}} \xleftarrow{R} \mathbb{Z}_N$. Since $\tau_i \xleftarrow{R} \mathbb{Z}_N$, the distributions of $\tau_i + \zeta_1$ and τ_i are identical. Since the other components $A_i, \vec{U}_i, \vec{J}_{j,i}$ are sampled using the same procedure as in $\mathsf{Hyb}_{2,L}^{(b)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$, we conclude that these components of gpp are correctly distributed.
- Algorithm $\mathcal{B}$ answers the queries using the same procedure as $\mathsf{Hyb}_{2,L}^{(b)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$.
- Next, the challenge ciphertext components $C_2, C_{3,\mathsf{aid}}, C_{4,\mathsf{aid}}$ are distributed exactly as in $\mathsf{Hyb}_{2,L}^{(b)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$ where $\zeta_2 \bmod p_2$ plays the role of $s \bmod p_2$. Since $s \xleftarrow{R} \mathbb{Z}_N$ in $\mathsf{Hyb}_{2,L}^{(b)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$, the distribution of $s \bmod p_2$ and $\zeta_2 \bmod p_2$ are identical.

Consider now the distribution of the challenge T:

- If $T = e(g_1, g_2)^{\alpha s}$, then $C_1 = \mu_b^* \cdot T = \mu_b^* \cdot Z^s$. In this case, algorithm $\mathcal{B}$ correctly simulates experiment $\mathsf{Hyb}_{3,L}^{(b)}$.
- If $T = e(g_1, g_2)^r$ where $r \xleftarrow{R} \mathbb{Z}_N$, the distribution of C_1 is uniform in $\mathbb{G}_T$, and algorithm $\mathcal{B}$ correctly simulates experiment $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$.

Thus, algorithm $\mathcal{B}$ breaks Assumption 3.4d with the same distinguishing advantage as $\mathcal{A}$ and the claim follows. □

Combining Lemmas 5.5 to 5.9, Theorem 5.4 holds, as required. □

5.2.3 Proof of Lemma 5.8

In this section, we give the proof of Lemma 5.8. To do so, we first define a sequence of intermediate hybrid experiments:

- $\mathsf{iHyb}_{\ell,0}^{(b)}$: Same as $\mathsf{Hyb}_{2,\ell-1}^{(b)}$ except the challenger makes some syntactic changes on how the cross terms $\vec{J}_{j,\ell}$ are generated. Specifically, for $j \neq \ell \in [L]$, the challenger sets

$$\vec{J}_{j,\ell} = (J_{j,\ell,1}, \dots, J_{j,\ell,P}) \text{ where } J_{j,\ell,a} = A_\ell^{u_{j,a}} (g_3 g_4)^{\gamma_{j,\ell,a}}.$$

- $\mathsf{iHyb}_{\ell,1}^{(b)}$: Same as $\mathsf{iHyb}_{\ell,0}^{(b)}$, except the challenger introduces a $\mathbb{G}_2$ component to A_ℓ. Specifically, the challenger sets $A_\ell = (g_1 g_2 g_3)^{t_\ell}$. Note that this modification propagates to all terms that depend on A_ℓ (e.g., B_ℓ and $J_{j,\ell}$).

- $\mathsf{iHyb}_{\ell,2}^{(b)}$: Same as $\mathsf{iHyb}_{\ell,1}^{(b)}$, except the challenger adds a gid-specific blinding value to the $\mathbb{G}_{2,4}$ subgroup of the challenge ciphertext:
  - Let GS_ℓ be the set of user identifiers $\mathsf{gid} \in \{0,1\}^\lambda$ registered to slot ℓ in the challenge ciphertext. Without loss of generality, we assume $|\mathsf{GS}_\ell| = P$.[4] Let $\mathsf{gid}_1, \mathsf{gid}_2, \dots, \mathsf{gid}_P$ be a fixed (e.g. lexicographic) ordering on the elements of GS_ℓ.
  - For $\mathsf{gid}_j \in \mathsf{GS}_\ell$, sample $\mathbf{w}_j \xleftarrow{\mathrm{R}} \mathbb{Z}_N^n$. Define $j_{\mathsf{aid}} \in [P]$ to be the index where $\mathsf{gid}_{j_{\mathsf{aid}}} = \mathsf{gid}_{\ell,\mathsf{aid}}$.
  - When preparing the challenge ciphertext, the challenger now constructs $C_{3,\mathsf{aid}}$ as

$$C_{3,\mathsf{aid}} = ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} \, (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_{j_{\mathsf{aid}}}} g_4^{\eta_{\mathsf{aid}}}.$$

- $\mathsf{iHyb}_{\ell,3}^{(b)}$: Same as $\mathsf{iHyb}_{\ell,2}^{(b)}$, but the challenger randomizes the $\mathbb{G}_2$ component of B_ℓ by computing

$$B_\ell = g^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}.$$

- $\mathsf{iHyb}_{\ell,4}^{(b)}$: Same as $\mathsf{iHyb}_{\ell,3}^{(b)}$, but the challenger removes the gid-specific blinding factor from the challenge ciphertext. Namely, the challenger now constructs $C_{3,\mathsf{aid}}$ as:

$$C_{3,\mathsf{aid}} = ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} \, g_4^{\eta_{\mathsf{aid}}}.$$

- $\mathsf{iHyb}_{\ell,5}^{(b)}$: Same as $\mathsf{iHyb}_{\ell,4}^{(b)}$, but the challenger removes the $\mathbb{G}_2$ subgroup component from A_ℓ. Namely, the challenger sets $A_\ell = (g_1 g_3)^{t_\ell}$.

- $\mathsf{iHyb}_{\ell,6}^{(b)}$: Same as $\mathsf{iHyb}_{\ell,5}^{(b)}$, but the challenger reverts to sampling $\vec{J}_{j,\ell}$ as before. Namely, the challenger computes

$$\vec{J}_{j,\ell} = (J_{j,\ell,1}, \dots, J_{j,\ell,P}) \text{ where } J_{j,\ell,a} = U_{j,a}^{t_\ell} (g_3 g_4)^{\gamma_{j,\ell,a}}.$$

We now show that each pair of adjacent hybrids are indistinguishable.

Lemma 5.10. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{Hyb}_{2,\ell-1}^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{iHyb}_{\ell,0}^{(b)}(\mathcal{A}) = 1].$$

[4]There is a maximum of P authorities associated with the challenge ciphertext, so in the security game, there can be at most P user identifiers associated with any single slot (e.g., if the slot is associated with a different gid for every authority). If GS_ℓ contains fewer than P user identifiers, the challenger pads GS_ℓ with arbitrary authorities until it contains exactly P (distinct) authorities.

Proof. The only difference between the two experiment is the challenger replaces $U_{j,a}^{t_\ell}$ in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$ with $A_\ell^{u_{j,a}}$ in $\mathsf{iHyb}_{\ell,0}^{(b)}$ when constructing $J_{j,\ell,a}$ for all $a \in [P]$. In both experiments, $U_{j,a} = (g_1 g_4)^{u_{j,a}}$ and $A_\ell = (g_1 g_3)^{t_\ell}$. Then, in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$, we can write

$$U_{j,a}^{t_\ell}(g_3 g_4)^{\gamma_{j,\ell,a}} = g_1^{u_{j,a}t_\ell} g_4^{u_{j,a}t_\ell}(g_3 g_4)^{\gamma_{j,\ell,a}} = g_1^{u_{j,a}t_\ell} g_3^{\gamma_{j,\ell,a}} g_4^{u_{j,a}t_\ell+\gamma_{j,\ell,a}}.$$

On the other hand, in $\mathsf{iHyb}_{\ell,0}^{(b)}$,

$$A_\ell^{u_{j,a}}(g_3 g_4)^{\gamma_{j,\ell,a}} = g_1^{u_{j,a}t_\ell} g_3^{u_{j,a}t_\ell}(g_3 g_4)^{\gamma_{j,\ell,a}} = g_1^{u_{j,a}t_\ell} g_3^{u_{j,a}t_\ell+\gamma_{j,\ell,a}} g_4^{\gamma_{j,\ell,a}}.$$

These terms are identical in the $\mathbb{G}_1$ subgroup and only differ in the $\mathbb{G}_3$ and $\mathbb{G}_4$ subgroups. However, the exponents $\gamma_{j,\ell,a}$ are uniformly random (over $\mathbb{Z}_N$) and only appear in $J_{j,\ell,a}$. Thus, over the randomness of $\gamma_{j,\ell,a}$, the $\mathbb{G}_3$ and $\mathbb{G}_4$ components of both these terms are uniformly random and independent of all other components in both cases. We conclude that these two distributions are identically distributed. □

Lemma 5.11. *Suppose Assumption 3.4b holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient and admissible adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. Suppose that there exists an efficient adversary $\mathcal{A}$ that can distinguish these two experiments with non-negligible advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 3.4b with the same advantage:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_3, g_4, X, Y, T)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_3 \in \mathbb{G}_3$, $g_4 \in \mathbb{G}_4$, $X = (g_1 g_2)^{s_{12}}$, $Y = (g_2 g_3)^{s_{23}}$ and $s_{12}, s_{23} \xleftarrow{R} \mathbb{Z}_N$, and either $T = (g_1 g_3)^r$ or $T = (g_1 g_2 g_3)^r$ for some $r \xleftarrow{R} \mathbb{Z}_N$. The components that depend on the challenge elements X, Y, T are colored for clarity.

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the slot bound 1^L as well as the authority bound 1^P.

3. Algorithm $\mathcal{B}$ samples $\alpha, \beta \xleftarrow{R} \mathbb{Z}_N$. It sets $Z = e(g_1, g_1)^\alpha$ and $h = g_1^\beta$.

4. For each $i \in [L]$, $a \in [P]$, and $j \neq i \in [L]$, algorithm $\mathcal{B}$ samples $t_i, \tau_i, u_{i,a}, \gamma_{j,i,a} \xleftarrow{R} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ constructs the (semi-functional) slot components as follows:

   - For $i < \ell$, algorithm $\mathcal{B}$ sets

   $$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = g_1^\alpha A_i^\beta Y^{\tau_i} g_4^{\tau_i} \quad , \quad \vec{U}_i = (U_{i,1}, \ldots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$

   - For $i = \ell$, algorithm $\mathcal{B}$ sets

   $$A_\ell = T \quad , \quad B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell} \quad , \quad \vec{U}_i = (U_{i,1}, \ldots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$

   - For $i > \ell$, algorithm $\mathcal{B}$ sets

   $$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad \vec{U}_i = (U_{i,1}, \ldots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$

   Note that the exponent r from the challenge plays the role of t_ℓ.

5. For $j \neq \ell \in [L]$, algorithm $\mathcal{B}$ computes the hash-function cross terms as follows:

   $$\vec{J}_{j,\ell} = (J_{j,\ell,1}, \ldots, J_{j,\ell,P}) \text{ where } J_{j,\ell,a} = A_\ell^{u_{j,a}}(g_3 g_4)^{\gamma_{\ell,j,a}}.$$

   For all $i \in [L] \setminus \{\ell\}$ and all $j \neq i$, it sets the hash-function cross terms as follows:

   $$\vec{J}_{j,i} = (J_{j,i,1}, \ldots, J_{j,i,P}) \text{ where } J_{j,i,a} = U_{j,a}^{t_i}(g_3 g_4)^{\gamma_{j,i,a}}.$$

Algorithm $\mathcal{B}$ gives the global parameters

$$\mathsf{gpp} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, \vec{U}_i\}_{i\in[L]}, \{\vec{J}_{j,i}\}_{j\neq i\in[L]}\right)$$

and the global encryption parameters $\mathsf{gep} = (\mathcal{G}, Z, g, h)$. to the adversary $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} \leftarrow 0$ and an (empty) dictionary D to keep track of the key-generation queries.

6. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{iHyb}^{(b)}_{\ell,0}$ and $\mathsf{iHyb}^{(b)}_{\ell,1}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{\text{R}} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, and $V_{j,i} = A_j^{r_i}$ for $j \neq i$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, \{V_{j,i}\}_{j\neq i\in[L]})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary D. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \le \mathsf{ctr}' \le \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[\mathsf{ctr}']$ and replies to $\mathcal{A}$ with sk'.

7. In the challenge phase, after $\mathcal{A}$ specifies a set of authorities $S_{\mathsf{enc}} \subseteq \mathcal{AU}$, an associated challenge policy $\varphi^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{P'\times n}$ and $\rho\colon S_{\mathsf{enc}} \to [P']$, and two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$. In addition, for each each slot $i \in [L]$ and attribute $\mathsf{aid} \in S_{\mathsf{enc}}$, it also specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}^*_{i,\mathsf{aid}})$. Algorithm $\mathcal{B}$ sets up the public keys $\mathsf{pk}_{i,\mathsf{aid}}$ as in $\mathsf{iHyb}^{(b)}_{\ell,0}$, $\mathsf{iHyb}^{(b)}_{\ell,1}$:
   - If $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_{i,\mathsf{aid}}] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$ and $r_{i,\mathsf{aid}} = \mathsf{sk}'$. Otherwise, the challenger aborts with output 0.
   - If $c_{i,\mathsf{aid}} = (\bot, r_{i,\mathsf{aid}})$, then the challenger checks that $\mathsf{pk}^*_{i,\mathsf{aid}} = \mathsf{KeyGen}(\mathsf{gpp}, i; r_{i,\mathsf{aid}})$. If not, the challenger aborts with output 0. Otherwise, it sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}^*_{i,\mathsf{aid}}$.

   Finally, for each index $i \in [L]$ and $\mathsf{aid} \in S_{\mathsf{enc}}$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_{i,\mathsf{aid}} = (T_{i,\mathsf{aid}}, \{V_{j,i,\mathsf{aid}}\}_{j\neq i\in[L]})$.

8. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:
   - **Message-embedding components:** Set $C_1 = \mu_b^* \cdot e(g_1, X)^\alpha$ and $C_2 = X$.
   - **Policy-specific components:** Sample $v_2, \ldots, v_n, w_2, \ldots w_n \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$ and $\mathbf{w} = [0, w_2, \ldots w_n]^{\mathsf{T}}$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, it samples blinding factor $\eta_{\mathsf{aid}} \xleftarrow{\text{R}} \mathbb{Z}_N$ and sets

   $$C_{3,\mathsf{aid}} = X^{\beta \mathbf{m}_k^{\mathsf{T}}\mathbf{v}' - \sum_{i\in[L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}} g_4^{\eta_{\mathsf{aid}}} \quad , \quad C_{4,k} = g_1^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}}.$$

9. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

In the above reduction, the challenge exponents $r, s_{12} \in \mathbb{Z}_N$ play the roles of $t_\ell, s \in \mathbb{Z}_N$ in the underlying hybrids. Thus, depending on the choice of T, either $A_\ell = T = (g_1g_3)^{t_\ell}$ or $A_\ell = T = (g_1g_2g_3)^{t_\ell}$. The former corresponds to the distribution of A_ℓ in $\mathsf{iHyb}^{(b)}_{\ell,0}$ while the latter corresponds to the distribution of A_ℓ in $\mathsf{iHyb}^{(b)}_{\ell,1}$. It suffices to argue that the remaining components are simulated exactly as required:

- First, the only other component of the public parameters gpp that is not generated using the same procedure as in $\mathsf{iHyb}^{(b)}_{\ell,0}$ and $\mathsf{iHyb}^{(b)}_{\ell,1}$ are the components B_i for $i < \ell$. For such B_i, observe that as long as $s_{23} \neq 0 \bmod p_2$ and $s_{23} \neq 0 \bmod p_3$ (which holds with overwhelming probability over the choice of $\tau \xleftarrow{\text{R}} \mathbb{Z}_N$), then the distributions

  $$\{(g_2g_3)^{s_{23}\tau_i} : \tau_i \xleftarrow{\text{R}} \mathbb{Z}_N\} \quad \text{and} \quad \{(g_2g_3)^{\tau_i} : \tau_i \xleftarrow{\text{R}} \mathbb{Z}_N\}$$

  are identically distributed. Since τ_i is only revealed in the $\mathbb{G}_2$ and $\mathbb{G}_3$ subgroup here, the distribution of B_i is distributed exactly as in the real scheme.

- Next, consider the challenge ciphertext. The reduction computes

  $$C_1 = \mu_b^* \cdot e(g_1, (g_1g_2)^{s_{12}})^\alpha = \mu_b^* \cdot e(g_1, g_1)^{\alpha s_{12}} = \mu_b^* \cdot Z^{s_{12}},$$

  which is exactly its distribution in $\mathsf{iHyb}^{(b)}_{\ell,0}$ and $\mathsf{iHyb}^{(b)}_{\ell,1}$. The remaining ciphertext components are generated exactly as described in $\mathsf{iHyb}^{(b)}_{\ell,0}$ and $\mathsf{iHyb}^{(b)}_{\ell,1}$.

Thus, with overwhelming probability, algorithm $\mathcal{B}$ correctly simulates an execution of $\mathsf{iHyb}_{\ell,0}^{(b)}$ or $\mathsf{iHyb}_{\ell,1}^{(b)}$. Correspondingly, algorithm $\mathcal{B}$ breaks Assumption 3.4b with advantage at least $\varepsilon - \mathsf{negl}(\lambda)$. □

Lemma 5.12. *Suppose Assumption 3.4c holds with respect to* CompGroupGen. *Then for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{iHyb}_{\ell,2}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{\ell,1}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

To prove Lemma 5.12, we will define another sequence of intermediate hybrids that iterates over each user identifier $\mathsf{gid}_j \in \mathsf{GS}_\ell$ registered to slot ℓ of the challenge ciphertext. We defer this to Section 5.2.4.

Lemma 5.13. *For all $\ell \in [L]$, all efficient and admissible adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{iHyb}_{\ell,3}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{\ell,2}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. We show that the distributions $\mathsf{iHyb}_{\ell,3}^{(b)}(\mathcal{A})$ and $\mathsf{iHyb}_{\ell,2}^{(b)}(\mathcal{A})$ are statistically indistinguishable. This argument will rely on the additional $\mathbb{G}_{2,4}$ randomization introduced in $\mathsf{iHyb}_{\ell,2}^{(b)}$. By construction, the only difference between the two experiments is the distribution of component B_ℓ in the $\mathbb{G}_2$ subgroup:

$$B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell} \qquad \text{in } \mathsf{iHyb}_{\ell,2}^{(b)}$$
$$B_\ell = g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell} \qquad \text{in } \mathsf{iHyb}_{\ell,3}^{(b)}.$$

In both experiments, $A_\ell = (g_1 g_2 g_3)^{t_\ell}$. Suppose that $t_\ell \neq 0 \bmod p_2$. Since $t_\ell \xleftarrow{\text{R}} \mathbb{Z}_N$, this holds with all but negligible probability. Consider the following relabeling of the variables in $\mathsf{iHyb}_{\ell,2}^{(b)}$:

- Let $\sigma^{(\beta)} \in \mathbb{Z}_N$ be the unique value where $\sigma^{(\beta)} = 0 \bmod p_1 p_3 p_4$ and $\sigma^{(\beta)} = (t_\ell)^{-1}\tau_\ell \bmod p_2$. Suppose we write $\beta = \beta' + \sigma^{(\beta)}$ where $\beta' \xleftarrow{\text{R}} \mathbb{Z}_N$.
- Let $\sigma^{(\mathbf{w})} \in \mathbb{Z}_N^n$ be the unique value where $\sigma^{(\mathbf{w})} = \mathbf{0} \bmod p_1 p_3 p_4$ and $\sigma^{(\mathbf{w})} = -s \cdot \sigma^{(\beta)} \cdot \mathbf{v}' \bmod p_2$. For each $j \in [P]$, let $\mathbf{w}_j = \mathbf{w}_j' + \sigma^{(\mathbf{w})}$ where $\mathbf{w}_j' \xleftarrow{\text{R}} \mathbb{Z}_N^n$.

By construction, these substitutions preserve the distribution of β and $\mathbf{w}_1, \ldots, \mathbf{w}_P$ in $\mathsf{iHyb}_{\ell,2}^{(b)}$. Consider the remaining components in the adversary's view with this variable substitution:

- Consider the components in the global public parameters. First, $h = g_1^\beta = g_1^{\beta'}$. Next $A_i = (g_1 g_3)^{t_i}$ for all $i \neq \ell$ and $A_\ell = (g_1 g_2 g_3)^{t_\ell}$. Consider the distribution of each B_i:
  - If $i < \ell$, then $B_i = g_1^\alpha A_i^\beta (g_2 g_3 g_4)^{\tau_i} = g_1^\alpha A_i^{\beta'} (g_2 g_3 g_4)^{\tau_i}$.
  - If $i = \ell$, then
  $$B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell} = g_1^\alpha g_1^{t_\ell \beta} g_2^{t_\ell \beta} g_3^{t_\ell \beta} (g_3 g_4)^{\tau_\ell} = g_1^\alpha g_1^{t_\ell \beta'} g_2^{t_\ell \beta' + \tau_\ell} g_3^{t_\ell \beta'} (g_3 g_4)^{\tau_\ell} = g_1^\alpha A_\ell^{\beta'} (g_2 g_3 g_4)^{\tau_\ell},$$
  since $\beta = \beta' \bmod p_1 p_3 p_4$ and $\beta = \beta' + (t_\ell)^{-1}\tau_\ell \bmod p_2$.
  - If $i > \ell$, then $B_i = g_1^\alpha A_i^\beta g_3^{\tau_i} = g_1^\alpha A_i^{\beta'} (g_3 g_4)^{\tau_i}$.

  The remaining components in the global public parameters gpp do not depend on β or $\mathbf{w}_1, \ldots, \mathbf{w}_P$, and are thus unchanged.
- Next, when responding to key-generation queries, the challenger does not need to refer to the exponents β or $\mathbf{w}_1, \ldots, \mathbf{w}_P$. Thus, the challenger's responses to the key-generation queries are unaffected by this substitution.

- Finally, consider the components in the challenge ciphertext. The components $C_1, C_2, C_{4,\mathsf{aid}}$ for aid $\in S_{\mathsf{enc}}$ are constructed without referring to the exponents β or $\mathbf{w}_1, \ldots, \mathbf{w}_P$. Thus, it suffices to consider the ciphertext components $C_{3,\mathsf{aid}}$. For clarity, we will write

$$C_{3,\mathsf{aid}} = (g_1 g_2)^{s\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}'} (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_{j_{\mathsf{aid}}}} \cdot C'_{3,\mathsf{aid}}.$$

where $C'_{3,\mathsf{aid}}$ represents the remaining components that can be constructed without referring to $\beta, \mathbf{w}_1, \ldots, \mathbf{w}_P$. Then, we have

$$\begin{aligned} C_{3,\mathsf{aid}} &= (g_1 g_2)^{s\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}'} (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_{j_{\mathsf{aid}}}} \cdot C'_{3,\mathsf{aid}} \\ &= g_1^{s\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}'} g_2^{s\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' + \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_{j_{\mathsf{aid}}}} g_4^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_{j_{\mathsf{aid}}}} \cdot C'_{3,\mathsf{aid}} \\ &= g_1^{s\beta' \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}'} g_2^{s(\beta' + \sigma^{(\beta)}) \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' + \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} (\mathbf{w}'_{j_{\mathsf{aid}}} - s\sigma^{(\beta)} \mathbf{v}')} g_4^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'_{j_{\mathsf{aid}}}} \cdot C'_{3,\mathsf{aid}} \\ &= g_1^{s\beta' \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}'} g_2^{s\beta' \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' + \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'_{j_{\mathsf{aid}}}} g_4^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'_{j_{\mathsf{aid}}}} \cdot C'_{3,\mathsf{aid}} \\ &= (g_1 g_2)^{s\beta' \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}'} (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'_{j_{\mathsf{aid}}}} \cdot C'_{3,\mathsf{aid}}. \end{aligned}$$

using the fact that $\sigma^{(\mathbf{w})} = -s\sigma^{(\beta)} \mathbf{v}' \bmod p_2$ and $\sigma^{(\beta)} = \sigma^{(\mathbf{w})} = 0 \bmod p_1 p_4$.

With this relabeling of variables, we have recovered the ciphertext distribution in $\mathsf{iHyb}_{\ell,3}^{(b)}$ (with randomness β', $\mathbf{w}'_1, \ldots, \mathbf{w}'_P$). Thus, the distributions $\mathsf{iHyb}_{\ell,3}^{(b)}$ and $\mathsf{iHyb}_{\ell,2}^{(b)}$ are statistically close. □

Lemma 5.14. *Suppose Assumption 3.4c holds with respect to* CompGroupGen. *Then for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{iHyb}_{\ell,4}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{\ell,3}^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 5.12. □

Lemma 5.15. *Suppose Assumption 3.4b holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{iHyb}_{\ell,5}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{\ell,4}^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 5.11. □

Lemma 5.16. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{iHyb}_{\ell,6}^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1].$$

Proof. This follows from a similar argument as the proof of Lemma 5.10. □

Lemma 5.8 now follows by appealing to Lemmas 5.10 to 5.16. □

5.2.4 Proof of Lemma 5.13 (GID Hybrids)

In this section, we give the proof of Lemma 5.13. To do so, we define a set of intermediate hybrids between $\mathsf{iHyb}_{\ell,2}^{(b)}$ and $\mathsf{iHyb}_{\ell,3}^{(b)}$ (see Section 5.2.3), where we iterate over the user identifiers $\mathsf{gid}_j \in \mathsf{GS}_\ell$ associated with slot ℓ of the challenge ciphertext. At a high level, our argument proceeds by first introducing a random $\mathbb{G}_2$ subgroup component to $H(\mathsf{gid}_j)$. Then, we use the fact that gid_j cannot be authorized on slot ℓ to show that this is statistically indistinguishable from directly randomizing the challenge ciphertext component $C_{3,\mathsf{aid}}$ for slots registered to gid_j. We can then undo the $H(\mathsf{gid}_j)$ randomization and proceed to the next user identifier. We now describe the intermediate hybrid experiments we use. Each hybrid in this sequence is indexed by $j \in [P]$.

- $\mathsf{gHyb}_{j,0}^{(b,\ell)}$ for $j \in [P]$: Same as $\mathsf{iHyb}_{\ell,1}^{(b)}$, but for all $j_{\mathsf{aid}} < j$ (recall $j_{\mathsf{aid}} \in [P]$ is the index where $\mathsf{gid}_{j_{\mathsf{aid}}} = \mathsf{gid}_{\ell,\mathsf{aid}}$), the challenger samples $\mathbf{w}_j \xleftarrow{R} \mathbb{Z}_N^n$ and sets

$$C_{3,\mathsf{aid}} = ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_{j_{\mathsf{aid}}}} g_4^{\eta_{\mathsf{aid}}}.$$

- $\mathsf{gHyb}_{j,1}^{(b,\ell)}$: Same as $\mathsf{gHyb}_{j,0}^{(b,\ell)}$, except the challenger randomizes the $\mathbb{G}_{2,4}$ component of the hash key used to compute Q_{aid}. Specifically, for each $i \in [P]$, the challenger samples $\zeta_{\ell,i} \xleftarrow{R} \mathbb{Z}_N$. Next it computes

$$\vec{U}'_\ell = (U'_{\ell,1}, \ldots, U'_{\ell,P}) \text{ where } U'_{\ell,i} = U_{\ell,i} (g_2 g_4)^{\zeta_{\ell,i}}.$$

Then, when constructing the challenge ciphertext, the challenger computes

$$Q_{\mathsf{aid}} = H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{j \in [L] \setminus \{\ell\}} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}).$$

- $\mathsf{gHyb}_{j,2}^{(b,\ell)}$: Same as $\mathsf{gHyb}_{j,1}^{(b,\ell)}$, but the challenger introduces a gid-specific blinding factor. Specifically, the challenger samples $\kappa_j \xleftarrow{R} \mathbb{Z}_N$ and for all aid where $j_{\mathsf{aid}} = j$, it sets

$$Q_{\mathsf{aid}} = H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot (g_2 g_4)^{\kappa_j} \cdot \prod_{j \in [L] \setminus \{\ell\}} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}).$$

- $\mathsf{gHyb}_{j,3}^{(b,\ell)}$: Same as $\mathsf{gHyb}_{j,2}^{(b,\ell)}$, but the challenger removes the hash key randomization from Q_{aid}. Specifically, when preparing the challenge ciphertext, the challenger now computes

$$Q_{\mathsf{aid}} = \begin{cases} \prod_{j \in [L]} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \cdot (g_2 g_4)^{\kappa_j} & j_{\mathsf{aid}} = j \\ \prod_{j \in [L]} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) & j_{\mathsf{aid}} \neq j. \end{cases}$$

- $\mathsf{gHyb}_{j,4}^{(b,\ell)}$: Same as $\mathsf{gHyb}_{j,3}^{(b,\ell)}$, but the challenger introduces $\mathbf{w}_j \xleftarrow{R} \mathbb{Z}_N^n$ in $C_{3,\mathsf{aid}}$ for all aid where $j_{\mathsf{aid}} = j$. Specifically, the challenger samples $\mathbf{w}_j \xleftarrow{R} \mathbb{Z}_N^n$. Then, for all aid where $j_{\mathsf{aid}} = j$, the challenger defines $C_{3,\mathsf{aid}}$ as follows:

$$C_{3,\mathsf{aid}} = ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_{j_{\mathsf{aid}}}} g_4^{\eta_{\mathsf{aid}}}.$$

- $\mathsf{gHyb}_{j,5}^{(b,\ell)}$: Same as $\mathsf{gHyb}_{j,4}^{(b,\ell)}$, but the challenger re-introduces the hash key randomization. Namely, the challenger now sets

$$Q_{\mathsf{aid}} = \begin{cases} H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{j \in [L] \setminus \{\ell\}} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \cdot (g_2 g_4)^{\kappa_j} & j_{\mathsf{aid}} = j \\ H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{j \in [L] \setminus \{\ell\}} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) & j_{\mathsf{aid}} \neq j \end{cases}$$

- $\mathsf{gHyb}_{j,6}^{(b,\ell)}$: Same as $\mathsf{gHyb}_{j,5}^{(b,\ell)}$, except the challenger removes the gid-specific blinding factor from Q_{aid} for all aid where $j_{\mathsf{aid}} = j$. Specifically for all aid where $j_{\mathsf{aid}} = j$, the challenger sets

$$Q_{\mathsf{aid}} = H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{j \in [L] \setminus \{\ell\}} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}})$$

We now show that each pair of adjacent hybrids are indistinguishable.

Lemma 5.17. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{iHyb}_{\ell,1}^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{gHyb}_{1,0}^{(b,\ell)}(\mathcal{A}) = 1].$$

Proof. Since there does not exist any indices $j \in [P]$ where $j < 1$, these two experiments are identical by definition. □

Lemma 5.18. *Suppose Assumption 3.4c holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all $j \in [P]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{gHyb}_{j,1}^{(b,\ell)}(\mathcal{A}) = 1] - \Pr[\mathsf{gHyb}_{j,0}^{(b,\ell)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

To prove Lemma 5.18, we will define another sequence of intermediate hybrid experiments to introduces the re-randomization factors into the $\mathbb{G}_{2,4}$-subgroup of the hash key components. This step corresponds to the "delayed gid programming" alluded to in Section 2. We give this hybrid sequence and analysis in Section 5.2.5.

Lemma 5.19. *For all $\ell \in [L]$, all $j \in [P]$, all adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{gHyb}_{j,2}^{(b,\ell)}(\mathcal{A}) = 1] = \Pr[\mathsf{gHyb}_{j,1}^{(b,\ell)}(\mathcal{A}) = 1].$$

Proof. We show that the distributions $\mathsf{gHyb}_{j,1}^{(b,\ell)}(\mathcal{A})$ and $\mathsf{gHyb}_{j,2}^{(b,\ell)}(\mathcal{A})$ are identically distributed. By construction, the only difference between the two experiments is the additional $(g_2 g_4)^{\kappa_j}$ blinding term in Q_{aid} in $\mathsf{gHyb}_{j,2}^{(b,\ell)}(\mathcal{A})$ whenever $j_{\mathsf{aid}} = j$. In the following, we will interpret each user identifier $\mathsf{gid} \in \{0,1\}^\lambda$ with an integer in the range $[2^\lambda]$. Then, let $\mathbf{V}_\ell \in \mathbb{Z}_N^{P \times P}$ be the Vandermonde matrix associated with the elements $\mathsf{GS}_\ell = \{\mathsf{gid}_1, \ldots \mathsf{gid}_P\}$:

$$\mathbf{V}_\ell = \begin{bmatrix} 1 & \mathsf{gid}_1 & \mathsf{gid}_1^2 & \cdots & \mathsf{gid}_1^{P-1} \\ 1 & \mathsf{gid}_2 & \mathsf{gid}_2^2 & \cdots & \mathsf{gid}_2^{P-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \mathsf{gid}_P & \mathsf{gid}_P^2 & \cdots & \mathsf{gid}_P^{P-1} \end{bmatrix} \in \mathbb{Z}_N^{P \times P}.$$

Since $\mathsf{gid}_1, \ldots, \mathsf{gid}_P$ are unique and moreover, $\mathsf{gid}_i \le 2^\lambda < p_k$ for all $k \in \{1,2,3,4\}$, we conclude that $\mathsf{gid}_i \ne 0 \bmod p_k$ for all k. Correspondingly, $\det(\mathbf{V}_\ell) = \prod_{i \ne j}(\mathsf{gid}_i - \mathsf{gid}_j) \ne 0 \bmod p_k$. In this case, $\mathbf{V}_\ell$ is invertible. Now, let $\mathbf{v}_{\ell,i}^{\mathsf{T}}$ denote the ith row of $\mathbf{V}_\ell$. Then, for any vector $\mathbf{u} = [u_1, \ldots, u_P]^{\mathsf{T}}$, we have

$$g^{\mathbf{v}_{\ell,i}^{\mathsf{T}}\mathbf{u}} = H(\vec{U}, \mathsf{gid}_i) \text{ where } \vec{U} = (g^{u_1}, \ldots, g^{u_P}).$$

Consider now the following relabeling of the variables in $\mathsf{gHyb}_{j,1}^{(b,\ell)}$:

- Let $\boldsymbol{\sigma}^{(\zeta_\ell)} = (\sigma^{(\zeta_{\ell,1})}, \ldots, \sigma^{(\zeta_{\ell,P})}) \in \mathbb{Z}_N^P$ be the unique vector where $\boldsymbol{\sigma}^{(\zeta_\ell)} = \mathbf{0} \bmod p_1 p_3$ and

$$\sigma^{(\zeta_\ell)} = \kappa_j \mathbf{V}_\ell^{-1} \mathbf{e}_j \bmod p_2 p_4,$$

  where $\mathbf{e}_j$ is the jth standard basis vector.

- Now, for $i \in [P]$, let $\boldsymbol{\zeta}_\ell = \boldsymbol{\zeta}'_\ell + \boldsymbol{\sigma}^{(\zeta_\ell)}$ where $\boldsymbol{\zeta}'_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_N^P$.

Observe that the distribution of $\boldsymbol{\zeta} = [\zeta_{\ell,1}, \ldots, \zeta_{\ell,P}]^{\mathsf{T}}$ remains uniform over $\mathbb{Z}_N^P$. Consider the distribution of the components in $\mathsf{gHyb}_{j,1}^{(b,\ell)}$ under this substitution. By design, the exponents $\zeta_{\ell,i}$ only appears in the definition of Q_{aid}, so this is the only component whose definition changes. Let $\mathbf{u}_\ell = [u_{\ell,1}, \ldots, u_{\ell,P}]^{\mathsf{T}}$. Then, under this substitution,

$$\begin{aligned} Q_{\mathsf{aid}} &= H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{j \ne \ell} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \\ &= (g_1 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \mathbf{u}_\ell} (g_2 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \zeta_\ell} \cdot \prod_{j \ne \ell} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \\ &= (g_1 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \mathbf{u}_\ell} (g_2 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} (\zeta'_\ell + \kappa_j \mathbf{V}_\ell^{-1} \mathbf{e}_j)} \cdot \prod_{j \ne \ell} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \end{aligned}$$

Now,

$$\kappa_j \mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \mathbf{V}_\ell^{-1} \mathbf{e}_j = \kappa_j \mathbf{e}_{j_{\mathsf{aid}}}^{\mathsf{T}} \mathbf{e}_j = \begin{cases} \kappa_j & j_{\mathsf{aid}} = j \\ 0 & \text{otherwise.} \end{cases}.$$

We conclude that for all aid where $j_{\mathsf{aid}} = j$,

$$\begin{aligned} Q_{\mathsf{aid}} &= (g_1 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \mathbf{u}_\ell} (g_2 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \boldsymbol{\zeta}'_\ell + \kappa_j} \cdot \prod_{j \neq \ell} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \\ &= H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot (g_2 g_4)^{\kappa_j} \cdot \prod_{j \neq \ell} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}), \end{aligned}$$

where $\vec{U}'_\ell = (U'_{\ell,1}, \ldots, U'_{\ell,P})$ and $U'_{\ell,i} = (g_1 g_4)^{u_{\ell,i}} (g_2 g_4)^{\zeta'_{\ell,i}} = U_{\ell,i} (g_2 g_4)^{\zeta'_{\ell,i}}$. For aid where $j_{\mathsf{aid}} \neq j$, we have

$$\begin{aligned} Q_{\mathsf{aid}} &= (g_1 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \mathbf{u}_\ell} (g_2 g_4)^{\mathbf{v}_{\ell,j_{\mathsf{aid}}}^{\mathsf{T}} \boldsymbol{\zeta}'_\ell} \cdot \prod_{j \neq \ell} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \\ &= H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{j \neq \ell} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}) \end{aligned}$$

This precisely coincides with the distribution of Q_{aid} in $\mathsf{gHyb}_{\ell,2}^{(b,\ell)}$ with randomness $\boldsymbol{\zeta}'_\ell$. Thus, the distributions $\mathsf{gHyb}_{j,1}^{(b,\ell)}$ and $\mathsf{gHyb}_{j,2}^{(b,\ell)}$ are identical. □

Lemma 5.20. *Suppose Assumption 3.4c holds with respect to* CompGroupGen. *Then for all $\ell \in [L]$, all $j \in [P]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{gHyb}_{j,3}^{(b,\ell)}(\mathcal{A}) = 1] - \Pr[\mathsf{gHyb}_{j,2}^{(b,\ell)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 5.18. Note that the additional $(g_2 g_4)^{\zeta_{\ell,i}}$ introduced in $\mathsf{gHyb}_{j,2}^{(b,\ell)}(\mathcal{A})$ can be simulated using the component $(g_2 g_4)^{s_{24}}$ given out in Assumption 3.4c. □

Lemma 5.21. *For all $\ell \in [L]$, all $j \in [P]$, all admissible adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{gHyb}_{j,4}^{(b,\ell)}(\mathcal{A}) = 1] - \Pr[\mathsf{gHyb}_{j,3}^{(b,\ell)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. We show that the distributions $\mathsf{gHyb}_{j,3}^{(b,\ell)}(\mathcal{A})$ and $\mathsf{gHyb}_{j,4}^{(b,\ell)}(\mathcal{A})$ are statistically indistinguishable. By construction, the only difference between the two experiments is the addition of a $(g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_j}$ blinding term in the ciphertext components $C_{3,\mathsf{aid}}$ where $j_{\mathsf{aid}} = j$. We show that these two distributions are identical conditioned on $s, \kappa_j \neq 0 \bmod p_2$. Since the challenger in both experiments samples $s, \kappa_j \xleftarrow{\text{R}} \mathbb{Z}_N$, we conclude that the two experiments are statistically indistinguishable. Formally, consider the following relabeling of the elements in $\mathsf{gHyb}_{j,3}^{(b,\ell)}(\mathcal{A})$ (where we assume $s, \kappa_j \neq 0 \bmod p_2$):

- Let $\sigma^{(\mathbf{w})} \in \mathbb{Z}_N^n$ be the unique value where $\sigma^{(\mathbf{w})} = \mathbf{0} \bmod p_1 p_3 p_4$ and $\sigma^{(\mathbf{w})} = (\kappa_j)^{-1} \cdot (\mathbf{w}_j - w_1 \cdot \mathbf{w}^*) \bmod p_2$. Let $w_1 \in \mathbb{Z}_N$ be the first component of $\mathbf{w}_j$. Suppose we write $\mathbf{w} = \mathbf{w}' + \sigma^{(\mathbf{w})}$ for some $\mathbf{w}' \xleftarrow{\text{R}} \mathbb{Z}_N^n$ where $\mathbf{w}'_1 = 0$ and $\mathbf{w}_j \xleftarrow{\text{R}} \mathbb{Z}_N^n$.
- Let $\mathsf{AS}_{\ell,j} \subseteq S_{\mathsf{enc}}$ be the set of attributes aid $\in S_{\mathsf{enc}}$ associated with the challenge ciphertext where $\mathsf{gid}_{\ell,\mathsf{aid}} = \mathsf{gid}_j$, and let $\mathcal{N}_\ell$ be the set of attributes aid where where slot (ℓ, aid) is not in the corruption set C: namely, $\mathcal{N}_\ell = \{\mathsf{aid} \mid (\ell, \mathsf{aid}) \notin C\}$.
- For aid $\in \mathsf{AS}_{\ell,j}$, let $\sigma^{(\eta_{\mathsf{aid}})} \in \mathbb{Z}_N$ be the unique value where $\sigma^{(\eta_{\mathsf{aid}})} = 0 \bmod p_1 p_2 p_3$ and $\sigma^{(\eta_{\mathsf{aid}})} = \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_j \bmod p_4$. Suppose we write $\eta_{\mathsf{aid}} = \eta'_{\mathsf{aid}} + \sigma^{(\eta_{\mathsf{aid}})}$.

- For aid $\in \mathcal{N}_\ell \cap \mathsf{AS}_{\ell,j}$, let $\sigma^{(r_{\ell,\mathsf{aid}})} \in \mathbb{Z}_N$ be the unique value where $\sigma^{(r_{\ell,\mathsf{aid}})} = 0 \bmod p_1 p_3 p_4$ and $\sigma^{(r_{\ell,\mathsf{aid}})} = -s^{-1} \cdot \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} w_1 \cdot \mathbf{w}^* \bmod p_2$. Suppose we write $r_{\ell,\mathsf{aid}} = r'_{\ell,\mathsf{aid}} + \sigma^{(r_{\ell,\mathsf{aid}})}$.

By construction, observe that these substitutions preserve the distribution of $\mathbf{w}$ and $r_{\ell,\mathsf{aid}}$ in $\mathsf{gHyb}_{j,1}^{(b,\ell)}$ (note that the first coefficient of $\sigma^{(\mathbf{w})}$ is 0, so $w_1 \cdot w^*$ has the same first coefficient as $\mathbf{w}_j$). Consider now the distribution of the $C_{3,\mathsf{aid}}$ terms in $\mathsf{gHyb}_{j,3}^{(b,\ell)}$ under this relabeling (note that this relabeling only affects the terms $C_{3,\mathsf{aid}}$ in the challenge ciphertext):

- For aid $\notin \mathsf{AS}_{\ell,j}$, recall that
$$Q_{\mathsf{aid}} = \prod_{i \in [L]} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}}). \tag{5.6}$$
For all $i \in [L]$ and $j \in [P]$, $U_{i,j} \in \mathbb{G}_{1,4}$, This means $Q_{\mathsf{aid}} \in \mathbb{G}_{1,4}$. Thus,
$$Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}} = Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}'+\sigma^{(\mathbf{w})}} = Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}'}.$$
Finally, the distributions of η_{aid} and r_{aid} are unchanged for aid $\notin \mathsf{AS}_{\ell,j}$, so we conclude that the distributions are identical in this case.

- Suppose aid $\in \mathsf{AS}_{\ell,j} \cap \mathcal{N}_\ell$. For these attributes, we use the fact that the *only* component that depends on $r_{\ell,\mathsf{aid}} \bmod p_2$ is the challenge ciphertext component $C_{3,\mathsf{aid}}$. In this case, the term $r_{\ell,\mathsf{aid}} \bmod p_2$ present in $\hat{T}_{\mathsf{aid}}^{-s}$ serves as fresh randomness in the $\mathbb{G}_2$ subgroup to blind the value of $\mathbf{w}$. First, we have
$$Q_{\mathsf{aid}} = \prod_{i \in [L]} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}}) \cdot (g_2 g_4)^{\kappa_j}.$$
Thus, using the fact that $U_{i,j} \in \mathbb{G}_{1,4}$ and the fact that $\mathbf{w} = \mathbf{w}' + \sigma^{(\mathbf{w})}$, we have
$$\begin{aligned} Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}} = Q_{\mathsf{aid}}^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}(\mathbf{w}'+\sigma^{(\mathbf{w})})} &= \left(\prod_{i \in [L]} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}}) \cdot (g_2 g_4)^{\kappa_j}\right)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}(\mathbf{w}'+\sigma^{(\mathbf{w})})} \\ &= (g_2 g_4)^{\kappa_j \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}(\mathbf{w}'+\sigma^{(\mathbf{w})})} \cdot \left(\prod_{i \in [L]} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}})\right)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}(\mathbf{w}'+\sigma^{(\mathbf{w})})} \\ &= (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\kappa_j \mathbf{w}'} (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\kappa_j \sigma^{(\mathbf{w})}} \cdot \left(\prod_{i \in [L]} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}})\right)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}'} \\ &= (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\kappa_j \mathbf{w}'} g_2^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}(\mathbf{w}_j - w_1 \mathbf{w}^*)} \cdot \left(\prod_{i \in [L]} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}})\right)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}'} \\ &= g_2^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}(\mathbf{w}_j - w_1 \mathbf{w}^*)} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}'}. \end{aligned} \tag{5.7}$$
Next, using the fact that $r_{\ell,\mathsf{aid}} = r'_{\ell,\mathsf{aid}} + \sigma^{(r_{\ell,\mathsf{aid}})}$, we have
$$\begin{aligned} ((g_1 g_2)^s)^{-r_{\ell,\mathsf{aid}}} &= ((g_1 g_2)^s)^{-r'_{\ell,\mathsf{aid}} - \sigma^{(r_{\ell,\mathsf{aid}})}} \\ &= ((g_1 g_2)^s)^{-r'_{\ell,\mathsf{aid}}} g_2^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} w_1 \cdot \mathbf{w}^*}. \end{aligned} \tag{5.8}$$
Finally, substituting $\eta_{\mathsf{aid}} = \eta'_{\mathsf{aid}} + \sigma^{(\eta_{\mathsf{aid}})}$ yields
$$g_4^{\eta_{\mathsf{aid}}} = g_4^{\eta'_{\mathsf{aid}} + \sigma^{\eta_{\mathsf{aid}}}} = g_4^{\eta'_{\mathsf{aid}}} g_4^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}_j}. \tag{5.9}$$

Consider now the value of $C_{3,\mathsf{aid}}$ with this setting of variables. Using Eqs. (5.7) to (5.9), we now have

$$\begin{aligned} C_{3,\mathsf{aid}} &= ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - r_{\ell,\mathsf{aid}} - \sum_{i \neq \ell} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} \cdot g_4^{\eta_{\mathsf{aid}}} \\ &= ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - r'_{\ell,\mathsf{aid}} - \sum_{i \neq \ell} r_{i,\mathsf{aid}}} g_2^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} w_1 \mathbf{w}^*} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'} g_2^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} (\mathbf{w}_j - w_1 \mathbf{w}^*)} \cdot g_4^{\eta'_{\mathsf{aid}}} g_4^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_j} \\ &= ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - r'_{\ell,\mathsf{aid}} - \sum_{i \neq \ell} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'} \cdot g_4^{\eta'_{\mathsf{aid}}} \cdot \underline{(g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_j}}. \end{aligned}$$

This is the ciphertext distribution in $\mathsf{gHyb}_{j,4}^{(b,\ell)}$ with randomness $\mathbf{w}', \eta_{\mathsf{aid}}, r'_{\ell,\mathsf{aid}}$.

- Suppose $\mathsf{aid} \in \mathsf{AS}_{\ell,j} \cap \overline{\mathcal{N}_\ell}$. In this case, the adversary knows the randomness $r_{\ell,\mathsf{aid}}$, so we rely on the linear secret sharing scheme. Specifically, since the adversary's set of attributes do *not* satisfy the challenge policy, there exists a vector $\mathbf{w}^*$ such that $\mathbf{w}^*$ has first coefficient 1 and is orthogonal to $\mathbf{m}_{\mathsf{aid}}$ for rows corresponding to these attributes. Similar to the previous case, we again use Eqs. (5.7) and (5.9) to rewrite $C_{3,\mathsf{aid}}$:

$$\begin{aligned} C_{3,\mathsf{aid}} &= ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}} \cdot g_4^{\eta_{\mathsf{aid}}} \\ &= ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v} - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}' + \sigma^{(\mathbf{w})}} \cdot g_4^{\eta'_{\mathsf{aid}} + \sigma^{(\eta_{\mathsf{aid}})}} \\ &= ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'} g_2^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} (\mathbf{w}_j - w_1 \mathbf{w}^*)} \cdot g_4^{\eta'_{\mathsf{aid}}} g_4^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_j} \\ &= ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'} \cdot g_4^{\eta'_{\mathsf{aid}}} \cdot \underline{(g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_j} g_2^{-\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} w_1 \mathbf{w}^*}}. \end{aligned}$$

  By definition of $\mathbf{w}^*$, we have $\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}^* = 0$ for $\mathsf{aid} \in \mathsf{AS}_{\ell,j} \cap \overline{\mathcal{N}_\ell}$. This means

$$C_{3,\mathsf{aid}} = ((g_1 g_2)^s)^{\beta \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}'} \cdot g_4^{\eta'_{\mathsf{aid}}} \cdot \underline{(g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}_j}}.$$

  Once again, we have recovered the ciphertext distribution in $\mathsf{gHyb}_{j,4}^{(b,\ell)}$ with randomness $\mathbf{w}', \eta_{\mathsf{aid}}, r'_{\ell,\mathsf{aid}}$.

In all three cases, we conclude that with this relabeling of variables, we recover the ciphertext distribution in $\mathsf{gHyb}_{j,4}^{(b,\ell)}$ (with randomness $\mathbf{w}', \eta_{\mathsf{aid}}, r'_{\ell,\mathsf{aid}}$). Thus, the distributions $\mathsf{gHyb}_{j,3}^{(b,\ell)}$ and $\mathsf{gHyb}_{j,4}^{(b,\ell)}$ are statistically close. □

Lemma 5.22. *Suppose Assumption 3.4c holds with respect to* CompGroupGen*. Then for all $\ell \in [L]$, all $j \in [P]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{gHyb}_{j,5}^{(b,\ell)}(\mathcal{A}) = 1] - \Pr[\mathsf{gHyb}_{j,4}^{(b,\ell)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 5.18. □

Lemma 5.23. *For all $\ell \in [L]$, all $j \in [P]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{gHyb}_{j,6}^{(b,\ell)}(\mathcal{A}) = 1] - \Pr[\mathsf{gHyb}_{j,5}^{(b,\ell)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 5.19. □

Lemma 5.24. *Suppose Assumption 3.4c holds with respect to* CompGroupGen*. Then for all $\ell \in [L]$, all $j \in [P]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{gHyb}_{j+1,0}^{(b,\ell)}(\mathcal{A}) = 1] - \Pr[\mathsf{gHyb}_{j,6}^{(b,\ell)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 5.18. □

Lemma 5.25. *For all $\ell \in [L]$, all $j \in [P]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, for all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{iHyb}_{\ell,2}^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{gHyb}_{P+1,0}^{(b,\ell)}(\mathcal{A}) = 1].$$

Proof. Since $P + 1 > P$, all of the user identifiers gid registered to slot ℓ (i.e., the elements of GS_ℓ) have a gid-specific blinding component. Thus these hybrids are identical experiments. □

Lemma 5.12 now follows from Lemmas 5.17 to 5.25 and a hybrid argument. □

5.2.5 Proof of Lemma 5.18 (Delayed gid Programming)

In this section, we give the proof of Lemma 5.18. The goal in this sequence of hybrids is to introduce fresh $\mathbb{G}_{2,4}$ randomness into the hash key U_ℓ that only appears for a *single* user identifier $\mathsf{gid}_{\ell,\mathsf{aid}}$. To do so, we first introduce randomization in the $\mathbb{G}_{2,4}$ subgroup to the hash key in the public parameters. Then, we rely on the additional randomness together with the randomness in the challenge ciphertext component $C_{3,\mathsf{aid}}$ to re-randomize the $\mathbb{G}_{2,4}$ component of $H(\mathsf{gid}_{\ell,\mathsf{aid}})$. We then remove the $\mathbb{G}_{2,4}$ re-randomization from the public parameters, leaving only the fresh statistical re-randomization in $C_{3,\mathsf{aid}}$. This is the "delayed gid programming" technique described in Section 2. We now define the intermediate hybrid experiments between $\mathsf{gHyb}_{j,0}^{(b,\ell)}$ and $\mathsf{gHyb}_{j,1}^{(b,\ell)}$, where we iterate over the identifiers $\mathsf{gid}_j \in \mathsf{GS}_\ell$. Our hybrid sequence is additionally indexed by $i^* \in [P]$.

- $\mathsf{kHyb}_{i^*,0}^{(b,\ell,j)}$ for $i^* \in [P]$: Same as $\mathsf{gHyb}_{j,0}^{(b,\ell)}$, but the challenger introduces fresh randomness in the $\mathbb{G}_{2,4}$ subgroup in the first i^*-1 components of $\vec{U}'_\ell$. Specifically, after sampling $U_{i,a}$ the challenger samples $\zeta_{\ell,a} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and sets
$$U'_{\ell,a} = \begin{cases} U_{\ell,a}(g_2g_4)^{\zeta_{\ell,a}} & a < i^* \\ U_{\ell,a} & a \ge i^*, \end{cases}$$
The challenger sets
$$\vec{U}'_\ell = (U'_{\ell,1},\ldots,U'_{\ell,P}) \text{ where } U'_{\ell,i} = U_{\ell,i}(g_2g_4)^{\zeta_{\ell,i}}.$$
When constructing Q_{aid}, the challenger now sets
$$Q_{\mathsf{aid}} = H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{j\in[L]\setminus\{\ell\}} H(\vec{U}_j, \mathsf{gid}_{j,\mathsf{aid}}).$$

- $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$: Same as $\mathsf{kHyb}_{i^*,0}^{(b,\ell,j)}$, except the challenger introduces a $\mathbb{G}_2$ component to U_{ℓ,i^*}: namely, $U_{\ell,i^*} = (g_1g_2g_4)^{u_{\ell,i^*}}$.

- $\mathsf{kHyb}_{i^*,2}^{(b,\ell,j)}$: Same as $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$, except the challenger randomizes the $\mathbb{G}_{2,4}$ component of U'_{ℓ,i^*}: specifically, the challenger samples $\zeta_{\ell,i^*} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and sets $U'_{\ell,i^*} = U_{\ell,i^*}(g_2g_4)^{\zeta_{\ell,i^*}}$.

We now show that each pair of adjacent hybrids are indistinguishable.

Lemma 5.26. *For all $\ell \in [L]$, $j \in [P]$, all adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*
$$\Pr[\mathsf{kHyb}_{1,0}^{(b,\ell,j)}(\mathcal{A}) = 1] = \Pr[\mathsf{gHyb}_{j,0}^{(b,\ell)}(\mathcal{A}) = 1]$$

Proof. By construction, the definition of $\vec{U}'_\ell$ in $\mathsf{gHyb}_{j,0}^{(b,\ell)}$ is identical to $\vec{U}_\ell$ in $\mathsf{kHyb}_{1,0}^{(b,\ell,j)}$. Thus, the output of these two experiments are identically distributed. □

Lemma 5.27. *Suppose Assumption 3.4c holds with respect to* CompGroupGen. *Then for all $\ell \in [L]$, all $j, i^* \in [P]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*
$$\left|\Pr[\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}(\mathcal{A}) = 1] - \Pr[\mathsf{kHyb}_{i^*,0}^{(b,\ell,j)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that can distinguish these two experiments with non-negligible advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 3.4c with nonnegligible advantage:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_3, g_4, X, Y, T)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_3 \in \mathbb{G}_3$, $g_4 \in \mathbb{G}_4$, $X = (g_1g_2)^{s_{12}}$, $Y = (g_2g_4)^{s_{24}}$ for some $s_{12}, s_{24} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, and either $T = (g_1g_4)^r$ or $T = (g_1g_2g_4)^r$ for some $r \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. The components that depend on the challenge elements X, Y, T are colored for clarity.

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the slot bound 1^L as well as the authority bound 1^P.

3. Algorithm $\mathcal{B}$ samples $\alpha, \beta \xleftarrow{R} \mathbb{Z}_N$. It sets $Z = e(g_1, g_1)^\alpha$, and $h = g_1^\beta$.

4. For each slot $i \in [L]$, $a \in [P]$, and $j \neq i \in [L]$, algorithm $\mathcal{B}$ samples $t_i, \tau_i, u_{i,a}, \gamma_{j,i,a} \xleftarrow{R} \mathbb{Z}_N$ (with the exception of u_{ℓ,i^*}, which algorithm $\mathcal{B}$ does not sample).

   - For $i < \ell$, algorithm $\mathcal{B}$ sets
   $$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = g_1^\alpha A_i^\beta Y^{\tau_i} g_3^{\tau_i} \quad , \quad \vec{U}_i = (U_{i,1}, \dots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$
   - For $i = \ell$, algorithm $\mathcal{B}$ sets
   $$A_\ell = (X g_3)^{t_i} \quad , \quad B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}.$$
   Additionally, for $i' \neq i^* \in [P]$, algorithm $\mathcal{B}$ samples $u_{\ell,i'} \xleftarrow{R} \mathbb{Z}_N$, and sets $U_{\ell,i'} = (g_1 g_4)^{u_{\ell,i'}}$. Algorithm $\mathcal{B}$ sets $U_{\ell,i^*} = T$.
   - For $i > \ell$, algorithm $\mathcal{B}$ sets
   $$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad \vec{U}_i = (U_{i,1}, \dots, U_{i,P}) \text{ where } U_{i,a} = (g_1 g_4)^{u_{i,a}}.$$

5. For $j \neq \ell \in [L]$, algorithm $\mathcal{B}$ computes the hash-function cross terms as follows:
   $$\vec{J}_{j,\ell} = (J_{j,\ell,1}, \dots, J_{j,\ell,P}) \text{ where } J_{j,\ell,a} = A_\ell^{u_{j,a}} (g_3 g_4)^{\gamma_{\ell,j,a}}$$
   For all $i \in [L] \setminus \{\ell\}$ and all $j \neq i$, it sets the hash-function cross terms as follows:
   $$\vec{J}_{j,i} = (J_{j,i,1}, \dots, J_{j,i,P}) \text{ where } J_{j,i,a} = U_{j,a}^{t_i} (g_3 g_4)^{\gamma_{j,i,a}}.$$
   Algorithm $\mathcal{B}$ gives the global parameters
   $$\mathsf{gpp} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, \vec{U}_i\}_{i \in [L]}, \{\vec{J}_{j,i}\}_{j \neq i \in [L]}\right)$$
   to the adversary $\mathcal{A}$. It also initializes a counter ctr $\leftarrow 0$ and an (empty) dictionary D to keep track of the key-generation queries.

6. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{kHyb}_{i^*,0}^{(b,\ell,j)}$ and $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{R} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, and $V_{j,i} = A_j^{r_i}$ for $j \neq i$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, \{V_{j,i}\}_{j \neq i \in [L]})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary D. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \leq \mathsf{ctr}' \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[\mathsf{ctr}']$ and replies to $\mathcal{A}$ with sk'.

7. In the challenge phase, after $\mathcal{A}$ specifies a set of authorities $S_{\mathsf{enc}} \subseteq \mathcal{AU}$, an associated challenge policy $\varphi^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{P' \times n}$ and $\rho \colon S_{\mathsf{enc}} \to [P']$, and two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$. In addition, for each each slot $i \in [L]$ and attribute $\mathsf{aid} \in S_{\mathsf{enc}}$, it also specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}_{i,\mathsf{aid}}^*)$. Algorithm $\mathcal{B}$ sets up the public keys $\mathsf{pk}_{i,\mathsf{aid}}$ as in $\mathsf{kHyb}_{i^*,0}^{(b,\ell,j)}$, $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$:

   - If $c_{i,\mathsf{aid}} \in \{1, \dots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_{i,\mathsf{aid}}] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$ and $r_{i,\mathsf{aid}} = \mathsf{sk}'$. Otherwise, the challenger aborts with output 0.
   - If $c_{i,\mathsf{aid}} = (\perp, r_{i,\mathsf{aid}})$, then the challenger checks that $\mathsf{pk}_{i,\mathsf{aid}}^* = \mathsf{KeyGen}(\mathsf{gpp}, i; r_{i,\mathsf{aid}})$. If not, the challenger aborts with output 0. Otherwise, it sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}_{i,\mathsf{aid}}^*$.

   Finally, for each index $i \in [L]$ and $\mathsf{aid} \in S_{\mathsf{enc}}$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_{i,\mathsf{aid}} = (T_{i,\mathsf{aid}}, \{V_{j,i,\mathsf{aid}}\}_{j \neq i \in [L]})$.

8. Next, to simulate the challenge ciphertext, algorithm $\mathcal{B}$ first samples $\zeta_{\ell,i} \xleftarrow{R} \mathbb{Z}_N$ for all $i < i^*$. Then, it constructs

$$U'_{\ell,a} = \begin{cases} U_{\ell,a} Y^{\zeta_{\ell,a}} & a < i^* \\ U_{\ell,a} & a \geq i^*. \end{cases}$$

Algorithm $\mathcal{B}$ then sets $\vec{U}'_\ell = (U'_{\ell,1}, \ldots, U'_{\ell,P})$. Next, algorithm $\mathcal{B}$ samples $\mathbf{w}_{j'} \xleftarrow{R} \mathbb{Z}_N^n$ for all $j' < j$. It constructs the challenge ciphertext as follows:

- **Message-embedding components:** Set $C_1 = \mu_b^* \cdot e(g_1, X)^\alpha$ and $C_2 = X$.
- **Policy-specific components:** Sample $v_2, \ldots, v_n, w_2, \ldots w_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^\top$ and $\mathbf{w} = [0, w_2, \ldots w_n]^\top$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, it samples blinding factor $\eta_{\mathsf{aid}} \xleftarrow{R} \mathbb{Z}_N$ and sets

$$Q_{\mathsf{aid}} = H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{i \in [L] \setminus \{\ell\}} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}})$$
$$C'_{3,\mathsf{aid}} = X^{\beta \mathbf{m}_k^\top \mathbf{v}' - \sum_{i \in [L]} r_{i,\mathsf{aid}}} \cdot (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^\top \mathbf{w}} g_4^{\eta_{\mathsf{aid}}}.$$

  Then, it constructs the challenge ciphertext components as

$$C_{3,\mathsf{aid}} \leftarrow \begin{cases} C'_{3,\mathsf{aid}} \cdot Y^{\mathbf{m}_{\mathsf{aid}}^\top \mathbf{w}_{j'}} & \mathsf{gid}_{\ell,\mathsf{aid}} = \mathsf{gid}_{j'} \wedge j' < j \\ C'_{3,\mathsf{aid}} & \text{otherwise} \end{cases}, \quad C_{4,k} = g_1^{\mathbf{m}_{\mathsf{aid}}^\top \mathbf{w}}.$$

9. At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which $\mathcal{B}$ also outputs.

In the above reduction, the challenge exponents $r, s_{12} \in \mathbb{Z}_N$ play the roles of $u_{\ell,j}, s \in \mathbb{Z}_N$ in the underlying hybrids, respectively. First, we argue that the components of the global public parameters gpp are correctly simulated:

- First, consider the component A_ℓ in the public parameters gpp. Hybrids $\mathsf{kHyb}_{j,0}^{(b,\ell,j)}$ and $\mathsf{gHyb}_{j,1}^{(b,\ell)}$ set $A_\ell = (g_1 g_2 g_3)^{t_\ell}$. The reduction algorithm sets $A_\ell = ((g_1 g_2)^{s_{12}} g_3)^{t_\ell} = (g_1 g_2)^{s_{12} t_\ell} g_3^{t_\ell}$. As long as $s_{12} \neq 0 \bmod p_1, p_2$, (which holds with overwhelming probability over $s_{12} \xleftarrow{R} \mathbb{Z}_N$), these distributions are uniform over $\mathbb{Z}_N$.
- Next, consider the elements B_i where $i < \ell$. Hybrids $\mathsf{kHyb}_{j,0}^{(b,\ell,j)}$ and $\mathsf{kHyb}_{j,1}^{(b,\ell,j)}$ set $B_i = g_1^\alpha A_i^\beta (g_2 g_3 g_4)^{\tau_i}$. The reduction sets $B_i = g_1^\alpha A_i^\beta (g_2 g_4)^{s_{24} \tau_i} g_3^{\tau_i}$. As long as $s_{24} \neq 0 \bmod p_2, p_4$ (which holds with overwhelming probability over the choice of $s_{24} \xleftarrow{R} \mathbb{Z}_N$), these distributions are uniform over $\mathbb{Z}_N$.

Finally, if $T = (g_1 g_4)^r$, then algorithm $\mathcal{B}$ simulates the public parameters according to the distribution in $\mathsf{kHyb}_{j,0}^{(b,\ell,j)}$. If $T = (g_1 g_2 g_4)^r$, then algorithm $\mathcal{B}$ simulates the public parameters according to the distribution in $\mathsf{kHyb}_{j,1}^{(b,\ell,j)}$. It suffices now to argue that the components of the challenge ciphertext are correctly simulated:

- First, $C_1 = \mu_b^* \cdot e(g_1, (g_1 g_2)^{s_{12}})^\alpha = \mu_b^* \cdot e(g_1, g_1)^{\alpha s_{12}} = Z^{s_{12}}$, which is distributed according to the specification of $\mathsf{kHyb}_{j,0}^{(b,\ell,j)}$ and $\mathsf{kHyb}_{j,1}^{(b,\ell,j)}$.
- Finally, consider the distribution of $C_{3,\mathsf{aid}}$. In the reduction, algorithm $\mathcal{B}$ simulates the re-randomization components $(g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^\top \mathbf{w}_{j'}}$ and $U'_{\ell,i} = U_{\ell,i} (g_2 g_4)^{\zeta_{\ell,i}}$ as

$$Y^{\mathbf{m}_{\mathsf{aid}}^\top \mathbf{w}_{j'}} = (g_2 g_4)^{\mathbf{m}_{\mathsf{aid}}^\top (s_{24} \mathbf{w}_{j'})}$$
$$U'_{\ell,i} = U_{\ell,i} Y^{\zeta_{\ell,i}} = U_{\ell,i} (g_2 g_4)^{s_{24} \zeta_{\ell,i}},$$

  respectively. However, since $\zeta_{\ell,i}$ and $\mathbf{w}_{j'}$ are only used in the construction of the challenge ciphertext, and moreover, they are independent and uniform. Thus, as long as $s_{24} \neq 0 \bmod p_2$ and $s_{24} \neq 0 \bmod p_4$, these two distributions are identically distributed.

We conclude that with overwhelming probability, algorithm $\mathcal{B}$ simulates the challenge ciphertext components exactly as required in $\mathsf{kHyb}_{j,0}^{(b,\ell,j)}$ and $\mathsf{kHyb}_{j,1}^{(b,\ell,j)}$. Correspondingly, algorithm $\mathcal{B}$ breaks Assumption 3.4c with advantage at least $\varepsilon - \mathsf{negl}(\lambda)$. $\square$

Lemma 5.28. *For all $\ell \in [L]$ and $j, i^* \in [P]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{kHyb}_{i^*,2}^{(b,\ell,j)}(\mathcal{A}) = 1] - \Pr[\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$$

Proof. We show that the distributions $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$ and $\mathsf{kHyb}_{i^*,2}^{(b,\ell,j)}$ are statistically indistinguishable. This argument will rely on the $\mathbb{G}_{2,4}$ randomization introduced in $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$. By construction, the only difference between the two experiments is the distribution of U'_{ℓ,i^*} in the $\mathbb{G}_{2,4}$ subgroup. In $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$,

$$U'_{\ell,i^*} = U_{\ell,i^*} = (g_1 g_2 g_4)^{u_{\ell,i^*}}$$

while in $\mathsf{kHyb}_{i^*,2}^{(b,\ell,j)}$,

$$U'_{\ell,i^*} = U_{\ell,i^*}(g_2 g_4)^{\zeta_{\ell,i^*}} = g_1^{u_{\ell,i^*}}(g_2 g_4)^{u_{\ell,i^*}+\zeta_{\ell,i^*}}.$$

Suppose that $u_{\ell,i^*} + \sum_{i<i^*} \zeta_{\ell,i} \neq 0 \bmod p_2$. Since $u_{\ell,i^*} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, this holds with overwhelming probability. Consider now the following relabeling of the variables in $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$:

- Let $\pi^{(\mathbf{w})} \in \mathbb{Z}_N$ be the unique value where $\pi^{(\mathbf{w})} = 1 \bmod p_1 p_3 p_4$ and
$$\pi^{(\mathbf{w})} = \left(u_{\ell,i^*} + \zeta_{\ell,i^*}\right) u_{\ell,i^*}^{-1} \bmod p_2.$$
Suppose we write $\mathbf{w} = \mathbf{w}' \cdot \pi^{(\mathbf{w})}$ for some $\mathbf{w}' \xleftarrow{\mathrm{R}} \mathbb{Z}_N^n$ (with first coefficient 0).
- Let $\pi^{(\zeta)} = (\pi^{(\mathbf{w})})^{-1} \in \mathbb{Z}_N$. For $i < i^*$ suppose we write $\zeta_{\ell,i} = \zeta'_{\ell,i} \cdot \pi^{(\zeta)}$ for some $\zeta'_{\ell,i} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.
- For $\mathsf{aid} \in S_{\mathsf{enc}}$, let $\sigma^{(\eta_{\mathsf{aid}})} \in \mathbb{Z}_N$ be the unique value where
$$\begin{aligned}\sigma^{(\eta_{\mathsf{aid}})} &= 0 \bmod p_1 p_2 p_3 \\ \sigma^{(\eta_{\mathsf{aid}})} &= \zeta_{\ell,i^*} \cdot \mathsf{gid}_{\ell,\mathsf{aid}}^{i^*} \cdot \mathbf{m}_{\mathsf{aid}}^{\mathsf{T}} \mathbf{w}' \bmod p_4.\end{aligned}$$
Suppose we write $\eta_{\mathsf{aid}} = \eta'_{\mathsf{aid}} + \sigma^{(\eta_{\mathsf{aid}})}$ for some $\eta'_{\mathsf{aid}} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.

Since $\mathbf{w}$ is a uniform random vector with first coefficient 0, these substitutions preserve the distribution of $\mathbf{w}, \zeta_{\ell,i}, \eta_{\mathsf{aid}}$ in $\mathsf{kHyb}_{i^*,1}^{(b,\ell,j)}$. By construction, these components only affect the value of $C_{3,\mathsf{aid}}$. Thus, consider the value of $C_{3,\mathsf{aid}}$ with this substitution. To simplify the exposition, we express $C_{3,\mathsf{aid}}$ as

$$C_{3,\mathsf{aid}} = (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}} \cdot g_4^{\eta_{\mathsf{aid}}} \cdot C'_{3,\mathsf{aid}}$$

where $C'_{3,\mathsf{aid}}$ represents the remaining components that can be considered without referring to $\mathbf{w}, \zeta_{\ell,i}, \eta_{\mathsf{aid}}$. Then, we can write

$$\begin{aligned}
C_{3,\mathsf{aid}} &= (Q_{\mathsf{aid}})^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}} \cdot g_4^{\eta_{\mathsf{aid}}} \cdot C'_{3,\mathsf{aid}} \\
&= \left(H(\vec{U}'_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \prod_{i \neq \ell} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}})\right)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}'\pi^{(\mathbf{w})}} \cdot g_4^{\eta'_{\mathsf{aid}}+\sigma^{(\eta_{\mathsf{aid}})}} \cdot C'_{3,\mathsf{aid}} \\
&= \left(\left(\prod_{i<i^*} (U'_{\ell,i})^{\mathsf{gid}_{\ell,\mathsf{aid}}^{i}}\right) \cdot (U'_{\ell,i^*})^{\mathsf{gid}_{\ell,\mathsf{aid}}^{i^*}} \cdot \left(\prod_{i>i^*} (U'_{\ell,i})^{\mathsf{gid}_{\ell,\mathsf{aid}}^{i}}\right) \cdot \prod_{i \neq \ell} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}})\right)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}' \cdot \pi^{(\mathbf{w})}} g_4^{\eta'_{\mathsf{aid}}} g_4^{\sigma^{(\eta_{\mathsf{aid}})}} \cdot C'_{3,\mathsf{aid}} \qquad (5.10) \\
&= \tilde{U} \cdot \left(\prod_{i \neq \ell} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}})\right)^{\mathbf{m}_{\mathsf{aid}}^{\mathsf{T}}\mathbf{w}' \cdot \pi^{(\mathbf{w})}} g_4^{\eta'_{\mathsf{aid}}} \cdot C'_{3,\mathsf{aid}},
\end{aligned}$$

where

$$\begin{aligned}
\tilde{U} &= \left(\left(\prod_{i<i^*} (U_{\ell,i}(g_2g_4)^{\zeta_{\ell,i}})^{\mathsf{gid}^i_{\ell,\mathsf{aid}}} \right) \cdot (g_1g_2g_4)^{u_{\ell,i^*}\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}} \cdot \left(\prod_{i>i^*} U_{\ell,i}^{\mathsf{gid}^i_{\ell,\mathsf{aid}}} \right) \right)^{\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\cdot\pi^{(\mathbf{w})}} g_4^{\sigma^{(\eta_{\mathsf{aid}})}} \\
&= \left(\left(\prod_{i<i^*} (U_{\ell,i}(g_2g_4)^{\zeta_{\ell,i}})^{\mathsf{gid}^i_{\ell,\mathsf{aid}}} \right) \cdot (g_1g_2g_4)^{u_{\ell,i^*}\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}} \cdot \left(\prod_{i>i^*} U_{\ell,i}^{\mathsf{gid}^i_{\ell,\mathsf{aid}}} \right) \right)^{\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\cdot\pi^{(\mathbf{w})}} g_4^{\zeta_{\ell,i^*}\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'}
\end{aligned} \tag{5.11}$$

For all $i \neq i^*$, we have $U_{\ell,i} \in \mathbb{G}_{1,4}$. This means

$$\begin{aligned}
\left(U_{\ell,i}(g_2g_4)^{\zeta_{\ell,i}}\right)^{\mathsf{gid}^i_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\pi^{(\mathbf{w})}} &= U_{\ell,i}^{\mathsf{gid}^i_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'} \left((g_2g_4)^{\zeta'_{\ell,i}\pi^{(\zeta)}}\right)^{\mathsf{gid}^i_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\pi^{(\mathbf{w})}} \\
&= (U_{\ell,i}(g_2g_4)^{\zeta'_{\ell,i}})^{\mathsf{gid}^i_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'} \\
(g_1g_2g_4)^{u_{\ell,i^*}\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\cdot\pi^{(w)}} &= (g_1g_4)^{u_{\ell,i^*}\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'} g_2^{\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'(u_{\ell,i^*}+\zeta_{\ell,i^*})} \\
&= \left(U_{\ell,i^*}g_2^{\zeta_{\ell,i^*}}\right)^{\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'} \\
U_{\ell,i}^{\mathsf{gid}^i_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\pi^{(\mathbf{w})}} &= U_{\ell,i}^{\mathsf{gid}^i_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'}
\end{aligned}$$

Thus, we can now write Eq. (5.11) as

$$\begin{aligned}
\tilde{U} &= \left(\left(\prod_{i<i^*} (U_{\ell,i}(g_2g_4)^{\zeta_{\ell,i}})^{\mathsf{gid}^i_{\ell,\mathsf{aid}}} \right) \cdot (g_1g_2g_4)^{u_{\ell,i^*}\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}} \cdot \left(\prod_{i>i^*} U_{\ell,i}^{\mathsf{gid}^i_{\ell,\mathsf{aid}}} \right) \right)^{\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\cdot\pi^{(\mathbf{w})}} g_4^{\zeta_{\ell,i^*}\mathsf{gid}^{i^*}_{\ell,\mathsf{aid}}\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'} \\
&= H(\vec{U}''_\ell, \mathsf{gid}_{\ell,\mathsf{aid}})^{\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'},
\end{aligned}$$

with an effective hash key $\vec{U}''_\ell$ defined as follows:

$$\vec{U}''_\ell = (U''_{\ell,1}, \ldots, U''_{\ell,P}) \quad \text{where} \quad U''_{\ell,a} = \begin{cases} U_{\ell,a}(g_2g_4)^{\zeta'_{\ell,a}} & a < i^* \\ U_{\ell,a}(g_2g_4)^{\zeta_{\ell,a}} & a = i^* \\ U_{\ell,a} & a > i^*. \end{cases}$$

Using the fact that $U_{i,a} \in \mathbb{G}_{1,4}$ for all $i \neq \ell$, we can now write Eq. (5.10) as

$$\begin{aligned}
C_{3,\mathsf{aid}} &= \tilde{U} \cdot \left(\prod_{i \neq \ell} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}}) \right)^{\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'\cdot\pi^{(\mathbf{w})}} g_4^{\eta'_{\mathsf{aid}}} \cdot C'_{3,\mathsf{aid}} \\
&= \left(H(\vec{U}''_\ell, \mathsf{gid}_{\ell,\mathsf{aid}}) \cdot \left(\prod_{i \neq \ell} H(\vec{U}_i, \mathsf{gid}_{i,\mathsf{aid}}) \right) \right)^{\mathbf{m}^{\mathsf{T}}_{\mathsf{aid}}\mathbf{w}'} \cdot g_4^{\eta'_{\mathsf{aid}}}
\end{aligned}$$

This is precisely the distribution of $C_{3,\mathsf{aid}}$ in $\mathsf{kHyb}^{(b,\ell,j)}_{i^*,2}$ with randomness $\mathbf{w}', \zeta'_{\ell,i}, \eta'_{\mathsf{aid}}$. Thus, we conclude that the distributions $\mathsf{kHyb}^{(b,\ell,j)}_{i^*,1}$ and $\mathsf{kHyb}^{(b,\ell,j)}_{i^*,2}$ are statistically close. □

Lemma 5.29. *Suppose Assumption 3.4c holds with respect to* CompGroupGen. *Then for all $\ell \in [L]$, all $j, i^* \in [P]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{kHyb}^{(b,\ell,j)}_{i^*+1,0}(\mathcal{A}) = 1] - \Pr[\mathsf{kHyb}^{(b,\ell,j)}_{i^*,2}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 5.27. □

Lemma 5.30. *For all $\ell \in [L]$, $j \in [P]$, all adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $\lambda \in \mathbb{N}$,*

$$\Pr[\mathsf{iHyb}^{(b)}_{\ell,1}(\mathcal{A}) = 1] = \Pr[\mathsf{kHyb}^{(b,\ell,j)}_{P+1,0}(\mathcal{A}) = 1].$$

Proof. Since $P + 1 > P$, these experiments are identical by construction. □

Lemma 5.18 now follows from Lemmas 5.26 to 5.30 and a hybrid argument. □

6 Lifting a Slotted Scheme to an Unslotted Scheme

To compile our slotted multi-authority registered ABE scheme into a standard multi-authority registered ABE (see Definition 4.1), we employ a similar "powers of two" transformation similar to the one proposed in [HLWW23] (and implicitly used in earlier works on registration-based cryptography [GHMR18, GHM⁺19]). At a high level, the [HLWW23] compiler instantiates a series of slotted schemes where the number of slots in the i^{th} scheme is 2^i. When users join the system, their key is added to each of the underlying schemes, and aggregation occurs whenever a scheme fills up (for scheme i, this occurs after 2^i users have joined the system). Each time scheme i fills up, the public key for scheme i is updated. Spacing out the number of slots by powers of two ensures that the public keys associated with the larger schemes are not updated too frequently. This ensures the number of decryption updates a user requires scales logarithmically with the number of registered users.

Extending this transformation to the multi-authority setting introduces a number of new challenges. The main challenge is the fact that the authorities are independent, and each authority could issue keys to different sets of registered users (recall that a user only registers with authorities for which they have the corresponding attribute). Thus, to extend the [HLWW23] transformation to this setting, we need to tackle two key challenges:

- **Associating a common slot for each user.** First, we have to ensure that for any user identifier gid, there is a *common* slot index i^* such that gid is registered in slot i^* across all of the target authorities. This is required because the underlying slotted multi-authority registered ABE scheme only supports decryption when the user's keys are registered to the *same* slot across all of the authorities.

- **Users aggregated in different schemes for different authorities.** Second, since each authority serves a differing number of users, a user registered to slot i^* might be aggregated as part of *different* slotted multi-authority registered ABE schemes. Recall in the basic powers-of-two approach from [HLWW23], there are $O(\log L)$ slotted schemes (each supporting a different power-of-two number of slots). Correctness for the underlying slotted multi-authority registered ABE scheme only supports decryption when a user is registered to the same slotted scheme across all of the authorities. Unfortunately, even if each user has a unique index i^* that is shared across all authorities, the powers-of-two transformation does not ensure that the slot i^* is always aggregated as part of the same slotted scheme across all of the authorities. Which slotted scheme contains i^* for each authority is a function of the number of users registered with the particular authority, and this can vary across authorities.

We tackle these challenge as follows:

- **Cover-free sets.** To address the first issue, we have each user register to *multiple* slots at each authority. Specifically, for each user identifier gid, we associate a set of indices S_{gid}. For each authority with whom the user (with identifier gid) wishes to register, the user would register their key in every *unused* slot in S_{gid}. Next, if we choose the sets S_{gid} from a *cover-free family* of sets [KS64], this ensures that every gid can be associated with a slot $i^* \in [L]$ that is *not* covered by the union $S_{\mathsf{gid}_1} \cup S_{\mathsf{gid}_2} \cup \cdots \cup S_{\mathsf{gid}_L}$ of up to L other user identifiers $\mathsf{gid}_1, \ldots, \mathsf{gid}_L \neq \mathsf{gid}$. Note that in our construction, the space of user identifiers is exponential in λ while the number of slots maintained by each authority is polynomially-bounded. (This is the reason we cannot simply use gid as the slot index for the user).

- **Using a single slotted scheme.** To handle the second issue, we consider a different variant of the powers-of-two approach where we use a *single* copy of a slotted multi-authority registered ABE scheme with L slots (as opposed to $\log L$ copies that each support a different power-of-two number of users). Next, we split each authority into $\log L$ "virtual" sub-authorities, where the i^{th} virtual sub-authority is responsible for up to 2^i users, exactly as in the original powers-of-two approach. The unused slots for the i^{th} virtual sub-authority are padded with a key for a dummy user (included as part of the global parameters). The i^{th} virtual sub-authority effectively functions as a slotted scheme with 2^i slots. Next, during encryption, one encrypts to the policy that replaces each authority with a disjunction over the $\log L$ virtual sub-authorities associated with the authority (i.e., it suffices to have a key associated with any virtual sub-authority of the authority). In this approach, there is just a single slotted scheme, and as such, we can rely on correctness of the underlying slotted scheme.

Cover-free families. We now recall the notion of a cover-free family of sets [KS64] that we use.

Definition 6.1 (Cover-Free Family [KS64, Wei23])**.** Let X be a set, and $\mathcal{F} \subset 2^X$ be a family of subsets of X. Then, we say $(X, \mathcal{F})$ is an r-cover free family if for all distinct $A, B_1, \ldots B_r \in \mathcal{F}$ we have that $A \not\subseteq \bigcup_{i \in [r]} B_i$.

Fact 6.2 (Cover-Free Family [EFF85, Example 3.2])**.** For any prime power q and any integer $t < q$, there exists a r-cover-free family $(X, \mathcal{F})$ where $r = \lfloor q/(t-1) \rfloor$, $|X| = q^2$, and $|\mathcal{F}| = q^t$. Moreover, there exists an explicit and efficient algorithm that takes as input an index $i \in [q^t]$ and outputs the i^{th} set in $\mathcal{F}$.

Lemma 6.3 (Cover-Free Family)**.** *Let λ be a security parameter and suppose $\mathcal{GID} \subseteq \{0,1\}^\lambda$. Then, for all polynomially-bounded $r = r(\lambda)$, there exists an efficiently-computable r-cover free family $(X, \mathcal{F})$ where $|X| = \mathsf{poly}(\lambda)$ together with an efficient injective mapping $f \colon \mathcal{GID} \to \mathcal{F}$. Moreover, we can take $X = [|X|]$.*

Proof. Take any prime power $q \in [r\lambda, 2r\lambda]$ and set $t = \lambda + 1$. Then Fact 6.2 yields an s-cover free family $(X, \mathcal{F})$ where $|X| = q^2 = O(r^2\lambda^2)$, $|\mathcal{F}| = q^{\lambda+1} > 2^\lambda \geq |\mathcal{GID}|$, and $s = \lfloor q/(t-1) \rfloor \geq r$. Since $s \geq r$, we conclude that $(X, \mathcal{F})$ is also r-cover free. Finally, the mapping $f \colon \mathcal{GID} \to \mathcal{F}$ is simply the explicit and efficient algorithm from Fact 6.2 (namely, $f(\mathsf{gid})$ is the gid^{th} set in $\mathcal{F}$). Since $|X| = \mathsf{poly}(\lambda, r)$, we can always relabel the elements of X with their index in the lexicographic ordering of the elements of X. This means we can always take $X = [|X|]$ without loss of generality. □

Lifting a slotted scheme to an unslotted scheme. We now describe our transformation:

Construction 6.4 (Slotted Multi-Authority Registered ABE to Unslotted Scheme)**.** Let λ be a security parameter. Let $L = L(\lambda)$ be the number of users, $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of authority identifiers, and $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of global user identifiers where $|\mathcal{GID}_\lambda| \leq 2^\lambda$. We use the following conventions and primitives in our construction:

- Without loss of generality, we assume that the bound on the number of users $L = 2^\ell$ is a power of two. Rounding the bound to the next power of two incurs at most a factor of 2 overhead.
- Let $\mathcal{AU}' = \{\mathcal{AU}'_\lambda\}$ be a set of authority identifiers where $\mathcal{AU}'_\lambda = \mathcal{AU}_\lambda \times [0, \ell]$. Next, let $\Pi_{\mathsf{sRABE}} = (\mathsf{sRABE.GlobalSetup}, \mathsf{sRABE.KeyGen}, \mathsf{sRABE.IsValid}, \mathsf{sRABE.Aggregate}, \mathsf{sRABE.Encrypt}, \mathsf{sRABE.Decrypt})$ be a slotted multi-authority registered ABE scheme with authority identifiers $\mathcal{AU}'$ and global identifier space $\mathcal{GID}$. For ease of exposition, we will assume that Π_{sRABE} supports single-use monotone Boolean formulas where there is an a priori bounded number of users per policy (e.g., Construction 5.1). Let $P = P(\lambda)$ be a bound on the maximum number of authorities associated with each policy.
- The multi-authority registered ABE scheme will internally maintain $\ell + 1$ slotted ABE schemes, where the k^{th} scheme is a slotted scheme that will "support" 2^k users.
- Each authority maintains auxiliary data $\mathsf{aux}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{D}_{0,\mathsf{aid}}, \mathsf{D}_{1,\mathsf{aid}}, \mathsf{D}_{2,\mathsf{aid}}, \mathsf{mpk}_{\mathsf{aid}})$ which contains the following data:
  - A counter $\mathsf{ctr}_{\mathsf{aid}}$ that keeps track of the number of registered users in the system.
  - A dictionary $\mathsf{D}_{0,\mathsf{aid}}$ that maps global identifiers gid to a counter ctr (indicating the number of registered users at the time user gid registered) together with the set of slots F'_{gid} to which user gid was registered.
  - A dictionary $\mathsf{D}_{1,\mathsf{aid}}$ that maps a slot index $i \in X$ to a pair $(\mathsf{gid}, \mathsf{pk})$ which specifies the public key assigned to slot i and the associated user identifier gid.
  - A dictionary $\mathsf{D}_{2,\mathsf{aid}}$ that maps a scheme index $k \in [0, \ell]$ and a user identifier gid to the helper decryption keys associated with scheme k and gid.
  - The current master public key $\mathsf{mpk}_{\mathsf{aid}} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$.

  If $\mathsf{aux} = \bot$, we parse it as $(\mathsf{ctr}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ where $\mathsf{ctr} = 0$, $\mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2 = \varnothing$, and $\mathsf{mpk} = (0, \bot, \ldots, \bot)$. This corresponds to a fresh scheme with no registered users.

- The master public keys $\mathsf{mpk}_{\mathsf{aid}}$ associated with each authority will take the form of $(\mathsf{ctr}, \mathsf{mpk}_0, \dots \mathsf{mpk}_\ell)$, where ctr is the total number of users registered to this authority, and each mpk_k is a master public key of the underlying slotted scheme Π_{sRABE}.
- The helper decryption keys $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}}$ will take the form of $(\mathsf{ctr}, F'_{\mathsf{gid}}, \mathsf{hsk}_0, \dots \mathsf{hsk}_\ell)$, where ctr is a counter (indicating the number of registered users at the time user gid registered), F'_{gid} is the set of slots to which user gid was registered, and each hsk_k is a collection of helper decryption keys for the underlying slotted scheme Π_{sRABE} associated with slots $i \in F'_{\mathsf{gid}}$.

We now construct a multi-authority registered ABE scheme $\Pi_{\mathsf{MA\text{-}RABE}} = (\mathsf{GlobalSetup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{UpdateKey}, \mathsf{Encrypt}, \mathsf{Decrypt})$ as follows:

- $\mathsf{GlobalSetup}(1^\lambda, 1^L, 1^P)$: On input the security parameter λ, a bound $L = 2^\ell$ on the number of users, and a bound P on the number of users appearing in a policy, the setup algorithm proceeds as follows:
  - First, the setup algorithm instantiates an $(L \cdot P)$-cover free family of sets $(X, \mathcal{F})$ where $|\mathcal{F}| \geq 2^\lambda \geq |\mathcal{GID}_\lambda|$. Let $f\colon \mathcal{GID} \to \mathcal{F}$ be the associated injective mapping (see Lemma 6.3). Recall that $X = [|X|]$.
  - Next, it runs the setup algorithm for the slotted scheme:
  $$(\mathsf{sRABE.gpp}, \mathsf{sRABE.gep}) \leftarrow \mathsf{sRABE.GlobalSetup}(1^\lambda, 1^{|X|}, 1^{P\cdot\ell}).$$
  - Finally, it samples "dummy" public keys for each slot $j \in [|X|]$ and $k \in [0, \ell]$:
  $$(\mathsf{pk}'_{j,k}, \mathsf{sk}'_{j,k}) \leftarrow \mathsf{sRABE.KeyGen}(\mathsf{sRABE.gpp}, j).$$

  It outputs the global public parameters
  $$\mathsf{gpp} = (\mathsf{sRABE.gpp}, f, \{(j, k, \mathsf{pk}'_{j,k})\}_{j \in X, k \in [0,\ell]})$$
  and the global encryption parameters $\mathsf{gep} = \mathsf{sRABE.gep}$. The message space $\mathcal{M}$ is the same as that associated with $\mathsf{sRABE.gpp}$.
- $\mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid})$: On input the common reference string $\mathsf{gpp} = (\mathsf{sRABE.gpp}, f, \{(j, k, \mathsf{pk}'_{j,k})\}_{j \in X, k \in [0,\ell]})$ and the user identifier gid, the key-generation algorithm first computes $F_{\mathsf{gid}} = f(\mathsf{gid}) \subseteq X$. Then, for each index $i \in F_{\mathsf{gid}}$ and $k \in [0, \ell]$, it generates $(\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{sRABE.KeyGen}(\mathsf{sRABE.gpp}, i)$. Output
$$\mathsf{pk} = (\{(i, k, \mathsf{pk}_{i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]}) \quad \text{and} \quad \mathsf{sk} = (\mathsf{gid}, \{(i, k, \mathsf{sk}_{i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]}).$$
- $\mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}_{\mathsf{aid}}, \mathsf{gid}, \mathsf{pk}_{\mathsf{gid}})$: On input the public parameters $\mathsf{gpp} = (\mathsf{sRABE.gpp}, f, \{(j, k, \mathsf{pk}'_{j,k})\}_{j \in X, k \in [0,\ell]})$, the auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ for an authority, a user identifier gid, and a public key $\mathsf{pk} = (\{(i, k, \mathsf{pk}_{i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]})$, the registration algorithm proceeds as follows:
  - First, it checks that $F_{\mathsf{gid}} = f(\mathsf{gid})$. Otherwise, the algorithm halts and outputs the current auxiliary data aux and master public key mpk.
  - For each $i \in F_{\mathsf{gid}}$ and $k \in [0, \ell]$, check that $\mathsf{sRABE.IsValid}(\mathsf{sRABE.gpp}, i, \mathsf{pk}_{i,k}) = 1$. Otherwise, the algorithm halts and outputs the current auxiliary data aux and master public key mpk.
  - If $\mathsf{D}_0[\mathsf{gid}] \neq \perp$, the algorithm halts and outputs the current auxiliary data aux and master public key mpk.
  - Define $F'_{\mathsf{gid}} = \{i \mid i \in F_{\mathsf{gid}} \wedge \mathsf{D}_1[i] = \perp\}$. Set $\mathsf{D}_0[\mathsf{gid}] = (\mathsf{ctr}, F'_{\mathsf{gid}})$. Then, for each $i \in F'_{\mathsf{gid}}$, set
  $$\mathsf{D}_1[i] = (\mathsf{gid}, \{(k, \mathsf{pk}_{i,k})\}_{k \in [0,\ell]}).$$
  - For each $k \in [0, \ell]$, check if $\mathsf{ctr} + 1 = 0 \bmod 2^k$. If so, the registration algorithm performs the following update procedure:

* For all $i \in X$, the registration algorithm first looks up $(\mathsf{gid}_i, \{(k', \mathsf{pk}_{i,k'})\}_{k' \in [0,\ell]}) = \mathsf{D}_1[i]$. If it is the case that $\mathsf{D}_0[\mathsf{gid}_i] \in [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$, then set

$$\mathsf{pk}_i^* = \mathsf{pk}_{i,k} \quad \text{and} \quad \mathsf{gid}_i^* = \mathsf{gid}_i.$$

Otherwise, if $\mathsf{D}_0[\mathsf{gid}_i] \notin [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$ or if $\mathsf{D}_1[i] = \perp$, let

$$\mathsf{pk}_i^* = \mathsf{pk}'_{i,k} \quad \text{and} \quad \mathsf{gid}_i^* = 0.$$

* Compute the aggregated parameters

$$(\mathsf{mpk}_k^*, \{(i, \mathsf{hsk}_{i,k}^*)\}_{i \in X}) = \mathsf{sRABE.Aggregate}\big(\mathsf{sRABE.gpp}, (\mathsf{gid}_1^*, \mathsf{pk}_1^*), \ldots, (\mathsf{gid}_{|X|}^*, \mathsf{pk}_{|X|}^*)\big).$$

* For each user identifier gid where $\mathsf{D}_0[\mathsf{gid}] = (\mathsf{ctr}_{\mathsf{gid}}, F'_{\mathsf{gid}})$ and $\mathsf{ctr}_{\mathsf{gid}} \in [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$, set

$$\mathsf{D}_2[\mathsf{gid}, k] = \big\{(i, \mathsf{hsk}_{i,k}^*) \mid i \in F'_{\mathsf{gid}}\big\}.$$

* Update $\mathsf{mpk}_k = \mathsf{mpk}_k^*$

– Finally, the registration algorithm increments $\mathsf{ctr} = \mathsf{ctr} + 1$ and outputs the new master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$ along with the updated auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$.

• $\mathsf{UpdateKey}(\mathsf{gpp}, \mathsf{aux}_{\mathsf{aid}}, \mathsf{gid})$: On input the global public parameters $\mathsf{gpp} = (\mathsf{sRABE.gpp}, f, \{(j, k, \mathsf{pk}'_{j,k})\}_{j \in X, k \in [0,\ell]})$, the auxiliary data $\mathsf{aux}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ for an authority, and a user identifier gid, the update algorithm looks up $(\mathsf{ctr}_{\mathsf{gid}}, F'_{\mathsf{gid}}) = \mathsf{D}_0[\mathsf{gid}]$. Then, for each $k \in [0, \ell]$, the update algorithm sets $\mathsf{hsk}_k = \mathsf{D}_2[\mathsf{gid}, k]$ and outputs the helper decryption key $\mathsf{hsk}_{\mathsf{gid}} = \big(\mathsf{ctr}_{\mathsf{gid}}, F'_{\mathsf{gid}}, \mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell\big)$.

• $\mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi), \{\mathsf{mpk}_{\mathsf{aid}}\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu)$: On input the encryption parameters $\mathsf{gep} = \mathsf{sRABE.gep}$, an access policy $(S_{\mathsf{enc}}, \varphi)$, the master public keys $\mathsf{mpk}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{mpk}_{\mathsf{aid},0}, \ldots, \mathsf{mpk}_{\mathsf{aid},\ell})$ associated with authorities $\mathsf{aid} \in S_{\mathsf{enc}}$, and a message $\mu \in \mathcal{M}$, the encryption algorithm first construct a new authority set

$$S'_{\mathsf{enc}} = \{(\mathsf{aid}, k) \mid \mathsf{aid} \in S_{\mathsf{enc}}, k \in [0, \ell]\}.$$

Then, it construct the extended policy φ' by substituting each appearance of attribute aid in the policy φ with the clause $\big(\bigvee_{k \in [0,\ell]} (\mathsf{aid}, k)\big)$. It then computes

$$\mathsf{ct}' \leftarrow \mathsf{sRABE.Encrypt}(\mathsf{sRABE.gep}, (S'_{\mathsf{enc}}, \varphi'), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid},k})\}_{(\mathsf{aid},k) \in S'_{\mathsf{enc}}}, \mu).$$

It outputs the ciphertext $\mathsf{ct} = \big(\mathsf{ct}', \{(\mathsf{aid}, \mathsf{ctr}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}\big)$.

• $\mathsf{Decrypt}(\mathsf{gpp}, S, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}})\}_{\mathsf{aid} \in S}, \mathsf{ct})$: On input the global parameters

$$\mathsf{gpp} = (\mathsf{sRABE.gpp}, f, \{(j, k, \mathsf{pk}'_{j,k})\}_{j \in X, k \in [0,\ell]}),$$

a set of attributes $S \subseteq S_{\mathsf{enc}}$, a collection of secret keys $\mathsf{sk}_{\mathsf{aid}} = (\mathsf{gid}, \{(i, k, \mathsf{sk}_{\mathsf{aid},i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]})$, a collection of helper keys $\mathsf{hsk}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid},\mathsf{gid}}, F'_{\mathsf{aid},\mathsf{gid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid},0}, \ldots, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid},\ell})$, and a ciphertext $\mathsf{ct} = (\mathsf{ct}', \{(\mathsf{aid}, \mathsf{ctr}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}})$, the decryption algorithm proceeds as follows:

– Take any $i^* \in \bigcap_{\mathsf{aid} \in S} F'_{\mathsf{gid},\mathsf{aid}}$. If no such i^* exists, output $\perp$.

– For each $\mathsf{aid} \in S$, let k_{aid} to be the index of the most significant bit where $\mathsf{ctr}_{\mathsf{aid}}$ and $\mathsf{ctr}_{\mathsf{aid},\mathsf{gid}}$ differ.

– If $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid},k_{\mathsf{aid}}} = \perp$, abort and output $\perp$. Otherwise, parse

$$\mathsf{hsk}_{\mathsf{aid},\mathsf{gid},k_{\mathsf{aid}}} = \{(i, \mathsf{hsk}_{\mathsf{aid},i,k_{\mathsf{aid}}}) \mid i \in F'_{\mathsf{gid},\mathsf{aid}}\}.$$

Output

$$m = \mathsf{sRABE.Decrypt}(\mathsf{sRABE.gpp}, S, i^*, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid},i^*,k_{\mathsf{aid}}}, \mathsf{hsk}_{\mathsf{aid},i^*,k_{\mathsf{aid}}})\}_{\mathsf{aid} \in S}, \mathsf{ct}').$$

Correctness, compactness, and efficiency. Recall the correctness game from Definition 4.3. We will show if the underlying slotted scheme Π_{sRABE} satisfies completeness, correctness and compactness, then the transformed scheme from Construction 6.4 also satisfies perfect correctness, compactness, and efficiency. To help analyze our construction, we use [HLWW23, Claim 6.4], which we restate below.

Claim 6.5 ([HLWW23, Claim 6.4])**.** *Let $x, y < 2^{\ell+1}-1$ be nonnegative integers with binary representations $x = x_\ell \cdots x_1 x_0$ and $y = y_\ell \cdots y_1 y_0$. Suppose $x < y$. Let $k_{x,y} = \max\{k \in [0,\ell] : x_k \neq y_k\}$. Namely, $k_{x,y}$ is the index of the most significant bit on which x and y differ. Then $k_{x,y} \leq k_{x,y+1}$. Moreover, if $k_{x,y} < k_{x,y+1}$, then $y + 1 = 0 \bmod 2^{k_{x,y+1}}$*

Theorem 6.6 (Correctness)**.** *Suppose Π_{sRABE} is complete and perfectly correct. Then Construction 6.4 is a perfectly correct MA-RABE scheme with respect to the following* IsReady *function:*

$$\mathsf{IsReady}(e, d, h) = \begin{cases} 1 & h \geq 2^{k_{d,e}} \\ 0 & \textit{otherwise}, \end{cases}$$

where $k_{d,e}$ is as defined in Claim 6.5.

Proof. We first recall the correctness game from Definition 4.3 instantiated with Construction 6.4:

- **Setup phase:** The challenger receives parameters $(1^\lambda, 1^L, 1^P)$, and generates the global parameters

$$\mathsf{gpp} = (\mathsf{sRABE.gpp}, f, \{(j, k, \mathsf{pk}'_{j,k})\}_{j \in X, k \in [0,\ell]}) \quad \text{and} \quad \mathsf{gep} = \mathsf{sRABE.gep}$$

  by computing $\mathsf{Setup}(1^\lambda, 1^L, 1^P)$. Then, the challenger initializes a dictionary D mapping authority identifiers aid to a tuple $(c, k, j, (\mathsf{mpk}_0, \ldots, \mathsf{mpk}_c), \mathsf{aux})$, where the components are defined as follows:

  - c represents the total number of users registered to this authority.
  - $k = (d, \mathsf{pk}_{\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{gid}^*})$ is a tuple representing the target user identifier $\mathsf{gid}^* \in \mathcal{GID}_\lambda$. Here, $d \in \mathbb{N}$ denotes the number of users registered with authority aid immediately after gid^* is registered, and $(\mathsf{pk}_{\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{gid}^*})$ are the public/secret keys associated with gid^* and authority aid (as sampled by the challenger in the security game). If the adversary has not yet registered a target user, then $k = \perp$.
  - $j = (h, \mathsf{hsk}_{\mathsf{gid}^*})$ is a tuple containing the counter value $h \in \mathbb{N}$ when the helper decryption key $\mathsf{hsk}_{\mathsf{gid}^*}$ for the target user gid^* was last requested (for authority aid). If the adversary has not yet registered a target user or requested an update, then $j = \perp$.
  - $(\mathsf{mpk}_0, \ldots, \mathsf{mpk}_c)$ is the list of master public keys associated with the authority after each registration.
  - aux is the auxiliary state of authority aid.

  At the beginning of the correctness game, the challenger initializes $c = 0$, $k = \perp$, $j = \perp$, $\mathsf{mpk}_0 = \perp$ and $\mathsf{aux} = \perp$.

- **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following queries:

  - **Register non-target key query:** In a non-target-key registration query, the adversary $\mathcal{A}$ specifies a user identifier gid, a public key $\mathsf{pk}_{\mathsf{gid}} = (\{(i, k, \mathsf{pk}_{i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]})$, and an authority $\mathsf{aid} \in \mathcal{AU}$. The challenger first retrieves the entry $(c, k, j, (\mathsf{mpk}_0, \ldots \mathsf{mpk}_c), \mathsf{aux}) = \mathsf{D}[\mathsf{aid}]$ and computes $\mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}, \mathsf{pk}_{\mathsf{gid}})$ as follows:

    - The challenger parses $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$.
    - The challenger checks that $F_{\mathsf{gid}} = f(\mathsf{gid})$. If not, the challenger outputs the current auxiliary data aux and master public key mpk.
    - For each $i \in F_{\mathsf{gid}}, k \in [0, \ell]$, the challenger checks that $\mathsf{IsValid}(\mathsf{sRABE.gpp}, i, \mathsf{pk}_{i,k}) = 1$. Otherwise, the challenger outputs the current auxiliary data aux and master public key mpk.
    - If $\mathsf{D}_0[\mathsf{gid}] \neq \perp$, the challenger outputs the current auxiliary data aux and master public key mpk.
    - Define $F'_{\mathsf{gid}} = \{i \mid i \in F_{\mathsf{gid}} \wedge \mathsf{D}_1[i] = \perp\}$. Set $\mathsf{D}_0[\mathsf{gid}] = (\mathsf{ctr}, F'_{\mathsf{gid}})$, and for each $i \in F'_{\mathsf{gid}}$, set $\mathsf{D}_1[i] = (\mathsf{gid}, \{\mathsf{pk}_{i,k}\}_{k \in [0,\ell]})$.

* For all $k \in [0, \ell]$, the challenger checks if $\mathsf{ctr} + 1 = 0 \bmod 2^k$. If so, then it performs the following updates:
   · For all $i \in X$, the challenger first looks up $(\mathsf{gid}_i, \{(k', \mathsf{pk}_{i,k'})\}_{k' \in [0,\ell]}) = \mathsf{D}_1[i]$. If it is the case that $\mathsf{D}_0[\mathsf{gid}_i] \in [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$, then set

$$\widetilde{\mathsf{pk}}_i = \mathsf{pk}_{i,k} \quad \text{and} \quad \widetilde{\mathsf{gid}}_i = \mathsf{gid}_i.$$

   Otherwise, if $\mathsf{D}_0[\mathsf{gid}_i] \notin [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$ or if $\mathsf{D}_1[i] = \perp$, let

$$\widetilde{\mathsf{pk}}_i = \mathsf{pk}'_{i,k} \quad \text{and} \quad \widetilde{\mathsf{gid}}_i = 0.$$

   · Compute the aggregated parameters

$$(\widetilde{\mathsf{mpk}}_k, \{(i, \widetilde{\mathsf{hsk}}_{i,k})\}_{i \in X}) = \mathsf{sRABE.Aggregate}\big(\mathsf{sRABE.gpp}, (\widetilde{\mathsf{gid}}_1, \widetilde{\mathsf{pk}}_1), \dots, (\widetilde{\mathsf{gid}}_{|X|}, \widetilde{\mathsf{pk}}_{|X|})\big).$$

   · For each user identifier gid where $\mathsf{D}_0[\mathsf{gid}] = (\mathsf{ctr}_{\mathsf{gid}}, F'_{\mathsf{gid}})$ and $\mathsf{ctr}_{\mathsf{gid}} \in [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$, set

$$\mathsf{D}_2[\mathsf{gid}, k] = \big\{(i, \widetilde{\mathsf{hsk}}_{i,k}) \mid i \in F'_{\mathsf{gid}}\big\}.$$

   · Update $\mathsf{mpk}_k = \mathsf{mpk}^*_k$
* Finally, the challenger increments $\mathsf{ctr} = \mathsf{ctr} + 1$ and outputs the new master public key $\mathsf{mpk}_{c+1} = (\mathsf{ctr}, \mathsf{mpk}_0, \dots \mathsf{mpk}_\ell)$ and updated auxiliary data $\mathsf{aux}' = \mathsf{aux}$.

The challenger updates its auxiliary data by setting $\mathsf{D}[\mathsf{aid}] := (c + 1, k, (\mathsf{mpk}_0, \dots \mathsf{mpk}_c, \mathsf{mpk}_{c+1}), \mathsf{aux}')$ and replies to $\mathcal{A}$ with $(\mathsf{mpk}_{c+1}, \mathsf{aux}')$.

– **Register target key query:** In a target-key registration query, the adversary specifies a target user identifier gid^* and an authority aid. If the adversary has made a previous register-target-key query with a user identifier $\mathsf{gid} \neq \mathsf{gid}^*$, then the challenger outputs $\perp$. In addition, if the adversary has already made a registration query for gid^* to authority aid, then the challenger also replies with $\perp$. Otherwise, the challenger retrieves $(c, k, (\mathsf{mpk}_0, \dots \mathsf{mpk}_c), \mathsf{aux}) = \mathsf{D}[\mathsf{aid}]$. Then, it samples

$$(\mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*}) \leftarrow \mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid}^*),$$

where $\mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*} = (\{\mathsf{pk}_{i,k,\mathsf{aid}}\}_{i \in F_{\mathsf{gid}^*}, k \in [0,\ell]})$ and $\mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*} = (\mathsf{gid}^*, \{\mathsf{sk}_{i,k,\mathsf{aid}}\}_{i \in F_{\mathsf{gid}^*}, k \in [0,\ell]})$. The challenger register this key using the same procedure described above for registering a non-target-key. Afterwards it updates

$$\mathsf{D}[\mathsf{aid}] = (c + 1, (d, \mathsf{pk}, \mathsf{sk}, \mathsf{hsk}), (\mathsf{mpk}_0, \dots \mathsf{mpk}_c, \mathsf{mpk}_{c+1}), \mathsf{aux}'),$$

where $d = c+1$, $\mathsf{pk} = \mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}$, and $\mathsf{sk} = \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*}$. The challenger replies to $\mathcal{A}$ with $(\mathsf{mpk}_{c+1}, \mathsf{aux}', \mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*})$.

– **Request update query:** In a request update query, the adversary sends an authority identifier aid. The challenger looks up $(c, (d, \mathsf{pk}_{\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{gid}^*}), (h, \mathsf{hsk}_{\mathsf{gid}^*}), (\mathsf{mpk}_0, \dots, \mathsf{mpk}_c), \mathsf{aux}) = \mathsf{D}[\mathsf{aid}]$. If $d = \perp$, the challenger halts and returns 0. Otherwise, the challenger computes $\mathsf{hsk}' \leftarrow \mathsf{UpdateKey}(\mathsf{gpp}, \mathsf{aux}, \mathsf{pk})$ and updates $h = c$ and $\mathsf{hsk} = \mathsf{hsk}'$ in $\mathsf{D}[\mathsf{aid}]$.

• **Challenge phase:** During the challenge phase, the adversary $\mathcal{A}$ makes an encryption query in the form of a policy $(S_{\mathsf{enc}}, \varphi)$, a list of indices $\{e_{\mathsf{aid}}\}_{\mathsf{aid} \in S_{\mathsf{enc}}}$, and a message m. For each $\mathsf{aid} \in S_{\mathsf{enc}}$, the challenger looks up $\mathsf{D}[\mathsf{aid}] = (c_{\mathsf{aid}}, (d_{\mathsf{aid}}, \mathsf{pk}_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*}), (h_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*}), (\mathsf{mpk}_{\mathsf{aid},0}, \dots, \mathsf{mpk}_{\mathsf{aid},c_{\mathsf{aid}}}), \mathsf{aux}_{\mathsf{aid}})$. If for any i, $e_{\mathsf{aid}} > c_{\mathsf{aid}}$, output $\perp$. Otherwise, the challenger responds with the ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid},e_{\mathsf{aid}}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, m)$ computed as follows:

   – For each $\mathsf{aid} \in S_{\mathsf{enc}}$, parse $\mathsf{mpk}_{\mathsf{aid},e_{\mathsf{aid}}} = (\mathsf{ctr}, \mathsf{mpk}_{\mathsf{aid},0}, \dots \mathsf{mpk}_{\mathsf{aid},\ell})$.
   – Let $S'_{\mathsf{enc}} = \{(\mathsf{aid}, k) \mid \mathsf{aid} \in S_{\mathsf{enc}}, k \in [0, \ell]\}$ and φ' be the policy obtained by substituting each appearance of attribute aid in φ with the clause $\big(\bigvee_{k \in [0,\ell]} (\mathsf{aid}, k)\big)$.

  - Compute the ciphertext $\mathsf{ct}' \leftarrow \mathsf{sRABE.Encrypt}(\mathsf{gep}, (S'_{\mathsf{enc}}, \varphi'), \{((\mathsf{aid}, k)\mathsf{mpk}_{\mathsf{aid},k})\}_{(\mathsf{aid},k)\in S'_{\mathsf{enc}}}, m)$.
  - Set $\mathsf{ct} = (\mathsf{ct}', \{(\mathsf{aid}, \mathsf{ctr}_{\mathsf{aid}})\}_{\mathsf{aid}\in S_{\mathsf{enc}}})$.

Adversary $\mathcal{A}$ then replies with a set $S \subseteq S_{\mathsf{enc}}$ of authorities. The challenger evaluates the decryption algorithm $\mathsf{Decrypt}(\mathsf{gpp}, S, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*})\}_{\mathsf{aid}\in S,\mathsf{ct}})$ as follows:

  - For each $\mathsf{aid} \in S$, check that $d_{\mathsf{aid}} \neq \perp$ and $\mathsf{IsReady}(e_{\mathsf{aid}}, d_{\mathsf{aid}}, h_{\mathsf{aid}}) = 1$. Otherwise, abort with output 0.
  - Parse

$$\begin{aligned} \mathsf{sk}_{\mathsf{aid},\mathsf{gid}^*} &= (\mathsf{gid}^*, \{(i, k, \mathsf{sk}_{\mathsf{aid},i,k})\}_{i\in F_{\mathsf{gid}^*}, k\in[0,\ell]}) \\ \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*} &= (\mathsf{ctr}_{\mathsf{aid},\mathsf{gid}^*}, F'_{\mathsf{aid},\mathsf{gid}^*}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*,0}, \ldots \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*,\ell}). \end{aligned}$$

  - For each $\mathsf{aid} \in S$, let k_{aid} to be the index of the most significant bit which $\mathsf{ctr}_{\mathsf{aid}}$ and $\mathsf{ctr}_{\mathsf{aid},\mathsf{gid}}$ differ (where bits are 0-indexed starting from the least significant bit).
  - If $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*,k_{\mathsf{aid}}} = \perp$, output $m' = \perp$. Otherwise, recall $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*,k_{\mathsf{aid}}} = \{(i, \mathsf{hsk}_{\mathsf{aid},i,k_{\mathsf{aid}}})\}_{i\in F'_{\mathsf{aid},\mathsf{gid}^*}}$.
  - Compute $m' = \mathsf{sRABE.Decrypt}(\mathsf{sRABE.gpp}, \{\mathsf{sk}_{\mathsf{aid},i^*,k_{\mathsf{aid}}} \mathsf{hsk}_{\mathsf{aid},i^*,k_{\mathsf{aid}}}\}_{\mathsf{aid}\in S}, \mathsf{ct}')$.

If S satisfies φ and $m' \neq m$, the challenger outputs $b = 1$. Otherwise, the challenger outputs $b = 0$.

We begin by showing the following invariant:

Lemma 6.7. *Let* $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ *be the auxiliary data for an arbitrary authority* aid *at any point in the correctness game* after *the adversary has made a target-key registration query. Write* $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{mpk}_0, \ldots \mathsf{mpk}_\ell)$. *Let* $\mathsf{pk}^*_{\mathsf{gid}} = (\mathsf{gid}^*, \{\mathsf{pk}^*_{i,k}\}_{i\in F_{\mathsf{gid}}, k\in[0,\ell]})$ *be the target key the challenger sampled in the register-target-key query. Then for all* $i' \in F'_{\mathsf{aid},\mathsf{gid}^*}$, *it holds that* $\mathsf{mpk}_{k'}$ *was the output of a call to* $\mathsf{sRABE.Aggregate}(\mathsf{gpp}, \cdot)$ *on a tuple of keys and user identifiers that included the target key* $(\mathsf{pk}^*_{i',k'}, \mathsf{gid}^*)$ *where* k' *is the index of the most significant bit where* $\mathsf{D}_0[\mathsf{gid}^*]$ *and* $\mathsf{ctr}_{\mathsf{aid}}$ *differ.*

Proof. We prove Lemma 6.7 via induction. The base case corresponds to the state of the challenger immediately after the adversary registers the target key. Let $\mathsf{aux}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk}_{\mathsf{aid}})$ be the auxiliary data at the beginning of the adversary's first target-key registration query. We start by showing the invariant holds immediately following the query:

- In a target-key registration query, algorithm $\mathcal{A}$ sends the user identifier gid^* and the authority aid.
- The challenger runs $\mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid}^*)$. This is done by first computing F_{gid^*}. Then, for each $k \in [0, \ell]$ and $i \in F_{\mathsf{gid}^*}$, the challenger computes $\mathsf{pk}^*_{i,k} \leftarrow \mathsf{sRABE.KeyGen}(\mathsf{sRABE.gpp}, i)$. This yields the public key $\mathsf{pk}^* = (\mathsf{gid}, \{(i, k, \mathsf{pk}^*_{i,k})\}_{i\in F_{\mathsf{gid}^*}, k\in[0,\ell]})$.
- Next, the challenger registers the key it just computed using $\mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}_{\mathsf{aid}}, \mathsf{gid}^*, \mathsf{pk}^*)$. By completeness of Π_{sRABE}, we have that for all $k \in [0, \ell]$, $\mathsf{sRABE.IsValid}(\mathsf{sRABE.gpp}, i, \mathsf{pk}^*_{i,k}) = 1$. Recall further that gid^* has not been registered to authority aid before (otherwise the target-key query aborts). Thus, $\mathsf{D}_0[\mathsf{gid}^*] = \perp$ and RegPK continues execution, where it computes $F'_{\mathsf{aid},\mathsf{gid}^*}$ and sets $\mathsf{D}_0[\mathsf{gid}^*] = \mathsf{ctr}_{\mathsf{aid}}$ before incrementing $\mathsf{ctr}_{\mathsf{aid}}$ by one.
- Take any arbitrary $i \in F'_{\mathsf{aid},\mathsf{gid}^*}$. Let k' be the index of the most significant bit where $\mathsf{ctr}_{\mathsf{aid}} = 0 \bmod 2^{k'}$. This means the $(k')^{\text{th}}$ bit is 1 in $\mathsf{ctr}_{\mathsf{aid}}$, as otherwise $\mathsf{ctr}_{\mathsf{aid}} = 0 \bmod 2^{k'+1}$ but 0 in $\mathsf{D}_0[\mathsf{gid}^*]$. By construction, this means $\mathsf{pk}^*_{i,k'}$ has been aggregated into $\mathsf{mpk}_{k'}$.

Next, we consider the auxiliary state $\mathsf{aux}_{\mathsf{aid}}$ after each subsequent non-target-key registration query made by $\mathcal{A}$. Since the only queries that affect $\mathsf{aux}_{\mathsf{aid}}$ are non-target-key registration queries, we ignore the encryption and decryption queries in the following analysis.

- Let $\mathsf{aux}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk}_{\mathsf{aid}})$ and $\mathsf{mpk}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{mpk}_{\mathsf{aid},0}, \ldots, \mathsf{mpk}_{\mathsf{aid},\ell})$ be the auxiliary state and master public key at the time of the key-generation query. The inductive hypothesis is that the invariant holds for $\mathsf{aux}_{\mathsf{aid}}$: namely, $\mathsf{mpk}_{\mathsf{aid},k'}$ was aggregated with $\mathsf{pk}^*_{i,k'}$ for $i \in F'_{\mathsf{aid},\mathsf{gid}^*}$ where k' is the index of the most significant differing bit between $\mathsf{ctr}_{\mathsf{aid}}$ and $\mathsf{D}_0[\mathsf{gid}^*]$.

- In a non-target-key registration query, algorithm $\mathcal{A}$ sends a public key pk to be registered. First, if this key fails the $\mathsf{IsValid}$ check or is associated with an existing user identifier, the challenger aborts and $\mathsf{mpk}_{\mathsf{aid}}$ is unchanged. In this case, the inductive hypothesis continues to hold.

- Suppose pk passes all of the validity checks. Let k'' be the index of the most significant bit for which $\mathsf{ctr}_{\mathsf{aid}} + 1$ (i.e., the value of the counter after pk is registered) differs from $\mathsf{D}_0[\mathsf{gid}^*]$. By Claim 6.5, $k' \le k''$. We now consider two possibilities:

  - Suppose $k' = k''$. This means that the $(k')^{\text{th}}$ bit of ctr and $\mathsf{ctr} + 1$ are the same. Correspondingly, this means that $(\mathsf{ctr}_{\mathsf{aid}} \bmod 2^{k'}) + 1 < 2^{k'}$. Thus, $\mathsf{mpk}'_{\mathsf{aid},k_{\mathsf{old}}} = \mathsf{mpk}_{\mathsf{aid},k_{\mathsf{old}}}$ is *unchanged* by the registration algorithm. Thus our inductive hypothesis is preserved.
  - Suppose $k' < k''$. By Claim 6.5, this means that $\mathsf{ctr}_{\mathsf{aid}} + 1 = 0 \bmod 2^{k''}$. By construction, this means the challenger will increment $\mathsf{ctr}_{\mathsf{aid}}$ and invoke $\mathsf{sRABE.Aggregate}$ on all gid where $\mathsf{D}_0[\mathsf{gid}] \in [\mathsf{ctr}_{\mathsf{aid}} - 2^{k''} + 1, \mathsf{ctr}_{\mathsf{aid}}]$. Since gid was registered in $\mathsf{mpk}_{k'}$, this means $\mathsf{D}_0[\mathsf{gid}^*]$ must have been in the range $[\mathsf{ctr}_{\mathsf{aid}} - 2 \cdot 2^{k'} + 1, \mathsf{ctr}_{\mathsf{aid}} - 2^{k'}]$. Since $k'' \ge k' + 1$, we conclude $\mathsf{ctr}_{\mathsf{aid}} - 2^{k''} + 1 \le \mathsf{ctr}_{\mathsf{aid}} - 2 \cdot 2^{k'} + 1$, and so $\mathsf{D}[\mathsf{gid}^*] \in [\mathsf{ctr}_{\mathsf{aid}} - 2^{k''} + 1, \mathsf{ctr}_{\mathsf{aid}}]$, and as such, will be aggregated in the updated $\mathsf{mpk}_{k''}$. Once more, the induction hypothesis holds.

The above argument shows that if the invariant holds at the beginning of a non-target-key registration query, then it continues to hold after the query. Lemma 6.7 now follows by induction. □

To complete the proof, we now argue if an adversary causes this scheme to output 1 in the multi-authority registered ABE correctness game, then it has found some set of inputs which violates correctness of the underlying slotted multi-authority registered ABE correctness. This follows from the following observations:

- First, the set $\bigcap_{\mathsf{aid} \in S} F'_{\mathsf{aid},\mathsf{gid}^*}$ is nonempty (i.e., there exists an index i^* where $i^* \in \bigcap_{\mathsf{aid} \in S} F'_{\mathsf{aid},\mathsf{gid}^*}$). To see this, recall that the access policy contains at most P authorities. Each authority can register up to L user identifiers. Thus, the maximum number of user identifiers that can be registered to an authority appearing in the challenge ciphertext is $L \cdot P$. Since $(X, \mathcal{F})$ is $(L \cdot P)$-cover-free, there exists an index $i^* \in F_{\mathsf{gid}^*}$ and $i^* \notin F_{\mathsf{gid}}$ for all other user identifiers gid registered with authorities $\mathsf{aid} \in S$. By construction, this means $i^* \in F'_{\mathsf{aid},\mathsf{gid}^*}$ for all $\mathsf{aid} \in S$.

- If the attribute set S satisfies the policy φ, then the attribute set $\{(\mathsf{aid}, k_{\mathsf{aid}})\}_{\mathsf{aid} \in S}$ satisfies policy φ'. This also follows by construction. Namely, each clause $\left(\bigvee_{k \in [0,\ell]} (\mathsf{aid}, k)\right)$ in φ' is fulfilled by any (aid, k), and in particular, by $(\mathsf{aid}, k_{\mathsf{aid}})$. Thus, φ' is satisfied as long as φ is satisfied by S.

- For all $\mathsf{aid} \in S$, the target identifier gid^* is registered with master public key $\mathsf{mpk}_{\mathsf{aid},k_{\mathsf{aid}}}$. Since the correctness game did not abort in the decryption step, for all $\mathsf{aid} \in S$, it holds that $d_{\mathsf{aid}} \neq \perp$. Since this is only updated in target-key queries, the adversary must have submitted a valid target-key registration query to aid. This property now follows from Lemma 6.7.

- The helper decryption keys in $\mathsf{D}_2[\mathsf{gid}^*, k_{\mathsf{aid}}]$ correspond to helper decryption keys of $\mathsf{mpk}_{\mathsf{aid},k_{\mathsf{aid}}}$. Furthermore, $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}^*,k_{\mathsf{aid}}} \neq \perp$. Observe that $\mathsf{D}_2[\mathsf{gid}^*, k_{\mathsf{aid}}]$ is set when $\mathsf{mpk}_{\mathsf{aid},k_{\mathsf{aid}}}$ is aggregated. In addition, it is *never* updated, as subsequent calls to update $\mathsf{mpk}_{\mathsf{aid},k_{\mathsf{aid}}}$ will be on disjoint intervals user identifiers. Next, observe that since the challenger outputs $b = 1$, it must be the case that $\mathsf{IsReady}(e_{\mathsf{aid}}, d_{\mathsf{aid}}, h_{\mathsf{aid}}) = 1$ for all $\mathsf{aid} \in S$. This means $h \ge 2^{k_{d_{\mathsf{aid}},h_{\mathsf{aid}}}} = 2^{k_{\mathsf{aid}}}$, and thus, was updated after $\mathsf{D}_2[\mathsf{gid}^*, k_{\mathsf{aid}}]$ was set.

Taken together, we can see that for all $\mathsf{aid} \in S$, the master public key $\mathsf{mpk}_{\mathsf{aid},k_{\mathsf{aid}}}$ and the associated helper decryption keys $\mathsf{hsk}_{\mathsf{aid},i^*,k_{\mathsf{aid}}}$ are generated by invoking $\mathsf{Aggregate}$ with input $(\mathsf{gid}^*, \mathsf{pk}_{\mathsf{aid},i^*,k_{\mathsf{aid}}})$ (and taking the helper decryption keys for slot i^*). Furthermore, $\mathsf{pk}_{\mathsf{aid},i^*,k_{\mathsf{aid}}}$ is a valid key (as otherwise RegPK does not aggregate it). Moreover, sRABE.Decrypt was invoked on a set of authorities that satisfy the access policy, so if the correctness experiment returned 1, it must be the case that sRABE.Decrypt returned a message $m' \neq m$. This violates correctness of the underlying slotted scheme. □

Theorem 6.8 (Compactness)**.** *Suppose* Π_{sRABE} *is compact. Then Construction 6.4 is compact.*

Proof. Observe that the master public key mpk simply consists of an ℓ-bit counter indicating the current number of registered users along with $\ell + 1$ master public keys $\mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell$ for the underlying slotted scheme. Since each mpk_i is a public key for a slotted scheme with $|X|$ slots supporting $P \cdot \ell$ authorities, the length of each mpk_i is bounded by $\mathrm{poly}(\lambda, \log |X|, P \cdot \ell)$ by compactness of Π_{sRABE}. Since $|X|, P \in \mathrm{poly}(\lambda)$, $\ell = \log L$, the overall size of each mpk is bounded by $\mathrm{poly}(\lambda, \log L)$. The compactness of the global encryption parameters follows similarly from the compactness of the underlying slotted scheme. □

Theorem 6.9 (Polylogarithmic Update Efficiency)**.** *The* $\mathsf{IsReady}$ *predicate in Eq. (4.1) satisfies the polylogarithmically-efficiency requirements from Remark 4.4.*

Proof. We consider each requirement separately:

- Observe that if $e \geq d$, then $k_{e,d}$ is a bit which $e = 1$. Thus $2^{k_{e,d}} \leq e$, so if $h \geq e$, $\mathsf{IsReady}$ is satisfied.
- Consider the intervals $[1, 1], [2, 3], \ldots, [2^{i-1}, 2^i - 1]$ of successive powers of 2. First, we can see $[L] = \bigcup_{i \in [\lceil \log L+1 \rceil]} [2^{i-1}, 2^i - 1]$. Fix some $h \in [2^{i-1}, 2^i - 1]$. Since $e \leq 2^i - 1$, the most significant bit of e which has value 1 must be at most bit $i - 1$. Thus, $k_{e,d} \leq i - 1$, and so $2^{k_{e,d}} \leq 2^{i-1} \leq h$. We conclude that $\mathsf{IsReady}(e, d, h) = 1$. □

Theorem 6.10 (Security)**.** *Suppose* Π_{sRABE} *is secure. Then Construction 6.4 is secure.*

Proof. Suppose there is an adversary $\mathcal{A}$ that breaks security of the multi-authority registered ABE scheme. We use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ for the underlying slotted scheme:

- **Setup phase:** Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on security parameter 1^λ. Algorithm $\mathcal{A}$ outputs the number of slots 1^L and the maximum policy size 1^P. Algorithm $\mathcal{B}$ then proceeds as follows:
  - As in the construction, we assume without loss of generality that $L = 2^\ell$ is a power of two.
  - Algorithm $\mathcal{B}$ instantiates an $(L \cdot P)$-cover free family of sets $(X, \mathcal{F})$ where $|\mathcal{F}| \geq 2^\lambda$ exactly as in Construction 6.4. Let $f : \mathcal{GID} \to \mathcal{F}$ be the associated injective mapping.
  - Algorithm $\mathcal{B}$ initializes dictionaries D_0 to map authorities aid to their state and master public key, D_1 to map public keys of the underlying slotted scheme pk_i to their challenger-assigned counter value ctr, D_2 to map (aid, gid) pairs to the associated public keys that were registered, and D_3 to map master public keys mpk to the set of public keys and user identifiers with which they were associated.
  - Algorithm $\mathcal{B}$ defines RegPK', which behaves exactly as RegPK. However, after each invocation of
  $$(\mathsf{mpk}, \{\mathsf{hsk}\}) \leftarrow \mathsf{sRABE.Aggregate}(\mathsf{sRABE.gpp}, (\mathsf{gid}_1, \mathsf{pk}_1), \ldots, (\mathsf{gid}_{|X|}, \mathsf{pk}_{|X|})),$$
  it sets $\mathsf{D}_3[\mathsf{mpk}] = \big((\mathsf{D}_1[\mathsf{pk}_1], \mathsf{gid}_1, \mathsf{pk}_1), \ldots, (\mathsf{D}_1[\mathsf{pk}_{|X|}], \mathsf{gid}_{|X|}, \mathsf{pk}_{|X|})\big)$.
  - Algorithm $\mathcal{B}$ sends the slot count $1^{|X|}$ and the maximum policy size $1^{P \cdot (\ell+1)}$ to the challenger and receives global public parameters sRABE.gpp and the encryption parameters sRABE.gep.
  - For each slot index $j \in X$ and each $k \in [0, \ell]$, algorithm $\mathcal{B}$ submits a key-generation query on index k to the challenger. The challenger responds with $(\mathsf{ctr}'_{j,k}, \mathsf{pk}'_{j,k})$. Algorithm $\mathcal{B}$ updates $\mathsf{D}_1[\mathsf{pk}'_{j,k}] = \mathsf{ctr}'_{j,k}$
  - Algorithm $\mathcal{B}$ sets
  $$\mathsf{gpp} = (\mathsf{sRABE.gpp}, f, \{(j, k, \mathsf{pk}'_{j,k})\}_{j \in X, k \in [0,\ell]})$$
  and $\mathsf{gep} = \mathsf{sRABE.gep}$. It gives $(\mathsf{gpp}, \mathsf{gep})$ to $\mathcal{A}$.
- **Pre-challenge query phase:** Algorithm $\mathcal{B}$ simulates the queries made by $\mathcal{A}$ as follows:

- **Register honest key query:** When algorithm $\mathcal{A}$ makes a registration query on an authority aid and user identifier gid, algorithm $\mathcal{B}$ first computes F_{gid}. Then, for each $i \in F_{\mathsf{gid}}$ and $k \in [0, \ell]$, it makes a key-generation query on index i to the challenger and receives $(\mathsf{ctr}_{i,k}, \mathsf{pk}_{i,k})$. Algorithm $\mathcal{B}$ sets $\mathsf{D}_1[\mathsf{pk}_{i,k}] = \mathsf{ctr}_{i,k}$ and constructs $\mathsf{pk}_{\mathsf{gid}} = (\mathsf{gid}, \{(i, k, \mathsf{pk}_{i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]})$. Algorithm $\mathcal{B}$ then looks up $(\mathsf{mpk}, \mathsf{aux}) = \mathsf{D}_0[\mathsf{aid}]$ and computes
$$(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}'(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}, \mathsf{pk}_{\mathsf{gid}}).$$
Algorithm $\mathcal{B}$ updates $\mathsf{D}_0[\mathsf{aid}] = \mathsf{mpk}', \mathsf{aux}'$, $\mathsf{D}_2[\mathsf{aid}, \mathsf{gid}] = \mathsf{pk}_{\mathsf{gid}}$ and returns $(\mathsf{mpk}', \mathsf{aux}', \mathsf{pk}_{\mathsf{gid}})$ to $\mathcal{A}$.
- **Register corrupt key query:** When algorithm $\mathcal{A}$ makes a corrupted registration query on authority aid, user identifier gid, and public key $\mathsf{pk}_{\mathsf{gid}}$, algorithm $\mathcal{B}$ first looks up $(\mathsf{mpk}, \mathsf{aux}) = \mathsf{D}_0[\mathsf{aid}]$. Then it computes $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}'(\mathsf{gpp}, \mathsf{aux}, \mathsf{gid}, \mathsf{pk}_{\mathsf{gid}})$. Parsing $\mathsf{pk}_{\mathsf{gid}} = (\mathsf{gid}, \{(i, k, \mathsf{pk}_{i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]})$, algorithm $\mathcal{B}$ sets $\mathsf{D}_1[\mathsf{pk}_{i,k}] = \perp$ for all $i \in F_{\mathsf{gid}}$ and $k \in [0, \ell]$. Finally, algorithm $\mathcal{B}$ updates $\mathsf{D}_0[\mathsf{aid}] = (\mathsf{mpk}', \mathsf{aux}')$ and responds with $(\mathsf{mpk}', \mathsf{aux}', \mathsf{pk}_{\mathsf{gid}})$ to $\mathcal{A}$.
- **Corrupt honest key query:** When algorithm $\mathcal{A}$ makes a corruption query on authority aid and user identifier gid, algorithm $\mathcal{B}$ looks up $\mathsf{pk}_{\mathsf{gid}} = \mathsf{D}_2[\mathsf{aid}, \mathsf{gid}]$, and parses $\mathsf{pk}_{\mathsf{gid}} = (\mathsf{gid}, \{\mathsf{pk}_{i,k}\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]})$. For each $i \in F_{\mathsf{gid}}$ and $k \in [0, \ell]$, algorithm $\mathcal{B}$ makes a key-corruption query on $\mathsf{ctr}_{i,k} = \mathsf{D}_1[\mathsf{pk}_{i,k}]$ to the challenger and receives $\mathsf{sk}_{i,k}$. Algorithm $\mathcal{B}$ responds with $\mathsf{sk}_{\mathsf{gid}} = (\mathsf{gid}, \{(i, k, \mathsf{sk}_{i,k})\}_{i \in F_{\mathsf{gid}}, k \in [0,\ell]})$.

- **Challenge phase:** When algorithm $\mathcal{A}$ makes a challenge query with messages (μ_0^*, μ_1^*) and a policy $(S_{\mathsf{enc}}, \varphi)$, algorithm $\mathcal{B}$ makes a challenge query of its own as follows:
  - First, algorithm $\mathcal{B}$ defines the challenge authority set to be $S'_{\mathsf{enc}} = \{(\mathsf{aid}, k)\}_{\mathsf{aid} \in S_{\mathsf{enc}}, k \in [0,\ell]}$. It defines the challenge policy φ' by replacing each appearance of attribute aid in φ with the clause $\left(\bigvee_{k \in [0,\ell]} (\mathsf{aid}, k)\right)$. Algorithm $\mathcal{B}$ outputs the challenge policy $(S'_{\mathsf{enc}}, \varphi')$ and the messages μ_0^*, μ_1^*.
  - Next, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, algorithm $\mathcal{B}$ looks up $(\mathsf{mpk}_{\mathsf{aid}}, \mathsf{aux}_{\mathsf{aid}}) = \mathsf{D}_0[\mathsf{aid}]$. It parses $\mathsf{mpk}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{mpk}_{\mathsf{aid},0}, \ldots, \mathsf{mpk}_{\mathsf{aid},\ell})$. For each $\mathsf{aid} \in S_{\mathsf{enc}}$ and $k \in [0, \ell]$, algorithm $\mathcal{B}$ specifies the tuple $\mathsf{D}_3[\mathsf{mpk}_{\mathsf{aid},k}]$ as the set of keys associated with authority (aid, k).

  The challenger replies with a ciphertext ct'. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with $\mathsf{ct} = (\mathsf{ct}', \{(\mathsf{aid}, \mathsf{ctr}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}})$.

- **Post-challenge query phase:** Algorithm $\mathcal{B}$ answers the post-challenge queries using the *same* procedure as the pre-challenge queries.
- **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which $\mathcal{B}$ also outputs.

We will observe that if $\mathcal{A}$ is an admissible adversary for the non-slotted multi-authority registered ABE security game, then algorithm $\mathcal{B}$ is an admissible adversary for the slotted security game. Recall that $\mathcal{A}$ is admissible if there does not exist a user identifier gid^* such that the set of corrupted attributes satisfy policy φ. We will show that a non-admissible adversary $\mathcal{B}$ corresponds to a non-admissible $\mathcal{A}$ (which would be a contradiction). Suppose that $\mathcal{B}$ submits a sequence of queries such that there exists gid^* for which the set of corrupted attributes S'_{gid^*} satisfies φ'. Let $S_{\mathsf{gid}^*} = \{\mathsf{aid} \mid (\mathsf{aid}, k) \in S'_{\mathsf{gid}^*}\}$. By construction of $\mathcal{B}$, this can only happen if $\mathcal{A}$ made a registration query on $(\mathsf{gid}^*, \mathsf{aid})$. We consider two cases:

- Suppose adversary $\mathcal{A}$ made an honest-key registration query on $(\mathsf{gid}^*, \mathsf{aid})$. Suppose $\mathcal{B}$ returns key $\mathsf{pk}_{\mathsf{gid}^*} = (\mathsf{gid}^*, \{(i, k, \mathsf{pk}_{i,k})\}_{i \in F_{\mathsf{gid}^*}, k \in [0,\ell]})$. Observe that since each $\mathsf{pk}_{i,k}$ is set to its corresponding counter value, these keys will correspond to uncorrupted keys in $\mathsf{mpk}_{\mathsf{aid}}$ so long as $\mathcal{B}$ does not submit a corruption query on them, which can only occur through $\mathcal{A}$ making a corrupt-key query on $(\mathsf{gid}^*, \mathsf{aid})$. In this case, this would also corrupt $(\mathsf{gid}^*, \mathsf{aid})$ for $\mathcal{B}$.
- Suppose adversary $\mathcal{A}$ made a corrupted-key registration query on $(\mathsf{gid}^*, \mathsf{aid})$. Observe in this case, the key $\mathsf{pk}_{\mathsf{gid}^*} = (\mathsf{gid}^*, \{(i, k, \mathsf{pk}_{i,k})\}_{i \in F_{\mathsf{gid}^*}, k \in [0,\ell]})$ submitted by $\mathcal{A}$ has $\mathsf{D}_1[\mathsf{pk}_{i,k}]$ set to $\perp$. As such, when $\mathcal{B}$ constructs the challenge query, these slots will have counter value set to $\perp$, meaning they are also be corrupted in $\mathcal{B}$'s challenge as well.

Observe that in either case, aid $\in S_{\mathsf{gid}^*}$. Finally, by construction of φ', if S'_{gid^*} satisfies φ', then S_{gid^*} satisfies φ, in which case adversary $\mathcal{A}$ is also not admissible, as required. Finally, algorithm $\mathcal{B}$ perfectly simulates an execution of the registered ABE security game for $\mathcal{A}$, so taken together, we can conclude that assuming security of the underlying Π_{sRABE} scheme, the above construction is a secure multi-authority registered ABE scheme. □

7 Multi-Authority Registered ABE from Obfuscation

In this section, we show how to construct an adaptively-secure multi-authority registered ABE supporting monotone Boolean circuit policies for an *unbounded* number of users using indistinguishability obfuscation ($i\mathcal{O}$) [BGI+01], function-binding hash functions [FWW23], computational secret sharing (CSS) for monotone Boolean circuits [Yao89, VNS+03], and non-committing encryption [CFGN96]. We note these primitives are all implied by $i\mathcal{O}$ together with (leveled) homomorphic encryption.

7.1 Building Blocks

We start by recalling the definitions of the building blocks we use in our construction.

Definition 7.1 (Indistinguishability Obfuscation [BGI+01, GGH+13])**.** Let $C = \{C_\lambda\}_{\lambda\in\mathbb{N}}$ be a family of polynomial-size circuits. An indistinguishability obfuscator $i\mathcal{O}$ is an efficient algorithm that takes as input the security parameter λ, a circuit $C \in C_\lambda$ and outputs a circuit C'. An $i\mathcal{O}$ scheme should satisfy the following properties:

- **Functionality-preserving:** For all security parameters $\lambda \in \mathbb{N}$, all $C \in C_\lambda$, and all inputs x, we have that $C'(x) = C(x)$ where $C' \leftarrow i\mathcal{O}(1^\lambda, C)$.
- **Security:** For all efficient (possibly non-uniform) adversaries $\mathcal{A} = (\mathsf{Samp}, \mathcal{A}')$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the following holds: if for all security parameters $\lambda \in \mathbb{N}$,

  $$\Pr[\forall x, C_0(x) = C_1(x) : (C_0, C_1, \mathsf{st}) \leftarrow \mathsf{Samp}(1^\lambda)] = 1 - \mathsf{negl}(\lambda),$$

  then

  $$\left|\Pr[\mathcal{A}'(\mathsf{st}, i\mathcal{O}(1^\lambda, C_0)) = 1] - \Pr[\mathcal{A}'(\mathsf{st}, i\mathcal{O}(1^\lambda, C_1)) = 1]\right| = \mathsf{negl}(\lambda),$$

  where $(C_0, C_1, \mathsf{st}) \leftarrow \mathsf{Samp}(1^\lambda)$.

Definition 7.2 (Function Binding Hash Function [FWW23])**.** Let λ be a security parameter. A function binding (FB) hash function with block length $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, output length $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$, function class $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda\in\mathbb{N}}$ where $\mathcal{F}_\lambda$ consists of functions $f : (\{0,1\}^{\ell_{\mathsf{blk}}})^* \to \{0,1\}^{\ell_{\mathsf{out}}}$, hash length $\ell_{\mathsf{hash}} = \ell_{\mathsf{hash}}(\lambda)$, and opening length $\ell_{\mathsf{open}} = \ell_{\mathsf{open}}(\lambda)$ is a tuple of efficient algorithms $\Pi_{\mathsf{FB}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{Setup}(1^\lambda, N) \to \mathsf{hk}$: On input the security parameter λ, and the bound on the number of blocks $N \le 2^\lambda$ (in binary), the setup algorithm outputs a hash key hk.
- $\mathsf{SetupBinding}(1^\lambda, N, f) \to \mathsf{hk}$: On input the security parameter λ, a bound on the number of blocks $N \le 2^\lambda$ (in binary), and a function $f \in \mathcal{F}_\lambda$ the setup algorithm outputs a hash key hk.
- $\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \to h$: On input the hash key hk and a message $\mathbf{x} = (x_1, \dots, x_n) \in (\{0,1\}^{\ell_{\mathsf{blk}}})^n$ for some $n \le N$, the hash algorithm *deterministically* outputs a hash $h \in \{0,1\}^{\ell_{\mathsf{hash}}}$.
- $\mathsf{Open}(\mathsf{hk}, \mathbf{x}, i) \to \pi_i$: On input the hash key hk, an input $\mathbf{x} \in (\{0,1\}^{\ell_{\mathsf{blk}}})^n$, and an index $i \in [L]$, the open algorithm outputs an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$.
- $\mathsf{Verify}(\mathsf{hk}, h, S, \{(i, x_i, \pi_i)\}_{i\in S}) \to \{0,1\}$: On input the hash key hk, a hash value $h \in \{0,1\}^{\ell_{\mathsf{hash}}}$, a set of indices $S \subseteq [N]$, and the values $x_i \in \{0,1\}^{\ell_{\mathsf{blk}}}$, and openings $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$ associated with said indices, the verification algorithm outputs a bit $b \in \{0,1\}$ indicating whether it accepts or rejects.

We require the following properties:

- **Correctness**: For all security parameters $\lambda \in \mathbb{N}$, all block sizes $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, all integers $N \in \mathbb{N}$, all index sets $S \subseteq [n]$ for any $n \leq N$, and any $\mathbf{x} \in (\{0,1\}^{\ell_{\mathsf{blk}}})^N$,

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, h, S, \{(i, x_i, \pi_i)\}_{i \in S}) = 1 : \begin{array}{r} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, N) \\ h \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \\ \pi_i \leftarrow \mathsf{Open}(\mathsf{hk}, \mathbf{x}, i) \end{array}\right] = 1.$$

- **Computational function hiding:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, define the function hiding game as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the number of blocks $N \in \mathbb{N}$ (in *binary*) and a function $f \in \mathcal{F}_\lambda$.
  2. The challenger samples $\mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, N)$, $\mathsf{hk}_1 \leftarrow \mathsf{SetupBinding}(1^\lambda, N, f)$ and gives hk_b to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

  We require that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

  in the function hiding game.

- **Statistically function binding:** For a security parameter λ, we say that a hash key hk is (statistically) function binding on a function $f \in \mathcal{F}_\lambda$ if for all inputs $\mathbf{x} \in (\{0,1\}^{\ell_{\mathsf{blk}}})^n$ and setting $h = \mathsf{Hash}(\mathsf{hk}, \mathbf{x})$, there does not exist a set $S \subseteq N$ and $\{(x_i^*, \pi_i^*)\}_{i \in S}$ where

  - $\mathsf{Verify}(\mathsf{hk}, h, S, \{(i, x_i^*, \pi_i^*)\}_{i \in S}) = 1$
  - For all $\mathbf{x}'$ such that $x_i' = x_i^*$, $f(\mathbf{x}') \neq f(\mathbf{x})$.

  We say a scheme Π_{FB} is statistically function binding if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $N \leq 2^\lambda$, and all $f \in \mathcal{F}_\lambda$,

$$\Pr[\mathsf{hk} \text{ is function binding for } f : \mathsf{hk} \leftarrow \mathsf{SetupBinding}(1^\lambda, N, f)] = 1 - \mathsf{negl}(\lambda).$$

- **Succinctness:** The hash length ℓ_{hash} and opening length ℓ_{open} are fixed polynomials in the security parameter λ (and independent of N).

In this work, we consider function-binding hash functions for the family of unique-block-selection predicates, which we define below:

Definition 7.3 (Unique Block Selection). Let $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ be an input length parameter and $g \colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ be a predicate. We say a function f_g is a unique-block-selection function if it can be written as

$$f_g(x_1, \ldots, x_n) = \begin{cases} x_i & \text{if } g(x_i) = 1 \text{ and } \forall j \neq i \in [n] : g(x_j) = 0 \\ \perp & \text{otherwise.} \end{cases}$$

We say $\mathcal{F} = \{\mathcal{F}_\lambda\}_{\lambda \in \mathbb{N}}$ is the class of all unique block selection functions with input length $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ and predicate size $s = s(\lambda)$ if $\mathcal{F}_\lambda$ contains all functions $f_g \colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ where the function g can be computed by a Boolean circuit of size s.

In Appendix C, we show how to construct a function-binding hash function for the class of unique-block-selection predicates (where the predicates g can be computed by polynomial-size Boolean circuits) using leveled homomorphic encryption. Our construction follows the same approach as that used to construct somewhere statistically binding hash functions [HW15] and function-binding hash functions for disjuctions of block-wise functions [FWW23].

Definition 7.4 (Non-Committing Encryption [CFGN96])**.** Let λ be a security parameter. A (weak)-non-committing encryption scheme with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ is a tuple of efficient algorithms $\Pi_{\mathsf{NCE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{CTSim}, \mathsf{RSim})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda) \to (\mathsf{pk}, \mathsf{sk})$: On input the security parameter λ, the key-generation algorithm outputs a public key pk and secret key sk.
- $\mathsf{Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: On input a public key pk and a message $m \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m$: On input a secret key sk and a ciphertext sk, the encryption algorithm outputs a message m (which could be a special symbol $\perp$).
- $\mathsf{CTSim}(1^\lambda) \to (\mathsf{pk}, \mathsf{ct}, \mathsf{td})$: On input the security parameter λ, the ciphertext-simulation algorithm outputs a simulated public key pk, a simulated ciphertext ct, along with a trapdoor td.
- $\mathsf{RSim}(\mathsf{td}, m) \to r$: On input a trapdoor td and a message m, the randomness-simulation algorithm outputs a string $r \in \{0,1\}^\rho$. Here, $\rho = \rho(\lambda)$ is the number of bits of randomness the KeyGen algorithm takes.

We require Π_{NCE} satisfy the following properties:

- **Correctness:** For all security parameters λ, all messages $m \in \mathcal{M}$, we have that

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) = m : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m) \end{array}\right] = 1.$$

- **Simulatability:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, we define the simulation security experiment as follows:

  1. If $b = 0$, the challenger samples randomness $r \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and computes $(\mathsf{pk}, \mathsf{sk}) = \mathsf{KeyGen}(1^\lambda; r)$. If $b = 1$, it samples $(\mathsf{pk}, \mathsf{ct}, \mathsf{td}) \leftarrow \mathsf{CTSim}(1^\lambda)$. It sends pk to $\mathcal{A}$.
  2. Algorithm $\mathcal{A}$ outputs a message $m \in \mathcal{M}$.
  3. If $b = 0$, the challenger computes $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m)$. If $b = 1$, it computes $r \leftarrow \mathsf{RSim}(\mathsf{td}, m)$. The challenger replies to $\mathcal{A}$ with (ct, r).
  4. Algorithm $\mathcal{A}$ responds with a bit b', which is the output of the experiment.

  We require that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

  in the simulatability game.

Weak non-committing encryption can be constructed from any simulatable public-key encryption scheme [DN00], which in turn can be constructed from most number theoretic assumptions, including the decisional Diffie-Hellman (DDH) assumption over pairing free groups [DN00] or the learning with errors (LWE) assumption [GPV08].

Definition 7.5 (Computational Secret Sharing for Monotone Circuits [Yao89, VNS+03])**.** Let λ be a security parameter. A computational secret sharing (CSS) scheme for monotone Boolean circuits with secret space $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda \in \mathbb{N}}$ is a pair of efficient algorithms $\Pi_{\mathsf{CSS}} = (\mathsf{Share}, \mathsf{Recon})$ with the following syntax:

- $\mathsf{Share}(1^\lambda, C, s) \to (\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$: On input the security parameter λ, a monotone Boolean circuit $C \colon \{0,1\}^n \to \{0,1\}$, and a secret $s \in \mathcal{S}_\lambda$, the share algorithm outputs a set of shares $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n)$, where sh_i is the "share" associated with the ith input to C.
- $\mathsf{Recon}(C, x, \{(i, \mathsf{sh}_i)\}_{i : x_i = 1}) \to s$: On input a Boolean circuit $C \colon \{0,1\}^n \to \{0,1\}$, an input $x \in \{0,1\}^n$, and a set of shares $\{(i, \mathsf{sh}_i)\}_{i : x_i = 1}$, the reconstruction algorithm outputs a secret s.

We require Π_{CSS} satisfy the following properties:

- **Correctness:** For all security parameters $\lambda \in \mathbb{N}$, secrets $s \in \mathcal{S}_\lambda$, and all monotone Boolean circuits $C \colon \{0,1\}^n \to \{0,1\}$, and inputs $x \in \{0,1\}^n$ where $C(x) = 1$, we have that

$$\Pr[\mathsf{Recon}(C, x, \{(i, \mathsf{sh}_i)\}_{i : x_i = 1}) = s : (\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, C, s)] = 1$$

- **Adaptive security:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, define the adaptive secret sharing security game as follows:

  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ chooses a monotone Boolean circuit $C \colon \{0,1\}^n \to \{0,1\}$ and a pair of secrets $s_0, s_1 \in \mathcal{S}_\lambda$.
  2. The challenger samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_n) \leftarrow \mathsf{Share}(1^\lambda, C, s_b)$.
  3. The adversary can now adaptively request shares from the challenger. Namely, the adversary can send an index $i \in [n]$ to the challenger and the challenger replies with the associated share sh_i.
  4. At the end of the game, the adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  Let $x \in \{0,1\}^n$ be the bit-vector where $x_i = 1$ for all indices $i \in [n]$ on which the adversary $\mathcal{A}$ requested a share. We say the adversary $\mathcal{A}$ is admissible if $C(x) = 0$. Then, the secret sharing scheme satisfies adaptive security if for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda)$$

  in the adaptive security game.

The works of [Yao89, VNS+03] showed how to construct *statically* secure computational secret sharing schemes from public-key encryption. In the static security setting, the adversary has to commit to the shares it wants to see at the beginning of the experiment. A recent work of Lu and Waters [LW25] show how to generically lift a statically-secure computational secret sharing scheme into an adaptively-secure scheme (with an overhead that is quadratic in the number of parties).

7.2 Slotted Multi-Authority Registered ABE for Monotone Boolean Circuits

Similar to our pairing-based construction from Section 5, we first show how to construct a "slotted" version of our multi-authority registered ABE scheme using indistinguishability obfuscation (and the additional primitives described above). Then, we can apply a generic transformation (e.g., Construction 6.4) to obtain the general notion (Definition 4.1). Note that unlike our pairing-based construction (Construction 5.1), our obfuscation-based scheme does *not* require that users register to the same slot across different authorities. In fact, the key-generation procedure in our obfuscation-based scheme will not require keys to be generated with respect to a slot. This means we can avoid the need for cover-free sets when lifting to the unslotted scheme. This is important to be able to support an unbounded number of users (recall that the use of cover-free sets in Construction 5.1 imposed a bound on the maximum number of parties). In Appendix B, we describe a slimmed-down version of the slotted-to-unslotted transformation that does not rely on cover-free sets (under the assumption that the underlying slotted scheme does not require users to have a common slot when registering to different authorities).

Construction 7.6 (Slotted Multi-Authority Attribute-Based Registration-Based Encryption.)**.** Let λ be a security parameter. Let $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda \in \mathbb{N}}$ be a universe of authority identifiers. Let $\mathcal{M}_\lambda = \{0,1\}^\lambda$ and $\mathcal{GID}_\lambda = \{0,1\}^\lambda$. Let $\Phi = \{\Phi_\lambda\}_{\lambda \in \mathbb{N}}$ be a family of access policies where Φ_λ consists of access policies (S_{enc}, C) where $C \colon 2^{|S_{\text{enc}}|} \to \{0,1\}$ is an arbitrary monotone Boolean circuit. Our construction relies on the following primitives.

- Let iO be a secure indistinguishability obfuscation scheme for general Boolean circuits.
- Let $\Pi_{\text{CSS}} = (\text{CSS.Share}, \text{CSS.Recon})$ be a computational secret sharing scheme with secret space $\mathcal{S} = \{\mathcal{S}_\lambda\}_{\lambda \in \mathbb{N}}$ where $\mathcal{S}_\lambda = \{0,1\}$ for all $\lambda \in \mathbb{N}$. We write $|\mathsf{S}|$ to denote the share size for Π_{CSS}. Note that $|\mathsf{S}|$ is a function of the security parameter λ.

- Let $\Pi_{\mathsf{NCE}} = (\mathsf{NCE.KeyGen}, \mathsf{NCE.Encrypt}, \mathsf{NCE.Decrypt}, \mathsf{NCE.CTSim}, \mathsf{NCE.RSim})$ be a non-committing encryption scheme with message space $\mathcal{M}_{\mathsf{NCE}} = \{0,1\}^{|S|}$. Let ρ be the randomness complexity of NCE.KeyGen and $|\mathsf{pk}_{\mathsf{NCE}}|$ be the size of the public keys for Π_{NCE}. Note that $\rho = \rho(\lambda)$ and $|\mathsf{pk}_{\mathsf{NCE}}|$ are both functions of the security parameter λ.
- For a parameter $R \in \mathbb{N}$, let $\mathcal{H}_R = \{H_R \colon \{0,1\}^\lambda \to [R]\}$ be a family of universal hash families. We assume that the functions in $\{\mathcal{H}_{R'}\}_{R' \le R}$ can be computed by a Boolean circuit of size $s_{\mathcal{H}}(R)$.
- Let $\Pi_{\mathsf{FB}} = (\mathsf{FB.Setup}, \mathsf{FB.SetupBinding}, \mathsf{FB.Hash}, \mathsf{FB.Open}, \mathsf{FB.Verify})$ be a a function-binding hash for the family of unique-block-selection predicates (Definition 7.3) with block length $\ell_{\mathsf{blk}} = \lambda + |\mathsf{pk}_{\mathsf{NCE}}|$ and predicate size $s = s_{\mathcal{H}}(2^\lambda) + 2\lambda$.

We construct a slotted multi-authority registered ABE scheme $\Pi_{\mathsf{sMA\text{-}RABE}} = (\mathsf{GlobalSetup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with authority identifiers $\mathcal{AU}$, user identifier space $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda \in \mathbb{N}}$, message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$, and policy family Φ as follows:

- $\mathsf{GlobalSetup}(1^\lambda, L)$: On input the security parameter λ and the number of slots L (in binary), the setup algorithm runs $\mathsf{hk} \leftarrow \mathsf{FB.Setup}(1^\lambda, L)$ and outputs the global public parameters $\mathsf{gpp} = \mathsf{hk}$.[5]
- $\mathsf{KeyGen}(\mathsf{gpp})$: On input the global public parameters gpp, the key-generation algorithm samples $r \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and $(\mathsf{pk}_{\mathsf{NCE}}, \mathsf{sk}_{\mathsf{NCE}}) \leftarrow \mathsf{NCE.KeyGen}(1^\lambda; r)$. It outputs the public key $\mathsf{pk} = \mathsf{pk}_{\mathsf{NCE}}$ and secret key $\mathsf{sk} = r$.
- $\mathsf{IsValid}(\mathsf{gpp}, \mathsf{pk})$: On input the global public parameters gpp, and a public key pk, the validation algorithm outputs 1 as long as $|\mathsf{pk}| = |\mathsf{pk}_{\mathsf{NCE}}|$.
- $\mathsf{Aggregate}(\mathsf{gpp}, (\mathsf{gid}_1, \mathsf{pk}_1), \ldots, (\mathsf{gid}_L, \mathsf{pk}_L))$: On input the global public parameters $\mathsf{gpp} = \mathsf{hk}$ and L user identifiers gid_i and associated public keys pk_i, the aggregation algorithm computes the master public key
$$\mathsf{mpk} = h_{\mathsf{aid}} = \mathsf{FB.Hash}(\mathsf{hk}, ((\mathsf{gid}_1, \mathsf{pk}_1), \ldots, (\mathsf{gid}_L, \mathsf{pk}_L))).$$
Here we treat each pair $(\mathsf{gid}_i, \mathsf{pk}_i)$ as a binary string of length $\{0,1\}^{\lambda + |\mathsf{pk}_{\mathsf{NCE}}|}$. Then, for each user $i \in [L]$, the aggregation algorithm computes openings
$$\pi_i \leftarrow \mathsf{FB.Open}\big(\mathsf{hk}, \big((\mathsf{gid}_1, \mathsf{pk}_1), \ldots, (\mathsf{gid}_L, \mathsf{pk}_L)\big), i\big).$$
It sets the helper secret key to $\mathsf{hsk}_i = (i, \mathsf{gid}_i, \mathsf{pk}_i, \pi_i)$. Finally, it outputs mpk and hsk_i for all $i \in [L]$.
- $\mathsf{Encrypt}(\mathsf{gpp}, (S_{\mathsf{enc}}, C), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu)$: On input global public parameters $\mathsf{gpp} = \mathsf{hk}$, a set of authorities S_{enc}, a monotone Boolean circuit $C \colon 2^{|S_{\mathsf{enc}}|} \to \{0,1\}$, the public key $\mathsf{mpk}_{\mathsf{aid}}$ for each $\mathsf{aid} \in S_{\mathsf{enc}}$, and a message $\mu \in \{0,1\}^\lambda$, the encryption algorithm defines the following program:

[5]Since the size of the global public parameters scales polylogarithmically with the number of users, there is no need to distinguish between the global public parameters gpp and the encryption parameters gep. For simplicity, when describing our scheme, we can always take gep = gpp. Thus, when describing (and analyzing) the scheme, we always substitute gep with gpp.

Constants: Hash key hk, set S_{enc}, master public keys $\{(\text{aid}, \text{mpk}_{\text{aid}})\}_{\text{aid}\in S_{\text{enc}}}$, a monotone Boolean circuit $C\colon 2^{S_{\text{enc}}} \to \{0,1\}$, and a message $\mu \in \{0,1\}^\lambda$

Inputs: A decryption set $S \subseteq S_{\text{enc}}$ and for each aid $\in S$, a secret key sk_{aid} and a helper decryption key hsk_{aid}.

1. For each aid $\in S$, let $\text{mpk}_{\text{aid}} = h_{\text{aid}}$, $\text{sk}_{\text{aid}} = r_{\text{aid}}$, and $\text{hsk}_{\text{aid}} = (i_{\text{aid}}, \text{gid}_{\text{aid}}, \text{pk}_{\text{aid}}, \pi_{\text{aid}})$.
2. Output μ if the following checks pass:
   - $C(S) = 1$.
   - There exists $\text{gid}^* = \text{gid}_{\text{aid}}$ for all aid $\in S$.
   - $\text{FB.Verify}(\text{hk}, h_{\text{aid}}, S_{\text{ind}}, \{(i_{\text{aid}}, (\text{gid}^*, \text{pk}_{\text{aid}}), \pi_{\text{aid}})\}_{\text{aid}\in S}) = 1$ where $S_{\text{ind}} = \{i_{\text{aid}} \mid \text{aid} \in S\}$.
   - For each aid $\in S$, $\text{pk}_{\text{aid}} = \text{pk}'_{\text{aid}}$ where let $(\text{pk}'_{\text{aid}}, \text{sk}'_{\text{aid}}) = \text{NCE.KeyGen}(1^\lambda; r_{\text{aid}})$.

   If any of the checks fail, output $\perp$.

Figure 1: Program $\text{Embed}[\text{hk}, S_{\text{enc}}, \{(\text{aid}, \text{mpk}_{\text{aid}})\}_{\text{aid}\in S_{\text{enc}}}, C, \mu]$.

We assume that the circuit $\text{Embed}[\text{gpp}, S_{\text{enc}}, \{(\text{aid}, \text{mpk}_{\text{aid}})\}_{\text{aid}\in S_{\text{enc}}}, C, \mu]$ is padded to the maximum size of any program that appears in the proof of Theorem 7.11. The encryption algorithm then computes the obfuscated program $C' \leftarrow i\mathcal{O}(1^\lambda, \text{Embed}[\text{hk}, S_{\text{enc}}, \{(\text{aid}, \text{mpk}_{\text{aid}})\}_{\text{aid}\in S_{\text{enc}}}, C, \mu])$ and outputs $\text{ct} = C'$.

- $\text{Decrypt}(S, \{(\text{aid}, \text{sk}_{\text{aid}}, \text{hsk}_{\text{aid}})\}_{\text{aid}\in S}, \text{ct})$: On input the attribute set S, the secret keys sk_{aid} and helper decryption keys hsk_{aid} for attributes aid $\in S$, and a ciphertext $\text{ct} = C'$, the decryption algorithm outputs $\mu = C'(S, \{(\text{aid}, \text{sk}_{\text{aid}}, \text{hsk}_{\text{aid}})\}_{\text{aid}\in S})$.

Theorem 7.7 (Completeness)**.** *Construction 7.6 satisfies completeness.*

Proof. Since the IsValid function outputs 1 on any public key of length $|\text{pk}_{\text{NCE}}|$, and by definition $|\text{pk}_{\text{NCE}}|$ is the size of NCE public keys, IsValid outputs 1 on any pk in the support of KeyGen. □

Theorem 7.8 (Correctness)**.** *If $i\mathcal{O}$ is correct and Π_{FB} is complete, then Construction 7.6 is correct.*

Proof. Take any $\lambda, L \in \mathbb{N}$, any decryption index $j^* \in [L]$, and any gpp = hk in the support of GlobalSetup. Take any policy (S_{enc}, C) and decryption set $S^* \subseteq S_{\text{enc}}$ where $C(S^*) = 1$. Take any target user identifier $\text{gid}^* \in \mathcal{GID}_\lambda$ and any collection of key-pairs $(\text{pk}_{\text{aid},j^*}, \text{sk}_{\text{aid},j^*})$ in the support of $\text{KeyGen}(1^\lambda)$ for aid $\in S^*$. Take any collection of user identifiers $\{(j, \text{aid}, \text{gid}_{j,\text{aid}})\}_{(j,\text{aid})\in[L]\setminus\{j^*\}\times S_{\text{enc}}}$ and for each aid $\in S_{\text{enc}}$, let

$$(\text{mpk}_{\text{aid}}, \text{hsk}_{1,\text{aid}}, \ldots, \text{hsk}_{L,\text{aid}}) = \text{Aggregate}(\text{gpp}, \{(j, \text{gid}_{j,\text{aid}}, \text{pk}_{j,\text{aid}})\}_{j\in[L]}),$$

where $\text{gid}_{j^*,\text{aid}} = \text{gid}^*$. Take any message μ and let $\text{ct} \leftarrow \text{Encrypt}(\text{gep}, (S_{\text{enc}}, C), \{(\text{aid}, \text{mpk}_{\text{aid}})\}_{\text{aid}\in S_{\text{enc}}}, \mu)$. Consider now $\text{Decrypt}(S^*, \{(\text{aid}, \text{sk}_{j^*,\text{aid}}, \text{hsk}_{j^*,\text{aid}})\}_{\text{aid}\in S^*}, \text{ct})$. By definition, the ciphertext ct is an obfuscation of the program $\text{Embed}[\text{hk}, S_{\text{enc}}, \{(\text{aid}, \text{mpk}_{\text{aid}})\}_{\text{aid}\in S_{\text{enc}}}, C, \mu]$. By correctness of the obfuscation scheme, it suffices to show that

$$\text{Embed}[\text{hk}, S_{\text{enc}}, \{(\text{aid}, \text{mpk}_{\text{aid}})\}_{\text{aid}\in S_{\text{enc}}}, C, \mu](S^*, \{(\text{aid}, \text{sk}_{j^*,\text{aid}}, \text{hsk}_{j^*,\text{aid}})\}_{\text{aid}\in S^*}) = \mu.$$

We check the conditions:

- By assumption, $C(S^*) = 1$, so the first condition is satisfied.
- By construction, the user identifier associated with $\text{hsk}_{j^*,\text{aid}}$ is $\text{gid}_{j^*,\text{aid}} = \text{gid}^*$, so the second condition is satisfied.

- By definition of Aggregate, we have

$$\mathsf{mpk}_{\mathsf{aid}} = h_{\mathsf{aid}} = \mathsf{FB.Hash}(\mathsf{hk}, ((\mathsf{gid}_{1,\mathsf{aid}}, \mathsf{pk}_{1,\mathsf{aid}}), \ldots, (\mathsf{gid}_{L,\mathsf{aid}}, \mathsf{pk}_{L,\mathsf{aid}})))$$

  and $\mathsf{hsk}_{i,\mathsf{aid}} = (i, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}_{i,\mathsf{aid}}, \pi_{i,\mathsf{aid}})$ where

$$\pi_{i,\mathsf{aid}} \leftarrow \mathsf{FB.Open}(\mathsf{hk}, ((\mathsf{gid}_{1,\mathsf{aid}}, \mathsf{pk}_{1,\mathsf{aid}}), \ldots, (\mathsf{gid}_{L,\mathsf{aid}}, \mathsf{pk}_{L,\mathsf{aid}})), i).$$

  Thus, for all $\mathsf{aid} \in S^*$, $\mathsf{FB.Verify}(\mathsf{hk}, h_{\mathsf{aid}}, S^*_{\mathsf{ind}}, \{(i_{\mathsf{aid}}, (\mathsf{gid}^*, \mathsf{pk}_{j^*\mathsf{aid}}), \pi_{j^*,\mathsf{aid}})\}_{\mathsf{aid} \in S^*_{\mathsf{ind}}}) = 1$ by perfect correctness of Π_{FB}. Thus, the third condition holds.

- Finally, by construction of $\mathsf{KeyGen}(1^\lambda)$, we have that $(\mathsf{pk}_{j^*,\mathsf{aid}}, \mathsf{sk}) = \mathsf{NCE.KeyGen}(1^\lambda; \mathsf{sk}_{j^*,\mathsf{aid}})$, and so the last condition holds.

Since all of the conditions are satisfied, Embed outputs μ, as required. □

Theorem 7.9 (Compactness)**.** *If Π_{FB} is succinct, then Construction 7.6 is compact.*

Proof. In Construction 7.6, the master public key is a function-binding hash on the input $((\mathsf{gid}_1, \mathsf{pk}_1), \ldots (\mathsf{gid}_L, \mathsf{pk}_L))$, and the global parameters is a hash key. The function-binding hash function needs to support the class of unique-block-selection predicates with predicate size $s = s_{\mathcal{H}}(2^\lambda)$. If we instantiate the universal hash family $\mathcal{H}_R$ with the classic Carter-Wegman construction [CW79] (i.e., evaluate a random linear function modulo a prime $R' \geq 2^\lambda$ and then reduce modulo R where). In this case, the size of the circuit $s_{\mathcal{H}}(R)$ is $\mathsf{poly}(\lambda, \log R)$. Correspondingly, $s(2^\lambda) = \mathsf{poly}(\lambda)$. In this case, succinctness of Π_{FB} ensures that the size of the global parameters as well as the size of the master public key is at most $\mathsf{poly}(\lambda)$, as required. □

Theorem 7.10 (Succinct Decryption Keys)**.** *If Π_{FB} is succinct, then Construction 7.6 has succinct decryption keys.*

Proof. By construction, the secret keys $\mathsf{sk}_{\mathsf{aid}}$ in Construction 7.6 are simply secret keys for Π_{NCE} on messages of length $|\mathsf{S}|$, where $|\mathsf{S}|$ is the share size for the computational secret sharing scheme Π_{CSS}. Since all of these primitives are instantiated independently of the number of users L, the size of $\mathsf{sk}_{\mathsf{aid}}$ depends only on the security parameter λ and not the total number of users. In addition, the helper decryption keys $\mathsf{hsk}_{\mathsf{aid}}$ output by Aggregate are openings of a function-binding hash function. By succinctness of Π_{FB}, the size of these components scale polylogarithmically with the number of users L. Note also that the decryption algorithm does not take the global parameters gpp as input (though even if it did, the global parameters for Construction 7.6) are also succinct. Correspondingly, we conclude that the scheme has succinct decryption keys. □

7.3 Security Analysis of Construction 7.6

In this section, we give the security proof for Construction 7.6. Specifically, we prove the following theorem:

Theorem 7.11 (Security)**.** *If iO is secure, Π_{FB} satisfies computational function hiding and statistical function binding, Π_{CSS} is adaptively secure, and Π_{NCE} is simulatable, then Construction 7.6 is secure.*

Proof. To prove Theorem 7.11, we will use a hybrid argument. We begin with a sequence of "outer" hybrids that highlights the general structure of our argument. Specifically, we modify the obfuscated program in the ciphertext to incrementally removes the ability for certain user identifiers to decrypt. To avoid the need to iterate over an exponential number of user identifiers (which would require an exponential number of hybrids), we take advantage of the fact that the number of users that the adversary can register in the security game is always polynomially-bounded (since the adversary is polynomially-bounded). This means we can partition the space of user identifiers $\mathcal{GID}_\lambda$ into polynomially-many disjoint subsets by hashing them into a polynomial-size set. By picking this set to be large enough (but still polynomial in the total number of registered user identifiers), we can ensure that all of the registered user identifiers gid are mapped onto distinct values under the hash function; this in turn allows us to prove security using a hybrid argument that analyzes one hash value at a time (of which there are at most a polynomial number). We now define our sequence of hybrid experiments. Each experiment is parameterized by an (implicit) security parameter λ, a bit $b \in \{0, 1\}$, and an adversary $\mathcal{A}$. We let $B_{\mathcal{A}}$ be an upper bound on the running time of adversary $\mathcal{A}$ (and thus, the number of unique user identifiers gid that the adversary might choose). Since $\mathcal{A}$ is efficient, we can bound $B_{\mathcal{A}} \leq 2^{\lambda/2}$.

- $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$: This is the real security game where the challenger encrypts message μ_b^*. We recall the main steps here:
  - **Setup phase:** In the setup phase, the adversary $\mathcal{A}$ sends the slot count 1^L to the challenger.[6] The challenger then samples the global parameters $\mathsf{gpp} = \mathsf{hk} \leftarrow \mathsf{FB.Setup}(1^\lambda, L)$ which is sent to the adversary. The challenger also initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary $\mathsf{D} = \varnothing$ to keep track of key-generation queries.
  - **Query phase:** The challenger responds to the adversary's queries as follows:
    - **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$. It then samples $r_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and sets $(\mathsf{pk}, \mathsf{sk}) = \mathsf{NCE.KeyGen}(1^\lambda; r_{\mathsf{ctr}})$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = \mathsf{pk}$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_{\mathsf{ctr}}$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to D.
    - **Corruption query:** If the adversary makes a corruption query on an index $1 \le \mathsf{ctr}^* \le \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[\mathsf{ctr}^*]$ and replies to $\mathcal{A}$ with sk'.
  - **Challenge phase:** In the challenge phase, the adversary specifies a set of authorities $S_{\mathsf{enc}} \subseteq \mathcal{AU}$ and a challenge policy (S_{enc}, C) where $C \colon 2^{S}_{\mathsf{enc}} \to \{0,1\}$ is a monotone Boolean circuit, and two messages $\mu_0^*, \mu_1^* \in \{0,1\}^\lambda$. For each slot $i \in [L]$ and authority $\mathsf{aid} \in S_{\mathsf{enc}}$, the adversary also specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}^*_{i,\mathsf{aid}})$. The challenger sets up the public keys $\mathsf{pk}_{i,\mathsf{aid}}$ as follows:
    - If $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_{i,\mathsf{aid}}] = (i', \mathsf{aid}', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$ and $\mathsf{aid} = \mathsf{aid}'$, the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$. Otherwise, the challenger aborts with output 0.
    - If $c_{i,\mathsf{aid}} = \perp$, then the challenger sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}^*_{i,\mathsf{aid}}$.

    The challenger computes the master public key for each authority $\mathsf{aid} \in S_{\mathsf{enc}}$ as
    $$\mathsf{mpk}_{\mathsf{aid}} = \mathsf{FB.Hash}(\mathsf{hk}, (\mathsf{gid}_{1,\mathsf{aid}}, \mathsf{pk}_{1,\mathsf{aid}}), \ldots, (\mathsf{gid}_{L,\mathsf{aid}}, \mathsf{pk}_{L,\mathsf{aid}})).$$
    The challenger then constructs the challenge ciphertext by constructing the obfuscated program
    $$C' \leftarrow i\mathcal{O}(\mathsf{Embed}[\mathsf{gpp}, S_{\mathsf{enc}}, \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, C, \mu_b^*]).$$
    It responds to $\mathcal{A}$ with the ciphertext $\mathsf{ct} = C'$.
  - **Post-challenge query phase:** Same as the pre-challenge query phase.
  - **Output phase:** At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

- $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$, except during the setup phase, the challenger samples a universal hash function $H \xleftarrow{\mathrm{R}} \mathcal{H}_R$ where $R = B^2_{\mathcal{A}}$. During the challenge phase, the challenger aborts if there exists $(i, \mathsf{aid}) \neq (i, \mathsf{aid}')$ where $\mathsf{gid}_{i,\mathsf{aid}} \neq \mathsf{gid}_{i,'\mathsf{aid}'}$ and $H(\mathsf{gid}_{i,\mathsf{aid}}) = H(\mathsf{gid}_{i,'\mathsf{aid}'})$.

- $\mathsf{Hyb}_{2,j}^{(b)}$ for $j \in [0, B^2_{\mathcal{A}}]$: Same as $\mathsf{Hyb}_1^{(b)}$, except in the challenge phase, the challenger constructs the challenge ciphertext as
  $$\mathsf{ct} = C' \leftarrow i\mathcal{O}(1^\lambda, \mathsf{Embed}'_j[\mathsf{hk}, S_{\mathsf{enc}}, \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, C, \mu_b^*, H])$$
  where Embed'_j is defined as follows:

[6]Later on in the security game, the adversary needs to specify a public key for each slot $i \in [L]$. For this reason, the number of slots an efficient adversary can choose is ultimately going to be polynomially-bounded. Thus, without loss of generality, we still require the adversary provide the number of slots in unary rather than binary.

Constants: Hash key hk, set S_{enc}, master public keys $\{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}$, a monotone Boolean circuit $C \colon 2^{S_{\mathsf{enc}}} \to \{0, 1\}$, a message $\mu \in \{0, 1\}^{\lambda}$, and a hash function $H \colon \mathcal{GID}_{\lambda} \to [B_{\mathcal{A}}^2]$

Inputs: A decryption set $S \subseteq S_{\mathsf{enc}}$ and for each $\mathsf{aid} \in S$, a secret key $\mathsf{sk}_{\mathsf{aid}}$ and a helper decryption key $\mathsf{hsk}_{\mathsf{aid}}$.

1. For each $\mathsf{aid} \in S$, let $\mathsf{mpk}_{\mathsf{aid}} = h_{\mathsf{aid}}$, $\mathsf{sk}_{\mathsf{aid}} = r_{\mathsf{aid}}$, and $\mathsf{hsk}_{\mathsf{aid}} = (i_{\mathsf{aid}}, \mathsf{gid}_{\mathsf{aid}}, \mathsf{pk}_{\mathsf{aid}}, \pi_{\mathsf{aid}})$.
2. Output μ if the following checks pass:
   - $C(S) = 1$.
   - There exists $\mathsf{gid}^* = \mathsf{gid}_{\mathsf{aid}}$ for all $\mathsf{aid} \in S$.
   - $\mathsf{FB.Verify}(\mathsf{hk}, h_{\mathsf{aid}}, S_{\mathsf{ind}}, \{(i_{\mathsf{aid}}, (\mathsf{gid}^*, \mathsf{pk}_{\mathsf{aid}}), \pi_{\mathsf{aid}})\}_{\mathsf{aid} \in S}) = 1$ where $S_{\mathsf{ind}} = \{i_{\mathsf{aid}} \mid \mathsf{aid} \in S\}$.
   - $H(\mathsf{gid}^*) > j$.
   - For each $\mathsf{aid} \in S$, $\mathsf{pk}_{\mathsf{aid}} = \mathsf{pk}'_{\mathsf{aid}}$ where let $(\mathsf{pk}'_{\mathsf{aid}}, \mathsf{sk}'_{\mathsf{aid}}) = \mathsf{NCE.KeyGen}(1^{\lambda}; r_{\mathsf{aid}})$.

   If any of the checks fail, output $\perp$.

Figure 2: Program $\mathsf{Embed}'_j[\mathsf{hk}, S_{\mathsf{enc}}, \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, C, \mu, H]$.

- $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$: Same as $\mathsf{Hyb}_{2,(B_{\mathcal{A}})^2}^{(b)}$, except the challenger computes the challenge ciphertext as $C' \leftarrow i\mathcal{O}(1^{\lambda}, \mathsf{Bot})$, where Bot is the program that outputs $\perp$ on all inputs. Note that as usual, the program is padded to the maximum size of any program that appears in the proof of Theorem 7.11.

As usual, for an adversary $\mathcal{A}$, we write $\mathsf{Hyb}^{(b)}(\mathcal{A})$ to denote the output distribution of an execution of $\mathsf{Hyb}^{(b)}$ with adversary $\mathcal{A}$. We now analyze each adjacent pair of hybrid distributions.

Lemma 7.12. *For all adversaries $\mathcal{A}$, $\Pr[\mathsf{Hyb}_{\mathsf{real}}^{(b)}(\mathcal{A}) = 1] \le 2 \cdot \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]$.*

Proof. By universality of $\mathcal{H}_R$, for any pair of distinct user identifiers $\mathsf{gid}_0 \neq \mathsf{gid}_1 \in \{0, 1\}^{\lambda}$, we have

$$\Pr[H(\mathsf{gid}_0) = H(\mathsf{gid}_1)] \le \frac{1}{R} = \frac{1}{B_{\mathcal{A}}^2}.$$

Since the total number of user identifiers appearing in the challenge phase can be bounded by the running time of the adversary $B_{\mathcal{A}}$, we can union bound the probability of *any* collision by

$$\frac{1}{B_{\mathcal{A}}^2} \cdot \binom{B_{\mathcal{A}}}{2} \le \frac{1}{B_{\mathcal{A}}^2} \frac{B_{\mathcal{A}}^2}{2} = \frac{1}{2}.$$ □

Lemma 7.13. *Suppose $i\mathcal{O}$ is secure. Then for all efficient adversaries $\mathcal{A}$ and all $b \in \{0, 1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,0}^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. By construction, these hybrids only differ in the distribution of the challenge ciphertext. In $\mathsf{Hyb}_1^{(b)}$, the challenge ciphertext is an obfuscation of Embed whereas in $\mathsf{Hyb}_{2,0}^{(b)}$, it is an obfuscation of the program Embed_0. By construction, the Embed'_0 program introduces an additional check that $H(\mathsf{gid}^*) > 0$, which is always satisfied. Thus, these programs are functionally equivalent, so indistinguishability follows by $i\mathcal{O}$ security. □

Lemma 7.14. *Suppose $i\mathcal{O}$ is secure, Π_{FB} satisfies computational function hiding and statistical function binding, Π_{CSS} is adaptively secure, and Π_{NCE} is simulatable, Then for all efficient adversaries $\mathcal{A}$, all $b \in \{0,1\}$, and all $j \in [B_{\mathcal{A}}^2]$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_{2,j-1}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,j}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

To prove Lemma 7.14, we will define a sequence of intermediate hybrids. We defer this to Section 7.3.1.

Lemma 7.15. *Suppose $i\mathcal{O}$ is secure. Then for all efficient adversaries $\mathcal{A}$ and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_{2,B_{\mathcal{A}}^2}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. By construction, these hybrids only differ in the distribution of the challenge ciphertext. In $\mathsf{Hyb}_{2,B_{\mathcal{A}}^2}^{(b)}$, the challenge ciphertext is an obfuscation of the program $\mathsf{Embed}'_{B_{\mathcal{A}}^2}$ whereas in $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$, it is an obfuscation of Bot. By construction, $\mathsf{Embed}'_{B_{\mathcal{A}}^2}$ will output something other than $\perp$ only if $H(\mathsf{gid}^*) > B_{\mathcal{A}}^2$. However, since the range of H is $[B_{\mathcal{A}}^2]$, this check is never satisfied. As such, $\mathsf{Embed}'_{B_{\mathcal{A}}^2}$ always outputs $\perp$. Thus, these programs are functionally equivalent and the lemma follows via $i\mathcal{O}$ security. □

Lemma 7.16. *For all adversaries $\mathcal{A}$,* $\Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(0)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(1)}(\mathcal{A}) = 1]$.

Proof. By definition, the challenger's behavior in these two experiments is independent of the bit b. As such, the output distributions of the experiments are identically distributed. □

Since $B_{\mathcal{A}} = \mathsf{poly}(\lambda)$, Theorem 7.11 now follows by combining Lemmas 7.12 to 7.16. □

7.3.1 Proof of Lemma 7.14

In the section, we show that hybrids $\mathsf{Hyb}_{2,j-1}^{(b)}$ to $\mathsf{Hyb}_{2,j}^{(b)}$ from the proof of Theorem 7.11 are computationally indistinguishable. To do so, we introduce a sequence of intermediate hybrids:

- $\mathsf{iHyb}_{j,0}^{(b)}$: Same as $\mathsf{Hyb}_{2,j-1}^{(b)}$, except the challenger defines the unique-block-selection (Definition 7.3) function f_g for the predicate

$$g = g_{H,j}((\mathsf{gid}, \mathsf{pk})) = \begin{cases} 1 & H(\mathsf{gid}) = j \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

  for the hash function $H \in \mathcal{H}_R$ sampled in $\mathsf{Hyb}_1^{(b)}$. Then, it defines the hash key to be $\mathsf{hk} \leftarrow \mathsf{SetupBinding}(1^\lambda, L, f_g)$.

- $\mathsf{iHyb}_{j,1}^{(b)}$: Same as $\mathsf{iHyb}_{j,0}^{(b)}$, except during the challenge phase, the challenger computes

$$(\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S_{\mathsf{enc}}|}) \leftarrow \mathsf{CSS.Share}(1^\lambda, C, 1).$$

  Next, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, if there exists a unique index i_{aid}^* where $H(\mathsf{gid}_{i_{\mathsf{aid}}^*, \mathsf{aid}}) = j$, then set

$$\mathsf{ct}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Encrypt}(\mathsf{pk}_{i^*,\mathsf{aid}}, \mathsf{sh}_{\mathsf{aid}}).$$

  Otherwise, set $\mathsf{ct}_{\mathsf{aid}} = \perp$. Note that if an index i_{aid}^* exists, then it is necessarily unique (otherwise, there is a collision in H and the challenger halts the experiment). Finally, the challenger constructs the challenge ciphertext as

$$\mathsf{ct} = C' \leftarrow i\mathcal{O}(\mathsf{Embed}''_j[\mathsf{gpp}, S_{\mathsf{enc}}, \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, C, \mu_b^*, H, \{(\mathsf{aid}, \mathsf{ct}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}])$$

  where Embed''_j is defined as follows:

Constants: Hash key hk, set S_{enc}, master public keys $\{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}$, a monotone Boolean circuit $C \colon 2^{S_{\mathsf{enc}}} \to \{0,1\}$, a message $\mu \in \{0,1\}^{\lambda}$, a hash function $H \colon \mathcal{GID}_{\lambda} \to [B^2_{\mathcal{A}}]$, and ciphertexts $\mathsf{ct}_{\mathsf{aid}}$ for each $\mathsf{aid} \in S_{\mathsf{enc}}$

Inputs: A decryption set $S \subseteq S_{\mathsf{enc}}$ and for each $\mathsf{aid} \in S$, a secret key $\mathsf{sk}_{\mathsf{aid}}$ and a helper decryption key $\mathsf{hsk}_{\mathsf{aid}}$.

1. For each $\mathsf{aid} \in S$, let $\mathsf{mpk}_{\mathsf{aid}} = h_{\mathsf{aid}}$, $\mathsf{sk}_{\mathsf{aid}} = r_{\mathsf{aid}}$, and $\mathsf{hsk}_{\mathsf{aid}} = (i_{\mathsf{aid}}, \mathsf{gid}_{\mathsf{aid}}, \mathsf{pk}_{\mathsf{aid}}, \pi_{\mathsf{aid}})$.
2. Output μ if the following checks pass:
   - $C(S) = 1$.
   - There exists $\mathsf{gid}^* = \mathsf{gid}_{\mathsf{aid}}$ for all $\mathsf{aid} \in S$.
   - $\mathsf{FB.Verify}(\mathsf{hk}, h_{\mathsf{aid}}, S_{\mathsf{ind}}, \{(i_{\mathsf{aid}}, (\mathsf{gid}^*, \mathsf{pk}_{\mathsf{aid}}), \pi_{\mathsf{aid}})\}_{\mathsf{aid} \in S}) = 1$. where $S_{\mathsf{ind}} = \{i_{\mathsf{aid}} \mid \mathsf{aid} \in S\}$.
   - $H(\mathsf{gid}^*) > j$.
   - If $H(\mathsf{gid}^*) = j$, then for all $\mathsf{aid} \in S$, check that the following conditions hold:
     * $(\mathsf{pk}_{\mathsf{aid}}, \mathsf{sk}_{\mathsf{aid}}) = \mathsf{NCE.KeyGen}(1^{\lambda}; r_{\mathsf{aid}})$;
     * $\mathsf{CSS.Recon}(C, S, \{(\mathsf{aid}, \mathsf{sh}_{\mathsf{aid}})\}_{\mathsf{aid} \in S}) = 1$ where $\mathsf{sh}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Decrypt}(\mathsf{sk}_{\mathsf{aid}}, \mathsf{ct}_{\mathsf{aid}})$.
   - For each $\mathsf{aid} \in S$, $\mathsf{pk}_{\mathsf{aid}} = \mathsf{pk}'_{\mathsf{aid}}$ where let $(\mathsf{pk}'_{\mathsf{aid}}, \mathsf{sk}'_{\mathsf{aid}}) = \mathsf{NCE.KeyGen}(1^{\lambda}; r_{\mathsf{aid}})$.

   If any of the checks fail, output $\perp$.

Figure 3: Program $\mathsf{Embed}''_j[\mathsf{hk}, S_{\mathsf{enc}}, \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, C, \mu, H, \{(\mathsf{aid}, \mathsf{ct}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}]$.

- $\mathsf{iHyb}^{(b)}_{j,2}$: Same as $\mathsf{iHyb}^{(b)}_{j,1}$, except the challenger answers the key-generation queries using simulated keys. In the challenge phase, the challenger uses simulated ciphertexts for the uncorrupted keys. Specifically, the challenger in this experiment behaves as follows:
  - **Setup phase:** Same as in $\mathsf{iHyb}^{(b)}_{j,1}$.
  - **Query phase:** The challenger responds to the adversary's queries as follows:
    * **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger starts by incrementing the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$. It then computes $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}) \leftarrow \mathsf{NCE.CTSim}(1^{\lambda})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. The challenger then adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}, \perp)$ to the dictionary D.
    * **Corruption query:** If the adversary makes a corruption query on an index $1 \le \mathsf{ctr}^* \le \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{ct}', \mathsf{td}', \mathsf{sk}') = \mathsf{D}[\mathsf{ctr}^*]$. If $\mathsf{sk}' \neq \perp$, then it returns sk'. Otherwise, the challenge computes $r' \leftarrow \mathsf{NCE.RSim}(\mathsf{td}', 0)$, and updates the mapping $\mathsf{ctr} \mapsto (i', \mathsf{pk}', \mathsf{ct}', \mathsf{td}', r')$ in D. The challenger replies with $\mathsf{sk}' = r'$.
  - **Challenge phase:** During the challenge phase, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, the challenger proceeds as follows:
    * If the index i^*_{aid} exists, and $c_{i^*_{\mathsf{aid}}, \mathsf{aid}} \in [\mathsf{ctr}]$, the challenger looks up the entry $(i^*, \mathsf{pk}^*, \mathsf{ct}^*, \mathsf{td}^*, \mathsf{sk}^*) = \mathsf{D}[c_{i^*_{\mathsf{aid}}, \mathsf{aid}}]$. If $\mathsf{sk}^* = \perp$ (i.e., the adversary specified an *uncorrupted* key for index i^*_{aid}), the challenger sets $\mathsf{ct}_{\mathsf{aid}} = \mathsf{ct}^*$. If $\mathsf{sk}^* \neq \perp$ (i.e., the adversary corrupted the key), then the challenger computes $\mathsf{ct}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Encrypt}(\mathsf{pk}_{i^*, \mathsf{aid}}, \mathsf{sh}_{\mathsf{aid}})$ exactly as in $\mathsf{iHyb}^{(b)}_{j,1}$.
    * Otherwise, if the index i^*_{aid} exists, but $c_{i^*_{\mathsf{aid}}, \mathsf{aid}} \notin [\mathsf{ctr}]$ (i.e., the adversary chose the key for index i^*_{aid}), then the challenger computes $\mathsf{ct}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Encrypt}(\mathsf{pk}_{i^*, \mathsf{aid}}, \mathsf{sh}_{\mathsf{aid}})$ exactly as in $\mathsf{iHyb}^{(b)}_{j,1}$.

  Finally, if no such index i^*_{aid} exists, the challenger sets $\mathsf{ct}_{\mathsf{aid}} = \perp$ as in $\mathsf{iHyb}^{(b)}_{j,1}$. The rest of the challenge phase proceeds as in $\mathsf{iHyb}^{(b)}_{j,1}$.

- **Post-challenge query phase:** Same as the pre-challenge query phase, except when responding to corruption queries, the challenger uses the following modified procedure:
    - **Corruption query:** If the adversary makes a corruption query on an index $1 \leq \mathsf{ctr}^* \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{ct}', \mathsf{td}', \mathsf{sk}') = \mathsf{D}[\mathsf{ctr}^*]$. If $\mathsf{sk}' \neq \perp$, return sk'. Otherwise, if there exists i^*_{aid} such that $\mathsf{ctr}^* = c_{i^*_{\mathsf{aid}},\mathsf{aid}}$ and $H(\mathsf{gid}_{i^*_{\mathsf{aid}},\mathsf{aid}}) = j$, compute $r' \leftarrow \mathsf{NCE.RSim}(\mathsf{td}', \mathsf{sh}_{\mathsf{aid}})$. If not, compute $r' \leftarrow \mathsf{NCE.RSim}(\mathsf{td}', 0)$. In either case, update the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}', \mathsf{ct}', \mathsf{td}', r')$ in D and reply with $\mathsf{sk}' = r'$.

- $\mathsf{iHyb}^{(b)}_{j,3}$: Same as $\mathsf{iHyb}^{(b)}_{j,2}$, except during the challenge phase, the challenger uses a secret sharing of 0 instead. Namely, in the challenge phase, the challenger now samples $(\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S_{\mathsf{enc}}|}) \leftarrow \mathsf{CSS.Share}(1^\lambda, C, 0)$.

- $\mathsf{iHyb}^{(b)}_{j,4}$: Same as $\mathsf{iHyb}^{(b)}_{j,3}$, except the challenger answers the key-generation queries according to the real scheme (instead of simulating them). Specifically, the challenger in this experiment proceeds as follows:
  - **Setup phase:** Same as in $\mathsf{iHyb}^{(b)}_{j,3}$.
  - **Pre-challenge query phase:** The challenger answers the queries exactly as in $\mathsf{Hyb}^{(b)}_{\mathsf{real}}$. Namely, it proceeds as follows:
    - **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$. It then samples $r_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and sets $(\mathsf{pk}, \mathsf{sk}) = \mathsf{NCE.KeyGen}(1^\lambda; r_{\mathsf{ctr}})$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = \mathsf{pk}$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_{\mathsf{ctr}}$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to D.
    - **Corruption query:** If the adversary makes a corruption query on an index $1 \leq \mathsf{ctr}^* \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[\mathsf{ctr}^*]$ and replies to $\mathcal{A}$ with sk'.
  - **Challenge phase:** As in $\mathsf{iHyb}^{(b)}_{j,3}$, the challenger now computes. $(\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S_{\mathsf{enc}}|}) \leftarrow \mathsf{CSS.Share}(1^\lambda, C, 0)$, Next, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, if there exists a unique index i^*_{aid} where $H(\mathsf{gid}_{i^*_{\mathsf{aid}},\mathsf{aid}}) = j$, then it sets

    $$\mathsf{ct}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Encrypt}(\mathsf{pk}_{i^*,\mathsf{aid}}, \mathsf{sh}_{\mathsf{aid}}).$$

    Otherwise, set $\mathsf{ct}_{\mathsf{aid}} = \perp$. Finally, the challenger constructs the challenge ciphertext as

    $$\mathsf{ct} = C' \leftarrow i\mathcal{O}(\mathsf{Embed}''_j[\mathsf{gpp}, S_{\mathsf{enc}}, \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, C, \mu^*_b, H, \{(\mathsf{aid}, \mathsf{ct}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}]).$$
  - **Post-challenge query phase:** Same as the pre-challenge query phase.

- $\mathsf{iHyb}^{(b)}_{j,5}$: Same as $\mathsf{iHyb}^{(b)}_{j,4}$, except the challenger no longer computes the ciphertexts $\mathsf{ct}_{\mathsf{aid}}$. Instead, it generates the challenge ciphertext as

  $$\mathsf{ct} = C' \leftarrow i\mathcal{O}(\mathsf{Embed}'_j[\mathsf{hk}, S_{\mathsf{enc}}, \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, C, \mu^*_b, H])$$

  where Embed' is the program from Fig. 2.

Lemma 7.17. *Suppose Π_{FB} is computationally function-hiding. Then for all efficient adversaries $\mathcal{A}$, bits $b \in \{0,1\}$, and $j \in [B^2_{\mathcal{A}}]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}^{(b)}_{2,j-1}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}^{(b)}_{j,0}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. The only difference between $\mathsf{Hyb}^{(b)}_{2,j-1}$ and $\mathsf{iHyb}^{(b)}_{j,0}$ is the challenger samples hk in “normal” mode in $\mathsf{Hyb}^{(b)}_{2,j-1}$ and in binding mode in $\mathsf{iHyb}^{(b)}_{j,0}$. The claim thus follows by the function hiding property of Π_{FB}. Formally, suppose there exists an efficient adversary $\mathcal{A}$ such that that can distinguish $\mathsf{Hyb}^{(b)}_{2,j-1}$ and $\mathsf{iHyb}^{(b)}_{j,0}$ with non-negligible advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks function hiding of Π_{FB} with the same advantage:

1. Algorithm $\mathcal{B}$ starts by running algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs the number of slots $L \in \mathbb{N}$.
2. Algorithm $\mathcal{B}$ samples a universal hash function $H \xleftarrow{\mathrm{R}} \mathcal{H}_R$, and outputs the message length L and challenge function f_g. It receives a hash key hk from the challenger.
3. Algorithm $\mathcal{B}$ sets $\mathsf{crs} = \mathsf{hk}$ and gives crs to $\mathcal{A}$.
4. Algorithm $\mathcal{B}$ simulates the rest of $\mathsf{Hyb}_{2,j-1}^{(b)}$ and $\mathsf{iHyb}_{j,0}^{(b)}$ exactly as prescribed. At the end of the game, adversary $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

By construction, if $\mathsf{hk} \leftarrow \mathsf{FB.Setup}(1^\lambda, L)$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,j-1}^{(b)}$ for $\mathcal{A}$. Likewise, if $\mathsf{hk} \leftarrow \mathsf{FB.SetupBinding}(1^\lambda, L, f_g)$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{iHyb}_{j,0}^{(b)}$ for $\mathcal{A}$. Thus, algorithm $\mathcal{B}$ succeeds with the same advantage ε, and the claim follows. □

Lemma 7.18. *Suppose iO is secure,* Π_{NCE} *and* Π_{CSS} *are correct, and* Π_{FB} *is statistically function-binding. Then for all efficient adversaries* $\mathcal{A}$*, bits* $b \in \{0,1\}$*, and* $j \in [B_{\mathcal{A}}^2]$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr[\mathsf{iHyb}_{j,0}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{j,1}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Recall the difference between these two hybrids is the use of program Embed'_j and Embed''_j. We will argue that with all but negligible probability, these programs are functionally equivalent. Observe that Embed'_j only performs the following additional checks when $H(\mathsf{gid}^*) = j$:

- $(\mathsf{pk}_{\mathsf{aid}}, \mathsf{sk}_{\mathsf{aid}}) = \mathsf{KeyGen}(1^\lambda; r_{\mathsf{aid}})$.
- $\mathsf{CSS.Recon}(C, S, \{(\mathsf{aid}, \mathsf{sh}_{\mathsf{aid}})\}_{\mathsf{aid} \in S}) = 1$ where $\mathsf{sh}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Decrypt}(\mathsf{sk}_{\mathsf{aid}}, \mathsf{ct}_{\mathsf{aid}})$.

Let $S^*, \{(\mathsf{aid}, \mathsf{sk}^*_{\mathsf{aid}}, \mathsf{hsk}^*_{\mathsf{aid}})\}_{\mathsf{aid} \in S^*}$ be an arbitrary input to these two programs. Since Embed''_j is the same as Embed'_j with additional abort conditions, it must be the case that if $\mathsf{Embed}'_j(S^*, \{(\mathsf{aid}, \mathsf{sk}^*_{\mathsf{aid}}, \mathsf{hsk}^*_{\mathsf{aid}})\}_{\mathsf{aid} \in S^*}) = \perp$, then

$$\mathsf{Embed}''_j(S^*, \{(\mathsf{aid}, \mathsf{sk}^*_{\mathsf{aid}}, \mathsf{hsk}^*_{\mathsf{aid}})\}_{\mathsf{aid} \in S^*}) = \perp.$$

Thus, consider some input where $\mathsf{Embed}'_j(S^*, \{(\mathsf{aid}, \mathsf{sk}^*_{\mathsf{aid}}, \mathsf{hsk}^*_{\mathsf{aid}})\}_{\mathsf{aid} \in S^*}) = \mu_b^*$. First, we parse $\mathsf{sk}^*_{\mathsf{aid}} = r^*_{\mathsf{aid}}$ and $\mathsf{hsk}^*_{\mathsf{aid}} = (i^*_{\mathsf{aid}}, \mathsf{gid}^*, \mathsf{pk}^*_{\mathsf{aid}}, \pi^*_{\mathsf{aid}})$ and consider two cases:

- Suppose for all $\mathsf{aid} \in S^*$, there exists a public key associated with some $\mathsf{gid}_{\mathsf{aid}}$ where $H(\mathsf{gid}_{\mathsf{aid}}) = j$ and moreover, the associated public key in the challenge phase satisfies $\mathsf{pk}_{i^*_{\mathsf{aid}}, \mathsf{aid}} = \mathsf{pk}^*_{\mathsf{aid}}$. In this case, the ciphertext $\mathsf{ct}_{\mathsf{aid}}$ is computed as $\mathsf{ct}_{\mathsf{aid}} \leftarrow \mathsf{Encrypt}(\mathsf{pk}^*_{\mathsf{aid}}, \mathsf{sh}_{\mathsf{aid}})$, where $(\mathsf{sh}_1, \ldots, \mathsf{sh}_{|S_{\mathsf{enc}}|}) \leftarrow \mathsf{CSS.Share}(1^\lambda, C, 1)$. Since $(\mathsf{pk}^*_{\mathsf{aid}}, \mathsf{sk}^*_{\mathsf{aid}})$ are in the support of $\mathsf{NCE.KeyGen}$, by perfect correctness of Π_{NCE},

  $$\mathsf{NCE.Decrypt}(\mathsf{sk}^*_{\mathsf{aid}}, \mathsf{ct}_{\mathsf{aid}}) = \mathsf{sh}_{\mathsf{aid}}.$$

  Since $C(S^*) = 1$ (otherwise, Embed_j would already output $\perp$), by perfect correctness of Π_{CSS}, we have that $\mathsf{CSS.Recon}(C, S^*, \{\mathsf{sh}_{\mathsf{aid}}\}_{\mathsf{aid} \in S^*}) = 1$. Thus, the additional checks of Embed''_j pass and so

  $$\mathsf{Embed}'_j(S^*, \{(\mathsf{aid}, \mathsf{sk}^*_{\mathsf{aid}}, \mathsf{hsk}^*_{\mathsf{aid}})\}_{\mathsf{aid} \in S^*}) = \mathsf{Embed}''_j(S^*, \{(\mathsf{aid}, \mathsf{sk}^*_{\mathsf{aid}}, \mathsf{hsk}^*_{\mathsf{aid}})\}_{\mathsf{aid} \in S^*}).$$

- Otherwise, suppose there exists some $\mathsf{aid} \in S^*$ where the above condition does not hold. From the check in $\mathsf{Hyb}_1^{(b)}$, the function H is injective on the user identifiers selected by the adversary. Thus, for every aid in S_{enc}, it must be the case that $f_g(\{(\mathsf{aid}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}_{i,\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}) = \perp$ or there exists a key $\mathsf{pk}_{i^*,\mathsf{aid}}$ aggregated in the challenge to some $\mathsf{gid}_{\mathsf{aid}}$ where $H(\mathsf{gid}_{\mathsf{aid}}) = j$. If Π_{NCE} is function binding, in the former case, there does not exist a valid opening to h_{aid} (with overwhelming probability over the choice of the hash key hk), and in the latter case, the only possible opening is to $\mathsf{pk}_{i^*,\mathsf{aid}}$. Thus, such an aid existing only occurs when hk is not function binding on $f_{H,j}$, which occurs with negligible probability.

Taking together, we can see that $\mathsf{Embed}'_j, \mathsf{Embed}''_j$ are functionally equivalent with all but negligible probability. This hybrid then follows from $i\mathcal{O}$ security. $\square$

Lemma 7.19. *Suppose* Π_{NCE} *is simulatable. Then for all efficient adversaries* $\mathcal{A}$*, bits* $b \in \{0,1\}$*, and* $j \in [B^2_{\mathcal{A}}]$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr[\mathsf{iHyb}^{(b)}_{j,1}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}^{(b)}_{j,2}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. To show this, we introduce a sequence of sub-hybrids, where we incrementally replace key-generation queries answered using the real encryption algorithm be be answered using the simulated one.

- $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'}$: Same as $\mathsf{iHyb}^{(b)}_{j,1}$, except when $\mathsf{ctr} \le \mathsf{ctr}'$, the challenger samples keys as in $\mathsf{iHyb}^{(b)}_{j,2}$. Specifically, the challenger initializes two dictionaries D, D'. We now give the differences relative to $\mathsf{iHyb}^{(b)}_{j,1}$:
  - **Query phase:** The challenger responds to the adversary's queries as follows:
    - **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr}+1$. If $\mathsf{ctr} \le \mathsf{ctr}'$ it computes $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}) \leftarrow \mathsf{NCE.CTSim}(1^\lambda)$, adds $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{ct}_{\mathsf{ctr}}, \mathsf{td}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}} = \perp)$ to the dictionary D', and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. Otherwise, the challenger proceeds as in $\mathsf{iHyb}^{(b)}_{j,1}$
    - **Corruption query:** If the adversary makes a corruption query on an index $1 \le \mathsf{ctr}^* \le \min(\mathsf{ctr}, \mathsf{ctr}')$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{ct}', \mathsf{td}', \mathsf{sk}') \leftarrow \mathsf{D}'[\mathsf{ctr}^*]$. If $\mathsf{sk}' \ne \perp$, return sk'. Otherwise, the challenge computes $r' \leftarrow \mathsf{RSim}(\mathsf{td}', 0)$, sets $\mathsf{sk}' = r'$ (and updates the entry $\mathsf{D}'[\mathsf{ctr}^*]$ accordingly) and then returns sk'. If $\mathsf{ctr}^* > \mathsf{ctr}'$, the challenger proceed as $\mathsf{iHyb}^{(b)}_{j,1}$
  - **Challenge Phase:** The challenge phase proceeds as in $\mathsf{iHyb}^{(b)}_{j,1}$, except for each $\mathsf{aid} \in S_{\mathsf{enc}}$, if i^*_{aid} exists, and $c_{i^*_{\mathsf{aid}},\mathsf{aid}} \in [\min(\mathsf{ctr}, \mathsf{ctr}')]$, thc challenger looks up the entry $(i^*, \mathsf{pk}^*, \mathsf{ct}^*, \mathsf{td}^*, \mathsf{sk}^*) = \mathsf{D}'[\mathsf{ctr}^*]$. If $\mathsf{sk}^* = \perp$, set $\mathsf{ct}_{\mathsf{aid}} = \mathsf{ct}^*$. Otherwise, compute $\mathsf{ct}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Encrypt}(\mathsf{pk}_{i^*,\mathsf{aid}}, \mathsf{sh}_{\mathsf{aid}})$. If no such i^*_{aid} exists, set $\mathsf{ct}_{\mathsf{aid}} = \perp$.
  - **Post-Challenge Query Phase:** Same as the pre-challenge query phase, except:
    - **Corruption query:** If the adversary makes a corruption query on an index $1 \le \mathsf{ctr}^* \le \min(\mathsf{ctr}, \mathsf{ctr}')$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{ct}', \mathsf{td}', \mathsf{sk}') = \mathsf{D}'[\mathsf{ctr}^*]$. If $\mathsf{sk}' \ne \perp$, return sk'. Otherwise, if there exists i^*_{aid} such that $\mathsf{ctr}^* = c_{i^*_{\mathsf{aid}},\mathsf{aid}}$, compute $r' \leftarrow \mathsf{RSim}(\mathsf{td}', \mathsf{sh}_{\mathsf{aid}})$. If not or $\mathsf{ctr} > \mathsf{ctr}'$, compute $r' \leftarrow \mathsf{RSim}(\mathsf{td}', 0)$. In either case, set $\mathsf{sk}' = r'$ and return sk' (and update the entry $\mathsf{D}'[\mathsf{ctr}^*]$.

Claim 7.20. *Suppose* Π_{NCE} *is simulatable. Then for all efficient adversaries* $\mathcal{A}$*, bits* $b \in \{0,1\}$*,* $j \in [B^2_{\mathcal{A}}]$*, and* $\mathsf{ctr}' \in [Q_{\mathcal{A}}]$*, where* $Q_{\mathcal{A}}$ *is a bound on the number of queries the adversary makes, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr[\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}(\mathcal{A}) = 1] - \Pr[\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose that there exists an efficient adversary $\mathcal{A}$ that can distinguish these two experiments with non-negligible probability ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks simulation security of Π_{NCE} with the same advantage:

1. **Setup phase:** Algorithm $\mathcal{B}$ receives the challenge public key pk^*, and runs $\mathcal{A}$. The challenger simulates the setup phase exactly as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$.
2. **Query phase:** The challenger responds to the adversary's queries as follows:
   - **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger starts by incrementing the counter $\mathsf{ctr} \leftarrow \mathsf{ctr}+1$. If $\mathsf{ctr} = \mathsf{ctr}'$, respond with $(\mathsf{ctr}', \mathsf{pk}^*)$. Otherwise, it proceeds as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$.
   - **Corruption query:** If the adversary makes a corruption query on an index ctr', algorithm $\mathcal{B}$ makes a query on message $m^* = 0$ to its challenger. The challenger responds with (ct^*, r^*). Algorithm $\mathcal{B}$ returns r^* to $\mathcal{A}$. Otherwise, algorithm $\mathcal{B}$ proceeds as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$.

3. **Challenge Phase:** The challenge phase proceeds exactly as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$, except for each $\mathsf{aid} \in S_{\mathsf{enc}}$, if i^*_{aid} exists and $c_{i^*_{\mathsf{aid}},\mathsf{aid}} = \mathsf{ctr}'$ and has not yet been corrupted, algorithm $\mathcal{B}$ queries the challenger on the message $m^* = \mathsf{sh}_{\mathsf{aid}}$. The challenger responds with (ct^*, r^*). Algorithm $\mathcal{B}$ sets $\mathsf{ct}_{\mathsf{aid}} = \mathsf{ct}^*$. Otherwise, it sets $\mathsf{ct}_{\mathsf{aid}}$ as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$.

4. **Post-challenge query phase:** Same as the pre-challenge query phase, except:

   - **Corruption query:** If the adversary makes a corruption query on index ctr' *and* ctr' refers to a public key that is associated with the challenge ciphertext (i.e., chosen by the adversary) and has not yet been corrupted, algorithm $\mathcal{B}$ returns the corresponding r^*. If ctr' was not included in the ciphertext, then $\mathcal{B}$ queries the challenger on message $m^* = 0$ and receives (ct^*, r^*). Algorithm $\mathcal{B}$ responds with r^*. For other indices, algorithm $\mathcal{B}$ proceeds as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$.

We argue that when $b = 0$, this is exactly $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$ and when $b = 1$, this is exactly $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'}$. To see this, we consider a few cases.

- Suppose ctr' was never corrupted. In this case, algorithm $\mathcal{A}$ only receives pk^* which corresponds exactly to the output of KeyGen or CTSim when $b = 0$ or $b = 1$. This is how the public key is generated in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$ and $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'}$, respectively.
- Suppose index ctr' was corrupted in a pre-challenge query. In this case, when $b = 1$, algorithm $\mathcal{B}$ receives (ct^*, r^*) generated from $\mathsf{RSim}(\mathsf{td}, 0)$, exactly as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'}$. When $b = 0$, the value r^* is the randomness used in KeyGen, which is distributed according to $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$.
- Suppose index ctr' was corrupted in a post-challenge query. Here, either ctr' is in the ciphertext, in which case when $b = 1$, algorithm $\mathcal{B}$ receives (ct^*, r^*) generated from $\mathsf{RSim}(\mathsf{td}, \mathsf{sh}_{\mathsf{aid}})$, exactly as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'}$. When $b = 0$, this ciphertext component is an honest encryption of $\mathsf{sh}_{\mathsf{aid}}$, as in $\mathsf{HybNCE}^{(b,j)}_{\mathsf{ctr}'-1}$. On the other hand, when ctr' is not in the ciphertext, the execution and analysis is identical to pre-challenge queries. □

Completing the proof. Let $Q_{\mathcal{A}}$ be a bound on the number of key generation queries adversary $\mathcal{A}$ makes. By construction, hybrids $\mathsf{HybNCE}^{(b,j)}_0$ and $\mathsf{HybNCE}^{(b,j)}_{Q_{\mathcal{A}}}$ are identical to $\mathsf{iHyb}^{(b)}_{j,1}$ and $\mathsf{iHyb}^{(b)}_{j,2}$ respectively. Since $Q_{\mathcal{A}}$ is polynomially bounded, Lemma 7.19 follows by a hybrid argument. □

Lemma 7.21. *Suppose Π_{CSS} is secure. Then for all efficient admissible adversaries $\mathcal{A}$, bits $b \in \{0,1\}$, and $j \in [B^2_{\mathcal{A}}]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{iHyb}^{(b)}_{j,2}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}^{(b)}_{j,3}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose that there exists an efficient adversary $\mathcal{A}$ that can distinguish these two experiments with non-negligible probability ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks adaptive security of Π_{CSS} with the same advantage:

1. **Setup phase:** Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ and simulates an execution of $\mathsf{iHyb}^{(b)}_{j,2}$. Algorithm $\mathcal{B}$ implements the setup phase exactly as described in $\mathsf{iHyb}^{(b)}_{j,2}$.

2. **Query Phase:** $\mathcal{B}$ proceeds exactly as $\mathsf{iHyb}^{(b)}_{j,2}$.

3. **Challenge Phase:** $\mathcal{B}$ proceeds as in $\mathsf{iHyb}^{(b)}_{j,2}$, except when generating the shares, algorithm $\mathcal{B}$ queries the challenger on the circuit C and the challenge secrets $s_0 = 1$ and $s_1 = 0$. Then, for each $\mathsf{aid} \in S_{\mathsf{enc}}$, if i^*_{aid} exists, and $c_{i^*_{\mathsf{aid}},\mathsf{aid}} \in [\mathsf{ctr}]$, algorithm $\mathcal{B}$ looks up $(i^*, \mathsf{pk}^*, \mathsf{ct}^*, \mathsf{td}^*, \mathsf{sk}^*) = \mathsf{D}[\mathsf{ctr}^*]$. If $\mathsf{sk}^* = \bot$, it sets $\mathsf{ct}_{\mathsf{aid}} = \mathsf{ct}^*$. Otherwise, algorithm $\mathcal{B}$ submits the index aid to the challenger and receives the share $\mathsf{sh}_{\mathsf{aid}}$. Finally, algorithm $\mathcal{B}$ computes $\mathsf{ct}_{\mathsf{aid}} \leftarrow \mathsf{NCE.Encrypt}(\mathsf{pk}_{i^*,\mathsf{aid}}, \mathsf{sh}_{\mathsf{aid}})$. If no such i^*_{aid} exists, algorithm $\mathcal{B}$ sets $\mathsf{ct}_{\mathsf{aid}} = \bot$.

4. **Post-challenge query phase:** Algorithm $\mathcal{B}$ proceeds as $\mathsf{iHyb}_{j,2}^{(b)}$, except when answering corruption queries, it proceeds as follows:

   - **Corruption query:** If the adversary makes a corruption query on an index $1 \le \mathsf{ctr}^* \le \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{ct}', \mathsf{td}', \mathsf{sk}') = \mathsf{D}[\mathsf{ctr}^*]$. If $\mathsf{sk}' \neq \bot$, return sk'. Otherwise, if there exists i^*_{aid} such that $\mathsf{ctr}^* = c_{i^*_{\mathsf{aid}}, \mathsf{aid}}$, algorithm $\mathcal{B}$ submits the index aid to the challenger. The challenger replies with $\mathsf{sh}_{\mathsf{aid}}$ and algorithm $\mathcal{B}$ computes $r' \leftarrow \mathsf{RSim}(\mathsf{td}', \mathsf{sh}_{\mathsf{aid}})$. If the index i^*_{aid} does not exist, then algorithm $\mathcal{B}$ computes $r' \leftarrow \mathsf{RSim}(\mathsf{td}', 0)$. In either case, algorithm $\mathcal{B}$ sets $\mathsf{sk}' = r'$ (and updates the entry in D). It replies to $\mathcal{A}$ with sk'.

5. **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which algorithm $\mathcal{B}$ also outputs.

By construction, algorithm $\mathcal{B}$ only queries the challenger for shares on indices aid where i^*_{aid} exists. By construction, this corresponds to indices where the associated user identifier gid^* is registered to a particular (corrupted) attribute. Since $\mathcal{A}$ is admissible, the set of attributes associated with any user identifier gid cannot satisfy policy C. As such, algorithm $\mathcal{B}$ is an admissible CSS adversary. By construction, we see that depending on whether the shares are of the value $s_0 = 0$ or $s_1 = 1$, algorithm $\mathcal{B}$ either perfectly simulates an execution of $\mathsf{iHyb}_{j,2}^{(b)}$ or $\mathsf{iHyb}_{j,3}^{(b)}$, as required. □

Lemma 7.22. *Suppose Π_{NCE} is simulatable. Then, for all efficient adversaries $\mathcal{A}$, bits $b \in \{0, 1\}$, and $j \in [B^2_{\mathcal{A}}]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{iHyb}_{j,3}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{j,4}^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. Follows from a similar argument as the proof of Lemma 7.19. □

Lemma 7.23. *Suppose $i\mathcal{O}$ is secure, Π_{NCE} and Π_{CSS} are correct, and FB is statistically function-binding. Then, for all efficient adversaries $\mathcal{A}$, bits $b \in \{0, 1\}$, and $j \in [B^2_{\mathcal{A}}]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{iHyb}_{j,4}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{j,5}^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 7.18. However, since $\{(\mathsf{aid}, \mathsf{sh}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}$ is a secret sharing of 0, by perfect correctness of Π_{CSS}, CSS.Recon will never return 1, and so Embed'_j will always return $\bot$ when $H(\mathsf{gid}^*) = j$. □

Lemma 7.24. *Suppose Π_{FB} is computationally function-hiding. Then for all efficient adversaries $\mathcal{A}$, bits $b \in \{0, 1\}$, and $j \in [B^2_{\mathcal{A}}]$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{iHyb}_{j,5}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,j}^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma 7.17. □

Combining Lemmas 7.17 to 7.19 and 7.21 to 7.24, we conclude Lemma 7.14 holds. □

Acknowledgments

Brent Waters is supported by NSF CNS-1908611, CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[AT24] Nuttapong Attrapadung and Junichi Tomida. A modular approach to registered ABE for unbounded predicates. In *CRYPTO*, 2024.

[Bei96] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution.* PhD thesis, Technion, 1996.

[BFM88] Manuel Blum, Paul Feldman, and Silvio Micali. Non-interactive zero-knowledge and its applications (extended abstract). In *STOC*, 1988.

[BGI$^+$01] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In *CRYPTO*, 2001.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, 2005.

[BLM$^+$24] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. Traitor tracing without trusted authority from registered functional encryption. In *ASIACRYPT*, 2024.

[BWY11] Mihir Bellare, Brent Waters, and Scott Yilek. Identity-based encryption secure against selective opening attack. In *TCC*, 2011.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

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

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In *IMACC*, 2021.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *STOC*, 1996.

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

[CHK03] Ran Canetti, Shai Halevi, and Jonathan Katz. A forward-secure public-key encryption scheme. In *EUROCRYPT*, 2003.

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[CW79] Larry Carter and Mark N. Wegman. Universal classes of hash functions. *J. Comput. Syst. Sci.*, 18(2), 1979.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[DDO$^+$01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In *CRYPTO*, 2001.

[DKL$^+$23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In *EUROCRYPT*, 2023.

[DKW21] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority ABE for DNFs from LWE. In *EUROCRYPT*, 2021.

[DKW23] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority ABE for NC^1 from computational-BDH. *Journal of Cryptology*, 36(2), 2023.

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In *CRYPTO*, 2000.

[DPY24] Pratish Datta, Tapas Pal, and Shota Yamada. Registered FE beyond predicates: (attribute-based) linear functions and more. In *ASIACRYPT*, 2024.

[EFF85] Paul Erdős, Peter Frankl, and Zoltán Füredi. Families of finite sets in which no set is covered by the union of r others. *Israel J. Math*, 51(1-2), 1985.

[FFM$^+$23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. In *ASIACRYPT*, 2023.

[FKdP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. In *ASIACRYPT*, 2023.

[FLS90] Uriel Feige, Dror Lapidot, and Adi Shamir. Multiple non-interactive zero knowledge proofs based on a single random string (extended abstract). In *FOCS*, 1990.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In *CRYPTO*, 2023.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme.* PhD thesis, Stanford University, USA, 2009.

[GGH$^+$13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[GHM$^+$19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *PKC*, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, 2018.

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *ACM CCS*, 2023.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.

[GLWW23] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Realizing flexible broadcast encryption: How to broadcast to a public-key directory. In *ACM CCS*, 2023.

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David Wu. Reducing the CRS size in registered ABE systems. In *CRYPTO*, 2024.

[GOS06] Jens Groth, Rafail Ostrovsky, and Amit Sahai. Perfect non-interactive zero knowledge for NP. In *EUROCRYPT*, 2006.

[GOS12] Jens Groth, Rafail Ostrovsky, and Amit Sahai. New techniques for noninteractive zero-knowledge. *J. ACM*, 59(3), 2012.

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

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

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In *CRYPTO*, 2020.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, 2015.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[KS64] William H. Kautz and Richard C. Singleton. Nonrandom binary superimposed codes. *IEEE Trans. Inf. Theory*, 10(4), 1964.

[LCLS08] Huang Lin, Zhenfu Cao, Xiaohui Liang, and Jun Shao. Secure threshold multi authority attribute based encryption without a central authority. In *INDOCRYPT*, 2008.

[LOS$^+$10] Allison B. 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.

[LPWW20] Benoît Libert, Alain Passelègue, Hoeteck Wee, and David J. Wu. New constructions of statistical NIZKs: Dual-mode DV-NIZKs and more. In *EUROCRYPT*, 2020.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC*, 2010.

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

[LW25] George Lu and Brent Waters. How to make any computational secret sharing scheme adaptively secure. In *CRYPTO*, 2025.

[Mer89] Ralph C. Merkle. A certified digital signature. In *CRYPTO*, 1989.

[MKE08] Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert. Distributed attribute-based encryption. In *ICISC*, 2008.

[PS25] Tapas Pal and Robert Schädlich. Registered functional encryption for attribute-weighted sums with access control. *IACR Cryptol. ePrint Arch.*, 2025.

[RW15] Yannis Rouselakis and Brent Waters. Efficient statically-secure large-universe multi-authority attribute-based encryption. In *Financial Cryptography*, 2015.

[Sah99] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, 1999.

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

[VNS$^+$03] Vinod Vaikuntanathan, Arvind Narayanan, K. Srinathan, C. Pandu Rangan, and Kwangjo Kim. On the power of computational secret sharing. In *INDOCRYPT*, 2003.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, 2009.

[Wei23] Ruizhong Wei. On cover-free families. *arXiv*, 2023.

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[WWW22] Brent Waters, Hoeteck Wee, and David J. Wu. Multi-authority ABE from lattices without random oracles. In *TCC*, 2022.

[Yao89] Andrew Chi-Chih Yao. Unpublished manuscript, 1989.

[ZZC+25] Ziqi Zhu, Kai Zhang, Zhili Chen, Junqing Gong, and Haifeng Qian. Black-box registered ABE from lattices. *IACR Cryptol. ePrint Arch.*, 2025.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In *ASIACRYPT*, 2023.

A Semi-Malicious Security to Full Security

In this section, we describe a simple compiler to generically lift a multi-authority registered ABE scheme with semi-malicious security (Definition 4.9) to one that satisfies the standard security notion (Definition 4.6). Recall that in the semi-malicious security game, the adversary is required to provide the key-generation randomness for the keys it chooses in the security game. The idea is to simply have users attach a non-interactive zero-knowledge (NIZK) proof of knowledge of the key-generation randomness to its public key. In the security proof, the reduction algorithm extracts the key-generation randomness and relies on security of the underlying semi-malicious scheme.

Simulation-sound extractable NIZKs. We start by recalling the formal definition of a simulation-sound extractable NIZK argument [BFM88, FLS90, Sah99, DDO+01].

Definition A.1 (Simulation-Sound Extractable NIZK)**.** Let $\mathcal{R} \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be an NP relation (where $n = n(\lambda)$ and $h = h(\lambda)$ are polynomials). A simulation-sound extractable NIZK for $\mathcal{R}$ is a tuple of efficient algorithms $\Pi_{\mathsf{NIZK}} = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax:

- $\mathsf{CRSGen}(1^\lambda) \to \mathsf{crs}$: On input the security parameter λ, the common reference string generation algorithm outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$: On input a common reference string crs, a statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, the prover algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, x, \pi) \to b$: On input a common reference string crs, a statement $x \in \{0,1\}^n$, and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

Moreover, we require Π_{NIZK} satisfy the following properties:

- **Completeness:** For all $\lambda \in \mathbb{N}$, statements $x \in \{0,1\}^n$, witnesses $w \in \{0,1\}^w$ where $\mathcal{R}(x, w) = 1$, it holds that
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w) \end{array}\right] = 1.$$
- **Perfect soundness:** For all adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$,
$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge x \notin L : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}(1^\lambda, \mathsf{crs}) \end{array}\right] = 0.$$
- **Computational zero-knowledge:** There exists an efficient simulator $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ such that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$\left|\Pr\left[\mathcal{A}^{O_0(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1 : \mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)\right] - \Pr\left[\mathcal{A}^{O_1(\mathsf{st}_{\mathcal{S}},\cdot,\cdot)}(\mathsf{crs}) = 1 : (\mathsf{crs}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)\right]\right| = \mathsf{negl}(\lambda),$$
where the oracles O_0 and O_1 are defined as follows:
  - $O_0(\mathsf{crs}, x, w)$: On input a common reference string crs, a statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, output $\mathsf{Prove}(\mathsf{crs}, x, w)$ if $\mathcal{R}(x, w) = 1$. Otherwise, if $\mathcal{R}(x, w) = 0$, output $\perp$.
  - $O_1(\mathsf{st}_{\mathcal{S}}, x, w)$: On input the simulation state $\mathsf{st}_{\mathcal{S}}$, a statement $x \in \{0,1\}^n$, and a witness $w \in \{0,1\}^h$, output $\mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, x)$ if $\mathcal{R}(x, w) = 1$. Otherwise, if $\mathcal{R}(x, w) = 0$, output $\perp$.

- **Simulation-sound extractability:** There exists an efficient extraction algorithm $\mathcal{E}$ such that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, x, \pi) = 1 \wedge \mathcal{R}(x, w) = 0 \wedge (x, \pi) \notin Q : \begin{array}{c} (\mathsf{crs}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda) \\ (x, \pi) \leftarrow \mathcal{A}^{\mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, \cdot)}(\mathsf{crs}) \\ w \leftarrow \mathcal{E}(\mathsf{st}_{\mathcal{S}}, x, \pi) \end{array}\right] = \mathsf{negl}(\lambda),$$

where $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ is the zero-knowledge simulator (from the computational zero-knowledge requirement) and Q is the set of queries x and simulated outputs π made by $\mathcal{A}$ to the $\mathcal{S}_2$ oracle.

Constructions of simulation-sound extractable NIZKs. A number of works [CHK03, GOS06, GOS12, LPWW20] have shown how to construct NIZKs for general NP languages from standard bilinear map assumptions (e.g., the subgroup decision assumption in composite-order pairing groups or the k-linear assumption in prime-order pairing groups). Moreover, we can achieve simulation-sound extractability via standard compilers (based on public-key encryption and one-time signatures [Sah99, DDO+01]).

Compiling semi-malicious security to full security. We now describe our compiler to lift a semi-malicious multi-authority registered ABE scheme into a fully secure multi-authority registered ABE scheme.

Construction A.2 (Semi-Malicious Multi-Authority Registered ABE to Full Security)**.** Let $\Pi_{\mathsf{SM}} = (\mathsf{SM.GlobalSetup}, \mathsf{SM.KeyGen}, \mathsf{SM.Aggregate}, \mathsf{SMEncrypt}, \mathsf{SM.Decrypt})$ be a semi-maliciously secure slotted multi-authority registered ABE scheme on authority identifier space $\mathcal{AU} = \{\mathcal{AU}_\lambda\}$, user identifier space $\mathcal{GID} = \{\mathcal{GID}_\lambda\}$, and policy space $\Phi = \{\Phi_\lambda\}$. Let $\Pi_{\mathsf{NIZK}} = (\mathsf{CRSGen}, \mathsf{Prove}, \mathsf{Verify})$ be a simulation-sound extractable NIZK for the relation

$$\mathcal{R}((\mathsf{gpp}, \mathsf{pk}, i), (\mathsf{sk}, r)) = 1 \quad \text{if and only if} \quad (\mathsf{pk}, \mathsf{sk}) = \mathsf{SM.KeyGen}(\mathsf{gpp}, i; r).$$

Let $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ be the associated zero-knowledge simulator and $\mathcal{E}$ be the extractor (see Definition A.1). We construct a slotted multi-authority registered ABE scheme $\Pi_{\mathsf{sMA\text{-}RABE}} = (\mathsf{GlobalSetup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with authority identity space $\mathcal{AU}$, user identifier space $\mathcal{GID}$, and policy space Φ as follows:

- $\mathsf{GlobalSetup}(1^\lambda, 1^L)$: On input the security parameter λ and the slot bound L, the setup algorithm initializes the underlying semi-malicious scheme and generates the common reference string for the NIZK:
  - Sample $(\mathsf{SM.gpp}, \mathsf{SM.gep}) \leftarrow \mathsf{SM.GlobalSetup}(1^\lambda, 1^L)$.
  - Sample $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$.

  Output $\mathsf{gpp} = (\mathsf{SM.gpp}, \mathsf{crs})$ and $\mathsf{gep} = \mathsf{SM.gep}$.

- $\mathsf{KeyGen}(\mathsf{gpp}, i)$: On input the global public parameters $\mathsf{gpp} = (\mathsf{SM.gpp}, \mathsf{crs})$, a slot index $i \in [L]$, the key-generation algorithm runs the underlying semi-malicious key generation algorithm and then provides a NIZK proof that the key is well-formed:
  - Let ρ be a bound on the number of bits of randomness taken by $\mathsf{SM.KeyGen}(\mathsf{SM.gpp}, \cdot)$. Sample $r \xleftarrow{\mathrm{R}} \{0,1\}^\rho$ and $(\mathsf{SM.pk}, \mathsf{SM.sk}) = \mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i; r)$.
  - Construct a proof $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, (\mathsf{SM.gpp}, \mathsf{SM.pk}, i), (\mathsf{SM.sk}, r))$.

  Output the public key $\mathsf{pk} = (\mathsf{SM.pk}, \pi)$ and the secret key $\mathsf{sk} = \mathsf{SM.sk}$.

- $\mathsf{IsValid}(\mathsf{gpp}, i, \mathsf{pk})$: On input the global public parameters $\mathsf{gpp} = (\mathsf{SM.gpp}, \mathsf{crs})$, the slot index $i \in [L]$, and a public key pk, the validity checking algorithm outputs $\mathsf{Verify}(\mathsf{crs}, (\mathsf{SM.gpp}, \mathsf{SM.pk}, i), \pi)$.

- $\mathsf{Aggregate}(\mathsf{gpp}, (\mathsf{gid}_1, \mathsf{pk}_1), \ldots, (\mathsf{gid}_L, \mathsf{pk}_L))$: On input the global public parameters $\mathsf{gpp} = (\mathsf{SM.gpp}, \mathsf{crs})$ and a collection of pairs $(\mathsf{gid}_i, \mathsf{pk}_i)$ for $i \in [L]$, the aggregation algorithm simply runs the underlying semi-malicious aggregation algorithm. In more detail, it computes and outputs

$$(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{SM.Aggregate}(\mathsf{SM.gpp}, (\mathsf{gid}_1, \mathsf{SM.pk}_1), \ldots, (\mathsf{gid}_L, \mathsf{SM.pk}_L)).$$

- $\mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu)$: On input the encryption parameters $\mathsf{gep} = \mathsf{SM.gep}$, the policy $(S_{\mathsf{enc}}, \varphi)$, a collection of authority public keys $\mathsf{mpk}_{\mathsf{aid}}$, and a message μ, the encryption algorithm output
$$\mathsf{ct} \leftarrow \mathsf{SM.Encrypt}(\mathsf{SM.gep}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu).$$
- $\mathsf{Decrypt}(\mathsf{gpp}, S, i, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S}, \mathsf{ct})$: On input the global public parameters $\mathsf{gpp} = (\mathsf{SM.gpp}, \mathsf{crs})$, a set of authorities S, an index $i \in [L]$, and a collection of secret keys $\mathsf{sk}_{\mathsf{aid}}$ and helper decryption keys $\mathsf{hsk}_{\mathsf{aid}}$ for $\mathsf{aid} \in S$, and a ciphertext ct, the decryption algorithm outputs
$$m \leftarrow \mathsf{SM.Decrypt}(\mathsf{SM.gpp}, S, i, \{(\mathsf{aid}, \mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S}, \mathsf{ct}).$$

Theorem A.3 (Completeness, Correctness, and Compactness)**.** *If Π_{NIZK} is complete, then Construction A.2 is complete. Moreover, if Π_{SM} satisfies correctness and compactness, and Π_{NIZK} satisfies perfect soundness, then Construction A.2 is correct and compact.*

Proof. We show each property individually. Completeness follows by completeness of the NIZK Π_{NIZK} while correctness and compactness are directly inherited from the underlying semi-malicious scheme.

- **Completeness:** By construction $\mathsf{IsValid}(\mathsf{gpp}, i, (\mathsf{SM.pk}, \pi))$ simply checks that
$$\mathsf{Verify}(\mathsf{crs}, (\mathsf{SM.gpp}, \mathsf{SM.pk}, i), \pi) = 1,$$
where $\pi = \mathsf{Prove}(\mathsf{crs}, (\mathsf{SM.gpp}, \mathsf{SM.pk}, i), (\mathsf{SM}, \mathsf{sk}, r))$ and $(\mathsf{SM.pk}, \mathsf{SM.sk}) = \mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i; r)$. By construction of the relation $\mathcal{R}$, we have $\mathcal{R}((\mathsf{SM.gpp}, \mathsf{SM.pk}, i), (\mathsf{SM}, \mathsf{sk}, r)) = 1$. Completeness now follows from perfect completeness of Π_{NIZK}.
- **Correctness:** Observe that the output of Aggregate, Encrypt, and Decrypt are exactly the same as the underlying semi-malicious scheme. Moreover, by perfect soundness of Π_{NIZK}, every public key pk where $\mathsf{IsValid}(\mathsf{gpp}, i, \mathsf{pk})$ is in the support of $\mathsf{KeyGen}(\mathsf{gpp}, i)$ (and thus, the support of $\mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i)$). Correctness now by correctness of the underlying semi-malicious scheme.
- **Compactness:** Since the global encryption parameters and the master public keys are exactly the same as the underlying semi-malicious scheme, this follows from compactness of the underlying scheme. □

Theorem A.4 (Security)**.** *Suppose Π_{SM} is a semi-maliciously-secure multi-authority registered ABE scheme and Π_{NIZK} is a simulation-sound extractable NIZK. Then Construction A.2 is a secure multi-authority registered ABE scheme.*

Proof. To argue security of our slotted multi-authority registered ABE scheme, we define a sequence of hybrid experiments. Each experiment is parameterized by an implicit security parameter λ, a bit $b \in \{0, 1\}$, and an adversary $\mathcal{A}$:

- $\mathsf{Hyb}_0^{(b)}$: This is the standard multi-authority ABE security experiment. We recall the main steps here:
  - **Setup phase:** On input the security parameter 1^λ, adversary $\mathcal{A}$ sends the slot count 1^L to the challenger. The challenger samples gpp and gep according to the specification of the real setup algorithm. Specifically, it samples $(\mathsf{SM.gpp}, \mathsf{SM.gep}) \leftarrow \mathsf{SM.GlobalSetup}(1^\lambda, 1^L)$ and $\mathsf{crs} \leftarrow \mathsf{CRSGen}(1^\lambda)$. The challenger sends $\mathsf{gpp} = (\mathsf{SM.gpp}, \mathsf{crs})$ and $\mathsf{gep} = \mathsf{SM.gep}$ to the adversary. The challenger also initializes a counter $\mathsf{ctr} \leftarrow 0$, a dictionary D, a set of honest authorities $\mathcal{H} \leftarrow \varnothing$, and a set of tuples $C \leftarrow \varnothing$.
  - **Pre-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
    - **Key-generation query:** In a key-generation query, the adversary specifies a slot index $i \in [L]$. The challenger responds by incrementing the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$, sampling $r_{\mathsf{ctr}} \xleftarrow{\mathrm{R}} \{0, 1\}^\rho$, and computing
$$(\mathsf{SM.pk}_{\mathsf{ctr}}, \mathsf{SM.sk}_{\mathsf{ctr}}) = \mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i; r_{\mathsf{ctr}})$$
$$\pi_{\mathsf{ctr}} \leftarrow \mathsf{Prove}(\mathsf{crs}, (\mathsf{SM.pk}_{\mathsf{ctr}}, \mathsf{SM.gpp}, i), (\mathsf{SM.sk}_{\mathsf{ctr}}, r_{\mathsf{ctr}})).$$
It replies with $\mathsf{pk}_{\mathsf{ctr}} = (\mathsf{ctr}, (\mathsf{SM.pk}_{\mathsf{ctr}}, \pi_{\mathsf{ctr}}))$ to $\mathcal{A}$. In addition, the challenger adds the mapping $\mathsf{ctr} \mapsto (i, (\mathsf{SM.pk}_{\mathsf{ctr}}, \pi_{\mathsf{ctr}}), \mathsf{SM.sk}_{\mathsf{ctr}})$ to the dictionary D.

    * **Key-corruption query:** In a key-corruption query, the adversary specifies an index $1 \leq c \leq \mathsf{ctr}$. In response, the challenger looks up the tuple $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[c]$ and replies to $\mathcal{A}$ with sk'.

  - **Challenge phase:** In the challenge phase, the adversary specifies a challenge policy $(S_{\mathsf{enc}}, \varphi^*)$ and two messages $\mu_0^*, \mu_1^* \in \mathcal{M}$. For each $\mathsf{aid} \in S_{\mathsf{enc}}$ and slot $i \in [L]$, the adversary specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}^*_{i,\mathsf{aid}})$. The challenger responds by constructing $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{SM.pk}_{i,\mathsf{aid}}, \pi_{i,\mathsf{aid}}$ as follows:
    * If $c_{i,\mathsf{aid}} \in \{1, \ldots, \mathsf{ctr}\}$, then the challenger looks up $(i', \mathsf{aid}', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[c_{i,\mathsf{aid}}]$ and sets $\mathsf{pk}_{i,\mathsf{aid}} = \mathsf{pk}'$. The challenger adds (i, aid) to C if the adversary issued a key-corruption query on index $c_{i,\mathsf{aid}}$.
    * If $c_{i,\mathsf{aid}} = \perp$, then the challenger adds (i, aid) to C
    * In both cases, the challenger then checks that $\mathsf{IsValid}(\mathsf{gpp}, i, \mathsf{pk}_{i,\mathsf{aid}}) = 1$. Specifically, the challenger checks that $\mathsf{Verify}(\mathsf{crs}, (\mathsf{SM.pk}_{i,\mathsf{aid}}, \mathsf{SM.gpp}, i), \pi_{i,\mathsf{aid}}) = 1$. If not, the challenger halts with output 0.

    The challenger computes $(\mathsf{mpk}_{\mathsf{aid}}, \mathsf{hsk}_{1,\mathsf{aid}}, \ldots, \mathsf{hsk}_{L,\mathsf{aid}}) \leftarrow \mathsf{SM.Aggregate}(\mathsf{SM.gpp}, \{(i, \mathsf{SM.pk}_{i,\mathsf{aid}})\}_{i \in [L]})$ for all $\mathsf{aid} \in S_{\mathsf{enc}}$ and replies with the ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{gep}, (S_{\mathsf{enc}}, \varphi^*)\{\mathsf{mpk}_{\mathsf{aid}}\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu_b^*)$.

  - **Post-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
    * **Key-corruption query:** Same as in the pre-challenge query phase. If the adversary corrupts a key with index $c_{i,\mathsf{aid}}$ and the adversary specified $c_{i,\mathsf{aid}}$ in the challenge phase, then the challenger adds (i, aid) to the corruption set C.

  - **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

- $\mathsf{Hyb}_1^{(b)}$: In this experiment, except the challenger uses a simulated CRS in the global parameters and simulated proofs when answering key-generation queries. The specific changes are as follows:
  - **Setup phase:** Same as $\mathsf{Hyb}_0^{(b)}$, except the challenger now samples $(\mathsf{crs}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$, where $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$ is the simulator associated with Π_{NIZK}.
  - **Pre-challenge query phase:**
    * **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query, the challenger replaces the proof with a simulated one. Specifically, it now computes $\pi_{\mathsf{ctr}} \leftarrow \mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, (\mathsf{SM.pk}_{\mathsf{ctr}}, \mathsf{SM.gpp}, i))$.

- $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$, except in the challenge phase, the challenger aborts if the adversary $\mathcal{A}$ produces a key that is valid (as determined by the $\mathsf{IsValid}$ function), but the extraction algorithm fails to produce the key-generation randomness associated with the provided key. Specifically, the challenger performs the additional check in the challenge phase:
  - **Challenge phase:** In addition to checking that $\mathsf{Verify}(\mathsf{crs}, (\mathsf{SM.pk}_{i,\mathsf{aid}}, \mathsf{SM.gpp}, i), \pi_{i,\mathsf{aid}}) = 1$, the challenger additionally checks that either
    * $\mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i; r_{i,\mathsf{aid}}) = (\mathsf{SM.pk}_{i,\mathsf{aid}}, \mathsf{SM.sk}_{\mathsf{aid}})$ where
    $$(\mathsf{SM.sk}_{\mathsf{aid}}, r_{i,\mathsf{aid}}) \leftarrow \mathcal{E}(\mathsf{st}_{\mathcal{S}}, (\mathsf{SM.pk}_{i,\mathsf{aid}}, \mathsf{SM.gpp}, i), \pi_{i,\mathsf{aid}})$$
    * $(\mathsf{SM.pk}_{i,\mathsf{aid}}, \pi_{i,\mathsf{aid}}) = (\mathsf{SM.pk}_{\mathsf{ctr}_{i,\mathsf{aid}}}, \pi_{\mathsf{ctr}_{i,\mathsf{aid}}})$ for some $\mathsf{ctr}_{i,\mathsf{aid}} \in [\mathsf{ctr}]$.

    If either check fails, the challenger aborts with output 0.

Lemma A.5. *Suppose Π_{NIZK} satisfies computational zero knowledge. Then, for all efficient adversaries $\mathcal{A}$ and all $b \in \{0, 1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] \right| \leq \mathsf{negl}(\lambda).$$

Proof. Suppose there exists an adversary $\mathcal{A}$ which can distinguish between $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$ with non-negligible probability. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks computational zero knowledge with the same advantage:

- At the beginning of the game, algorithm $\mathcal{B}$ receives a common reference string crs, and has access to a prover oracle $\mathcal{O}(\cdot,\cdot)$.
- Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ and simulates an execution of the multi-authority registered ABE security game as follows:
  - **Setup Phase:** Same as $\mathsf{Exp}_0^{(b)}$, except algorithm $\mathcal{B}$ uses crs received from the challenger when constructing the global parameters gpp.
  - **Pre-challenge query phase:**
    - **Key-generation query:** Same as $\mathsf{Hyb}_0^{(b)}$, except algorithm $\mathcal{B}$ generates the proof π_{ctr} by querying its oracle. Namely, algorithm $\mathcal{B}$ computes $\pi_{\mathsf{ctr}} \leftarrow \mathcal{O}((\mathsf{SM.pk}_{\mathsf{ctr}}, \mathsf{SM.gpp}, i), (\mathsf{SM.sk}_{\mathsf{ctr}}, r_{\mathsf{ctr}}))$.
    - **Key-corruption query:** Algorithm $\mathcal{B}$ responds to key-corruption queries exactly as described in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$.
  - **Challenge phase:** Algorithm $\mathcal{B}$ implements the challenge phase exactly as described in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$.
  - **Post-challenge query phase:** Algorithm $\mathcal{B}$ responds to post-challenge queries exactly as described in $\mathsf{Hyb}_0^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$.
  - **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

By construction, if the CRS is generated using CRSGen and the oracle is implemented by $\mathsf{Prove}(\mathsf{crs},\cdot,\cdot)$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_0^{(b)}$. If the CRS and the proofs are simulated, then $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_1^{(b)}$. The claim follows. □

Lemma A.6. *Suppose Π_{NIZK} satisfies simulation-sound extractability. Then, for all efficient adversaries $\mathcal{A}$ and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. We define an intermediate sequence of hybrid experiments. Let $S_{\mathsf{enc}} = \{\mathsf{aid}_1, \ldots, \mathsf{aid}_{|S_{\mathsf{enc}}|}\}$ be the challenge set of authorities. Then, we define an intermediate experiment as follows:

- $\mathsf{Hyb}_{1,(i^*,a^*)}^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except the challenger implements the challenge phase as follows:
  - **Challenge phase:** For slots where $i < i^*$ or where ($i = i^*$ and $\mathsf{aid} = \mathsf{aid}_a$ for some $a < a^*$), the challenger additionally checks that either
    - $\mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i; r_{i,\mathsf{aid}}) = (\mathsf{SM.pk}_{i,\mathsf{aid}}, \mathsf{SM.sk}_{\mathsf{aid}})$ where
    $$(\mathsf{SM.sk}_{\mathsf{aid}}, r_{i,\mathsf{aid}}) \leftarrow \mathcal{E}(\mathsf{st}_{\mathcal{S}}, (\mathsf{SM.pk}_{i,\mathsf{aid}}, \mathsf{SM.gpp}, i), \pi_{i,\mathsf{aid}})$$
    - $(\mathsf{SM.pk}_{i,\mathsf{aid}}, \pi_{i,\mathsf{aid}}) = (\mathsf{SM.pk}_{\mathsf{ctr}_{i,\mathsf{aid}}}, \pi_{\mathsf{ctr}_{i,\mathsf{aid}}})$ for some $\mathsf{ctr}_{i,\mathsf{aid}} \in [\mathsf{ctr}]$.

    If either check fails, then the challenger halts with output 0.

Observe that $\mathsf{Hyb}_{1,(1,1)}^{(b)}$ is identical to $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_{1,(L,|S_{\mathsf{enc}}|+1)}^{(b)}$ is identical to $\mathsf{Hyb}_2^{(b)}$. Now we show that adjacent experiments are computationally indistinguishable as long as Π_{NIZK} is simulation-sound extractable. Suppose there exists an adversary $\mathcal{A}$ which can distinguish between $\mathsf{Hyb}_{1,(i^*,a^*-1)}^{(b)}$ and $\mathsf{Hyb}_{1,(i^*,a^*)}^{(b)}$ with non-negligible probability. We use $\mathcal{A}$ to construct an adversary for the simulation-sound extractability game:

- On input the security parameter 1^λ and the common reference string crs, algorithm $\mathcal{B}$ starts simulating an execution of $\mathsf{Hyb}_1^{(b)}$. In the setup phase, it uses crs to simulate the public parameters gpp.
- When answering key-generation queries, algorithm $\mathcal{B}$ uses oracle access to $\mathcal{S}_2(\mathsf{st}_{\mathcal{S}},\cdot)$ to simulate the proof π_{ctr}. Everything else is simulated according to the specification of $\mathsf{Hyb}_1^{(b)}$.

- In the challenge phase, let $\mathsf{aid}^* = \mathsf{aid}_{a^*}$. After $\mathcal{A}$ outputs its challenge tuples $(c_{i^*,\mathsf{aid}^*}, \mathsf{gid}_{i^*,\mathsf{aid}^*}, \mathsf{pk}_{i^*,\mathsf{aid}^*})$, parse $\mathsf{pk}_{i^*,\mathsf{aid}^*} = \mathsf{SM.pk}_{i^*,\mathsf{aid}^*}, \pi_{i^*,\mathsf{aid}^*}$ and output the statement $(\mathsf{SM.pk}_{i^*,\mathsf{aid}^*}, \mathsf{SM.gpp}, i^*)$ along with the proof $\pi_{i^*,\mathsf{aid}^*})$.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}^{(b)}_{1,(i^*,a^*-1)}$ and $\mathsf{Hyb}^{(b)}_{1,(i^*,a^*)}$ for algorithm $\mathcal{A}$. Next, the only difference between the two experiments is the additional check for index $i = i^*$ and $a = a^*$ in $\mathsf{Hyb}^{(b)}_{1,(i^*,a^*)}$. Thus, if the outputs of this experiment are different, then $\mathcal{A}$ must have produced a public key $\mathsf{SM.pk}_{i^*,\mathsf{aid}^*}$ and a proof π_{i^*,aid^*} such that the following three conditions all hold:

- $\mathsf{SM.pk}_{i^*,\mathsf{aid}^*}$ was not the result of a prior key-generation query;
- $(\mathsf{SM.pk}_{i^*,\mathsf{aid}^*}, \mathsf{SM.sk}_{\mathsf{aid}}) \neq \mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i^*; r_{i^*,\mathsf{aid}^*})$, where

$$(\mathsf{SM.sk}_{\mathsf{aid}}, r_{i^*,\mathsf{aid}^*}) \leftarrow \mathcal{E}(\mathsf{st}_{\mathcal{S}}, (\mathsf{SM.pk}_{i^*,\mathsf{aid}^*}, \mathsf{SM.gpp}, i^*), \pi_{i^*,\mathsf{aid}^*});$$

- $\mathsf{Verify}(\mathsf{crs}, (\mathsf{SM.pk}_{i^*,\mathsf{aid}^*}, \mathsf{SM.gpp}, i^*), \pi_{i^*,\mathsf{aid}^*}) = 1$.

This means $\mathcal{B}$ breaks simulation-sound extractability of Π_{NIZK} and the claim holds. Finally, since there are $|S_{\mathsf{enc}}|$ authorities (i.e., $a \in [|S_{\mathsf{enc}}|]$), experiments $\mathsf{Hyb}^{(b)}_{1,(i^*-1,|S_{\mathsf{enc}}|)}$ and $\mathsf{Hyb}^{(b)}_{1,(i^*,1)}$ are identical. The claim now follows by a hybrid argument. ◻

Lemma A.7. *Suppose* Π_{SM} *is a semi-maliciously-secure multi-authority registered ABE scheme. Then, for all efficient and admissible adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,*

$$\left|\Pr[\mathsf{Hyb}^{(1)}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(0)}_2(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Suppose there exists an efficient algorithm $\mathcal{A}$ that can distinguish between experiments $\mathsf{Hyb}^{(0)}_2$ and $\mathsf{Hyb}^{(1)}_2$ with non-negligible advantage. We use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ that achieves the same advantage:

- **Setup phase:** On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on the same security parameter. Algorithm $\mathcal{A}$ outputs the number of slots 1^L, which algorithm $\mathcal{B}$ forwards to the challenger. The challenger responds with $\mathsf{SM.gpp}$ and $\mathsf{SM.gep}$. Algorithm $\mathcal{B}$ samples $(\mathsf{crs}, \mathsf{st}_{\mathcal{S}}) \leftarrow \mathcal{S}_1(1^\lambda)$ and sets $\mathsf{gpp} = (\mathsf{SM.gpp}, \mathsf{crs})$ and $\mathsf{gep} = \mathsf{SM.gep}$. It gives gpp and gep to $\mathcal{A}$. In addition, algorithm $\mathcal{B}$ also initializes an empty dictionary Q.
- **Pre-challenge query phase:** Algorithm $\mathcal{B}$ responds to queries as follows:
  - **Key-generation query:** After algorithm $\mathcal{A}$ outputs an index i, algorithm $\mathcal{B}$ makes a key-generation query to its challenger. The challenger responds with a pair $(\mathsf{ctr}, \mathsf{SM.pk}_i)$. Algorithm $\mathcal{B}$ runs $\pi_i \leftarrow \mathcal{S}_2(\mathsf{st}_{\mathcal{S}}, (\mathsf{SM.gpp}, \mathsf{SM.pk}_i, i))$, and adds the mapping $(\mathsf{SM.pk}_i, \pi_i) \mapsto \mathsf{ctr}$ to Q. Algorithm $\mathcal{B}$ replies to $\mathcal{A}$ with $(\mathsf{ctr}, (\mathsf{SM.pk}_i, \pi_i))$.
  - **Key-corruption query:** When $\mathcal{A}$ makes a key-corruption query on a counter value c, algorithm $\mathcal{B}$ makes a key-corruption query on the same counter c. The challenger responds with $\mathsf{SM.sk}$, which algorithm $\mathcal{B}$ forwards to $\mathcal{A}$.
- **Challenge phase:** In the challenge phase, the adversary specifies a policy $(S_{\mathsf{enc}}, \varphi^*)$ and two messages $\mu_0^*, \mu_1^* \in \mathcal{M}$. In addition, for each $\mathsf{aid} \in S_{\mathsf{enc}}$ and slot $i \in [L]$, the adversary specifies a tuple $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}^*_{i,\mathsf{aid}})$. Algorithm $\mathcal{B}$ forwards the policy $(S_{\mathsf{enc}}, \varphi^*)$ and challenge messages μ_0^*, μ_1^* to its challenger. It constructs the challenge tuples for each $\mathsf{aid} \in S_{\mathsf{enc}}$ and slot $i \in [L]$ as follows:
  - If $c_{i,\mathsf{aid}} \in [\mathsf{ctr}]$, algorithm $\mathcal{B}$ simply forwards $(c_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{pk}^*_{i,\mathsf{aid}})$ to its challenger.
  - If $c_{i,\mathsf{aid}} = \perp$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_{i,\mathsf{aid}} = (\mathsf{SM.pk}_{i,\mathsf{aid}}, \pi_{i,\mathsf{aid}})$. We consider two possibilities:
    1. If $\mathsf{SM.pk}_{i,\mathsf{aid}} = \mathsf{SM.pk}_{\mathsf{ctr}_{i,\mathsf{aid}}}$ and $\pi_{i,\mathsf{aid}} = \pi_{\mathsf{ctr}_{i,\mathsf{aid}}}$ for some $\mathsf{ctr}_{i,\mathsf{aid}} \in [\mathsf{ctr}]$, then algorithm $\mathcal{B}$ forwards the tuple $(\mathsf{ctr}_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{SM.pk}_{i,\mathsf{aid}})$ to C.

2. Otherwise, algorithm $\mathcal{B}$ runs $(\mathsf{SM.sk}_{i,\mathsf{aid}}, r_{i,\mathsf{aid}}) \leftarrow \mathcal{E}(\mathsf{st}_{\mathcal{S}}, (\mathsf{SM.gpp}, \mathsf{SM.pk}_{i,\mathsf{aid}}, i), \pi_{i,\mathsf{aid}})$. If

$$\mathsf{SM.KeyGen}(\mathsf{SM.gpp}, i; r_{i,\mathsf{aid}}) \neq (\mathsf{SM.pk}_{i,\mathsf{aid}}, \mathsf{SM.sk}_{i,\mathsf{aid}}),$$

then algorithm $\mathcal{B}$ aborts with output 0. Otherwise it gives the tuple $(\bot, r_{i,\mathsf{aid}}, \mathsf{gid}_{i,\mathsf{aid}}, \mathsf{SM.pk}_{i,\mathsf{aid}})$ to its challenger.

The challenger responds with a challenge ciphertext ct^*, which $\mathcal{B}$ forwards to $\mathcal{A}$.

- **Post-challenge query phase:** Algorithm $\mathcal{B}$ responds to post-challenge queries using the same procedure as in the pre-challenge query phase.
- **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

By construction, algorithm $\mathcal{B}$ perfectly simulates an execution of the semi-malicious multi-authority registered ABE security game. Since algorithm $\mathcal{B}$ corrupts the same set of keys as $\mathcal{A}$, it is admissible whenever $\mathcal{A}$ is admissible. As such, if ct^* is an encryption of μ_0^*, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_2^{(0)}$, whereas if ct^* is an encryption of μ_1^*, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_2^{(1)}$. The claim follows. □

Theorem A.4 now follows from Lemmas A.5 to A.7. □

B Lifting a Slotted to an Unslotted Scheme (Unbounded)

In this section, we give a simplified version of the transformation from Section 6 that supports an unbounded number of users (i.e., this transformation can be applied to our $i\mathcal{O}$ construction from Construction 7.6 to obtain a multi-authority registered ABE scheme that supports an unbounded number of users). Since our $i\mathcal{O}$ scheme does not require public keys to be generated with respect to a particular slot, and decryption correctness holds regardless of which "slots" a particular user identifier are registered to between authorities, we can omit the use of cover-free sets in the transformation below. For this reason, this transformation also preserves the succinct decryption property in the underlying scheme.

Construction B.1 (Slotted Multi-Authority Registered ABE to Unslotted Scheme). Let λ be a security parameter. Let $L = L(\lambda)$ be the number of users, $\mathcal{AU} = \{\mathcal{AU}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of authority identifiers, and $\mathcal{GID} = \{\mathcal{GID}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of global user identifiers where $|\mathcal{GID}_\lambda| \le 2^\lambda$. We use the following conventions and primitives in our construction:

- Without loss of generality, we assume that the bound on the number of users $L = 2^\ell$ is a power of two. Rounding the bound to the next power of two incurs at most a factor of 2 overhead.
- Let $\mathcal{AU}' = \{\mathcal{AU}'_\lambda\}$ be a set of authority identifiers where $\mathcal{AU}'_\lambda = \mathcal{AU}_\lambda \times [0,\ell]$. Next, let $\Pi_{\mathsf{sRABE}} = (\mathsf{sRABE.GlobalSetup}, \mathsf{sRABE.KeyGen}, \mathsf{sRABE.IsValid}, \mathsf{sRABE.Aggregate}, \mathsf{sRABE.Encrypt}, \mathsf{sRABE.Decrypt})$ be a slotted multi-authority registered ABE scheme with authority identifiers $\mathcal{AU}'$ and global identifier space $\mathcal{GID}$. For ease of exposition, we will assume that Π_{sRABE} supports polynomial-size Boolean circuits.
- The multi-authority registered ABE scheme will internally maintain $\ell + 1$ slotted ABE schemes, where the k^{th} scheme is a slotted scheme that will "support" 2^k users. The slotted scheme will have authority identifier universe $\mathcal{AU}' = \mathcal{AU} \times [0,\ell]$.
- Each authority contains auxiliary data $\mathsf{aux}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{D}_{0,\mathsf{aid}}, \mathsf{D}_{1,\mathsf{aid}}, \mathsf{mpk}_{\mathsf{aid}})$ which contains the following data:
  - A counter $\mathsf{ctr}_{\mathsf{aid}}$ that keeps track of the number of registered users in the system.
  - A dictionary $\mathsf{D}_{0,\mathsf{aid}}$ that maps global identifiers gid to a counter ctr (indicating the number of registered users at the time user gid registered).
  - A dictionary $\mathsf{D}_{1,\mathsf{aid}}$ that maps a counter value ctr to the corresponding set of public keys and helper decryption keys $(\mathsf{gid}, \{(k, \mathsf{pk}_k, \mathsf{hsk}_k)\}_{k \in [0,\ell]})$.

- The current master public key $\mathsf{mpk}_{\mathsf{aid}} = (\mathsf{ctr}, \mathsf{mpk}_0, \dots, \mathsf{mpk}_\ell)$.

  If $\mathsf{aux} = \perp$, we parse it as $(\mathsf{ctr}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{mpk})$ where $\mathsf{ctr} = 0$, $\mathsf{D}_0, \mathsf{D}_1 = \varnothing$, and $\mathsf{mpk} = (0, \perp, \dots, \perp)$. This corresponds to a fresh scheme with no registered users.

- The master public keys $\mathsf{mpk}_{\mathsf{aid}}$ associated with each authority will take the form of $(\mathsf{ctr}, \mathsf{mpk}_0, \dots \mathsf{mpk}_\ell)$, where ctr is the total number of users registered to this authority, and each mpk_k is a master public key of the underlying slotted scheme Π_{sRABE}.
- Helper decryption keys $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid}}$ will take the form of $(\mathsf{ctr}, \mathsf{hsk}_0, \dots \mathsf{hsk}_\ell)$, where ctr is a counter (indicating the number of registered users at the time user gid registered) and each hsk_k is a helper decryption key for the underlying slotted scheme Π_{sRABE}.

We now construct a multi-authority registered ABE scheme $\Pi_{\mathsf{MA\text{-}RABE}} = (\mathsf{GlobalSetup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{UpdateKey}, \mathsf{Encrypt}, \mathsf{Decrypt})$ as follows:

- $\mathsf{Setup}(1^\lambda, L)$: On input the security parameter λ, and a bound on the number of users $L = 2^\ell$, the setup algorithm runs the setup algorithm for the slotted scheme:

$$\mathsf{sRABE.gpp} \leftarrow \mathsf{sRABE.GlobalSetup}(1^\lambda, 1^L).$$

  Then, it samples "dummy" public keys for each slot $j \in [L]$ and $k \in [0, \ell]$:

$$\left(\mathsf{pk}'_{j,k}, \mathsf{sk}'_{j,k}\right) \leftarrow \mathsf{sRABE.KeyGen}(\mathsf{sRABE.gpp}, j).$$

  It outputs the global public parameters

$$\mathsf{gpp} = (\mathsf{sRABE.gpp}, \{\mathsf{pk}'_{j,k}\}_{j \in [L], k \in [0,\ell]}).$$

- $\mathsf{KeyGen}(\mathsf{gpp}, \mathsf{gid})$: On input the common reference string $\mathsf{gpp} = \mathsf{sRABE.gpp}$ and a user identifier gid, the key-generation algorithm generates several public/secret key-pairs for the underlying scheme. Specifically, for $k \in [0, \ell]$, it generates $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{sRABE.KeyGen}(\mathsf{sRABE.gpp})$. Output $\mathsf{pk} = \{(k, \mathsf{pk}_k)\}_{k \in [0,\ell]}$ and $\mathsf{sk} = \{(k, \mathsf{sk}_k)\}_{k \in [0,\ell]}$.
- $\mathsf{RegPK}(\mathsf{gpp}, \mathsf{aux}_{\mathsf{aid}}, \mathsf{gid}, \mathsf{pk})$: On input the common reference string $\mathsf{gpp} = (\mathsf{sRABE.gpp}, \{\mathsf{pk}'_{j,k}\}_{j \in [L], k \in [0,\ell]})$, the auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{mpk})$, a user identifier gid and a public key $\mathsf{pk} = (\{(k, \mathsf{pk}_k)\}_{k \in [0,\ell]})$, the registration algorithm proceeds as follows:
  - For each $k \in [0, \ell]$, check that $\mathsf{sRABE.IsValid}(\mathsf{sRABE.gpp}, \mathsf{pk}_k) = 1$. Otherwise, the algorithm halts and outputs the current auxiliary data aux and master public key mpk.
  - If $\mathsf{D}_0[\mathsf{gid}] \neq \perp$, the algorithm halts and outputs the current auxiliary data aux and master public key mpk. Otherwise, update $\mathsf{D}_0[\mathsf{gid}] = \mathsf{ctr}$.
  - For $k^* \in [0, \ell]$, check if $\mathsf{ctr} + 1 = 0 \bmod 2^{k^*}$. If so, registration algorithm performs the following update procedure:
    * For $i \in [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$, look up $(\mathsf{gid}_i, \{(\mathsf{pk}_{i,k}, \mathsf{hsk}_{i,k})\}_{k \in [0,\ell]}) = \mathsf{D}_0[i]$, and set

$$\mathsf{pk}^*_i = \mathsf{pk}_{i,k^*} \quad \text{and} \quad \mathsf{gid}^*_i = \mathsf{gid}_i.$$

      Otherwise set

$$\mathsf{pk}^*_i = \mathsf{pk}'_{i,k} \quad \text{and} \quad \mathsf{gid}_i = 0.$$

    * Compute the aggregated parameters

$$\left(\mathsf{mpk}^*_k, \{(i, \mathsf{hsk}^*_{i,k})\}_{i \in [L]}\right) = \mathsf{sRABE.Aggregate}(\mathsf{sRABE.gpp}, (\mathsf{gid}^*_1, \mathsf{pk}^*_1), \dots, (\mathsf{gid}^*_L, \mathsf{pk}^*_L)).$$

    * For each $i \in [\mathsf{ctr} - 2^k + 1, \mathsf{ctr}]$, update $\mathsf{hsk}_{i,k^*} = \mathsf{hsk}^*_{\mathsf{ctr},k^*}$ in D_1. Finally, update $\mathsf{mpk}_k = \mathsf{mpk}^*_k$

- Finally, the registration algorithm increments $\mathsf{ctr} = \mathsf{ctr} + 1$ and outputs the new master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots \mathsf{mpk}_\ell)$ and updated auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{mpk})$.

- $\mathsf{UpdateKey}(\mathsf{gpp}, \mathsf{aux}_{\mathsf{aid}}, \mathsf{gid})$: On input the global public parameters gpp, the auxiliary data for an authority $\mathsf{aux}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_0, \mathsf{D}_1, \mathsf{mpk})$, and a user identifier gid, the update algorithm looks up $\mathsf{ctr}_{\mathsf{gid}} = \mathsf{D}_0[\mathsf{gid}]$. It then looks up $(\mathsf{gid}, \{(\mathsf{pk}_k, \mathsf{hsk}_k\}_{k \in [0,\ell]}) = \mathsf{D}_1[\mathsf{ctr}_{\mathsf{gid}}]$. The update algorithm outputs $(\mathsf{ctr}_{\mathsf{gid}}, \mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell)$.

- $\mathsf{Encrypt}(\mathsf{gpp}, (S_{\mathsf{enc}}, \varphi), \{(\mathsf{aid}, \mathsf{mpk}_{\mathsf{aid}})\}_{\mathsf{aid} \in S_{\mathsf{enc}}}, \mu)$: On input the global parameters gpp, a set of authorities S_{enc}, a policy φ, the master public keys $\mathsf{mpk}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid}}, \mathsf{mpk}_{\mathsf{aid},0}, \ldots, \mathsf{mpk}_{\mathsf{aid},\ell})$, and a message $\mu \in \mathcal{M}$, the encryption algorithm generates a new authority set

$$S'_{\mathsf{enc}} = \{(\mathsf{aid}, k) \mid \mathsf{aid} \in S_{\mathsf{enc}}, k \in [0, \ell]\},$$

and constructs the extended policy φ' by substituting each appearance of attribute aid in the policy φ with the clause $\left(\bigvee_{k \in [0,\ell]} (\mathsf{aid}, k)\right)$. It then computes

$$\mathsf{ct}' \leftarrow \mathsf{sRABE.Encrypt}(\mathsf{gpp}, (S'_{\mathsf{enc}}, \varphi'), \{((\mathsf{aid}, k), \mathsf{mpk}_{\mathsf{aid},k})\}_{(\mathsf{aid},k) \in S'_{\mathsf{enc}}}, \mu).$$

It outputs the ciphertext $(\mathsf{ct}', \{\mathsf{ctr}_{\mathsf{aid}}\}_{\mathsf{aid} \in S_{\mathsf{enc}}})$.

- $\mathsf{Decrypt}(S, \{(\mathsf{sk}_{\mathsf{aid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid}})\}_{\mathsf{aid} \in S}, \mathsf{ct})$: On input the attribute set $S \subseteq S_{\mathsf{enc}}$, a collection of secret keys $\mathsf{sk}_{\mathsf{aid}} = (\mathsf{gid}, \{\mathsf{sk}_{\mathsf{aid},k}\}_{k \in [0,\ell]})$, a collection of helper keys

$$\mathsf{hsk}_{\mathsf{aid}} = (\mathsf{ctr}_{\mathsf{aid},\mathsf{gid}}, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid},0}, \ldots, \mathsf{hsk}_{\mathsf{aid},\mathsf{gid},\ell}),$$

and a ciphertext $\mathsf{ct} = (\mathsf{ct}', \{\mathsf{ctr}_{\mathsf{aid}}\}_{\mathsf{aid} \in S_{\mathsf{enc}}})$, the decryption algorithm proceeds as follows:

  - For each $\mathsf{aid} \in S$, let k_{aid} be the index of the most significant bit where $\mathsf{ctr}_{\mathsf{aid}}$ and $\mathsf{ctr}_{\mathsf{aid},\mathsf{gid}}$ differ.
  - If $\mathsf{hsk}_{\mathsf{aid},\mathsf{gid},k_{\mathsf{aid}}} = \perp$, abort and output $\perp$.

  Otherwise, output $\mathsf{sRABE.Decrypt}(\{\mathsf{sk}_{\mathsf{aid},k_{\mathsf{aid}}} \mathsf{hsk}_{\mathsf{aid},k_{\mathsf{aid}}}\}_{\mathsf{aid} \in S}, \mathsf{ct}')$.

The correctness and security analysis follow analogously to that of Construction 6.4 (see Section 6).

C Function-Binding Hash Functions for Unique Block Selection

In this section, we show how to construct a function-binding hash function (Definition 7.2) for the unique-block-selection predicate (Definition 7.3) we need in Construction 7.6. Our construction uses leveled homomorphic encryption in a similar way as constructions of somewhere statistically binding hash functions [HW15] and function-binding hash functions for disjunctions of block-wise predicates [FWW23]. We start by recalling the notion of leveled homomorphic encryption we use.

Definition C.1 (Leveled Homomorphic Encryption)**.** A leveled homomorphic encryption (LHE) scheme [Gen09] with message space $\mathcal{M} = \{\mathcal{M}_\lambda\}$ is a tuple $\Pi_{\mathsf{LHE}} = (\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt}, \mathsf{Eval})$ with the following syntax:

- $\mathsf{KeyGen}(1^\lambda, 1^L) \to (\mathsf{pk}, \mathsf{sk})$: On input the security parameter λ and a depth bound L, the key-generation algorithm outputs a public/secret key-pair $(\mathsf{pk}, \mathsf{sk})$.

- $\mathsf{Encrypt}(\mathsf{pk}, m) \to \mathsf{ct}$: On input a public key pk and a message $m \in \mathcal{M}_\lambda$, the encryption algorithm outputs a ciphertext ct.

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}) \to m/\perp$: On input a secret key sk and a ciphertext ct, the decryption algorithm outputs a message m or $\perp$ in case of decryption error.

- $\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \to \mathsf{ct}'$: On input a public key pk, a Boolean circuit $C \colon \mathcal{M}_\lambda \to \mathcal{M}_\lambda$, the evaluation algorithm outputs a ciphertext ct'. This algorithm is deterministic.

Moreover, we require Π_{LHE} satisfy the following properties:

- **Correctness:** For all $\lambda, L \in \mathbb{N}$ and $x \in \mathcal{M}$, and Boolean circuits $C \colon \mathcal{M} \to \mathcal{M}$ of depth at most L:

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}') = C(x) : \begin{array}{c} (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x) \\ \mathsf{ct}' \leftarrow \mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct}) \end{array}\right] = 1.$$

- **Compactness:** There exists a polynomial p_{Eval} such that for all $\lambda, L \in \mathbb{N}$, all $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L)$, all inputs $x \in \mathcal{M}$, ciphertexts $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, x)$ and circuits C of depth $\leq L$,

$$|\mathsf{Eval}(\mathsf{pk}, C, \mathsf{ct})| \leq p_{\mathsf{Eval}}(\lambda, L, |C(x)|).$$

- **Semantic security:** For a bit $b \in \{0, 1\}$ and an adversary $\mathcal{A}$, we define the semantic security game as follows:
  1. On input the security parameter 1^λ, the adversary $\mathcal{A}$ outputs a depth bound 1^L.
  2. The challenger samples $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(1^\lambda, 1^L)$ and replies to $\mathcal{A}$ with pk.
  3. The adversary $\mathcal{A}$ outputs two messages $m_0, m_1 \in \mathcal{M}$. The challenger responds with $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_b)$.
  4. The adversary outputs a bit $b' \in \{0, 1\}$, which is the output of the experiment.

  We say the encryption scheme is semantically secure if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda).$$

- **Polylogarithmic decryption depth:** There exists a polynomial p_{Decrypt} such that for all $\lambda, L \in N$ and all $(\mathsf{pk}, \mathsf{sk})$ in the support of $\mathsf{KeyGen}(1^\lambda, 1^L)$, the decryption circuit computing $\mathsf{Decrypt}(\mathsf{sk}, \cdot)$ has depth $\leq p_{\mathsf{Decrypt}}(\log \lambda, \log L)$.

Binary tree notation. Similar to [FWW23], we use a Merkle-tree like structure [Mer89] in our construction. First, a Merkle tree of depth ℓ is a complete binary tree with 2^ℓ leaf nodes. We say the leaf nodes are at level 0 and the root node is at level ℓ. We associate each leaf node with an arbitrary fixed-length value. The values of internal nodes in levels $k \in [\ell]$ are obtained by applying a two-to-one hash function to the two values associated with its child nodes. We index the nodes by a pair $(k, i) \in [0, \ell] \times [1, 2^{\ell - k}]$, where k denotes the level and i denotes the node index within the level. Under our conventions, the node (k, i) is the parent node of $(k - 1, 2i - 1)$ and $(k - 1, 2i)$.

It is often helpful to refer to structured collections of nodes, such as paths or sub-trees. For a leaf node at index i, let $\mathsf{path}(i)$ be the set of nodes along the path from leaf i to the root. For a set of leaves S, we define $\mathsf{ST}(S)$ to denote the union of all paths associated with the leaves in S. A node's sibling $\mathsf{sib}(k, i)$ is defined to be the (unique) node with the same parent ($(k, i + 1)$ when $i = 1 \bmod 2$ and $(k, i - 1)$ otherwise). Finally, we use $\mathsf{dangling}(S)$ or $\mathsf{dangling}(i)$ to denote the set of siblings of $\mathsf{ST}(S)$ or $\mathsf{path}(i)$ respectively.

Construction C.2 (Function-Binding Hash)**.** Let λ be a security parameter. Without loss of generality, we assume that the maximum number of blocks N is always a power of two. We construct a function-binding hash with block length $\ell_{\mathsf{blk}} \in \mathsf{poly}(\lambda)$ for the class of family of unique-block-selection functions (Definition 7.3) induced by polynomial-size predicates (of size at most $s = s(\lambda)$). We define the following building blocks and conventions:

- Let $\Pi_{\mathsf{LHE}} = (\mathsf{LHE.KeyGen}, \mathsf{LHE.Encrypt}, \mathsf{LHE.Decrypt}, \mathsf{LHE.Eval})$ be a leveled homomorphic encryption scheme. Our construction will require encrypting messages of the following form:
  - The first type of messages is a pair (c, x_i), where $c \in \{0, 1, 2\}$ and $x_i \in \{0, 1\}^{\ell_{\mathsf{blk}}}$.
  - The second type of messages consists of a secret key for the Π_{LHE} scheme.

  We take the message space of Π_{LHE} to be sufficiently large enough to encrypt both types of messages.

- For a block $x_i \in \{0,1\}^{\ell}_{\mathsf{blk}}$, let $C_{x_i}(g)$ be the universal circuit that takes as input the describe of a predicate $g\colon \{0,1\}^{\ell_{\mathsf{blk}}} \to \{0,1\}$ and outputs
$$C_{x_i}(g) = \begin{cases} (1, x_i) & g(x_i) = 1 \\ (0, 0^{\ell_{\mathsf{blk}}}) & \text{otherwise.} \end{cases}$$
Let d_0 be the depth of the universal circuit C_{x_i} for evaluating circuits of size at most $s = s(\lambda)$. Note that $d_0 = \mathsf{poly}(s) \in \mathsf{poly}(\lambda)$.

- Let $C_{\mathsf{merge},d}[\mathsf{ct}_0, \mathsf{ct}_1](\mathsf{sk})$ be a circuit (parameterized by depth d) that takes as input an LHE secret key sk (supporting depth d computation) and first computes
$$(c_0, x_0) = \mathsf{LHE.Decrypt}(\mathsf{sk}, \mathsf{ct}_0)$$
$$(c_1, x_1) = \mathsf{LHE.Decrypt}(\mathsf{sk}, \mathsf{ct}_1).$$
Then it outputs
$$C_{\mathsf{merge},d}[\mathsf{ct}_0, \mathsf{ct}_1](\mathsf{sk}) = \begin{cases} (1, x_0) & c_0 = 1 \wedge c_1 = 0 \\ (1, x_1) & c_0 = 0 \wedge c_1 = 1 \\ (0, 0^{\ell_{\mathsf{blk}}}) & c_0 = 0 \wedge c_1 = 0 \\ (2, 0^{\ell_{\mathsf{blk}}}) & \text{otherwise.} \end{cases}$$

- For $k \in [\ell]$, we recursively define d_k to be the depth of the circuit $C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_0, \mathsf{ct}_1](\mathsf{sk})$. Note that d_0 is defined as above. Since Π_{LHE} supports polylogarithmic-depth decryption, the circuit $C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_0, \mathsf{ct}_1]$ can be implemented in $p_{\mathsf{Decrypt}}(\log \lambda, \log d_{k-1}) + O(1)$ depth, meaning we can write $d_k = \mathsf{poly}(\lambda, \log k)$.

We now construct the function-binding hash function $\Pi_{\mathsf{FB}} = (\mathsf{Setup}, \mathsf{SetupBinding}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ as follows:

- $\mathsf{Setup}(1^\lambda, N)$: On input the security parameter λ and a bound $N = 2^\ell$ on the number of inputs, the setup algorithm samples encryption keys $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{d_k})$ for each $k \in [0, \ell]$. Then, it computes the ciphertext $\mathsf{ct}_0 \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_0, g_\perp)$, where $g_\perp$ is the function that outputs 0 on all inputs. For each $k \in [\ell]$, compute $\mathsf{ct}_k \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_k, \perp)$. Finally, it outputs $\mathsf{hk} = \{(k, \mathsf{pk}_k, \mathsf{ct}_k)\}_{k \in [0,\ell]}$.

- $\mathsf{SetupBinding}(1^\lambda, N, g)$: On input the security parameter λ, a bound $N = 2^\ell$ on the number of inputs, and a predicate g, the setup algorithm samples encryption keys $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{d_k})$ for each $k \in [0, \ell]$. Then, it computes the ciphertext $\mathsf{ct}_0 \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_0, g)$. For each $k \in [\ell]$, it computes $\mathsf{ct}_k \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_k, \mathsf{sk}_{k-1})$. Finally, it outputs $\mathsf{hk} = \{(k, \mathsf{pk}_k, \mathsf{ct}_k)\}_{k \in [0,\ell]}$.

- $\mathsf{Hash}(\mathsf{hk}, \mathbf{x})$: On input a hash key $\mathsf{hk} = \{(k, \mathsf{pk}_k, \mathsf{ct}_k)\}_{k \in [0,\ell]}$ and an input $\mathbf{x} = (x_1, \ldots, x_n)$ for $n \le N$, the hash algorithm constructs a Merkle tree (on N nodes). First, it associates a ciphertext with each of the N leaf nodes:
  - If $i \le n$, set $\mathsf{ct}_{i,0} = \mathsf{LHE.Eval}(\mathsf{pk}_0, C_{x_i}, \mathsf{ct}_0)$.
  - If $i > n$, set $\mathsf{ct}_{i,0} = \mathsf{LHE.Encrypt}(\mathsf{pk}_0, (0, 0^{\ell_{\mathsf{blk}}}); 0^\lambda)$ using fixed randomness 0^λ.

  Then, for each $k \in [\ell]$ and $i \in [2^{\ell-k}]$, the hash algorithm computes the value of each internal node (k, i) as follows:
$$\mathsf{ct}_{k,i} = \mathsf{LHE.Eval}(\mathsf{pk}_k, C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_{k-1,2i-1}, \mathsf{ct}_{k-1,2i}], \mathsf{ct}_k). \tag{C.1}$$
  The output is the value of the root note $h = (\mathsf{ct}_{\ell,1}, n)$.

- $\mathsf{Open}(\mathsf{hk}, \mathbf{x}, i)$: On input the hash key hk and an input $\mathbf{x} = (x_1, \ldots, x_n)$, the prover constructs the Merkle tree as in Hash. Then it outputs the opening π_i as the values of the nodes in $\mathsf{dangling}(i)$.

- $\mathsf{Verify}(\mathsf{hk}, h, S, \{(i, x_i, \pi_i)\}_{i \in S})$: On input the hash key hk, the hash $h = (\mathsf{ct}^*, n)$, a set of indices $S \subseteq [n]$, and the values x_i and openings π_i for each index $i \in S$, the verification algorithm proceeds as follows for each $i \in S$:
  - Compute the leaf node $\mathsf{ct}_{i,0} = \mathsf{LHE.Eval}(\mathsf{pk}_0, C_{x_i}, \mathsf{ct}_0)$.

- Next, it parses π_i to be the value of the the nodes in $\mathsf{dangling}(i)$. For each $k \in [\ell]$, it finds the sibling of the previous level's node in $\mathsf{dangling}(i)$, and recomputes $\mathsf{path}(i)$ using Eq. (C.1). Whenever there exists an index i' where $\mathsf{path}(i)$ intersects $\mathsf{dangling}(i')$, the verification algorithm checks that the associated values match.

This proceeds until it computes the value $\mathsf{ct}_{\ell,1}$ associated with the root node. The verification algorithm outputs 1 if $\mathsf{ct}^* = \mathsf{ct}_{\ell,1}$ and outputs 0 otherwise.

Theorem C.3 (Correctness)**.** *Construction C.2 is correct.*

Proof. By construction, the Open and Verify algorithms compute $\mathsf{path}(i)$ on the same inputs and using the same procedure. Since these are deterministic algorithm, they will always produce the same root, so correctness holds. $\square$

Theorem C.4 (Computational Function Hiding)**.** *Suppose Π_{LHE} is semantically secure. Then Construction C.2 satisfies computational function hiding.*

Proof. We start by defining a modified $\mathsf{Setup}_{k'}$ algorithm indexed by $k' \in [\ell]$:

- $\mathsf{Setup}_{k'}(1^\lambda, N, g)$ On input the security parameter λ and a bound $N = 2^\ell$ on the number of inputs, the setup algorithm samples encryption keys $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{d_k})$ for each $k \in [0, \ell]$. Then, it computes the ciphertext $\mathsf{ct}_0 \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_0, g)$ For each $k \in [k']$, it computes $\mathsf{ct}_k \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_k, \mathsf{sk}_{k-1})$. Otherwise, for $k \in [k'+1, \ell]$, it computes $\mathsf{ct}_k \leftarrow \mathsf{LHE.Encrypt}(\mathsf{pk}_k, \perp)$. Finally, it outputs $\mathsf{hk} = \{(k, \mathsf{pk}_k, \mathsf{ct}_k)\}_{k \in [0,\ell]}$.

We will argue through a sequence of hybrids that the hash keys output by $\mathsf{Setup}_{k'-1}$ and $\mathsf{Setup}_{k'}$ are computationally indistinguishable for all $k' \in [\ell]$. Since the hash key output by Setup_ℓ is exactly $\mathsf{SetupBinding}$, function hiding follows. More formally, we define a series of hybrids as follows:

- $\mathsf{Hyb}_{k'}^{(b)}$: This is the computational function hiding game, where the challenger samples the hash key hk using $\mathsf{Setup}_{k'}$ instead of $\mathsf{SetupBinding}$. Formally, the experiment proceeds as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the number of blocks $N \in \mathbb{N}$ (in *binary*) and a predicate function g.
  2. The challenger samples $\mathsf{hk}_0 \leftarrow \mathsf{Setup}(1^\lambda, N)$, $\mathsf{hk}_1 \leftarrow \mathsf{Setup}_{k'}(1^\lambda, N, g)$ and gives hk_b to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

Lemma C.5. *Suppose Π_{LHE} is semantically secure. Then for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$*

$$\left|\Pr[\mathsf{Hyb}_0^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0^{(1)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Assume there exists an efficient adversary $\mathcal{A}$ which has with non-negligible advantage ε in $\mathsf{Hyb}_0^{(b)}$. We use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ that breaks semantic security of Π_{LHE} with the same advantage:

- On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts by sending the depth parameter 1^{d_0} to the challenger. The challenger replies with the public key pk_0^*.
- Algorithm $\mathcal{B}$ begins running $\mathcal{A}$ on the same security parameter. Algorithm $\mathcal{A}$ sends a length parameter N and a predicate g.
- Algorithm $\mathcal{B}$ chooses $g_\perp$ and g as its challenge messages and receives a challenge ciphertext ct^*.
- Algorithm $\mathcal{B}$ constructs hk as in $\mathsf{Setup}(1^\lambda, N)$, but sets $\mathsf{pk}_0 = \mathsf{pk}^*$ and $\mathsf{ct}_0 = \mathsf{ct}^*$. It gives hk to $\mathcal{A}$.
- At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

By construction, if ct^* is an encryption of $g_\perp$, then hk is generated according to the specification of Setup, whereas if ct^* is an encryption of g, then hk is generated according to the specification of Setup_0. Note that since ct_1 is an encryption of $\perp$ (and not sk_0) in both Setup and Setup_0, algorithm $\mathcal{B}$ can perfectly simulate $\mathsf{Hyb}_0^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks semantic security with the same advantage as $\mathcal{A}$, which proves the claim. □

Lemma C.6. *Suppose Π_{LHE} is semantically secure. Then for all $k' \in [\ell]$, all $b \in \{0,1\}$, and all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_{k'-1}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{k'}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. This follows from a similar argument as the proof of Lemma C.5. Assume there exists an efficient adversary $\mathcal{A}$ that can distinguish $\mathsf{Hyb}_{k'-1}^{(b)}$ from $\mathsf{Hyb}_{k'}^{(b)}$ with non-negligible advantage ε. We use $\mathcal{A}$ to construct a new adversary $\mathcal{B}$ that breaks semantic security of Π_{LHE} with the same advantage:

- On input the security parameter 1^λ, algorithm $\mathcal{B}$ starts by sending the depth parameter 1^{d_k} to the challenger. The challenger replies with the public key $\mathsf{pk}_{k'}^*$.
- Algorithm $\mathcal{B}$ starts running $\mathcal{A}$ on the same security parameter. Algorithm $\mathcal{A}$ sends a length parameter N and a predicate g.
- Algorithm $\mathcal{B}$ samples $(\mathsf{pk}_{k'-1}, \mathsf{sk}_{k'-1}) \leftarrow \mathsf{LHE.Setup}(1^\lambda, 1^{d_{k'-1}})$, and sends challenge messages $\perp$ and $\mathsf{sk}_{k'-1}$ to the challenger. The challenger replies with a ciphertext ct^*.
- Algorithm $\mathcal{B}$ constructs hk_0 according to the specification of $\mathsf{Setup}(1^\lambda, N)$ and hk_1 according to the specification of $\mathsf{Setup}_{k'-1}(1^\lambda, N, g)$ except it substitutes $\mathsf{pk}_{k'-1}$, $\mathsf{pk}_{k'}^*$ and ct^* received from the previous steps in place of $\mathsf{pk}_{k'-1}$, $\mathsf{pk}_{k'}$ and $\mathsf{ct}_{k'}$, respectively. Finally, it forwards hk_b to $\mathcal{A}$.
- At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

By construction, if ct^* is an encryption of $\perp$, then hk_1 is generated according to the specification of $\mathsf{Setup}_{k'-1}$, whereas if ct^* is an encryption of $\mathsf{sk}_{k'-1}$, then hk_1 is generated according to the specification of $\mathsf{Setup}_{k'}$. As before, note that since $\mathsf{ct}_{k'+1}$ is an encryption of $\perp$ (and not $\mathsf{sk}_{k'}$) in both $\mathsf{Setup}_{k'-1}$ and $\mathsf{Setup}_{k'}$, so algorithm $\mathcal{B}$ can perfectly simulate these hybrids. Thus, algorithm $\mathcal{B}$ breaks semantic security with the same advantage as $\mathcal{A}$, which proves the claim. □

Combining Lemmas C.5 and C.6, we conclude Construction C.2 satisfies computational function hiding. □

Theorem C.7 (Statistical Function Binding). *Suppose Π_{LHE} satisfies perfect correctness. Then Construction C.2 satisfies statistical function binding.*

Proof. Our construction is in fact perfectly function binding. To reason about this, we first show the following invariant:

Lemma C.8. *Let g be any predicate (of circuit size at most $s = s(\lambda)$) and suppose the hash key is sampled as $\mathsf{hk} \leftarrow \mathsf{SetupBinding}(1^\lambda, N, g)$ for some $N = 2^\ell$. Take any input $\mathbf{x} = (x_1, \ldots, x_n)$ where $n \le N$ and let $h = \mathsf{Hash}(\mathsf{hk}, \mathbf{x})$. Let $\{(i, x_i^*, \pi_i^*)\}_{i \in S}$ be a collection of values and openings for an arbitrary subset of leaf nodes $S \subseteq [n]$. Suppose moreover that*

$$\mathsf{Verify}(\mathsf{hk}, h, S, \{(i, x_i^*, \pi_i^*)\}_{i \in S}) = 1.$$

Recall that each proof π_i^* contains the values of the nodes in $\mathsf{dangling}(i)$. This is sufficient to compute $\mathsf{path}(i)$. Then, the following invariant holds for all of the nodes along $\mathsf{ST}(S)$[7]:

- *Let (k, i) be an arbitrary node in $\mathsf{ST}(S)$. Then, the associated value $\mathsf{ct}_{k,i}$ is an encryption of (c^*, x^*), where the following properties hold:*

    1. *If $c^* = 0$, then there are no leaf nodes x_j where $g(x_j) = 1$.*

[7]Recall that $\mathsf{ST}(S)$ denotes the union of all paths from the nodes in S to the root.

2. *If $c^* = 1$, there is at most one leaf node x_j in the sub-tree rooted at (k, i) where $g(x_j) = 1$. In addition:*
   (a) *If such an x_j exists, then $x_j = x^*$.*
   (b) *If the sub-tree rooted at (k, i) is complete, such an x_j exists.*
3. *If $c^* = 2$ and the sub-tree rooted at (k, i) is complete, there are at least two leaf nodes $x_j, x_{j'}$ where $j \neq j'$ and $g(x_j) = 1 = g(x_{j'})$.*

Proof. We argue that this invariant holds for the nodes in $\mathsf{ST}(S)$ inductively (from the leaves to the root):

- **Base case:** Suppose $k = 0$. Recall that for a leaf node x_j, this hash is computed by homomorphically evaluating C_{x_j} on ct_0, where ct_0 an encryption of the function g. By definition of g, this produces an encryption of $(1, x_j)$ when $g(x_j) = 1$ and an encryption of $(0, \perp)$ otherwise. Since this is the only leaf node in this sub-tree, the invariant holds.

- **Inductive step:** Consider an arbitrary node $(k, i) \in \mathsf{ST}(S)$. We consider two cases:
  - Suppose both child nodes of (k, i) are in $\mathsf{ST}(S)$. First, we can apply our invariant to the ciphertexts associated with these two child nodes, $\mathsf{ct}_{k-1,2i-1}, \mathsf{ct}_{k-1,2i}$. By construction, from Eq. (C.1), the value $\mathsf{ct}_{k,i}$ is obtained by homomorphically evaluating $C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_{k-1,2i-1}, \mathsf{ct}_{k-1,2i}]$ on ct_k. By perfect correctness of Π_{LHE}, this will decrypt to some pair of values (c_0, x_0) and (c_1, x_1) respectively. We consider the following sub-cases:
    * Suppose $c_0 = c_1 = 0$. From the induction hypothesis (Condition 1), this means $(k-1, 2i-1)$ and $(k-1, 2i)$ have no leaves which satisfy predicate g. Since these are the only children of (k, i), it follows that (k, i) also has no such leaves that satisfy g. In this case, $C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_{k-1,2i-1}, \mathsf{ct}_{k-1,2i}]$ outputs $(0, 0^{\ell_{\mathsf{blk}}})$. Correspondingly, the induction invariant is preserved.
    * Suppose $c_0 = 1$ and $c_1 = 0$. In this case, $C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_{k-1,2i-1}, \mathsf{ct}_{k-1,2i}]$ outputs $(1, x_0)$. We argue that this satisfies the induction invariant. By the induction hypothesis (Conditions 1 and 2), the node $(k-1, 2i-1)$ has at most one child which satisfies g while the node $(k-1, 2i)$ has none. This means (k, i) also has at most one child that satisfies the predicate, which satisfies Condition 2. We now consider the additional property:
      · If the sub-tree rooted at (k, i) contains a leaf x_j that satisfies the predicate g, then it must be contained in the sub-tree rooted at $(k-1, 2i-1)$. By the induction hypothesis (Condition 2a), this means $x_j = x_0$, so Condition 2a continues to hold.
      · Next, suppose the sub-tree rooted at (k, i) is complete. This also means the sub-tree rooted at $(k-1, 2i-1)$ is also complete, so Condition 2b continues to hold.
    * Suppose $c_0 = 0$ and $c_1 = 1$. This proceeds analogous to the previous case.
    * Finally, suppose that either $c_0 = 2$, $c_1 = 2$, or $c_0 = c_1 = 1$. In this case, $C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_{k-1,2i-1}, \mathsf{ct}_{k-1,2i}]$ outputs $(2, 0^{\ell_{\mathsf{blk}}})$. If the sub-tree rooted at (k, i) is not complete, then Condition 3 holds vacuously. Consider the case where the sub-tree rooted at (k, i) is complete. This means that the sub-trees rooted at $(k-1, 2i-1)$ and $(k-1, 2i)$ are also complete. By the induction hypothesis, this means that either there are at least two leaves which satisfy g in one of the sub-trees (applying Condition 3 of the induction invariant when $c_0 = 2$ or $c_1 = 2$) *or* there is exactly one such leaf node in each sub-tree (applying Condition 2b to both sub-trees when $c_0 = c_1 = 1$). In either case, there are at least two leaf nodes in the sub-tree rooted at (k, i) that satisfy the predicate, and the claim holds.
  - Suppose that only one child of (k, i) is in $\mathsf{ST}(S)$. By construction, this means the sub-tree rooted at (k, i) cannot be complete. Without loss of generality, suppose $(k-1, 2i-1) \in \mathsf{ST}(S)$. This means $(k-1, 2i) \in \mathsf{dangling}(S)$. In this case, we simply apply our induction invariant to $(k-1, 2i-1)$. As in the previous case, let (c_0, x_0) be the decryption of $\mathsf{ct}_{k-1,2i-1}$, and let (c^*, x^*) be the output of $C_{\mathsf{merge},d_{k-1}}[\mathsf{ct}_{k-1,2i-1}, \mathsf{ct}_{k-1,2i}]$. We consider two sub-cases:
    * Suppose $c_0 = 0$. By the induction hypothesis, there are no leaves x_j in the sub-tree rooted at $(k-1, 2i-1)$ such that $g(x_j) = 1$. Since $(k-1, 2i)$ is a dangling node, the only leaves of (k, i) in $\mathsf{ST}(S)$ are the leaves in the sub-tree rooted at $(k-1, 2i-1)$. Thus, we conclude there are no leaves in

the sub-tree rooted at (k, i) where $g(x_j) = 1$. This satisfies the induction invariant whether $c^* = 0$ or $c^* = 1$.

  * Suppose $c_0 = 1$. By the induction hypothesis, there is at most one leaf x_j in the sub-tree rooted at $(k-1, 2i-1)$ where $g(x_j) = 1$. By construction of C_{merge}, if $c_0 = 1$, then $c^* \geq 1$. Since $(k-1, 2i-1)$ is a dangling node and has no leaves in the sub-tree rooted at $\mathsf{ST}(S)$, we again conclude that there is at most one leaf in the sub-tree rooted at (k, i) where $g(x_j) = 1$. Moreover, note that if the sub-tree rooted at $(k-1, 2i-1)$ contains such a leaf node x_j, then by our induction hypothesis (Condition 2a), we have $x_0 = x_j$. By construction of C_{merge}, if $c^* = 1$, then it will also be the case that $x^* = x_0 = x_j$, as required. □

Corollary C.9. *Suppose the hash key is sampled as* $\mathsf{hk} \leftarrow \mathsf{SetupBinding}(1^\lambda, N, g)$ *for some* $N = 2^\ell$. *Take any input* $\mathbf{x} = (x_1, \ldots, x_n)$ *where* $n \leq N$ *and let* $h = (\mathsf{ct}^*, n) = \mathsf{Hash}(\mathsf{hk}, \mathbf{x})$. *Then, the ciphertext* ct^* *is an encryption of* (c^*, x^*) *where*

- $c^* = 0$ *if* $g(x_j) = 0$ *for all* $j \in [n]$;
- $c^* = 1$ *if there exists a* unique *index* $j \in [n]$ *where* $g(x_j) = 1$ *and moreover* $x_j = x^*$; *and*
- $c^* = 1$ *if there exist* $j \neq j' \in [n]$ *where* $g(x_j) = 1 = g(x_{j'})$.

Proof. This follows by applying Lemma C.8 to the set of "real" openings $\{(i, x_i, \mathsf{Open}(\mathsf{hk}, \mathbf{x}, i))\}_{i \in [n]}$. By correctness (Theorem C.3), this is a verifying set of openings, so we can apply our invariant to $\mathsf{ct}_{\ell,1} = \mathsf{ct}^*$. □

Now that we have established this invariant, we will show the contrapositive of the statistical function binding statement. Namely, take any $\mathsf{hk} \leftarrow \mathsf{SetupBinding}(1^\lambda, N, g)$, any input $\mathbf{x} = (x_1, \ldots x_n)$ where $n \leq N$, and let $h = (\mathsf{ct}^*, n) = \mathsf{Hash}(\mathsf{hk}, \mathbf{x})$. Take set of input/opening pairs $\{(i, x_i^*, \pi_i^*)\}_{i \in S}$ such that $\mathsf{Verify}(\mathsf{hk}, h, S, \{(i, x_i^*, \pi_i^*)\}_{i \in S}) = 1$. We now show that we can always find an extension $\mathbf{x}' = (x_1', \ldots, x_n')$ where $x_i' = x_i^*$ for all $i \in S$ and for which $f_g(\mathbf{x}) = f_g(\mathbf{x}')$. For notational convenience, let $x^0, x^1 \in \{0,1\}^{\ell_{\mathsf{blk}}}$ be arbitrary values where $g(x^0) = 0$ and $g(x^1) = 1$. In addition, let pk' be an arbitrary fixed public key. We consider the following cases based on the number of indices $i \in [n]$ where $g(x_i) = 1$:

- Suppose $|\{i \in [n] : g(x_i) = 1\}| = 0$. In this case $f_g(\mathbf{x}) = \perp$. By Corollary C.9, we conclude ct^* is an encryption of the pair $(0, x_j)$. Since $\{(i, x_i^*, \pi_i^*)\}_{i \in S}$ verifies with respect to the same root h, we can appeal to Lemma C.8 to conclude that for all $i \in S$, $g(x_i^*) = 0$. Thus, we can define the extension $\mathbf{x}' = (x_1', \ldots x_n')$ where
$$x_i' = \begin{cases} x_i^* & i \in S \\ x^0 & i \notin S. \end{cases} \tag{C.2}$$
Since $g(x_i^*) = 0 = g(x^0)$, we conclude that $f_g(\mathbf{x}') = \perp$, making this a satisfying extension.

- Suppose $|\{i \in [n] : g(x_i) = 1\}| = 1$. Let $i^* \in [n]$ be the unique index for which $g(x_{i^*}) = 1$. Then, by definition, $f_g(\mathbf{x}) = x_{i^*}$. By Corollary C.9, the ciphertext ct^* is also an encryption of $(1, x_{i^*})$. Since $\{(i, x_i^*, \pi_i^*)\}_{i \in S}$ verifies with respect to the same root h, we can appeal to Lemma C.8 to conclude that there is *at most* one index $i' \in S$ where $x_{i'}^*$ where $g(x_{i'}^*) = 1$. We consider the two possibilities:

  – Suppose that $g(x_i^*) = 0$ for all $i \in S$. By Lemma C.8, we conclude that $S \neq [n]$; specifically if $S = [n]$, then the root node of $\mathsf{ST}(S)$ is a complete tree in which case Lemma C.8 stipulates that there is exactly one leaf node $x_{i'}^*$ where $g(x_{i'}^*) = 1$. Now take $i'' \in [n]$ to be the smallest index in $[n] \setminus S$. We define our extension $\mathbf{x}' = (x_1', \ldots, x_n')$ as
$$x_i' = \begin{cases} x_i^* & i \in S \\ x_{i^*} & i = i'' \\ x^0 & \text{otherwise.} \end{cases}$$
By construction, $f_g(\mathbf{x}') = x_{i^*}$, as required.

- Suppose there exists exactly one index $i' \in S$ where $g(x^*_{i'}) = 1$. From Condition 2a of Lemma C.8, this manes ct^* is an encryption of $(1, x^*_{i'})$. As argued above, ct^* is also an encryption of $(1, x_{i^*})$. Since Π_{LHE} is perfectly correct, this means $x_{i^*} = x^*_{i'}$. Thus, we can define the extension $\mathbf{x}' = (x'_1, \dots x'_n)$ as in Eq. (C.2). By construction, this satisfies $f_g(\mathbf{x}') = x^*_{i'} = x_{i^*}$, as required.

- Suppose $|\{i \in [n] : g(x_i) = 1\}| > 1$. By definition, this means $f_g(\mathbf{x}) = \perp$. By Corollary C.9, this means ct^* is an encryption of the form $(2, x^*)$. We again consider the number of indices $i \in S$ for which $g(x^*_i) = 1$:

  - Suppose for all $i \in S$, we have $g(x^*_i) = 0$. In this case, we can again take $\mathbf{x}'$ as in Eq. (C.2) to be our extension. In this case $f_g(\mathbf{x}') = \perp$. Technically, $\mathbf{x}'$ does not contain any blocks which satisfy the predicate whereas $\mathbf{x}$ contains more than two blocks. However, the behavior of f_g is the same in these two cases.
  - Suppose there is exactly one index $i' \in S$ where $g(x^*_{i'}) = 1$. Similar to above, we can appeal to Lemma C.8 to conclude that $S \neq [n]$ (if $S = [n]$, then Lemma C.8 implies that ct^* is an encryption of the form $(1, x^*)$). As such, we can define our extension $\mathbf{x}' = (x'_1, \dots, x'_n)$ as

$$x'_i = \begin{cases} x^*_i & i \in S \\ x^1 & \text{otherwise.} \end{cases}$$

    By construction, this means $f_g(\mathbf{x}') = \perp$, as required.
  - Suppose there are at least two indices $i \neq i' \in S$ where $g(x^*_i) = 1 = g(x^*_{i'})$. Then, we can again take $\mathbf{x}'$ as in Eq. (C.2) to be our extension. By construction, this means $f_g(\mathbf{x}') = 0$, as required. □

Theorem C.10 (Succinctness)**.** *Suppose Π_{LHE} is compact and has polylogarithmic decryption depth. Then Construction C.2 is succinct.*

Proof. Recall the hash consists of an encryption of a tuple (c, x_i) of length $\ell_{\mathsf{blk}} + \log N$ which supports a computation of depth $d_\ell \leq \mathsf{poly}(\lambda, \log \ell)$. Since $\ell_{\mathsf{blk}} = \mathsf{poly}(\lambda)$, compactness of Π_{LHE} allows us to bound the size of the ciphertext by $\mathsf{poly}(\lambda, \log N)$. Next, each opening contains up to $\log N$ ciphertexts. As argued above, each ciphertext has size $\mathsf{poly}(\lambda, \log N)$, so the overall opening has size $\mathsf{poly}(\lambda, \log N)$, as required. □