Registered Attribute-Based Encryption

Susan Hohenberger[*] George Lu[†] Brent Waters[‡] David J. Wu[§]

Abstract

Attribute-based encryption (ABE) generalizes public-key encryption and enables fine-grained control to encrypted data. However, ABE upends the traditional trust model of public-key encryption by requiring a single *trusted* authority to issue decryption keys. If an adversary compromises the central authority and exfiltrates its secret key, then the adversary can decrypt *every* ciphertext in the system.

This work introduces registered ABE, a primitive that allows users to generate secret keys on their own and then register the associated public key with a "key curator" along with their attributes. The key curator aggregates the public keys from the different users into a single *compact* master public key. To decrypt, users occasionally need to obtain helper decryption keys from the key curator which they combine with their own secret keys. We require that the size of the aggregated public key, the helper decryption keys, the ciphertexts, as well as the encryption/decryption times to be polylogarithmic in the number of registered users. Moreover, the key curator is entirely *transparent* and maintains no secrets. Registered ABE generalizes the notion of registration-based encryption (RBE) introduced by Garg et al. (TCC 2018), who focused on the simpler setting of identity-based encryption.

We construct a registered ABE scheme that supports an a priori bounded number of users and policies that can be described by a linear secret sharing scheme (e.g., monotone Boolean formulas) from assumptions on composite-order pairing groups. Our approach deviates sharply from previous techniques for constructing RBE and only makes *black-box* use of cryptography. All existing RBE constructions (a weaker notion than registered ABE) rely on heavy non-black-box techniques. The encryption and decryption costs of our construction are comparable to those of vanilla pairing-based ABE. Two limitations of our scheme are that it requires a structured reference string whose size scales quadratically with the number of users (and linearly with the size of the attribute universe) and the running time of registration scales linearly with the number of users.

Finally, as a feasibility result, we construct a registered ABE scheme that supports general policies and an arbitrary number of users from indistinguishability obfuscation and somewhere statistically binding hash functions.

1 Introduction

Attribute-based encryption (ABE) [SW05, GPSW06] extends traditional public-key encryption to enable fine-grained access control to encrypted data. For instance, in a ciphertext-policy ABE, secret keys are associated with attributes, and ciphertexts are associated with decryption policies. A secret key sk_x for an attribute x can decrypt a ciphertext ct_P with policy P only if the attribute satisfies the ciphertext's policy (i.e., $P(x) = 1$). In contrast, with vanilla public-key encryption, decryption is all-or-nothing: if a user has the secret key, she can decrypt *every* ciphertext encrypted under the respective public key and if the user does not know the secret key, she cannot decrypt *any* ciphertext.

While ABE is a versatile cryptographic primitive for enabling fine-grained control to encrypted data, it significantly changes the trust model compared to standard public-key encryption. In an ABE scheme, a central *trusted* authority is required to issue the secret decryption keys associated with each user. Critically, this central authority needs to

[*]Johns Hopkins University. Email: susan@cs.jhu.edu.
[†]UT Austin. Email: gclu@cs.utexas.edu
[‡]UT Austin and NTT Research. Email: bwaters@cs.utexas.edu.
[§]UT Austin. Email: dwu4@cs.utexas.edu.

retain a *long-term* secret key. If the central authority is compromised by an adversary at *any* point, then the adversary gains the ability to decrypt *all* ciphertexts in the system. This makes ABE inherently vulnerable to key exfiltration attacks, and the long-term secret key must be carefully protected for the lifetime of the system. In contrast, with standard public-key encryption, users can generate their *own* public/secret keys, and they do not have to entrust their secret keys to any central party. Public-key encryption do not open users up to a *central* point of failure. The combination of built-in key escrow and vulnerability to key exfiltration is a common impediment to deploying ABE.

Registration-based encryption. Garg et al. [GHMR18] introduced the notion of registration-based encryption (RBE) to address the key-escrow problem in the setting of identity-based encryption (IBE). In an IBE scheme [Sha84, BF01, Coc01], secret keys and ciphertexts are associated with identities and decryption succeeds if the identities associated with the secret key and ciphertexts match; an IBE scheme is a special case of ABE for the equality policy. In an RBE scheme, the central authority is replaced by a "key curator." The role of the key curator is *not* to issue secret decryption keys, but instead, to *aggregate* public keys from registered users into a *short* master public key mpk.

In more detail, users in an RBE scheme generate their own public/secret keys (like in traditional public-key encryption), and then register their *public* keys together with their identity with the key curator. The key curator then updates the master public key of the scheme. Like IBE, the master public key can be used to encrypt a message to any identity. If the identity corresponds to that of a registered user, then the user can decrypt the message using their secret key and a *publicly-computable* helper decryption key that binds the user's public key to the current master public key. Since the master public key of the RBE scheme changes whenever new users join the system, users must periodically refresh their helper decryption keys over the lifetime of the system. Note that the helper decryption keys for each user can be computed publicly, and importantly, in an RBE system, the key curator does *not* possess any secret information. The efficiency requirement is that if L users register, then each user only needs to update their decryption key at most $O(\log L)$ times over the lifetime of the system. The size of each update should also be short (i.e., $\mathsf{poly}(\lambda, \log L)$, where λ is a security parameter). In addition, like IBE, the master public key must be short: $|\mathsf{mpk}| \leq \mathsf{poly}(\lambda, \log L)$.

A challenge: non-black-box use of cryptography. In recent years, a number of works have constructed registration-based encryption [GHMR18, GHM+19, GV20, CES21] from standard assumptions such as CDH, factoring, or LWE assumptions. However, all of the existing constructions make heavy *non-black-box* use of cryptography. Existing constructions either apply indistinguishability obfuscation to a cryptographic hash function [GHMR18] or use a hash garbling scheme to traverse a Merkle tree [GHM+19, GV20, CES21]. The latter approach chains together a sequence of garbled circuits (proportional to the length of the identity), where each garbled circuit reads one bit of the input and outputs a set of labels for the next garbled circuit; the final garbled circuit is a garbling of the encryption algorithm for a public-key encryption scheme. The heavy use of non-black-box cryptography in both approaches render existing schemes completely impractical. Even in spite of recent optimization efforts [CES21], a *single* ciphertext in a system supporting 2 billion users is estimated to be 4.5 *terabytes*.

This work: registered ABE. In this work, we introduce a generalization of RBE called registered ABE to address the key escrow problem and remove the need for long-term secret keys in the context of ABE. We introduce a new set of techniques for realizing registered ABE with only *black-box* use of cryptography. Our work extends registration-based encryption in two key ways:

- **Functionality:** Our scheme is attribute-based rather than identity-based, and is capable of supporting any access control policy that can be described by a linear secret sharing scheme (which includes monotone Boolean formulas). This matches the state-of-the-art in pairing-based ABE schemes. We refer to our new primitive as a *registered ABE* scheme. Our scheme includes RBE as a special case if we instantiate the scheme for the class of equality policies. Much like RBE provides a solution to the key-escrow problem for the setting of IBE, registered ABE provides an analogous solution in the setting of ABE.

- **Black-box use of cryptography:** Our construction does not make any non-black-box use of cryptography. The key-generation, encryption, and decryption algorithms in our scheme is comparable to that of existing pairing-based ABE schemes (e.g., [LOS+10]). Our approach departs from the hash garbling approach used in all existing constructions of RBE [GHMR18, GHM+19, GV20, CES21] and instead, takes an aggregation-based approach that is conceptually similar to those used in the construction of pairing-based vector commitments [CF13, LM19] and batch arguments [WW22].

We construct a registered ABE scheme from static assumptions on composite-order pairing groups (Assumption 5.2). We rely on the same assumptions as those used previously to construct IBE [LW10] and ABE [LOS+10].

A limitation of our scheme is that it imposes an *a priori* bound L on the number of users in the system, and security relies on a one-time trusted sampling of a common reference string (CRS). We note that this setup only needs to be done *once* and the same CRS can be reused across different systems. The size of the CRS is quadratic in L while the registration time is linear in L. However, the size of the master public key, the size of the helper decryption keys, as well as the encryption and decryption times, all scale polylogarithmically with L. As with standard RBE, the key curator is a deterministic algorithm and does not need to store any secret information. We also note that our scheme is limited to a polynomial-size attribute universe and the size of the CRS, the master public key, and each user's helper decryption key scale linearly with the size of the attribute universe.

While the CRS in our scheme is *structured*[1] and needs to be sampled by a trusted party (or using an MPC protocol), this is the only trusted component in our system. Thereafter, the behavior of the key curator is deterministic and auditable. As long as the adversary does not compromise this *one-time* setup, security holds. This is in contrast to traditional ABE where users must always trust the central authority who holds the *long-term* secret key. If the authority is compromised at *any* point in time and the adversary successfully exfiltrates the authority's secret key, then they gain the ability to decrypt every ciphertext in the system. Thus, even with a structured CRS, the registered ABE model still represents a significant reduction in trust compared to the traditional ABE model.

We summarize our main instantiation with the following (informal) characterization of Corollary 6.9:

Theorem 1.1 (Informal)**.** *Let λ be a security parameter. Let $\mathcal{U}$ be an attribute space and $\mathcal{P}$ be a set of policies that can be described by a linear secret sharing scheme over $\mathcal{U}$. Let L be a bound on the number of users. Then, under reasonable assumptions on a composite-order pairing group, there exists a registered ABE scheme that supports up to L users with attribute universe $\mathcal{U}$ and policy space $\mathcal{P}$ with the following properties:*

- *The size of the CRS and the size of the auxiliary data maintained by the key curator is $L^2 \cdot \mathrm{poly}(\lambda, |\mathcal{U}|, \log L)$.*
- *The running time of key-generation and registration is $L \cdot \mathrm{poly}(\lambda, |\mathcal{U}|, \log L)$.*
- *The size of the master public key and the helper decryption keys are both $|\mathcal{U}| \cdot \mathrm{poly}(\lambda, \log L)$.*
- *The size of a ciphertext is $|P| \cdot \mathrm{poly}(\lambda, \log L)$, where P is the size of the ciphertext policy.*

Note that only the key-generation, registration, and update algorithms depend on the (long) CRS. The running time of encryption and decryption are all polylogarithmic *in the number of users L.*

In addition to the above scheme based on composite-order bilinear maps, we also show how to construct a registered ABE scheme for an *arbitrary* number of users and supporting arbitrary policies (on a super-polynomial size attribute space) using indistinguishability obfuscation [BGI+01, BGI+12] and somewhere statistically binding hash functions [HW15]. Coupled with the work of Jain et al. [JLS21, JLS22], this yields a registered ABE scheme from falsifiable assumptions. We view this latter result as primarily a feasibility result for constructing registered ABE schemes capable of supporting general policies and an arbitrary number of users.

1.1 Related Work

Many previous works have explored mechanisms to address the key-escrow limitation inherent to IBE and ABE. One approach is based on threshold cryptography [BF01, CHSS02, PS08, KG10] where the master secret key is secret-shared across multiple independent authorities; this way, no single authority has the ability to decrypt ciphertexts. A similar notion in the setting of ABE is multi-authority ABE [Cha07, LCLS08, MKE08, CC09, LW11, RW15, DKW21a, DKW21b,

[1]Previous constructions of registration-based encryption [GHMR18, GHM+19, GV20, CES21] only assumed a *uniform* random string rather than a structured reference string.

WWW22] where anyone can become an authority and issue secret keys corresponding to the set of attributes within their domain. Policies in a multi-authority ABE scheme are in turn formulated with respect to the attributes of one or more authorities. Nonetheless, the keys in threshold and decentralized systems are still issued by entities other than the user, and if a sufficient number of the key-issuing entities are compromised or corrupted, then the schemes no longer ensure confidentiality.

Other techniques have focused on adding accountability to the central authority [Goy07, GLSW08] or introducing hybrid notions that combine IBE and traditional public-key directories [AP03]. However, none of these approaches completely eliminate the key-escrow problem inherent to notions like IBE and ABE.

Registration-based encryption was first introduced by Garg et al. [GHMR18] who also gave a construction from indistinguishability obfuscation and somewhere statistically binding hash functions. They also gave a "weakly-efficient" scheme (where registration runs in time that is polynomial in the number of registered users) from simpler assumptions like CDH or LWE. Subsequently, [GHMR18] provided a fully-efficient construction (where registration runs in time that is polylogarithmic in the number of registered users) from assumptions like CDH or LWE. Cong et al. [CES21] subsequently improved the concrete efficiency of their scheme. Goyal and Vusirikala [GV20] then showed how to augment RBE with protection against malicious key curators. All of these existing constructions (including the *weakly-efficient* ones) rely on non-black-box use of cryptography (e.g., obfuscation or hash garbling techniques).

2 Technical Overview

In this work, we construct a ciphertext-policy registered ABE scheme that supports any access policy that can be described by a linear secret sharing scheme (see Section 2.1 and Definition 3.2). In the following description, we let $\mathcal{U}$ be the universe of attributes. We will assume that $\mathcal{U}$ is polynomial-size (i.e., we are in the small universe setting). We additionally assume that there is an *a priori* bound L on the maximum number of users that can be registered, and moreover, that there is a (trusted) setup algorithm that samples a common reference string crs that will be used for key-generation, registration, and computing the helper information for decryption. In our setting, we allow the size of the crs to be $\mathsf{poly}(\lambda, L)$. The key curator initializes the master public key mpk to the empty string.

When a user wants to join the system, it first samples a public/secret key-pair $(\mathsf{pk}, \mathsf{sk})$. To register, the user provides their public key pk along with their set of attributes $S \subseteq \mathcal{U}$ to the key curator.[2] The key curator then *aggregates* the key into the master public key mpk and produces an updated key mpk'. In addition, the key curator computes a helper decryption key hsk and gives it to the user. In our setting, we allow the key-generation and registration process to be slow (i.e., running in time $\mathsf{poly}(\lambda, L)$).[3] However, the size of the master public key mpk, the secret key sk, and helper decryption key hsk for each user must be short (i.e., $\mathsf{poly}(\lambda, \log L)$). Each time a user registers, the master public key needs to be updated; this means users will need to periodically obtain an updated helper decryption key corresponding to the most recent master public key. As in RBE, we require that over the lifetime of the system, the user only needs to request $O(\log L)$ many updates from the key curator.

In a registered ABE scheme, encryption only requires knowledge of the master public key mpk (and *not* the long common reference string). The encryption algorithm takes in the master public key mpk, the access policy P, and a message μ and outputs a ciphertext ct. In turn, every registered user whose set of attributes S satisfy the policy is able to decrypt using their secret key sk and the helper decryption key hsk. Neither the encryption nor decryption algorithms require knowledge of the crs, and the running time of all of these algorithms scale with $\mathsf{poly}(\lambda, \log L, |P|)$. Notably, in a registered ABE scheme, there is an initial slow *one-time* process for generating and registering keys. Encryption and decryption are both fast (comparable to *standard* ABE).

Slotted registered ABE. Our construction of registered ABE proceeds in two steps. First, we define and construct an intermediate primitive that we call "slotted registered ABE" (Section 4.1). We then show how to compile a slotted registered ABE scheme into a registered ABE scheme (Section 6).

[2] Just like in RBE, the key curator first verifies the attributes claimed by the user before proceeding. This step is analogous to the checks certificate authorities perform in the public-key infrastructure before issuing a certificate or what the central authority would do in a standard ABE setting before issuing a decryption key. A difference is that the key curator possesses no secret information.

[3] This roughly coincides with the notion of *weak efficiency* in the work of Garg et al. [GHMR18].

In a slotted registered ABE scheme, we specify a *fixed* number of users L at setup, and moreover, each user is associated with a slot index $i \in [L]$. Public keys in a slotted registered ABE scheme are generated with respect to a particular slot. In addition, we replace the registration algorithm with an *aggregation* algorithm that takes as input a collection of L public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$, one for each slot, along with their associated attribute sets $S_1, \ldots, S_L \subseteq \mathcal{U}$, and outputs the master public key mpk together with the helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$ associated with each slot. The main difference is that aggregation takes all L keys at once and outputs the master public key (which is then fixed). In contrast, in (non-slotted) registered ABE, the public keys are registered one at a time, and the master public key is updated after each registration. We provide the formal definition of a slotted registered ABE scheme in Section 4.1 and show how to construct a slotted registered ABE scheme from assumptions on a composite-order pairing group in Section 5. We note that our scheme assumes a polynomial-size attribute universe and the sizes of the master public key and the helper decryption keys scale linearly with the size $|\mathcal{U}|$ of the attribute universe. We provide an overview of our slotted registered ABE scheme in Section 2.1.

From slotted registered ABE to registered ABE. To go from a slotted registered ABE scheme to a registered ABE scheme, we use a simple "powers-of-two" approach that was also used implicitly in previous constructions [GHMR18, GHM+19]. Suppose we want to support a maximum of $L = 2^\ell$ users. Our construction uses $\ell + 1$ copies of the slotted registered ABE scheme, where the k^{th} copy is a slotted ABE with exactly 2^k slots (with k ranging from 0 to ℓ). The master public key mpk consists of $\ell + 1$ master public keys $\mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell$, one for each of the underlying schemes. Initially, $\mathsf{mpk}_k = \perp$ for all k. The first user registers to an empty slot in each of the $\ell + 1$ instances. At this point, the first slotted registered ABE scheme (with 1 slot) is full, and the key curator computes mpk_0 and updates its value in mpk. When subsequent users join the system, they continue to register to the next vacant slot in each of the $\ell + 1$ instances (if one exists). If scheme k fills up (i.e., there is a key associated with each of its 2^k slots), the key curator updates mpk_k in the master public key and then *removes* all of the registered keys from schemes $0, \ldots, k - 1$ (since all of those users' public keys are now aggregated as part of mpk_k).[4] Subsequent registrations will reuse schemes $0, \ldots, k - 1$ since these are no longer full. On every registration, exactly one of the master public keys mpk_k is updated. When this occurs, all of the users who are now registered in the k^{th} scheme will need to obtain a decryption key update from the key curator. By design, this process can only happen at most $\ell + 1 = O(\log L)$ times, so this satisfies the efficiency requirements on the registered ABE scheme. To encrypt a message with respect to $\mathsf{mpk} = (\mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$, the encrypter encrypts the message to each mpk_k to obtain ct_k. The ciphertext is $\mathsf{ct} = (\mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$. To decrypt, a user who is currently registered in mpk_k takes ct_k and decrypts. Overall this powers-of-two approach incurs $O(\log L)$ overhead on the size of the public parameters, the ciphertext size, and the encryption time compared to the slotted scheme, but now supports efficient updates. We describe and analyze this transformation in Section 6 (Construction 6.1). We summarize the properties of our final registered ABE scheme in Corollary 6.9 (and Theorem 1.1).

Registered ABE for unbounded users from obfuscation. Our pairing-based registered ABE construction only supports a bounded number of users. A natural question is whether we can construct registered ABE that supports an *arbitrary* number of users. In Section 7, we show the feasibility of such a scheme using indistinguishability obfuscation [BGI+01, BGI+12] and somewhere statistically binding hash functions [HW15]. Our registered ABE (for arbitrary circuit predicates) is a direct generalization of the RBE scheme of Garg et al. [GHMR18] from indistinguishability obfuscation. Here, we describe a slotted version of the scheme. Given a collection of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ along with their attribute sets $S_1, \ldots, S_L$, we first construct a Merkle hash tree on values $(\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)$. The master public key is the root of the Merkle tree. A ciphertext consists of an obfuscated program that takes as input an index $i \in [L]$, the public key pk_i and its accompanying secret key sk_i, the set of attributes S_i, and a Merkle proof of opening for the value (pk_i, S_i) at index i. The obfuscated program checks that (1) the opening with respect to the hash root (hard-coded) is valid; (2) S_i satisfies the ciphertext policy (also hard-coded); and (3) sk_i is the secret key associated with pk_i. If all of these checks pass, it outputs the message m. This approach directly yields a registered ABE for an arbitrary number of users and which supports general circuit policies. We give the full construction in Section 7 (Construction 7.4). We leave the question of constructing registered ABE that supports an unbounded number of users without obfuscation (or without needing non-black-box use of cryptography) as an intriguing open problem.

[4] For ease of notation in the formal description (Section 6 and Construction 6.1), we do not implement this "clearing out" step explicitly. However, the construction is functionally behaving in this manner.

2.1 Constructing Slotted Registered ABE from Pairings

In this section, we provide a general overview of our slotted registered ABE scheme from composite-order pairing groups. The full construction and analysis are provided in Section 5. Together with the slotted-to-full transformation from Section 6, we obtain a registered ABE for an a priori bounded number of users.

Composite-order pairing groups. Our construction relies on composite-order pairing groups where the group order N is a product of three primes $N = p_1 p_2 p_3$. Then, a (symmetric) composite-order pairing group consists of two cyclic groups $\mathbb{G}$ and $\mathbb{G}_T$, each of order N. Let g be a generator of $\mathbb{G}$. By the Chinese remainder theorem, we can write $\mathbb{G} \cong \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_3$, where $\mathbb{G}_i$ is the subgroup of $\mathbb{G}$ order p_i and is generated by $g_i = g^{N/p_i}$. Additionally, there exists an efficiently-computable, non-degenerate bilinear map $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ called the *pairing*. For all exponents $a, b \in \mathbb{Z}_N$, we have that $e(g^a, g^b) = e(g, g)^{ab}$. Again by the Chinese remainder theorem, the subgroups $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ are orthogonal: namely $e(g_i, g_j) = 1$ for all $i \neq j$ where $i, j \in \{1, 2, 3\}$.

Linear secret sharing schemes. Like numerous other pairing-based ABE schemes [GPSW06, LOS+10, LW11], we design a (ciphertext-policy) ABE scheme that supports access policies which can be described by a linear secret sharing scheme (LSSS). Very briefly, a linear secret sharing scheme is specified by a share-generating matrix $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$, where each row of $\mathbf{M}$ is associated with a distinct attribute $x_1, \ldots, x_K$. We say a set of attributes $\{x_i\}_{i \in S}$ is authorized if and only if there exists a vector $\boldsymbol{\omega}_S \in \mathbb{Z}_N^{|S|}$ such that $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}} = [1, 0, \cdots, 0]$, where $\mathbf{M}_S$ is the matrix formed by taking the subset of rows indexed by $S \subseteq [K]$. In other words, the attributes $\{x_i\}_{i \in S}$ satisfy the policy if and only if $\mathbf{e}_1^{\mathsf{T}}$ is in the row-span of $\mathbf{M}_S$. Given an LSSS matrix $\mathbf{M}$, we can secret share a value $s \in \mathbb{Z}_N$ by sampling $v_2, \ldots, v_n \xleftarrow{\text{R}} \mathbb{Z}_N$, constructing the vector $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$ and computing the vector of shares $\mathbf{u} = \mathbf{M}\mathbf{v} \in \mathbb{Z}_q^K$. The i^{th} component $u_i \in \mathbb{Z}_N$ is the share associated with attribute x_i. Given an authorized set of attributes $\{x_i\}_{i \in S}$ and the subset of shares $\mathbf{u}_S \in \mathbb{Z}_N^{|S|}$ associated with S, reconstructing the secret corresponds to computing $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{u}_S = \boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S \mathbf{v} = \mathbf{e}_1^{\mathsf{T}} \mathbf{v} = s$.

Slotted registered ABE overview. In a slotted registered ABE scheme with L slots, users register a public key pk along with a set of attributes $S \subseteq \mathcal{U}$ to a particular slot $i \in [L]$. In our construction, the decryption algorithm implicitly enforces the following two checks:

- **Slot-specific check:** The user possesses a secret key associated with some slot i in the scheme.
- **Attribute-specific check:** The attributes associated with the slot i satisfy the ciphertext policy. In our construction, this check shares a similar structure to the Lewko et al. [LOS+10] ciphertext-policy ABE scheme.

Thus, when describing our scheme, we roughly partition the components of the CRS, the master public key, and the ciphertext based on whether they are "slot-specific" or "attribute-specific."

A single slot scheme. We start by describing a simple version of our scheme with just a *single* slot.[5] The single-slot scheme highlights the core components of our construction. Subsequently, we describe how to extend the single-slot scheme into a multi-slot scheme. An important difference between registered ABE and vanilla ABE is the fact that the master public keys in a registered ABE can depend on the set of attributes that have been registered so far. Thus, in the single-slot setting that just supports a single user, the user's attributes are directly embedded into the master public key. Let $\mathcal{U}$ be a (polynomial-size) universe of attributes and let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ be a composite-order pairing group with $N = p_1 p_2 p_3$. We now describe the main components of the scheme:

- The components of the common reference string crs can be partitioned into three general categories:
  - **General components:** The general component is used for blinding the message and linking together the slot-specific and attribute-specific decryption procedures. These components will subsequently be included as part of the master public key. Concretely, we sample exponents $\alpha, \beta \xleftarrow{\text{R}} \mathbb{Z}_N$ and include $Z \leftarrow e(g_1, g_1)^{\alpha}$ and $h \leftarrow g_1^{\beta}$ in the CRS.

[5]Note that a single-slot scheme by itself is trivial to construct. We can simply define the master public key to be the public key and set of attributes associated with the slot. However, for describing our construction, it is simpler to first illustrate the mechanics in the single-slot setting and then build up to the full multi-slot construction.

  - **Slot-specific components:** Each slot in the CRS is associated with a set of group elements. In the single-slot setting, we have two elements $A \leftarrow (g_1 g_3)^t$ and $B \leftarrow g_1^\alpha h^t g_3^\tau$, where $t \xleftarrow{\textsc{r}} \mathbb{Z}_N$ is a slot-specific exponent, $\alpha \in \mathbb{Z}_N$ is the "general" exponent from above, and $\tau \xleftarrow{\textsc{r}} \mathbb{Z}_N$ is a blinding factor.
  - **Attribute-specific components:** For each attribute $w \in \mathcal{U}$, the CRS contains a group element $U_w \leftarrow g_1^{u_w}$, where $u_w \xleftarrow{\textsc{r}} \mathbb{Z}_N$ is the attribute-specific exponent associated with w.

  Putting all the pieces together, the CRS in the single-slot setting consists of the following terms:

$$\mathsf{crs} = \big(\mathcal{G}\,,\, g_1\,,\, g_3\,,\, Z\,,\, h\,,\, (A, B)\,,\, \{U_w\}_{w \in \mathcal{U}}\big).$$

- To sample a new public/secret key-pair, the user samples $r \xleftarrow{\textsc{r}} \mathbb{Z}_N$ and sets it as their secret key $\mathsf{sk} = r$. The user sets the public key to be $\mathsf{pk} = T = g_1^r$.
- When the user registers their public key $\mathsf{pk} = T = g_1^r$ along with their set of attributes $S \subseteq \mathcal{U}$, the key curator sets $\hat{T} = T$ and $\hat{U}_w = U_w$ if $w \notin S$ and $\hat{U}_w = 1$ if $w \in S$. The key curator then outputs the master public key

$$\mathsf{mpk} = \big(\mathcal{G}\,,\, g_1\,,\, h\,,\, Z\,,\, \hat{T}\,,\, \{\hat{U}_w\}_{w \in \mathcal{U}}\big). \tag{2.1}$$

  As we will see later on, $\hat{T}$ is the attribute-independent key aggregated across all of the slots while $\hat{U}_w$ is the key associated with attribute w aggregated across all of the slots.

- The helper decryption key for the user is just the slot-specific components $A = (g_1 g_3)^t$ and $B = g_1^\alpha h^t g_3^\tau$ from the CRS.
- To encrypt a message $\mu \in \mathbb{G}_T$ to a policy $(\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ is the share-generating matrix associated with the policy, and $\rho \colon [K] \to \mathcal{U}$ is an injective row-labeling function that maps the rows of $\mathbf{M}$ onto the particular attribute to which it corresponds, the encrypter samples $s \xleftarrow{\textsc{r}} \mathbb{Z}_N$ and $h_1, h_2 \xleftarrow{\textsc{r}} \mathbb{G}_1$ such that $h_1 h_2 = h$. Namely, h_1 and h_2 function as a secret sharing of h. The ciphertext then consists of the following:
  - **Message-embedding components:** Let $C_1 \leftarrow \mu \cdot Z^s = \mu \cdot e(g_1, g_1)^{\alpha s}$. Let $C_2 \leftarrow g_1^s$.
  - **Attribute-specific component:** Let $\mathbf{v} = [s, v_2, \ldots, v_n]^\intercal$, where $v_2, \ldots, v_n \xleftarrow{\textsc{r}} \mathbb{Z}_N$. For each $k \in [K]$, set $C_{3,k} \leftarrow h_2^{\mathbf{m}_k^\intercal \mathbf{v}} \hat{U}_{\rho(k)}^{-s}$. Here $\mathbf{m}_k^\intercal \in \mathbb{Z}_N^n$ denotes the kth row of $\mathbf{M}$.
  - **Slot-specific component:** Set $C_4 \leftarrow h_1^s \hat{T}^{-s}$.

  The ciphertext is

$$\mathsf{ct} = \big((\mathbf{M}, \rho)\,,\, C_1\,,\, C_2\,,\, \{C_{3,k}\}_{k \in [K]}\,,\, C_4\big).$$

  Note that if we ignore the slot-specific ciphertext component, then the structure of the ciphertexts in our scheme coincides with those in the ciphertext-policy ABE scheme of Lewko et al. [LOS+10].[6] However, once we move beyond the single-slot setting, we will need to introduce additional components into the *aggregated* public key. This leads to a more complex decryption procedure and requires a more intricate security analysis compared to [LOS+10]. We discuss some of these details below and refer to Section 5 for the complete details.

The decryption algorithm can be decomposed into two main components: the first ensures the user's attributes satisfy the policy, and the second ensures the user's public key is bound to a specific slot. We describe these two steps below:

- **Policy check:** Let $S' = \{k \in [K] : \rho(k) \in S\}$ be the subset of the user's attributes that are associated with the policy $(\mathbf{M}, \rho)$. Suppose S' satisfies the policy $(\mathbf{M}, \rho)$. This means there exists a vector $\boldsymbol{\omega}_{S'} \in \mathbb{Z}_N^{|S'|}$ such that $\boldsymbol{\omega}_{S'}^\intercal \mathbf{M}_{S'} = \mathbf{e}_1^\intercal$. Moreover, by construction, $\hat{U}_w = 1$ for all $w \in S'$. In particular, this means that $C_{3,k} = h_2^{\mathbf{m}_k^\intercal \mathbf{v}}$ for all

[6]The scheme of Lewko et al. [LOS+10] also includes a row-specific blinding factor $s_k \xleftarrow{\textsc{r}} \mathbb{Z}_N$ associated with each row of $\mathbf{M}$. We do not need this additional randomization in our security analysis.

$k \in S'$. Using $\boldsymbol{\omega}_{S'}$ and $h_2^{\mathbf{m}_k^{\mathsf{T}}\mathbf{v}}$, the decrypter can compute $h_2^{\boldsymbol{\omega}_{S'}^{\mathsf{T}}\mathbf{M}_{S'}\mathbf{v}} = h_2^{\mathbf{e}_1^{\mathsf{T}}\mathbf{v}} = h_2^s$. Finally, the decryption algorithm can pair $A = (g_1 g_3)^t$ with h_2^s to obtain

$$D_{\mathsf{attrib}} = e(h_2^s, A) = e(h_2^s, (g_1 g_3)^t) = e(h_2, g_1)^{st},$$

since $h_2 \in \mathbb{G}_1$. Essentially, the decrypter should only be able to recover $e(h_2, g_1)^{st}$ if its set of attributes satisfy the policy. We note here that if an attribute $\rho(k) \notin S$, then $\hat{U}_{\rho(k)} \neq 1$; this property effectively "prevents" the decrypter from using $C_{3,k}$ during decryption since it would not be able to remove the extra $U_{\rho(k)}^{-s}$ component. The formal security analysis is more delicate and we defer to Section 5 for the exact analysis.

- **Slot check:** For the slot component, the decrypter takes its secret key r and computes

$$\begin{aligned} D_{\mathsf{slot}} = e(C_4, A) \cdot e(C_2, A^r) &= e\big(h_1^s g_1^{-sr}, (g_1 g_3)^t\big) \cdot e\big(g_1^s, (g_1 g_3)^{rt}\big) \\ &= e(h_1, g_1)^{st} \cdot e(g_1, g_1)^{-srt} \cdot e(g_1, g_1)^{srt} \\ &= e(h_1, g_1)^{st}, \end{aligned} \tag{2.2}$$

since $\hat{T} = g_1^r$ and $h_1 \in \mathbb{G}_1$. Essentially, the decrypter should only be able to recover $e(h_1, g_1)^{st}$ if it knows the secret key associated with the slot.

Recall now that h_1 and h_2 are a multiplicative secret sharing of h (i.e., $h_1 h_2 = h$). This means that if both of the policy check and the slot check passes (and in fact, *only* in this case), the decrypter is able to recover $e(h, g_1)^{st}$. This can now be combined with the message-embedding ciphertext components to recover the original message:

$$\frac{C_1 \cdot e(h, g_1)^{st}}{e(C_2, B)} = \frac{\mu \cdot e(g_1, g_1)^{\alpha s} e\big(h, g_1\big)^{st}}{e\big(g_1^s, g_1^\alpha h^t g_3^\tau\big)} = \frac{\mu \cdot e(g_1, g_1)^{\alpha s} \cdot e\big(h, g_1\big)^{st}}{e(g_1, g_1)^{\alpha s} \cdot e(h, g_1)^{st}} = \mu,$$

again using the fact that $h \in \mathbb{G}_1$.

Extending to multiple slots via key aggregation. To extend to an L-slot scheme, we essentially "concatenate" L *independent* copies of the single-slot scheme in the CRS. Specifically, for each slot $i \in [L]$, the CRS contains a set of slot-specific components and a set of attribute-specific components (in addition to the same set of general components from the single-slot scheme):

- **Slot-specific components:** Sample a slot-specific exponent $t_i \xleftarrow{R} \mathbb{Z}_N$ and a blinding factor $\tau_i \xleftarrow{R} \mathbb{Z}_N$, and let $A_i \leftarrow (g_1 g_3)^{t_i}$ and $B_i \leftarrow g_1^\alpha h^{t_i} g_3^{\tau_i}$.
- **Attribute-specific components:** For each attribute $w \in \mathcal{U}$, sample an attribute-specific exponent $u_{i,w} \xleftarrow{R} \mathbb{Z}_N$ and let $U_{i,w} \leftarrow g_1^{u_{i,w}}$.

The CRS consists of the general components, the slot-specific components, and the attribute-specific components for each of the slots:

$$\mathsf{crs} = \big(\mathcal{G}\,,\, g_1\,,\, g_3\,,\, Z\,,\, h\,,\, \{(A_i, B_i)\}_{i \in [L]}\,,\, \{U_{i,w}\}_{i \in [L], w \in \mathcal{U}}\big).$$

Next, we need a way to *aggregate* the public keys for the different slots into a single *compact* master public key mpk. Let $\{\mathsf{pk}_i\}_{i \in [L]}$ be a collection of public keys where $\mathsf{pk}_i = T_i = g_1^{r_i}$ is the public key associated with slot i. Let $S_i \subseteq \mathcal{U}$ be the set of attributes associated with pk_i. Our aggregation mechanism is simple: the aggregated public key components $\hat{T}, \hat{U}_w$ simply correspond to the *product* of the components associated with each slot:

$$\hat{T} = \prod_{j \in [L]} T_j \quad \text{and} \quad \hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_{j,w}.$$

The structure of the mpk is the same as in Eq. (2.1). Importantly, the size of the master public key is *independent* of the number of slots. The encryption algorithm also remains the same as in the single-slot case.

Cross term cancellation for decryption. When a message is encrypted with respect to an *aggregated* key, the ciphertext components are now a function of the exponents associated with *all* of the slots. However, the decrypter only has a key for a single slot (e.g., r_i), so the decrypter needs additional helper information in order to decrypt. To illustrate this, consider the decryption relation associated with the slot check (Eq. (2.2)). Suppose we are decrypting for slot i (using secret exponent r_i). Then,

$$\begin{aligned} D_{\text{slot}} = e(C_4, A_i) \cdot e\big(C_2, A_i^{r_i}\big) &= e\big(h_1^s \hat{T}^{-s}, (g_1 g_3)^{t_i}\big) \cdot e\big(g_1^s, (g_1 g_3)^{r_i t_i}\big) \\ &= e(h_1, g_1)^{s t_i} \cdot e(g_1, g_1)^{-s r_i t_i} \prod_{j \neq i} e(g_1, g_1)^{-s r_j t_i} \cdot e(g_1, g_1)^{s r_i t_i} \\ &= e(h_1, g_1)^{s t_i} \prod_{j \neq i} e(g_1, g_1)^{-s r_j t_i}, \end{aligned} \tag{2.3}$$

using the fact that $\hat{T} = \prod_{j \in [L]} T_j = \prod_{j \in [L]} g_1^{r_j}$. This is the same expression from Eq. (2.2) in the single-slot setting, *except* we have an extra term $\prod_{j \neq i} e(g_1, g_1)^{-s r_j t_i}$ from the slots $j \neq i$. We refer to these terms as the "cross-terms" since they correspond to an interaction between the secret key for slot j with the slot exponent for slot i. We thus require a way to eliminate the cross terms. Here, we take an approach that is often encountered when using pairings for aggregation (e.g., aggregating openings for vector commitments [CF13, LM19] or aggregating proofs in the case of batch arguments [WW22]). The strategy is to have the user for slot i provide the cross-terms $V_{j,i} = A_j^{r_i} = (g_1 g_3)^{r_i t_j}$ for each $j \neq i$ as part of its public key pk_i. Given all of the cross-terms from all of the users, the key curator can compute a helper decryption key component $\hat{V}_i = \prod_{j \neq i} V_{i,j} = \prod_{j \neq i} (g_1 g_3)^{r_j t_i}$ for each slot i. Given $\hat{V}_i$, the decrypter can now compute

$$e(C_2, \hat{V}_i) = \prod_{j \neq i} e\big(g_1^s, (g_1 g_3)^{r_j t_i}\big) = \prod_{j \neq i} e(g_1, g_1)^{s r_j t_i},$$

which precisely cancels out the extra term in Eq. (2.3). Finally, observe that the additional helper decryption component is just a single group element and is again, *independent* of the number of slots. This means that the size of the master public key, the size of the helper decryption components, as well as the encryption and decryption times are independent of the number of slots. Only the (one-time) key-generation and registration costs scale with the number of slots. We introduce a similar cross-term cancellation approach for the attribute-specific components and refer to Section 5.2 for the full description and analysis.

Security analysis. To prove security of our construction, we follow the dual-system methodology [Wat09, LW10]. While traditional dual-system proofs modify the distribution of the secret keys and the ciphertexts given out in the security game, in the registered ABE setting, we modify the distribution of the *slot parameters* and the ciphertexts. In more detail, in the security proof, we introduce modified ciphertexts (referred to as "semi-functional ciphertexts") and slot components (referred to as "semi-functional slots"). Keys registered to a semi-functional slot can be used to decrypt normal ciphertexts (i.e., those output by the honest encryption algorithm) and keys registered to a normal slot can be used to decrypt semi-functional ciphertexts. However, a key registered to a semi-functional slot is unable to decrypt a semi-functional ciphertext. The proof then proceeds via a hybrid argument where we first switch the challenge ciphertext from a normal ciphertext to a semi-functional one. Then, we switch the parameters associated with each slot from normal to semi-functional. In the final experiment then, all of the slots are semi-functional, as is the challenge ciphertext. Since keys associated with semi-functional slots cannot be used to decrypt a semi-functional ciphertext, arguing semantic security in the final experiment is straightforward. We give the full proof in Section 5. Here, we highlight two of the technical challenges that arise in the proof:

- **Malformed public keys:** In registered ABE, the adversary is allowed to submit *arbitrary* public keys to the key curator. In the security proof (and even for correctness), it will be important that the public keys are well-formed (and in particular, that the cross-terms are properly constructed). To enable this, we introduce a validity-check mechanism that uses the pairing to check that the components of the public key are properly computed. In the security proof (Claim 5.12), we show that the only public keys an efficient adversary can construct that pass the validity check are those in the support of the honest key-generation algorithm. Note that an alternative approach to rule out malformed public keys is to have users include a non-interactive zero-knowledge proof of knowledge of their public key that certifies well-formedness of the public key. However, doing so generically would either bring in random oracles [FS86] or require making non-black-box use of cryptography. Hence, we opt for a simpler algebraic mechanism that integrates directly with the rest of our construction.

- **Arguing semantic security.** A standard proof strategy for arguing security of an ABE scheme based on linear secret sharing is to construct a sequence of hybrid experiments such that in the final experiment, the challenge ciphertext *information-theoretically* hides the message by the security of the linear secret sharing scheme. This strategy applies if all of the keys the adversary possesses do *not* satisfy the challenge policy, and indeed, this property is enforced in the standard ABE security experiment. In registered ABE, the scenario is slightly different since there are two possibilities we have to consider:
  - The adversary knows the secret key associated with slot i and the attributes associated with slot i do *not* satisfy the challenge policy; or
  - The adversary does *not* know the secret key associated with slot i. In this case, it could be the case that the attributes associated with slot i do satisfy the challenge policy.

  Handling these two cases requires two different information-theoretic arguments: the first relies on the linear secret sharing scheme while the second relies on the secret key r_i for slot i to be hidden from the view of the adversary. Setting up these information-theoretic arguments requires slightly different distributions on the slot components. Consequently, we rely on two *different* sequence of hybrid experiments to handle the two cases. We refer to Section 5 (and specifically, the proof of Lemma 5.16) for more details.

We refer to Section 5 for the full construction and analysis of our slotted registered ABE scheme.

3 Preliminaries

Throughout this work, we write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$, and $[0, n]$ to denote the set $\{0, \ldots, n\}$. We use bold uppercase letters (e.g., $\mathbf{M}$) to denote matrices and bold lowercase letters (e.g., $\mathbf{v}$) to denote vectors. We use non-boldface letters to refer to their components (e.g., $\mathbf{v} = [v_1, \ldots, v_n]$). For a positive integer $N \in \mathbb{N}$, we write $\mathbb{Z}_N$ to denote the integers modulo N.

We write $\mathsf{poly}(\lambda)$ to denote a function that is $O(\lambda^c)$ for some constant $c \in \mathbb{N}$ and $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all $c \in \mathbb{N}$. We say that 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. We say that two families of distributions $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_2 = \{\mathcal{D}_{2,\lambda}\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if no efficient algorithm can distinguish them with non-negligible probability. We say they are statistically indistinguishable if the statistical distance $\Delta(\mathcal{D}_1, \mathcal{D}_2)$ is bounded by a negligible function in λ.

Access structures and linear secret sharing. We also recall the definition of monotone access structures and linear secret sharing which we will use in this work.

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 $B, C \in 2^S$, if $B \in \mathbb{A}$ and $B \subseteq C$, then $C \in \mathbb{A}$.

Definition 3.2 (Linear Secret Sharing Scheme [Bei96]). Let $\mathcal{P}$ be a set of parties. A linear secret sharing scheme over a ring $\mathbb{Z}_N$ for $\mathcal{P}$ is a pair $(\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_N^{\ell \times n}$ is a "share-generating" matrix and $\rho \colon [\ell] \to \mathcal{P}$ 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 $v_2, \ldots, v_n \xleftarrow{\text{R}} \mathbb{Z}_N$ 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 $S \subseteq \mathcal{P}$ 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 $\mathbf{e}_1^{\mathsf{T}} = [1, 0, \dots, 0]$ denotes the first elementary basis vector. Conversely, if $S \subseteq$ is an *unauthorized* sets 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 $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$).

Remark 3.3 (One-Use Restriction)**.** In this work, we construct a registered ABE scheme (Section 5) that supports any policy that can be described by a linear secret sharing scheme (Definition 3.2), with the restriction that each attribute is associated with at most one row of $\mathbf{M}$. This corresponds to policies $(\mathbf{M}, \rho)$ where the row-labeling function ρ is *injective*. As shown in Lewko et al. [LOS+10, §2.2], it is straightforward to extend a scheme with the one-use restriction into one where attributes can be used up to k times by expanding the public parameters and secret keys by a factor of k (i.e., split each attribute into k independent copies).

Remark 3.4 (Monotone Boolean Formulas)**.** Our pairing-based registered ABE construction (Section 5) supports monotone access policies that can be described by any (one-use) linear secret sharing scheme. As a special case, this captures the class of monotone Boolean formulas. There are multiple ways to take a monotone Boolean formula and express it as a linear secret sharing scheme; we refer to [LW11, Appendix G] for one such approach.

4 Registered Attribute-Based Encryption

In this section, we introduce the notion of a registered attribute-based encryption scheme for a polynomial-size attribute space. Our definition is an adaptation of the notion of registration-based encryption (RBE) [GHMR18] to the more general attribute-based setting. We compare some features of our definition with RBE in Remark 4.6.

Definition 4.1 (Registered Attribute-Based Encryption)**.** Let λ be a security parameter. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ be a universe of attributes and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of policies on $\mathcal{U}$. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$ be the message space. A registered attribute-based encryption scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and message space $\mathcal{M}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{R\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}) \to \mathsf{crs}$: On input the security parameter λ and the size of the attribute universe $\mathcal{U}$, the setup algorithm outputs a common reference string crs.

- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{aux}) \to (\mathsf{pk}, \mathsf{sk})$: On input the common reference string crs, and a (possibly empty) state aux, the key-generation algorithm outputs a public key pk and a secret key sk.

- $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S_{\mathsf{pk}}) \to (\mathsf{mpk}, \mathsf{aux}')$: On input the common reference string crs, a (possibly empty) state aux, a public key pk, and a set of attributes $S_{\mathsf{pk}} \subseteq \mathcal{U}$, the registration algorithm *deterministically* outputs the master public key mpk and an updated state aux′.

- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu) \to \mathsf{ct}$: On input the master public key mpk, an access policy $P \in \mathcal{P}$, and a message $\mu \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.

- $\mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}) \to \mathsf{hsk}$: On input the common reference string crs, a state aux, and a public key pk, the update algorithm *deterministically* outputs a helper decryption key hsk.

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct}) \to \mathcal{M} \cup \{\perp, \mathsf{GetUpdate}\}$: On input the master public key mpk, a secret key sk, a helper decryption key hsk, and a ciphertext ct, the decryption algorithm either outputs a message $\mu \in \mathcal{M}$, a special symbol $\perp$ to indicate a decryption failure, or a special flag GetUpdate that indicates an updated helper decryption key is needed to decrypt.

Correctness and efficiency. We now define the correctness and efficiency requirements on a registered ABE scheme. At a high level, correctness says that if a user properly registers her public key along with a set of attributes, then she can use her secret key to decrypt all future ciphertexts ct encrypted under the resulting (and any subsequent) master public key, provided that her set of attributes satisfy the policy associated with the ciphertext. Notably, this should hold even if malicious users register (possibly-malformed) keys. In other words, as long as the key curator is semi-honest, an adversary cannot register "bad" keys to cause decryption to fail for an honest user. The main efficiency requirements we impose is that the size of the master public key and the size of each user's helper decryption key should be compact (i.e., polylogarithmic in the total number of users). We compare our notion with the RBE definition in Remark 4.6. We now give the formal definition:

Definition 4.2 (Correctness and Efficiency of Registered ABE)**.** Let $\Pi_{\mathsf{R\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ be a registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and message space $\mathcal{M}$. For a security parameter λ and an adversary $\mathcal{A}$, we define the following game between $\mathcal{A}$ and the challenger:

- **Setup phase:** The challenger starts by sampling the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|})$. It then initializes the auxiliary input $\mathsf{aux} \leftarrow \perp$ and initial master public key $\mathsf{mpk}_0 \leftarrow \perp$. It also initializes a counter $\mathsf{ctr}[\mathsf{reg}] \leftarrow 0$ to keep track of the number of registration queries and another counter $\mathsf{ctr}[\mathsf{enc}] \leftarrow 0$ to keep track of the number of encryption queries. Finally, it initializes $\mathsf{ctr}[\mathsf{reg}]^* \leftarrow \infty$ as the index for the target key (which will also be updated during the course of the game). Finally, it gives crs to $\mathcal{A}$.
- **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 public key pk and a set of attributes $S \subseteq \mathcal{U}$. The challenger first increments the counter $\mathsf{ctr}[\mathsf{reg}] \leftarrow \mathsf{ctr}[\mathsf{reg}] + 1$ and then registers the key by computing $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S)$. The challenger updates its auxiliary data by setting $\mathsf{aux} \leftarrow \mathsf{aux}'$ and replies to $\mathcal{A}$ with $(\mathsf{ctr}[\mathsf{reg}], \mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux})$.
  - **Register target key query:** In a target-key registration query, the adversary specifies a set of attributes $S^* \subseteq \mathcal{U}$. If the adversary has previously made a target-key registration query, then the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr}[\mathsf{reg}] \leftarrow \mathsf{ctr}[\mathsf{reg}] + 1$, samples $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and registers $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*, S^*)$. It computes the helper decryption key $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$. The challenger updates its auxiliary data by setting $\mathsf{aux} \leftarrow \mathsf{aux}'$, stores the index of the target identity $\mathsf{ctr}[\mathsf{reg}]^* \leftarrow \mathsf{ctr}[\mathsf{reg}]$, and replies to $\mathcal{A}$ with $(\mathsf{ctr}[\mathsf{reg}], \mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{hsk}^*, \mathsf{sk}^*)$.
  - **Encryption query:** In an encryption query, the adversary submits the index $\mathsf{ctr}[\mathsf{reg}]^* \leq i \leq \mathsf{ctr}[\mathsf{reg}]$ of a public key,[7] a message $\mu_{\mathsf{ctr}[\mathsf{enc}]} \in \mathcal{M}$, and a policy $P_{\mathsf{ctr}[\mathsf{enc}]} \in \mathcal{P}$. If the adversary has not yet registered a target key, or if the target set of attributes S^* does not satisfy the policy $P_{\mathsf{ctr}[\mathsf{enc}]}$, the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr}[\mathsf{enc}] \leftarrow \mathsf{ctr}[\mathsf{enc}] + 1$ and computes $\mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}_i, P_{\mathsf{ctr}[\mathsf{enc}]}, \mu_{\mathsf{ctr}[\mathsf{enc}]})$. The challenger replies to $\mathcal{A}$ with $(\mathsf{ctr}[\mathsf{enc}], \mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]})$.
  - **Decryption query:** In a decryption query, the adversary submits a ciphertext index $1 \leq j \leq \mathsf{ctr}[\mathsf{enc}]$. The challenger computes $m_j' \leftarrow \mathsf{Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_j)$. If $m_j' = \mathsf{GetUpdate}$, then the challenger computes an updated helper decryption key $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$ and recomputes $m_j' \leftarrow \mathsf{Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_j)$. If $m_j' \neq m_j$, the experiment halts with outputs $b = 1$.

  If the adversary has finished making queries and the experiment has not halted (as a result of a decryption query), then the experiment outputs $b = 0$.

We say that $\Pi_{\mathsf{R\text{-}ABE}}$ is correct and efficient if for all (possibly unbounded) adversaries $\mathcal{A}$ making at most a polynomial number of queries, the following properties hold:

- **Correctness:** There exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the above game. We say the scheme satisfies *perfect correctness* if $\Pr[b = 1] = 0$.

[7] Since we are requiring correctness to hold with respect to the target key, we only consider ciphertexts encrypted to master public keys constructed *after* the target key has been registered.

- **Compactness:** Let N be the number of registration queries the adversary makes in the above game. There exists a universal polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for all $i \in [N]$, $|\mathsf{mpk}_i| = \mathsf{poly}(\lambda, |\mathcal{U}|, \log i)$. We also require that the size of the helper decryption key hsk^* satisfy $|\mathsf{hsk}^*| = \mathsf{poly}(\lambda, |\mathcal{U}|, \log N)$ (at *all* points in the game).

- **Update efficiency:** Let N be the number of registration queries the adversary makes in the above game. Then, in the course of the above game, the challenger invokes the update algorithm Update at most $O(\log N)$ times, where each invocation runs in $\mathsf{poly}(\log N)$ time in the RAM model of computation. Specifically, we model Update as a RAM program that has *random* access to its input; thus, the running time of Update in the RAM model can be *smaller* than the input length.

Registered ABE security. The security requirement for a registered ABE scheme is analogous to the standard ABE security notion. Namely, semantic security should hold for a ciphertext associated with a policy P if the user only has keys registered to attribute sets $S_1, \ldots, S_k$ which do *not* satisfy the policy. In the security game, we allow the adversary the ability to register users with a set of attributes that *do* satisfy the challenge policy, provided the adversary does *not* know the user's secret key (i.e., they are generated honestly by the challenger). In addition, the adversary is allowed to register (arbitrary) public keys for attribute sets of its choosing, provided that none of them satisfy the challenge policy. We give the formal definition below:

Definition 4.3 (Security of Registered ABE)**.** Let $\Pi_{\mathsf{R\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ be a registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and message space $\mathcal{M}$. 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 samples the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|})$. It then initializes the auxiliary input $\mathsf{aux} \leftarrow \bot$, the initial master public key $\mathsf{mpk} \leftarrow \bot$, a counter $\mathsf{ctr} \leftarrow 0$ for the number of honest-key-registration queries the adversary has made, an empty set of keys $C \leftarrow \varnothing$ (to keep track of corrupted public keys), and an empty dictionary mapping public keys to registered attribute sets $\mathsf{D} \leftarrow \varnothing$. For notational convenience, if $\mathsf{pk} \notin \mathsf{D}$, then we define $\mathsf{D}[\mathsf{pk}] \coloneqq \varnothing$. to be the empty set. The challenger gives the crs to $\mathcal{A}$.

- **Query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
  - **Register corrupted key query:** In a corrupted-key-registration query, the adversary $\mathcal{A}$ specifies a public key pk and a set of attributes $S \subseteq \mathcal{U}$. The challenger registers the key by computing $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S)$. The challenger updates its copy of the public key $\mathsf{mpk} \leftarrow \mathsf{mpk}'$, its auxiliary data $\mathsf{aux} \leftarrow \mathsf{aux}'$, adds pk to C, and updates $D[\mathsf{pk}] \leftarrow D[\mathsf{pk}] \cup \{S\}$. It replies to $\mathcal{A}$ with $(\mathsf{mpk}', \mathsf{aux}')$.
  - **Register honest key query:** In an honest-key-registration query, the adversary specifies a set of attributes $S \subseteq \mathcal{U}$. The challenger increments the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and samples $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{aux})$, and registers $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}_{\mathsf{ctr}}, S)$. The challenger updates its public key $\mathsf{mpk} \leftarrow \mathsf{mpk}'$, its auxiliary data $\mathsf{aux} \leftarrow \mathsf{aux}'$, and $D[\mathsf{pk}_{\mathsf{ctr}}] \leftarrow D[\mathsf{pk}_{\mathsf{ctr}}] \cup \{S\}$. It replies to $\mathcal{A}$ with $(\mathsf{ctr}, \mathsf{mpk}', \mathsf{aux}', \mathsf{pk}_{\mathsf{ctr}})$.
  - **Corrupt honest key query:** In a corrupt-honest-key query, the adversary specifies an index $1 \le i \le \mathsf{ctr}$. Let $(\mathsf{pk}_i, \mathsf{sk}_i)$ be the i^{th} public/secret key the challenger samples when responding to the i^{th} honest-key-registration query. The challenger adds pk_i to C and replies to $\mathcal{A}$ with sk_i.

- **Challenge phase:** The adversary $\mathcal{A}$ chooses two messages $\mu_0^*, \mu_1^* \in \mathcal{M}$ and an access policy $P^* \in \mathcal{P}$. The challenger replies with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P^*, \mu_b^*)$.

- **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$.

Let $\mathcal{S} = \{S \in \mathsf{D}[\mathsf{pk}] : \mathsf{pk} \in C\}$ be the set of corrupted attributes. We say that an adversary $\mathcal{A}$ is admissible if the challenge policy P^* is not satisfied by any attribute set $S \in \mathcal{S}$. Note that it could be the case that P^* is satisfied by the attributes S from an honest key query (that was not subsequently corrupted). We say that a 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.

Definition 4.4 (Bounded Registered ABE)**.** We say a registered ABE scheme $\Pi_{\text{R-ABE}}$ is *bounded* if there is an *a priori* bound on the number of registered users in the system. In this setting, we modify Setup to takes as input an additional bound parameter 1^L which specifies the maximum number of registered users. In the correctness and security definitions (Definitions 4.2 and 4.3), we allow the adversary to specify the bound 1^L at the beginning, and in the games themselves, the adversary can make up to L queries (the challenger answers subsequent registration queries with $\perp$).

Remark 4.5 (Adaptive Corruptions)**.** We could also consider a version of registered ABE which allows for corruption queries after the challenge phase. However, such a definition is in fact equivalent to the current definition, since the adversary can always make all admissible corruption queries in the pre-challenge phase before submitting the challenge policy. This step relies on the fact that the registration algorithm is *deterministic*, so its behavior can be entirely simulated by the adversary. In the slotted setting (Section 4.1), we give a formal proof of equivalence for these two notions (see Lemma 4.10). The analogous argument applies to the full registered ABE security game (Definition 4.3).

Remark 4.6 (Comparison with Registration-Based Encryption)**.** The correctness and security definitions of our registered ABE scheme are essentially generalizations of the corresponding definitions for registration-based encryption introduced by Garg et al. [GHM+19] to the setting of attribute-based encryption. The main difference is that we do not impose efficiency requirements on the running time of the key-generation and registration algorithms whereas Garg et al. require that they run in time $\text{poly}(\lambda, \log n)$, where n is the number of registered users. In the case of our main pairing-based construction (Corollary 6.9), the running time of key-generation and registration scale with the *bound* on the *maximum* number of registered users. This roughly corresponds to the notion of "weak efficiency" in the language of Garg et al. Our obfuscation-based registered ABE scheme in Section 7 supports efficient key-generation and registration. It is an interesting question to construct a registered ABE scheme with efficient key-generation and registration without obfuscation (or making non-black-box use of cryptographic primitives).

A difference in the security definition is we additionally allow the adversary to register an honest user, and then later on corrupt the user and learn its secret key. Registration-based encryption did not allow for corruption queries.

Remark 4.7 (Transparent Key Curator)**.** When using a registered ABE scheme, a key curator is responsible for maintaining the auxiliary data and processing user registrations. Just like in RBE, the key curator in a registered ABE scheme is entirely *transparent* and maintains no secrets. Indeed, both the registration and update algorithms are *deterministic*, so an independent party (or a user) can audit the key curator and verify whether it is behaving honestly or not. Note however that we cannot prevent a malicious key curator from registering a set of attributes that allow it to decrypt all ciphertexts (this is analogous to a malicious key curator registering a key for a target user of its choosing in the setting of RBE). However, such activity is always detectable by an external auditor.

Remark 4.8 (Universe Size)**.** Definition 4.2 allows the size of the CRS, the master public key and the helper decryption keys (and by extension, the size of the ciphertext) to scale with the size of the attribute universe $\mathcal{U}$. This means our definition is currently tailored for a polynomial-size attribute space. We could define an analogous "large-universe" version of our definition where the size of the CRS, the master public key and helper decryption keys scale with $\text{poly}(\log |\mathcal{U}|)$; in this case, the Setup algorithm would take the universe size in *binary* rather than *unary*. Our pairing-based construction (Section 5) only supports a polynomial-size attribute universe while our obfuscation-based construction (Section 7) supports an arbitrary universe size. It is an interesting question to extend our pairing-based construction to the large-universe setting where the set of attributes can be an arbitrary bit-string. Note that even a small-universe ABE captures notions like identity-based encryption (e.g., the number of attributes would be linear in the *bit-length* of the identity).

4.1 Slotted Registered Attribute-Based Encryption

In this section, we formally introduce the notion of a slotted registered ABE scheme which is the core building block underlying our pairing-based construction (Section 5) and obfuscation-based construction (Section 7). Then in Section 6, we show how to compile a slotted registered ABE scheme into a standard registered ABE scheme.

Definition 4.9 (Slotted Registration-Based Encryption)**.** Let λ be a security parameter. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be a universe of attributes and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies on $\mathcal{U}$. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be the message space. A slotted registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and message space $\mathcal{M}$ is a tuple of efficient algorithms $\Pi_{\mathsf{sRBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L) \to \mathsf{crs}$: On input the security parameter λ, the size of the universe $\mathcal{U}$, and the number of slots L, the setup algorithm outputs a common reference string crs.
- $\mathsf{KeyGen}(\mathsf{crs}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the common reference string crs, a slot index $i \in [L]$, the key-generation algorithm outputs a public key pk_i and a secret key sk_i for slot i.
- $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) \to \{0,1\}$: On input the common reference string crs, a slot index $i \in [L]$, and a public key pk_i, the key-validation algorithm outputs a bit $b \in \{0,1\}$ indicating whether pk_i is valid or not. This algorithm is *deterministic*.
- $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)) \to (\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L)$: On input the common reference string crs and a list of public keys and the associated attributes $(\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)$, the aggregate algorithm outputs the master public key mpk and a collection of helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$. This algorithm is *deterministic*.
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu) \to \mathsf{ct}$: On input the master public key mpk, an access policy $P \in \mathcal{P}$, and a message $\mu \in \mathcal{M}$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct}) \to m$: On input a decryption key sk, the helper decryption key hsk, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M} \cup \{\bot\}$. This algorithm is *deterministic*.

Moreover, the above algorithms should satisfy the following properties:

- **Completeness:** For all parameters $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, all attribute universes $\mathcal{U}$, and all indices $i \in [L]$,
$$\Pr\left[\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L); (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)\right] = 1.$$
- **Correctness:** We say Π_{sRBE} is correct if for all security parameters $\lambda \in \mathbb{N}$, all attribute universes $\mathcal{U}$, all slot lengths $L \in \mathbb{N}$, all indices $i \in [L]$, if we sample $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$, then for all collections of public keys $\{\mathsf{pk}_j\}_{j\neq i}$ (which may be correlated with pk_i) where $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$, all messages $\mu \in \mathcal{M}$, all sets of attributes $S_1, \ldots, S_L \subseteq \mathcal{U}$, all policies $P \in \mathcal{P}$ where S_i satisfies policy P, the following holds:
$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{hsk}_i, \mathsf{ct}) = \mu : \begin{array}{c} (\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P, \mu) \end{array}\right] = 1,$$
where the probability is taken over the randomness in Setup, KeyGen, and Encrypt.
- **Compactness:** There exists a universal polynomial $\mathrm{poly}(\cdot,\cdot,\cdot)$ such that the length of the master public key and individual helper secret keys output by Aggregate are $\mathrm{poly}(\lambda, |\mathcal{U}|, \log L)$.
- **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:** The adversary $\mathcal{A}$ sends a slot count 1^L to the challenger. The challenger then samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$ and gives crs to $\mathcal{A}$. The challenger also initializes a counter $\mathsf{ctr} \leftarrow 0$, a dictionary D, and a set of slot indices $C \leftarrow \varnothing$.
  - **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 $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$ and replies with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$ to $\mathcal{A}$. The challenger adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary D.

        * **Corruption query:** In a 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:** For each slot $i \in [L]$, adversary $\mathcal{A}$ must specify a tuple $(c_i, S_i, \mathsf{pk}_i^*)$ where either $c_i \in \{1, \ldots, \mathsf{ctr}\}$ to reference a challenger-generated key or $c_i = \bot$ to reference a key outside this set. The adversary also specifies a challenge policy $P^* \in \mathcal{P}$ and two messages $\mu_0^*, \mu_1^* \in \mathcal{M}$. The challenger responds by first constructing pk_i as follows:

        * If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, then the challenger looks up the entry $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, then the challenger sets $\mathsf{pk}_i \leftarrow \mathsf{pk}'$. Moreover, if the adversary previously issued a "corrupt identity" query on index c_i, then the challenger adds the slot index i to C. Otherwise, if $i \ne i'$, then the experiment halts.
        * If $c_i = \bot$, then the challenger checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, the experiment halts. If the key is valid, the challenger sets $\mathsf{pk}_i \leftarrow \mathsf{pk}_i^*$ and adds the slot index i to C.

      The challenger computes $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$ and replies with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P^*, \mu_b^*)$. Note that because Aggregate is *deterministic* and can be run by $\mathcal{A}$ itself, there is no need to additionally 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:

        * **Corruption query:** In a corruption query, the adversary specifies an index $c \in \{1, \ldots, \mathsf{ctr}\}$. In response the challenger looks up the tuple $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[c]$ and replies to $\mathcal{A}$ with sk'. Moreover, if the adversary registered a tuple of the form (c, S, pk^*) in the challenge phase for some choice of $S \subseteq \mathcal{U}$ and pk^*, then the challenger adds the slot index $i' \in [L]$ to 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 for all corrupted slot indices $i \in C$, the set S_i does not satisfy P^* (i.e., the attributes associated with a corrupted slot does not satisfy the challenge policy). Finally, we say that a slotted registration-based encryption scheme 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.

The security requirement in Definition 4.9 allows the adversary to issue additional corruption queries in a post-challenge query phase. However, as we show below, it suffices to argue security in the simpler setting where there are no post-challenge queries. Security in the setting without post-challenge queries implies security in the setting with post-challenge queries.

Lemma 4.10 (Security without Post-Challenge Queries)**.** *Suppose a slotted registered ABE scheme* $\Pi_{\mathsf{sRBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ *is secure against all efficient adversaries* $\mathcal{A}$ *that does not make any post-challenge queries. Then it is also a secure slotted registered ABE scheme (in the sense of Definition 4.9).*

Proof. Let $\mathcal{A}$ be an efficient adversary that wins the slotted registered ABE security game with non-negligible probability ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the slotted registered ABE security game that wins with the same advantage, but makes *no* post-challenge queries:

- **Setup phase:** Algorithm $\mathcal{A}$ outputs a slot count 1^L. Algorithm $\mathcal{B}$ forwards 1^L to the challenger. The challenger replies to $\mathcal{B}$ with a $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$. Algorithm $\mathcal{B}$ gives crs to $\mathcal{A}$. Algorithm $\mathcal{B}$ also initializes an (empty) dictionary D.

- **Pre-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:

- **Key-generation query:** When algorithm $\mathcal{A}$ makes a key-generation query on a slot index $i \in [L]$, algorithm $\mathcal{B}$ forwards the same key-generation query to its challenger. Let $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$ be the challenger's response to the query. Algorithm $\mathcal{B}$ then adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \perp)$ to its dictionary D. It replies with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$ to $\mathcal{A}$.
- **Corruption query:** When algorithm $\mathcal{A}$ makes a corruption query on an index c, algorithm $\mathcal{B}$ forwards the same query to its challenger. Let $\mathsf{sk}_{\mathsf{ctr}}$ be the challenger's response. Algorithm $\mathcal{B}$ then looks up the value $(i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}') \leftarrow \mathsf{D}[\mathsf{ctr}]$ and updates $\mathsf{D}[\mathsf{ctr}]$ to $(i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$. It then replies to $\mathcal{A}$ with $\mathsf{sk}_{\mathsf{ctr}}$.

- **Challenge phase:** For each slot $i \in [L]$, adversary $\mathcal{A}$ outputs a tuple $(c_i, S_i, \mathsf{pk}_i^*)$ where either $c_i \in [N]$ or $c_i = \perp$, where N is the number of key-generation queries the adversary made. In addition, algorithm $\mathcal{A}$ outputs a challenge policy $P^* \in \mathcal{P}$ and two messages $\mu_0^*, \mu_1^* \in \mathcal{M}$. For each $c \in [N]$, algorithm $\mathcal{B}$ then checks if for all slot indices $i \in [L]$ where $c_i = c$, the associated set of of attributes S_i does not satisfy P^* (i.e., $P^*(S_i) = 0$). If this holds, then $\mathcal{B}$ submits a corrupt-identity query on c to obtain a secret key sk_c. Algorithm $\mathcal{B}$ looks up the mapping $(i, \mathsf{pk}_c, \mathsf{sk}') \leftarrow \mathsf{D}[c]$ and updates its value to $(i, \mathsf{pk}_c, \mathsf{sk}_c)$. At this point, algorithm $\mathcal{B}$ issues a challenge query on the same challenge $\big(\mu_0^*, \mu_1^*, P^*, \{(c_i, S_i, \mathsf{pk}_i^*)\}_{i \in [L]}\big)$. Let ct^* be the challenge ciphertext. Algorithm $\mathcal{B}$ forwards ct^* to $\mathcal{A}$.
- **Post-challenge query phase:** Adversary $\mathcal{A}$ can now issue additional corruption queries. On input an index c, algorithm $\mathcal{B}$ looks up the value $(i, \mathsf{pk}_c, \mathsf{sk}_c) \leftarrow \mathsf{D}[c]$. If $\mathsf{sk}_c = \perp$, algorithm $\mathcal{B}$ aborts. Otherwise, it replies to $\mathcal{A}$ with sk_c.
- **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which $\mathcal{B}$ also outputs.

By construction, algorithm $\mathcal{B}$ is a slotted registered ABE adversary that does not make any post-challenge corruption queries. Moreover, algorithm $\mathcal{B}$ perfectly simulates an execution of the security game for $\mathcal{A}$, provided that $\mathcal{A}$ does not issue any post-challenge queries that cause $\mathcal{B}$ to abort. Now, algorithm $\mathcal{B}$ aborts only if the value $\mathsf{D}[c]$ is of the form $(i, \mathsf{pk}, \perp)$. This will be the case only if $\mathcal{A}$ submits a post-challenge corrupt-identity query on an index $c \in [N]$ and there exists a tuple $(c_i, S_i, \mathsf{pk}_i^*)$ in the challenge tuple where $c_i = c$ and S_i satisfies the challenge policy P. Such queries are not allowed if $\mathcal{A}$ is admissible. Thus, $\mathcal{B}$ perfectly simulates the security game for $\mathcal{A}$. Moreover, if $\mathcal{A}$'s pre-challenge queries are admissible, then all of algorithm $\mathcal{B}$'s queries are admissible. This is because the additional queries algorithm $\mathcal{B}$ makes in the security game (after $\mathcal{A}$ outputs the challenge) are admissible by construction. □

5 Slotted Registered ABE from Pairings

In this section, we show how to construct a slotted registered ABE scheme for policies describable by a linear secret sharing scheme using composite-order bilinear maps.

5.1 Preliminaries: Composite-Order Pairing Groups

Our pairing-based construction of slotted registered ABE will rely on composite-order pairing groups [BGN05]. We recall the formal definition below:

Definition 5.1 (Three-Prime Composite-Order Bilinear Group [BGN05])**.** A (symmetric) three-prime composite-order bilinear group generator is an efficient algorithm CompGroupGen that takes as input the security parameter λ and outputs a description $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, g, e)$ of a bilinear group where p_1, p_2, p_3 are distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order $N = p_1 p_2 p_3$, 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 $N = p_1 p_2 p_3$ and generator g. In the following, we will write $\mathbb{G}_1 = \langle g^{p_2 p_3} \rangle$ to denote the subgroup of $\mathbb{G}$ of order p_1. We define $\mathbb{G}_2$ and $\mathbb{G}_3$ analogously. By the Chinese Remainder Theorem, if g_1, g_2, g_3 are generators of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$, respectively, then $g_1 g_2 g_3 \in \mathbb{G}$ is a generator of $\mathbb{G}$, and moreover, every element $h \in \mathbb{G}$ can be *uniquely* written as $g_1^{x_1} g_2^{x_2} g_3^{x_3}$ where $x_1 \in \mathbb{Z}_{p_1}$, $x_2 \in \mathbb{Z}_{p_2}$, and $x_3 \in \mathbb{Z}_{p_3}$. 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 two 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. Finally, we state a simple implication (Lemma 5.3) from [LW10] of the assumptions that will be useful in our security analysis.

Assumption 5.2 (Subgroup Decision Assumptions [LW10])**.** Let CompGroupGen be a three-prime composite-order bilinear group generator. Let $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$, $N = p_1 p_2 p_3$, $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, and $g_1 \xleftarrow{\text{R}} \mathbb{G}_1$, $g_2 \xleftarrow{\text{R}} \mathbb{G}_2$, and $g_3 \xleftarrow{\text{R}} \mathbb{G}_3$. We now define several pairs of distributions $\mathcal{D}_0, \mathcal{D}_1$ where each distribution $\mathcal{D}_b = (D, T_b)$ consists of a set of common components D and a challenge element T_b. We say that each assumption below holds with respect to CompGroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

Assumption 5.2a: Sample $r \xleftarrow{\text{R}} \mathbb{Z}_N$, and let

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

Assumption 5.2b: Sample $s_{12}, s_{23}, r \xleftarrow{\text{R}} \mathbb{Z}_N$, and let

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

Assumption 5.2c: Sample $\alpha, s, t_1, t_2, r \xleftarrow{\text{R}} \mathbb{Z}_N$, and let

$$D = \left(\mathcal{G}, g_1, g_2, g_3, g_1^\alpha g_2^{t_1}, g_1^s g_2^{t_2}\right), \qquad T_0 = e(g_1, g_1)^{\alpha s}, \qquad T_1 = e(g, g)^r.$$

Lemma 5.3 (Hardness of Factoring [LW10, Lemma 5])**.** *Let* CompGroupGen *be a composite-order bilinear group generator where Assumption 5.2b holds. Then, for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\Pr\left[1 < \gcd(x, N) < N : \begin{array}{c} (\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda), \\ N \leftarrow p_1 p_2 p_3, \mathcal{G} \leftarrow (\mathbb{G}, \mathbb{G}_T, N, g, e), \\ g_1 \xleftarrow{\text{R}} \mathbb{G}_1, g_3 \xleftarrow{\text{R}} \mathbb{G}_3, s_{12}, s_{23} \xleftarrow{\text{R}} \mathbb{Z}_N \\ x \leftarrow \mathcal{A}\left(\mathcal{G}, g_1, g_3, (g_1 g_2)^{s_{12}}, (g_2 g_3)^{s_{23}}\right), \end{array}\right] = \mathsf{negl}(\lambda).$$

In words, given $\left(\mathcal{G}, g_1, g_3, (g_1 g_2)^{s_{12}}, (g_2 g_3)^{s_{23}}\right)$, no efficient adversary can output a non-trivial factor of N.

5.2 Slotted Registered ABE from Composite-Order Pairing Groups

In this section, we show how to construct a slotted registered ABE scheme from composite-order pairing groups.

Construction 5.4 (Slotted Attribute-Based Registration-Based Encryption)**.** Let CompGroupGen be a composite-order bilinear group generator, let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ be a (polynomial-size) attribute space, and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of policies that can be described by a (one-use) linear secret sharing scheme (Definition 3.2 and Remark 3.3) over $\mathcal{U}$. We construct a slotted attribute-based registration-based encryption scheme $\Pi_{\mathsf{R\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with message space $\mathcal{M} = \mathbb{G}_T$, attribute space $\mathcal{U}$, and policy space $\mathcal{P}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$: On input the security parameter λ, the size of the attribute space $\mathcal{U}$, and the number of slots L, the setup algorithm starts by sampling $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ be the subgroups of $\mathbb{G}$ of orders p_1, p_2, p_3, respectively. The setup algorithm now constructs the following quantities:
  - Let $N = p_1 p_2 p_3$ and let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ be the (public) group description.

  - Sample generators $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_3 \xleftarrow{R} \mathbb{G}_3$ and exponents $\alpha, \beta \xleftarrow{R} \mathbb{Z}_N$. Let $h \leftarrow g_1^{\beta}$.
  - For each slot index $i \in [L]$, sample exponents $t_i, \delta_i \xleftarrow{R} \mathbb{Z}_N$ and a slot blinding factor $\tau_i \xleftarrow{R} \mathbb{Z}_N$. Construct the slot components as follows:

$$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^{\alpha} A_i^{\beta} g_3^{\tau_i} \quad , \quad P_i \leftarrow (g_1 g_3)^{\delta_i}.$$

    Then, for each attribute $w \in \mathcal{U}$ and each slot $i \in [L]$, sample an exponent $u_{i,w} \xleftarrow{R} \mathbb{Z}_N$, and for each $j \in [L]$ with $j \neq i$, sample a blinding factor $\gamma_{i,j,w} \xleftarrow{R} \mathbb{Z}_N$. Construct the attribute components $U_{i,w}$ and $W_{i,j,w}$ as follows:

$$U_{i,w} \leftarrow g_1^{u_{i,w}} \quad , \quad W_{i,j,w} \leftarrow A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}}.$$

  - Finally, compute $Z \leftarrow e(g_1, g_1)^{\alpha}$ and output the common reference string

$$\mathsf{crs} = \big(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{i,j,w}\}_{i \neq j, w \in \mathcal{U}}\big) \tag{5.1}$$

- $\mathsf{KeyGen}(\mathsf{crs}, i)$: On input the common reference string crs (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

$$T_i \leftarrow g_1^{r_i} \quad , \quad Q_i \leftarrow P_i^{r_i} \quad , \quad R_i \leftarrow g_3^{r_i}.$$

  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:

$$\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i}) \quad \text{and} \quad \mathsf{sk}_i = r_i.$$

  Note that this particular key-generation algorithm does *not* depend on the set of attributes.

- $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$: On input the common reference string crs (with components given by Eq. (5.1)), a slot index $i \in [L]$, and a purported public key $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$, the key-validation algorithm first affirms that each of the components in pk_i is a valid group element (i.e., an element in $\mathbb{G}$). If so, it then checks

$$e(g_3, T_i) = 1 = e(g_1, R_i) \quad \text{and} \quad e(T_i, P_i) = e(g_1, Q_i) \quad \text{and} \quad e(R_i, P_i) = e(g_3, Q_i).$$

  Next, for each $j \neq i$, the algorithm checks that

$$e(g_1, V_{j,i}) = e(T_i, A_j) \quad \text{and} \quad e(g_3, V_{j,i}) = e(R_i, A_j).$$

  If all checks pass, it outputs 1; otherwise, it outputs 0.

- $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$: On input the common reference string crs (with components given by Eq. (5.1)), a collection of L public keys $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ together with their attribute sets $S_i \subseteq \mathcal{U}$, the aggregation algorithm starts by computing the attribute-independent public key $\hat{T}$ and the attribute-independent slot key $\hat{V}_i$ for each $i \in [L]$:

$$\hat{T} = \prod_{j \in [L]} T_j \quad , \quad \hat{V}_i = \prod_{j \neq i} V_{i,j}.$$

  Next, for each attribute $w \in \mathcal{U}$, it computes the attribute-specific public key $\hat{U}_w$ and the attribute-specific slot key $\hat{W}_{i,w}$ for each $i \in [L]$:

$$\hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_{j,w} \quad , \quad \hat{W}_{i,w} = \prod_{j \neq i : w \notin S_j} W_{i,j,w}$$

  Finally, it outputs the master public key mpk and the slot-specific helper decryption keys hsk_i where

$$\mathsf{mpk} = \big(\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}}\big) \quad \text{and} \quad \mathsf{hsk}_i = \big(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}}\big).$$

- $\mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{M}, \rho), \mu)$: On input the master public key $\mathsf{mpk} = (\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}})$, a policy $(\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$ is an injective row-labeling function, and a message $\mu \in \mathbb{G}_T$, the encryption algorithm starts by sampling a secret exponent $s \xleftarrow{R} \mathbb{Z}_N$ and $h_1, h_2 \xleftarrow{R} \mathbb{G}_1$ such that $h = h_1 h_2$. Then, it constructs the ciphertext components as follows:

- **Message-embedding components:** First, let $C_1 \leftarrow \mu \cdot Z^s$ and $C_2 \leftarrow g_1^s$.
- **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ for the linear secret sharing scheme and let $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$. For each $k \in [K]$, set $C_{3,k} \leftarrow h_2^{\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s}$, where $\mathbf{m}_k^{\mathsf{T}} \in \mathbb{Z}_p^n$ denotes the k^{th} row of $\mathbf{M}$.
- **Slot-specific component:** Set $C_4 \leftarrow h_1^s \hat{T}^{-s}$.

It then outputs the ciphertext

$$\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}\}_{k \in [K]}, C_4\big).$$

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input the secret key $\mathsf{sk} = r$, the helper key $\mathsf{hsk} = \big(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}}\big)$, where $\mathsf{mpk} = (\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}})$, and the ciphertext $\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}\}_{k \in [K]}, C_4\big)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$ is an injective row-labeling function, the decryption algorithm proceeds as follows:
  - If the set of attributes S_i is not authorized by $(\mathbf{M}, \rho)$, then the decryption algorithm outputs $\perp$.
  - Otherwise, let $I = \{k \in [K] : \rho(k) \in S_i\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes $S_i \subseteq \mathcal{U}$. Write the elements as $I = \{k_1, \ldots, k_{|I|}\}$.
  - Let $\mathbf{M}_{S_i}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I. Since S_i is authorized, let $\boldsymbol{\omega}_{S_i} \in \mathbb{Z}_N^{|I|}$ be a vector such that $\boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} = \mathbf{e}_1^{\mathsf{T}}$.
  - Then, compute and output

$$\underbrace{C_1 \cdot e(C_4, A_i) \cdot e(C_2, A_i^r \hat{V}_i)}_{D_{\text{slot}}} \cdot \underbrace{\prod_{1 \le j \le |I|} \Big(e(C_{3,k_j}, A_i) \cdot e(C_2, \hat{W}_{i,\rho(k_j)})\Big)^{\omega_{S_i,j}}}_{D_{\text{attrib}}} / e(C_2, B_i). \tag{5.2}$$

  We will refer to D_{slot} as the *slot-specific* decryption component and D_{attrib} as the *attribute-specific* decryption component.

Correctness and security analysis. We now provide the correctness (Theorems 5.6 to 5.8) and security analysis (Theorem 5.9) of Construction 5.4. Taken together, we obtain the following corollary:

Corollary 5.5 (Slotted Registered ABE from Pairings). *Let λ be a security parameter, $L = L(\lambda)$ be the number of slots, and $\mathcal{M}$, $\mathcal{U}$, $\mathcal{P}$, be the message space, attribute space, and policy space from Construction 5.4, respectively. Assuming Assumption 5.2 holds with respect to* CompGroupGen*, Construction 5.4 is a secure slotted registered ABE scheme with the following efficiency properties:*

- *The size of the CRS is $L^2 \cdot |\mathcal{U}| \cdot \mathsf{poly}(\lambda)$.*
- *The size of the master public key* mpk *and each helper decryption key* hsk_i *for any slot $i \in [L]$ is $|\mathcal{U}| \cdot \mathsf{poly}(\lambda)$. Notably, this is* independent *of the number of slots (i.e., registered users).*
- *The size of a ciphertext associated with policy $P = (\mathbf{M}, \rho) \in \mathcal{P}$ is $|P| \cdot \mathsf{poly}(\lambda)$.*

Theorem 5.6 (Completeness). *Construction 5.4 is complete.*

Proof. Fix a security parameter λ the number of slots L. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$. Take any index $i \in [L]$ and let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. By construction of KeyGen, we can write $\mathsf{pk}_i = \big(T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i}\big)$, where

$$T_i = g_1^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad R_i = g_3^{r_i} \quad , \quad V_{j,i} = A_j^{r_i}$$

for some $r_i \in \mathbb{Z}_N$ and where A_i and P_i are elements from crs. We now consider each of the pairing checks in IsValid and appeal to orthogonality:

- $e(g_3, T_i) = e(g_3, g_1^{r_i}) = e(g_3, g_1)^{r_i} = 1$.

- $e(g_1, R_i) = e(g_1, g_3^{r_i}) = e(g_1, g_3)^{r_i} = 1$.
- $e(T_i, P_i) = e(g_1^{r_i}, P_i) = e(g_1, P_i^{r_i}) = e(g_1, Q_i)$.
- $e(R_i, P_i) = e(g_3^{r_i}, P_i) = e(g_3, P_i^{r_i}) = e(g_3, Q_i)$.
- $e(g_1, V_{j,i}) = e(g_1, A_j^{r_i}) = e(g_1^{r_i}, A_j) = e(T_i, A_j)$.
- $e(g_3, V_{j,i}) = e(g_3, A_j^{r_i}) = e(g_3^{r_i}, A_j) = e(R_i, A_j)$.

Since all of the pairing checks pass, $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$ outputs 1 and completeness holds. □

Theorem 5.7 (Correctness). *Construction 5.4 is correct.*

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

- Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$ where $\mathsf{crs} = \big(\mathcal{G}, Z, g_1, h, g_3, \{(A_i, B_i, P_i)\}_{i\in[L]}, \{U_{i,w}, W_{i,j,w}\}_{i\neq j, w\in\mathcal{U}}\big)$. By construction, the slot components can be written as $A_i = (g_1 g_3)^{t_i}$, $B_i = g_1^\alpha A_i^\beta g_3^{\tau_i}$, and $P_i = (g_1 g_3)^{\delta_i}$. The attribute components can be written as $U_{i,w} = g_1^{u_{i,w}}$ and $W_{i,j,w} = A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}}$.
- Let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. Then, we can write $\mathsf{sk}_i = r_i$ and $\mathsf{pk}_i = \big(T_i, Q_i, R_i, \{V_{j,i}\}_{j\neq i}\big)$ where
$$T_i = g_1^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad R_i = g_3^{r_i} \quad , \quad V_{j,i} = A_j^{r_i} = (g_1 g_3)^{t_j r_i}. \tag{5.3}$$
- Take any set of public keys $\{\mathsf{pk}_j\}_{j\neq i}$ where $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j)$ holds. Since pk_j satisfies the IsValid predicate, we can write $\mathsf{pk}_j = \big(T_j, Q_j, R_j, \{V_{\ell,j}\}_{\ell\in[L]\setminus\{j\}}\big)$.
- For each $j \in [L]$, let $S_j \subseteq \mathcal{U}$ be the attributes associated with pk_j.
- The master public key mpk and i^{th} slot-specific helper decryption key hsk_i can then be written as follows:
$$\mathsf{mpk} = \big(\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w\in\mathcal{U}}\big) \quad \text{and} \quad \mathsf{hsk}_i = \big(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w\in\mathcal{U}}\big),$$
where $\hat{T} = \prod_{j\in[L]} T_j$, $\hat{V}_i = \prod_{j\neq i} V_{i,j}$, and
$$\hat{U}_w = \prod_{j\in[L]: w\notin S_j} U_{j,w} = \prod_{j\in[L]: w\notin S_j} g_1^{u_{j,w}}$$
$$\hat{W}_{i,w} = \prod_{j\neq i: w\notin S_j} W_{i,j,w} = \prod_{j\neq i: w\notin S_j} (g_1 g_3)^{t_i u_{j,w}} g_3^{\gamma_{i,j,w}}$$
- Let $(\mathbf{M}, \rho)$ be the challenge policy where $\mathbf{M} \in \mathbb{Z}_p^{K\times n}$ and $\rho\colon [K] \to \mathcal{U}$ is an injective row-labeling function. Take any message $\mu \in \mathcal{M}$. The challenge ciphertext ct can be written as $\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}\}_{k\in[K]}, C_4\big)$ where $\mathcal{P} = (\mathbf{M}, \rho)$ is the challenge policy, $C_1 = \mu \cdot Z^s$, $C_2 = g_1^s$, $C_{3,k} = h_2^{\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s}$, $C_4 = h_1^s \hat{T}^{-s}$, $h_1 h_2 = h$, and $\mu \in \mathbb{G}_T$ is the challenge message.

We now show that $\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{hsk}_i, \mathsf{ct})$ outputs μ. Let $I = \{k \in [K] : \rho(k) \in S_i\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes S_i. Write the elements of I as $I = \{k_1, \ldots, k_{|I|}\}$. Let $\mathbf{M}_{S_i}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I, and let $\boldsymbol{\omega}_{S_i} \in \mathbb{Z}_N^{|I|}$ be a vector such that $\boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} = \mathbf{e}_1^{\mathsf{T}}$. We break up the decryption relation (Eq. (5.2)) into several pieces and analyze them individually:

- **Policy check:** First, consider $D_{\text{attrib}} = \prod_{1 \le j \le |I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_2, \hat{W}_{i,\rho(k_j)}) \right)^{\omega_{S_i,j}}$. By definition,

$$e(C_{3,k_j}, A_i) = e\left(h_2^{\mathbf{m}_{k_j}^{\mathsf{T}} \mathbf{v}} \hat{U}_{\rho(k_j)}^{-s}, (g_1 g_3)^{t_i} \right) = e(h_2, g_1)^{t_i \mathbf{m}_{k_j}^{\mathsf{T}} \mathbf{v}} \prod_{\ell \in [L] : \rho(k_j) \notin S_\ell} e(g_1, g_1)^{-t_i s u_{\ell,\rho(k_j)}}$$

$$e(C_2, \hat{W}_{i,\rho(k_j)}) = \prod_{\ell \in [L] \setminus \{i\} : \rho(k_j) \notin S_\ell} e\left(g_1^s, W_{i,\ell,\rho(k_j)} \right) = \prod_{\ell \in [L] \setminus \{i\} : \rho(k_j) \notin S_\ell} e(g_1, g_1)^{t_i s u_{\ell,\rho(k_j)}}$$

since $h_2, \hat{U}_{\rho(k_j)} \in \mathbb{G}_1$. By construction, $\rho(k_j) \in S_i$, so

$$\prod_{\ell \in [L] : \rho(k_j) \notin S_\ell} e(g_1, g_1)^{-t_i s u_{\ell,\rho(k_j)}} = \prod_{\ell \in [L] \setminus \{i\} : \rho(k_j) \notin S_\ell} e(g_1, g_1)^{-t_i s u_{\ell,\rho(k_j)}},$$

and so we can write

$$e(C_{3,k_j}, A_i) e(C_2, \hat{W}_{i,\rho(k_j)}) = e(h_2, g_1)^{t_i \mathbf{m}_{k_j}^{\mathsf{T}} \mathbf{v}}.$$

Finally noting that $\mathbf{e}_1^{\mathsf{T}} \mathbf{v} = s$, we have

$$\begin{aligned} D_{\text{attrib}} = \prod_{1 \le j \le |I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_2, \hat{W}_{i,\rho(k_j)}) \right)^{\omega_{S_i,j}} &= e(h_2, g_1)^{t_i \sum_{1 \le j \le |I|} \omega_{S_i,j} \mathbf{m}_{k_j}^{\mathsf{T}} \mathbf{v}} \\ &= e(h_2, g_1)^{t_i \boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} \mathbf{v}} \\ &= e(h_2, g_1)^{t_i \mathbf{e}_1^{\mathsf{T}} \mathbf{v}} = e(h_2, g_1)^{s t_i}. \end{aligned}$$

- **Slot check:** Next, consider the component $D_{\text{slot}} = e(C_4, A_i) e(C_2, A_i^{r_i} \hat{V}_i)$. By definition,

$$e(C_4, A_i) = e(h_1^s \hat{T}^{-s}, (g_1 g_3)^{t_i}) = e(h_1, g_1)^{s t_i} \prod_{j \in [L]} e(T_j, g_1 g_3)^{-s t_i} = e(h_1, g_1)^{s t_i} \prod_{j \in [L]} e(T_j, A_i)^{-s}$$

$$e(C_2, A_i^{r_i} \hat{V}_i) = e(g_1^s, (g_1 g_3)^{r_i t_i} \hat{V}_i) = e(g_1, g_1)^{s r_i t_i} \prod_{j \ne i} e(g_1, V_{i,j})^s.$$

since $h_1 \in \mathbb{G}_1$. Now, since we know for all $j \in [L]$, $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$, we have that for all $j \ne i$, $e(g_1, V_{i,j}) = e(T_j, A_i)$. Thus, using Eq. (5.3), we can now write

$$\begin{aligned} D_{\text{slot}} = e(C_4, A_i) e(C_2, A_i^{r_i} \hat{V}_i) &= \left(e(h_1, g_1)^{s t_i} e(T_i, A_i)^{-s} \prod_{j \ne i} e(T_j, A_i)^{-s} \right) \left(e(g_1, g_1)^{s r_i t_i} \prod_{j \ne i} e(g_1, V_{i,j})^s \right) \\ &= e(h_1, g_1)^{s t_i} e(T_i, A_i)^{-s} e(g_1, g_1)^{s r_i t_i} \\ &= e(h_1, g_1)^{s t_i} e(g_1^{r_i}, (g_1 g_3)^{t_i})^{-s} e(g_1, g_1)^{s r_i t_i} = e(h_1, g_1)^{s t_i} \end{aligned}$$

- **Message reconstruction:** Using the fact that $h = h_1 h_2$, and combining the above relations, we have that

$$D_{\text{slot}} \cdot D_{\text{attrib}} = e(h_1, g)^{s t_i} e(h_2, g_1)^{s t_i} = e(h, s)^{s t_i}.$$

Next, using the fact that $h = g_1^\beta$, we have

$$e(C_2, B_i) = e(g_1^s, g_1^\alpha A_i^\beta g_3^{\tau_i}) = e(g_1, g_1)^{\alpha s} e(g_1^s, (g_1 g_3)^{\beta t_i}) = e(g_1, g_1)^{\alpha s} e(h, g_1)^{s t_i}.$$

Thus, putting everything together, Eq. (5.2) becomes

$$\frac{C_1 \cdot D_{\text{slot}} \cdot D_{\text{attrib}}}{e(C_2, B_i)} = \frac{\mu \cdot e(g_1, g_1)^{\alpha s} e(h, g_1)^{s t_i}}{e(g_1, g_1)^{\alpha s} e(h, g_1)^{s t_i}} = \mu. \qquad \square$$

Theorem 5.8 (Compactness)**.** *Construction 5.4 is compact.*

Proof. This follows by inspection. The master public key mpk consists of the group description and $O(|\mathcal{U}|)$ group elements. Since the group description and each individual group element can be represented in $\mathsf{poly}(\lambda)$ bits, the size of the master public key is bounded by $\mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$ bits. Likewise, the helper decryption key consists of the master public key along with $O(|\mathcal{U}|)$ group elements. Thus, the size of hsk_i is also $\mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$ bits. □

Theorem 5.9 (Security)**.** *Suppose Assumption 5.2 holds with respect to* CompGroupGen*. Then, Construction 5.4 is secure.*

Proof. Our proof follows the dual-system methodology [Wat09, LW10], where we introduce modified ciphertexts (referred to as "semi-functional ciphertexts") and slot components (referred to as "semi-functional slots"). Keys registered to a semi-functional slot can be used to decrypt normal ciphertexts (i.e., those output by the honest encryption algorithm) and keys registered to a normal slot can be used to decrypt semi-functional ciphertexts. However, a key registered to a semi-functional slot is unable to decrypt a semi-functional ciphertext. The proof then proceeds via a sequence of games where we first switch the challenge ciphertext from a normal ciphertext to a semi-functional one. Then, we switch the parameters associated with each slot from normal to semi-functional. In the final experiment, all of the slots are semi-functional, as is the challenge ciphertext. Since keys associated with semi-functional slots cannot be used to decrypt a semi-functional ciphertext, arguing semantic security in the final experiment is straightforward. We now specify the structure of our semi-functional slots and ciphertexts.

- **Semi-functional ciphertext:** Semi-functional ciphertexts contain additional components in the $\mathbb{G}_2$ subgroup. Specifically, suppose $\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}\}_{k \in [K]}, C_4\big) \leftarrow \mathsf{Encrypt}(\mathsf{crs}, \mathsf{mpk}, \mathsf{id}, \mu)$. Then, a semi-functional ciphertext has the following structure:

$$\mathsf{ct}' = \Big((\mathbf{M}, \rho)\,,\, C_1\,,\, C_2 g_2^{\zeta_2}\,,\, \big\{C_{3,k} g_2^{\zeta_{3,k}}\big\}_{k \in [K]}\,,\, \big(C_4 g_2^{\zeta_4}\,,\,\big)\Big),$$

  for some choice of exponents $\zeta_2, \zeta_{3,k}, \zeta_4 \in \mathbb{Z}_N$. We note that while the proof does not construct ct' by sampling the exponents $\zeta_2, \zeta_{3,k}, \zeta_4$ directly, the components in the semi-functional ciphertexts can be written in this form. We refer to the specific hybrid argument for details on how the individual components are sampled.

- **Semi-functional slot:** The slot components for a semi-functional slot at index $i \in [L]$ are generated exactly as the normal slot components except we change the distribution of B_i and P_i:

$$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^{\alpha} A_i^{\beta} (g_2 g_3)^{\tau_i} \quad , \quad P_i \leftarrow g^{\delta_i}.$$

We now define our sequence of hybrid experiments. By Lemma 4.10, we can assume without loss of generality that the adversary submits all of its queries *before* submitting the challenge ciphertext. Let $b \in \{0, 1\}$ be a bit.

- $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$: 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 a slot count 1^L the challenger, who samples the common reference string crs according to the specification of the real setup algorithm:
    * The challenger initializes a counter $\mathsf{ctr} \leftarrow 0$ and an (empty) dictionary D to keep track of the key-generation queries.
    * Let $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $N = p_1 p_2 p_3$ and $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ be the group description.
    * Sample generators $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_3 \xleftarrow{R} \mathbb{G}_3$, exponents $\alpha, \beta \xleftarrow{R} \mathbb{Z}_N$, and let $h \leftarrow g_1^{\beta}$.
    * For each slot $i \in [L]$, sample $t_i, \delta_i, \tau_i \xleftarrow{R} \mathbb{Z}_N$. Construct the slot components as follows:

$$A_i = (g_1 g_3)^{t_i} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} g_3^{\tau_i} \quad , \quad P_i = (g_1 g_3)^{\delta_i}.$$

    Then, for each attribute $w \in \mathcal{U}$ and each slot $i \in [L]$, sample an exponent $u_{i,w} \xleftarrow{R} \mathbb{Z}_N$. For each $j \in [L]$ with $j \neq i$, sample a blinding factor $\gamma_{i,j,w} \xleftarrow{R} \mathbb{Z}_N$ and construct the Attribute-specific component $U_{i,w}$ and $W_{i,j,w}$ as follows:

$$U_{i,w} = g_1^{u_{i,w}} \quad , \quad W_{i,j,w} = A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}}.$$

    * Finally, compute $Z \leftarrow e(g_1, g_1)^\alpha$ and output the common reference string

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{i,j,w}\}_{i \neq j, w \in \mathcal{U}}\right) \tag{5.4}$$

  - **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 $r_i \xleftarrow{\text{R}} \mathbb{Z}_N$. It then computes $T_i \leftarrow g_1^{r_i}$, $Q_i \leftarrow P_i^{r_i}$, $R_i \leftarrow g_3^{r_i}$, and $V_{j,i} \leftarrow A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ 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.
    * **Corruption query:** If the adversary makes a corruption query on an index $1 \leq i \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[i]$ and replies to $\mathcal{A}$ with sk'.
  - **Challenge phase:** In the challenge phase, the adversary specifies a challenge policy $P^* = (\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$ is an injective row-labeling function, two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. The challenger sets up the public keys pk_i as follows:
    * If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, the challenger sets $\mathsf{pk}_i \leftarrow \mathsf{pk}'$. Otherwise, the challenger aborts with output 0.
    * If $c_i = \perp$, then the challenger checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, the challenger aborts with output 0. Otherwise, it sets $\mathsf{pk}_i \leftarrow \mathsf{pk}_i^*$.

    For each public key pk_i, the challenger parses it as $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$. Next, the challenger computes the attribute-independent public key $\hat{T}$ and the attribute-independent slot key $\hat{V}_i$ for each $i \in [L]$:

$$\hat{T} = \prod_{j \in [L]} T_j \quad , \quad \hat{V}_i = \prod_{j \neq i} V_{i,j}.$$

    Then, for each attribute $w \in \mathcal{U}$, it computes the attribute-specific public key $\hat{U}_w$ and the attribute-specific slot key $\hat{W}_{i,w}$ for each $i \in [L]$ as follows:

$$\hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_{j,w} \quad , \quad \hat{W}_{i,w} = \prod_{j \neq i : w \notin S_j} W_{i,j,w}.$$

    The challenger then constructs the challenge ciphertext by sampling a secret exponent $s \xleftarrow{\text{R}} \mathbb{Z}_N$ and $h_1, h_2 \xleftarrow{\text{R}} \mathbb{G}_1$ such that $h = h_1 h_2$. It then constructs the challenge ciphertext components as follows:
    * **Message-embedding components:** First, let $C_1 \leftarrow \mu_b^* \cdot Z^s$ and $C_2 \leftarrow g_1^s$.
    * **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{\text{R}} \mathbb{Z}_N$ for the linear secret sharing scheme and let $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, for each $k \in [K]$, set $C_{3,k} \leftarrow h_2^{\mathbf{m}_k^{\mathsf{T}} \mathbf{v}} \hat{U}_{\rho(k)}^{-s}$, where $\mathbf{m}_k^{\mathsf{T}}$ denotes the k^{th} row of $\mathbf{M}$.
    * **Slot-specific component:** Set $C_4 \leftarrow h_1^s \hat{T}^{-s}$.

    It replies to $\mathcal{A}$ with the challenge ciphertext

$$\mathsf{ct}^* = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}\}_{k \in [K]}, C_4\right).$$

  - **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 for the following (primarily syntactic) changes:
  - **Setup phase:** The challenger samples $\beta_1, \beta_2 \xleftarrow{\text{R}} \mathbb{Z}_N$ and sets $\beta \leftarrow \beta_1 + \beta_2$. It sets $h \leftarrow g_1^\beta$ as in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$. In addition, instead of sampling the secret exponent s during the challenge phase, the challenger samples $s \xleftarrow{\text{R}} \mathbb{Z}_N$ during the setup phase and then sets $P_i \leftarrow (g_1^s g_3)^{\delta_i}$ for all $i \in [L]$.

  - **Challenge phase:** When simulating the challenge ciphertext, the challenger sets $h_1 \leftarrow g_1^{\beta_1}$ and $h_2 \leftarrow g_1^{\beta_2}$, where $\beta_1, \beta_2 \in \mathbb{Z}_N$ are the exponents sampled during the setup phase. Then it constructs the challenge ciphertext components as follows:
    - **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, for each $k \in [K]$, set
$$C_{3,k} \leftarrow (g_1^s)^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]: \rho(k) \notin S_i} u_{i,\rho(k)}}.$$
    - **Slot-specific component:** Set
$$C_4 \leftarrow (g_1^s)^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right).$$

- $\mathsf{Hyb}_{2,0}^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$, except the challenge ciphertext is replaced by a semi-functional ciphertext, and simultaneously, we lift the P_i component to be in the full group (rather than $\mathbb{G}_1 \times \mathbb{G}_3$). Namely, during the setup phase, the challenger constructs P_i as follows for each $i \in [L]$:
$$P_i \leftarrow ((g_1 g_2)^s g_3)^{\delta_i}.$$
Then, in the challenge phase, after the adversary has chosen its attribute sets S_i and corresponding public keys $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ for each slot $i \in [L]$, the challenger constructs the challenge ciphertext components as follows:
  - **Message-embedding components:** Let $C_1 \leftarrow \mu_b^* \cdot Z^s$ and $C_2 \leftarrow (g_1 g_2)^s$.
  - **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, for each $k \in [K]$, set
$$C_{3,k} \leftarrow ((g_1 g_2)^s)^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L]: \rho(k) \notin S_i} u_{i,\rho(k)}}.$$
  - **Slot-specific component:** Set
$$C_4 \leftarrow ((g_1 g_2)^s)^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

- $\mathsf{Hyb}_{2,\ell}^{(b)}$: Same as $\mathsf{Hyb}_{2,\ell-1}^{(b)}$, except we change slot ℓ to a semi-functional slot. Specifically, during the setup phase, the challenger samples the slot components A_ℓ, B_ℓ, and P_ℓ as follows:
$$A_\ell \leftarrow (g_1 g_3)^{t_\ell} \quad , \quad B_\ell \leftarrow g_1^\alpha A_\ell^\beta (g_2 g_3)^{\tau_\ell} \quad , \quad P_\ell \leftarrow ((g_1 g_2)^s g_3)^{\delta_\ell}.$$

- $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$: Same as $\mathsf{Hyb}_{2,L}$ except when constructing the challenge ciphertext, the challenger samples $C_1 \xleftarrow{R} \mathbb{G}_T$. Importantly, this distribution is *independent* of the message.

For an adversary $\mathcal{A}$ and a hybrid experiment Hyb, we write $\mathsf{Hyb}(\mathcal{A})$ to denote the output of an execution of Hyb with adversary $\mathcal{A}$. We now show that the output distributions of each adjacent pair of hybrid experiments are computationally indistinguishable.

Lemma 5.10. *Suppose Assumption 5.2a 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 $\lambda \in \mathbb{N}$,*
$$\left| \Pr[\mathsf{Hyb}_{\mathsf{real}}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. As we show below, the two experiments are *statistically indistinguishable* if all of the public keys pk_i^* the adversary specifies in the challenge phase either satisfy $\mathsf{pk}_i^* = \bot$ or pk_i^* is in the support of the honest key-generation algorithm (i.e., for every $i \in [L]$, there exists r_i such that pk_i^* is the public key output by $\mathsf{KeyGen}(\mathsf{crs}, i)$). We now show that under Assumption 5.2a, the only public keys pk_i^* that an *efficient* adversary can construct and which satisfy the validity check $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$ are those that are in the support of the honest key-generation algorithm. To do so, we start by characterizing the set of possible strategies available to an efficient adversary.

Claim 5.11. *For a security parameter λ, define the following game between an adversary $\mathcal{A}$ and a challenger:*

1. *The challenger starts by sampling $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It sets $N = p_1 p_2 p_3$, $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ and samples $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_3 \xleftarrow{R} \mathbb{G}_3$, $s \xleftarrow{R} \mathbb{Z}_N$, and $Z \leftarrow (g_1 g_3)^s$. The challenger gives the tuple $(\mathcal{G}, g_1, g_3, Z)$ to $\mathcal{A}$.*
2. *Algorithm $\mathcal{A}$ outputs a tuple $(A, B, C) \in \mathbb{G}^3$.*
3. *The game outputs $b = 1$ if the following relations are satisfied:*
$$e(g_3, A) = 1 = e(g_1, B) \quad \text{and} \quad e(A, Z) = e(g_1, C) \quad \text{and} \quad e(B, Z) = e(g_3, C),$$
and moreover, there does not exist $r \in \mathbb{Z}_N$ such that $A = g_1^r$, $B = g_3^r$, and $C = Z^r$.

Suppose Assumption 5.2a holds with respect to CompGroupGen. *Then, for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the above security game.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ such that $\Pr[b = 1] = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2a:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_3, T)$, where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_3 \in \mathbb{G}_3$, and either $T = g_1^r$ or $T = (g_1 g_2)^r$
2. Algorithm $\mathcal{B}$ samples exponents $\gamma_A, \gamma_B \xleftarrow{R} \mathbb{Z}_N$ and computes $Z = g_1^{\gamma_A} g_3^{\gamma_B}$.
3. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on input $(\mathcal{G}, g_1, g_3, Z)$ to obtain a triple (A, B, C).
4. Algorithm $\mathcal{B}$ computes $Z' \leftarrow C/(A^{\gamma_A} B^{\gamma_B})$ and outputs 1 if $e(Z', T) = 1$ and 0 otherwise.

In the real security game (Claim 5.11), the element $Z = (g_1 g_3)^s = g_1^{s \bmod p_1} g_3^{s \bmod p_3}$. Since $s \xleftarrow{R} \mathbb{Z}_N$, by the Chinese Remainder Theorem, the individual exponents $s \bmod p_1$ and $s \bmod p_3$ are independent and uniform over $\mathbb{Z}_{p_1}$ and $\mathbb{Z}_{p_3}$, respectively. Thus, algorithm $\mathcal{B}$ perfectly simulates the security game for $\mathcal{A}$. Thus, with probability at least ε, algorithm $\mathcal{A}$ outputs a tuple (A, B, C) such that $e(g_1, B) = 1 = e(g_3, A)$, $e(A, Z) = e(g_1, C)$, and $e(B, Z) = e(g_3, C)$. Moreover, there does not exist $r \in \mathbb{Z}_N$ such that $A = g_1^r$, $B = g_3^r$, and $C = Z^r$. We now argue that in this case, over the choice of $\gamma_A, \gamma_B \xleftarrow{R} \mathbb{Z}_N$, it will be the case that $Z' \in \mathbb{G}_2 \setminus \{1\}$ with overwhelming probability.

- First, we show that Z' does not have any non-trivial component in the $\mathbb{G}_1$ and $\mathbb{G}_3$ subgroups (i.e., $Z' \in \mathbb{G}_2$). This is equivalent to checking that $e(g_1 g_3, Z') = 1$. First, using the fact that $e(g_1, C) = e(A, Z)$, $e(g_3, C) = e(B, Z)$, and $Z = g_1^{\gamma_A} g_3^{\gamma_B}$, we can write
$$e(g_1 g_3, Z') = \frac{e(g_1 g_3, C)}{e(g_1 g_3, A^{\gamma_A}) e(g_1 g_3, B^{\gamma_B})} = \frac{e(A, Z) e(B, Z)}{e(g_1 g_3, A^{\gamma_A}) e(g_1 g_3, B^{\gamma_B})} = \frac{e(A, g_1^{\gamma_A} g_3^{\gamma_B}) e(B, g_1^{\gamma_A} g_3^{\gamma_B})}{e(g_1 g_3, A^{\gamma_A}) e(g_1 g_3, B^{\gamma_B})}$$
Next, since $e(g_1, B) = 1 = e(g_3, A)$, we have
$$e(g_1 g_3, Z') = \frac{e(A, g_1^{\gamma_A} g_3^{\gamma_B}) e(B, g_1^{\gamma_A} g_3^{\gamma_B})}{e(g_1 g_3, A^{\gamma_A}) e(g_1 g_3, B^{\gamma_B})} = \frac{e(A, g_1)^{\gamma_A} e(B, g_3)^{\gamma_B}}{e(g_1, A)^{\gamma_A} e(g_3, B)^{\gamma_B}} = 1.$$
- Next, at least one of the group elements A, B, C must contain a non-trivial component in the $\mathbb{G}_2$ subgroup. Suppose otherwise: namely that $A = (g_1 g_3)^{r_A}$, $B = (g_1 g_3)^{r_B}$, and $C = (g_1 g_3)^{r_C}$ for some $r_A, r_B, r_C \in \mathbb{Z}_N$. Then, the conditions imply the following:
  - Since $e(g_3, A) = e(g_3, g_3)^{r_A \bmod p_3} = 1$, it must be the case that $r_A \bmod p_3 = 0$. Thus, $A = g_1^{r_A \bmod p_1}$.
  - Since $e(g_1, B) = e(g_1, g_1)^{r_B \bmod p_1} = 1$, it must be the case that $r_B \bmod p_1 = 0$. Thus, $B = g_3^{r_B \bmod p_3}$.

- Finally, $e(g_1, C) = e(A, Z)$ means that $e(g_1, g_1)^{r_C \bmod p_1} = e(A, Z) = e(g_1, g_1)^{\gamma_A r_A \bmod p_1}$. Analogously, $e(g_3, C) = e(B, Z)$ means that $e(g_3, g_3)^{r_C \bmod p_3} = e(B, Z) = e(g_3, g_3)^{\gamma_B r_B \bmod p_3}$. Putting these together, this means that $r_C = \gamma_A r_A \bmod p_1$ and $r_C = \gamma_B r_B \bmod p_3$. Take any $r \in \mathbb{Z}_N$ such that $r = r_A \bmod p_1$ and $r = r_B \bmod p_3$. Then, we can write

$$C = (g_1 g_3)^{r_C} = g_1^{r_C \bmod p_1} g_3^{r_C \bmod p_3} = g_1^{\gamma_A r_A \bmod p_1} g_3^{\gamma_B r_B \bmod p_3} = (g_1^{\gamma_A} g_3^{\gamma_B})^r = Z^r.$$

This contradicts the assumption that there does not exist $r \in \mathbb{Z}_N$ such that $A = g_1^r$, $B = g_3^r$, and $C = Z^r$.

- Thus, at least one of A, B, C must contain a non-trivial component in the $\mathbb{G}_2$ subgroup. We consider two cases:
  - Suppose that at least one of A or B has a non-trivial component in the $\mathbb{G}_2$ subgroup. By the Chinese Remainder Theorem, γ_A and γ_B are uniform over $\mathbb{Z}_N$, so $\gamma_A \bmod p_2$ and $\gamma_B \bmod p_2$ are uniform over $\mathbb{Z}_{p_2}$ and more importantly, *independent* of the view of the adversary. Thus, $\gamma_A \bmod p_2$ and $\gamma_B \bmod p_2$ are uniform over $\mathbb{Z}_{p_2}$ even given A, B, C. Since at least one of A or B contains a component in $\mathbb{G}_2$, this means that $Z' = C/(A^{\gamma_A} B^{\gamma_B})$ is uniform over $\mathbb{G}_2$. Correspondingly, $Z' \neq 1$ with probability $1 - 1/p_2 = 1 - \mathsf{negl}(\lambda)$.
  - Suppose that A and B do not have a component in the $\mathbb{G}_2$ subgroup. Then, C must have a non-trivial component in the $\mathbb{G}_2$ subgroup, and correspondingly, $Z' = C/(A^{\gamma_A} B^{\gamma_B})$ must also have a non-trivial component in the $\mathbb{G}_2$ subgroup.

Putting the pieces together, if algorithm $\mathcal{A}$ succeeds, then with overwhelming probability, $Z' \in \mathbb{G}_2 \setminus \{1\}$. In this case, if $T = g_1^r$, then $e(Z', T) = 1$ and if $T = (g_1 g_2)^r$, then $e(Z', T) \neq 1$ (unless $r = 0$). Correspondingly, algorithm $\mathcal{B}$ breaks Assumption 5.2a with probability $\varepsilon - \mathsf{negl}(\lambda)$. □

Using Claim 5.11, we now show that the only public keys pk_i^* the efficient adversary can construct that pass the validity check are those in the support of the honest key-generation algorithm.

Claim 5.12. *For each index $i \in [L]$, let pk_i^* be the public key algorithm $\mathcal{A}$ outputs for slot i in the challenge phase in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$. Suppose Assumption 5.2a holds with respect to* CompGroupGen*. Then, for all indices $i \in [L]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0, 1\}$, if* $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$*, then with probability* $1 - \mathsf{negl}(\lambda)$*, there exists $r_i \in \mathbb{Z}_N$ such that pk_i^* is the public key output of* $\mathsf{KeyGen}(\mathsf{crs}, i; r_i)$*.*

Proof. Take any index $i \in [L]$. Let pk_i^* be the public key algorithm $\mathcal{A}$ chooses for index i in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$. Parse $\mathsf{pk}_i^* = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$. Suppose $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*) = 1$.

- We first show that there exists $r_i \in \mathbb{Z}_N$ such that $T_i = g_1^{r_i}$, $R_i = g_3^{r_i}$, and $Q_i = P_i^{r_i}$ where $P_i = (g_1 g_3)^{\delta_i}$ is the component in the CRS. Suppose otherwise. Then, we use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ that wins the game in Claim 5.11:
  1. At the beginning of the game, algorithm $\mathcal{B}$ receives a tuple $(\mathcal{G}, g_1, g_3, Z)$ from the challenger, where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ and $Z = (g_1 g_3)^\delta$ for some $\delta \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.
  2. Algorithm $\mathcal{B}$ guesses an index $i^* \xleftarrow{\mathrm{R}} [L]$ and uses $(\mathcal{G}, g_1, g_3)$ to construct the CRS components according to $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$. It uses element Z in place of element P_{i^*}. All of the other components are sampled according to the procedure in Setup.
  3. Algorithm $\mathcal{B}$ gives crs to $\mathcal{A}$.
  4. After $\mathcal{A}$ outputs (i, pk_i), algorithm $\mathcal{B}$ aborts if $i \neq i^*$. Otherwise, it parses $\mathsf{pk}_{i^*} = (T_{i^*}, Q_{i^*}, R_{i^*}, \{V_{j,i^*}\}_{j \neq i^*})$ and outputs $(T_{i^*}, R_{i^*}, Q_{i^*})$.

  By construction, algorithm $\mathcal{B}$ perfectly simulates the distribution of the common reference string, and moreover, the index i^* is perfectly hidden from $\mathcal{A}$. Thus, with probability $1/L$, $i = i^*$. In this case, if $\mathsf{IsValid}(\mathsf{crs}, i^*, \mathsf{pk}_{i^*}^*)$ holds, then

$$e(g_3, T_{i^*}) = 1 = e(g_1, R_{i^*}) \quad \text{and} \quad e(T_{i^*}, P_{i^*}) = e(g_1, Q_{i^*}) \quad \text{and} \quad e(R_{i^*}, P_{i^*}) = e(g_3, Q_{i^*}).$$

Suppose now that there does not exist $r_{i^*} \in \mathbb{Z}_N$ where $T_{i^*} = g_1^{r_{i^*}}$, $R_{i^*} = g_3^{r_{i^*}}$, and $Q_{i^*} = P_{i^*}^{r_{i^*}}$. This means that algorithm $\mathcal{B}$ wins the game in Claim 5.11. Correspondingly, if algorithm $\mathcal{A}$ outputs a malformed key with probability ε, then algorithm $\mathcal{B}$ succeeds with probability ε/L, which proves the claim.

- Next, we show that for all $j \neq i$, there exists $r_{j,i} \in \mathbb{Z}_N$ such that $T_i = g_1^{r_{j,i}}$, $R_i = g_3^{r_{j,i}}$, $V_{j,i} = A_j^{r_{j,i}}$, and $A_j = (g_1, g_3)^{t_j}$ is the component in the CRS. This follows by a similar argument as in the previous case and appealing again to Claim 5.11.

Thus, we have shown that for all tuples (i, pk_i^*) satisfying $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*) = 1$ output by an efficient adversary $\mathcal{A}$, it must be the case that there exists $r_i, r_{j,i} \in \mathbb{Z}_N$ for all $j \neq i$ such that

$$T_i = g_1^{r_i} = g_1^{r_{j,i}} \quad \text{and} \quad R_i = g_3^{r_i} = g_3^{r_{j,i}} \quad \text{and} \quad Q_i = P_i^{r_i} \quad \text{and} \quad V_{j,i} = A_j^{r_{j,i}}.$$

The requirement on T_i ensures that $r_i = r_{j,i} \bmod p_1$ for all $j \neq i$. Similarly, the requirement on R_i ensures that $r_i = r_{j,i} \bmod p_3$. By construction, each of the A_j's are contained in $\mathbb{G}_1 \times \mathbb{G}_3$. Then,

$$T_i = g_1^{r_i} \quad \text{and} \quad R_i = g_3^{r_i} \quad \text{and} \quad Q_i = P_i^{r_i} \quad \text{and} \quad V_{j,i} = A_j^{r_{j,i}} = A_j^{r_{j,i} \bmod p_1 p_3} = A_j^{r_i \bmod p_1 p_3} = A_j^{r},$$

for all $j \neq i$, and the claim follows. □

Returning now to the proof of Lemma 5.10, we can first appeal to Claim 5.12 to conclude that for all efficient adversaries $\mathcal{A}$, in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$, the public keys $\mathsf{pk}_1^*, \ldots, \mathsf{pk}_L^*$ chosen by $\mathcal{A}$ in the challenge phase are either $\perp$, do not satisfy the $\mathsf{IsValid}$ predicate, or are in the support of the honest key-generation algorithm. Thus, if the challenger does not abort, then it must be the case that for all $i \in [L]$, there exists $r_i \in \mathbb{Z}_N$ such that pk_i is the public key output of $\mathsf{KeyGen}(\mathsf{crs}, i; r_i)$. In particular, all of the keys pk_i sampled by the challenger in an (honest) key-generation query already satisfy this property. Thus, for each $i \in [L]$, we can write

$$T_i = g_1^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad R_i = g_3^{r_i} \quad , \quad V_{j,i} = A_j^{r_i}. \tag{5.5}$$

Then, in both $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$ and $\mathsf{Hyb}_1^{(b)}$, the following relations hold:

$$\hat{T} = \prod_{i \in [L]} T_i = \prod_{i \in [L]} g_1^{r_i} \quad \text{and} \quad \hat{U}_{\rho(k)} = \prod_{i \in [L] : \rho(k) \notin S_i} U_{i,\rho(k)} = g_1^{\sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}}. \tag{5.6}$$

We now consider the components in the two experiments:

- In both experiments, h, h_1, h_2 is uniform over $\mathbb{G}_1$ subject to the constraint $h = h_1 h_2$. Moreover, since $\beta_1, \beta_2 \xleftarrow{R} \mathbb{Z}_N$, $\beta = \beta_1 + \beta_2$ is also uniform over $\mathbb{Z}_N$ in $\mathsf{Hyb}_1^{(b)}$, so the distribution of β matches that in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$.

- Consider the distribution of P_i in the two experiments. In $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$,

$$P_i = (g_1 g_3)^{\delta_i} = g_1^{\delta_i \bmod p_1} g_3^{\delta_i \bmod p_3}.$$

  Since δ_i is uniform over $\mathbb{Z}_N$ (and independent of all other quantities), $\delta_i \bmod p_1$ and $\delta_i \bmod p_3$ are independently uniform over $\mathbb{Z}_{p_1}$ and $\mathbb{Z}_{p_3}$, respectively, by the Chinese remainder theorem. In $\mathsf{Hyb}_1^{(b)}$,

$$P_i = (g_1^s g_3)^{\delta_i} = g_1^{s\delta_i \bmod p_1} g_3^{\delta_i \bmod p_3}.$$

  Since δ_i is still uniform over $\mathbb{Z}_N$ (and independent of all other quantities), the distribution of $s\delta_i \bmod p_1$ is uniform over $\mathbb{Z}_{p_1}$ as long as $s \neq 0 \bmod p_1$ (which holds with overwhelming probability since $s \xleftarrow{R} \mathbb{Z}_N$). As such, the distribution of P_i in these two experiments is statistically indistinguishable.

- Consider the attribute-specific component in the challenge ciphertext. By Eq. (5.6), in $\mathsf{Hyb}_1^{(b)}$, for each $k \in [K]$,

$$C_{3,k} = (g_1^s)^{\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}' - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}} = h_2^{s\mathbf{m}_k^\mathsf{T} \mathbf{v}'} \hat{U}_{\rho(k)}^{-s} = h_2^{\mathbf{m}_k^\mathsf{T} \mathbf{v}''} \hat{U}_{\rho(k)}^{-s},$$

  where $\mathbf{v}'' = s\mathbf{v}' = [s, sv_2, \ldots, sv_n]^\mathsf{T}$. This matches the distribution in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$ with the substitution $\mathbf{v} \mapsto \mathbf{v}''$.

- Finally, consider the slot-specific component in the challenge ciphertext in $\mathsf{Hyb}_1^{(b)}$. By Eq. (5.5),

$$\frac{Q_i^{\delta_i^{-1}}}{R_i} = \frac{P_i^{r_i \delta_i^{-1}}}{g_3^{r_i}} = \frac{g_1^{sr_i} g_3^{r_i}}{g_3^{r_i}} = g_1^{sr_i}.$$

  By Eq. (5.6), in $\mathsf{Hyb}_1^{(b)}$,

$$C_4 = (g_1^s)^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right) = h_1^s \prod_{i \in [L]} g_1^{-sr_i} = h_1^s \hat{T}^{-s}.$$

  C_4 is distributed identically to in $\mathsf{Hyb}_{\mathsf{real}}^{(b)}$. □

Lemma 5.13. *Suppose Assumption 5.2a 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 $\lambda \in \mathbb{N}$,*

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

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that distinguishes between $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_{2,0}^{(b)}$ with non-negligible advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2a with the same advantage:

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

2. Algorithm $\mathcal{B}$ starts by sampling $\alpha, \beta_1, \beta_2 \xleftarrow{R} \mathbb{Z}_N$. It sets $Z \leftarrow e(g_1, g_1)^\alpha$, $\beta \leftarrow \beta_1 + \beta_2$, and $h \leftarrow g_1^\beta$.

3. For each slot $i \in [L]$, sample $t_i, \delta_i, \tau_i \xleftarrow{R} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ constructs the slot components as follows:

$$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^\alpha A_i^\beta g_3^{\tau_i} \quad , \quad P_i \leftarrow (T g_3)^{\delta_i}.$$

   Then, for each attribute $w \in \mathcal{U}$ and each slot $i \in [L]$, algorithm $\mathcal{B}$ samples $u_{i,w} \xleftarrow{R} \mathbb{Z}_N$. In addition, for each $j \neq i$, it samples $\gamma_{i,j,w} \xleftarrow{R} \mathbb{Z}_N$. It constructs the attribute components $U_{i,w}$ and $W_{i,j,w}$ as follows:

$$U_{i,w} \leftarrow g_1^{u_{i,w}} \quad , \quad W_{i,j,w} \leftarrow A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}}.$$

   Algorithm $\mathcal{B}$ gives the common reference string

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{i,j,w}\}_{i \neq j, w \in \mathcal{U}} \right)$$

   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.

4. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_{2,0}^{(b)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{R} \mathbb{Z}_N$. It then computes $T_i \leftarrow g_1^{r_i}$, $Q_i \leftarrow P_i^{r_i}$, $R_i \leftarrow g_3^{r_i}$, and $V_{j,i} \leftarrow A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ 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 $1 \leq i \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[i]$ and replies to $\mathcal{A}$ with sk'.

5. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy P^*, the messages μ_0^*, μ_1^*, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{Hyb}_1^{(b)}$ and $\mathsf{Hyb}_{2,0}^{(b)}$:

   - If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i \leftarrow \mathsf{pk}'$. Otherwise, algorithm $\mathcal{B}$ aborts with output 0.
   - If $c_i = \perp$, then algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, it aborts with output 0. Otherwise, it sets $\mathsf{pk}_i \leftarrow \mathsf{pk}_i^*$.

   Finally, for each $i \in [L]$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$.

6. Algorithm $\mathcal{B}$ parses the challenge policy as $P^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:

   - **Message-embedding components:** Set $C_1 \leftarrow \mu_b^* \cdot e(g_1, T)^\alpha$ and $C_2 \leftarrow T$.
   - **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^\mathsf{T}$. For each $k \in [K]$, set
   $$C_{3,k} \leftarrow T^{\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}' - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}}.$$
   - **Slot-specific component:** Set
   $$C_4 \leftarrow T^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

7. 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 $\mathsf{Hyb}_1^{(b)}$. Alternatively, when $T = (g_1 g_2)^s$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,0}^{(b)}$. Thus, algorithm $\mathcal{B}$ breaks Assumption 5.2a with the same advantage ε. □

Lemma 5.14. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*

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

Proof. We introduce an intermediate hybrid $\mathsf{iHyb}_\ell^{(b)}$:

- $\mathsf{iHyb}_\ell^{(b)}$: Same as $\mathsf{Hyb}_{2,\ell-1}^{(b)}$, except we change the distribution of A_ℓ in the CRS. Specifically, during the setup phase, the challenger samples A_ℓ, B_ℓ, P_ℓ as
$$A_\ell \leftarrow g^{t_\ell} \quad , \quad B_\ell \leftarrow g_1^\alpha A_\ell^\beta g_3^{\tau_\ell} \quad , \quad P_\ell \leftarrow ((g_1 g_2)^s g_3)^{\delta_\ell}.$$
  In the previous hybrid $\mathsf{Hyb}_{2,\ell-1}^{(b)}$, we have that $A_\ell \leftarrow (g_1 g_3)^{t_\ell}$.

We now show that for all efficient adversaries $\mathcal{A}$, the output distributions of $\mathsf{Hyb}_{2,\ell-1}^{(b)}(\mathcal{A})$ and $\mathsf{iHyb}_\ell^{(b)}(\mathcal{A})$ are computationally indistinguishable, as are those of $\mathsf{iHyb}_\ell^{(b)}(\mathcal{A})$ and $\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A})$.

Lemma 5.15. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*

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

Proof. By construction, the only difference between these two hybrids is component A_ℓ in the CRS. 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 Assumption 5.2b with the same advantage:

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

2. Algorithm $\mathcal{B}$ starts by sampling $\alpha, \beta_1, \beta_2 \xleftarrow{R} \mathbb{Z}_N$. It sets $Z \leftarrow e(g_1, g_1)^\alpha$, $\beta = \beta_1 + \beta_2$, and $h \leftarrow g_1^\beta$.

3. For each $i \in [L]$, algorithm $\mathcal{B}$ samples $t_i, \delta_i, \tau_i \xleftarrow{R} \mathbb{Z}_N$.

   - For $i < \ell$, algorithm $\mathcal{B}$ sets
   $$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^\alpha A_i^\beta Y^{\tau_i} \quad , \quad P_i \leftarrow (X g_3)^{\delta_i}.$$
   - For $i = \ell$, algorithm $\mathcal{B}$ sets
   $$A_\ell \leftarrow T \quad , \quad B_\ell \leftarrow g_1^\alpha A_\ell^\beta g_3^{\tau_\ell} \quad , \quad P_\ell \leftarrow (X g_3)^{\delta_\ell}.$$
   - For $i > \ell$, algorithm $\mathcal{B}$ sets
   $$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^\alpha A_i^\beta g_3^{\tau_i} \quad , \quad P_i \leftarrow (X g_3)^{\delta_i}.$$

   Then, for each attribute $w \in \mathcal{U}$ and slot $i \in [L]$, sample $u_{i,w} \xleftarrow{R} \mathbb{Z}_N$ and for each $j \neq i$, sample $\gamma_{i,j,w} \xleftarrow{R} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ then constructs the attribute-specific slot components $U_{i,w}$ and $W_{i,j,w}$ as in $\mathsf{Hyb}_{2,\ell-1,2}^{(b)}$ and $\mathsf{iHyb}_\ell^{(b)}$:
   $$U_{i,w} = g_1^{u_{i,w}} \quad , \quad W_{i,j,w} = A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}}.$$
   Algorithm $\mathcal{B}$ gives the common reference string
   $$\mathsf{crs} = \big(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{i,j,w}\}_{i \neq j, w \in \mathcal{U}}\big)$$
   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.

4. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$ and $\mathsf{iHyb}_\ell^{(b)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{R} \mathbb{Z}_N$. It then computes $T_i \leftarrow g_1^{r_i}$, $Q_i \leftarrow P_i^{r_i}$, $R_i \leftarrow g_3^{r_i}$, and $V_{j,i} \leftarrow A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ 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 $1 \leq i \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[i]$ and replies to $\mathcal{A}$ with sk'.

5. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy P^*, the messages μ_0^*, μ_1^*, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$ and $\mathsf{iHyb}_\ell^{(b)}$:

   - If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i \leftarrow \mathsf{pk}'$. Otherwise, algorithm $\mathcal{B}$ aborts with output 0.
   - If $c_i = \perp$, then algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, it aborts with output 0. Otherwise, it sets $\mathsf{pk}_i \leftarrow \mathsf{pk}_i^*$.

   Finally, for each $i \in [L]$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$.

6. Algorithm $\mathcal{B}$ parses the challenge policy as $P^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:

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

- **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^\mathsf{T}$. For each $k \in [K]$, set
$$C_{3,k} \leftarrow X^{\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}' - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}}.$$
- **Slot-specific component:** Set
$$C_4 \leftarrow X^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

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

In the reduction, the exponent $s_{12} \xleftarrow{\text{R}} \mathbb{Z}_N$ plays the role of $s \xleftarrow{\text{R}} \mathbb{Z}_N$ in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$ and $\mathsf{iHyb}_\ell^{(b)}$. Note that in the reduction,
$$C_1 = \mu_b^* \cdot e(g_1, (g_1 g_2)^{s_{12}})^\alpha = \mu_b^* \cdot e(g_1, g_1)^{\alpha s_{12}} = Z^{s_{12}},$$
which matches the distribution in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$ and $\mathsf{iHyb}_\ell^{(b)}$. Next, consider the distribution of B_i for $i < \ell$. 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 $s_{23} \xleftarrow{\text{R}} \mathbb{Z}_N$), then the distributions
$$\{Y^{\tau_i} = (g_2 g_3)^{s_{23}\tau_i} : \tau_i \xleftarrow{\text{R}} \mathbb{Z}_N\} \quad \text{and} \quad \{(g_2 g_3)^{\tau_i} : \tau_i \xleftarrow{\text{R}} \mathbb{Z}_N\}$$
are identical. Consider now the distribution of A_ℓ:

- If $T = (g_1 g_3)^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$, then this is exactly the distribution in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$.
- If $T = g^t$ for some $t \xleftarrow{\text{R}} \mathbb{Z}_N$, then this is exactly the distribution in $\mathsf{iHyb}_\ell^{(b)}$.

Thus, we conclude that algorithm $\mathcal{B}$ breaks Assumption 5.2b with advantage at least $\varepsilon - \mathsf{negl}(\lambda)$. □

Lemma 5.16. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*
$$\left| \Pr[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda). \tag{5.7}$$

Proof. Our analysis will depend on whether the adversary knows the secret key associated with slot ℓ or not. Let $(c_i, S_i, \mathsf{pk}_i^*)$ be the tuples adversary $\mathcal{A}$ outputs for each slot $i \in [L]$ in the challenge phase. Let ctr be the number of key-generation queries the adversary has made at the beginning of the challenge phase. We say that event NonCorrupt occurs if
$$c_\ell \in \{1, \ldots, \mathsf{ctr}\} \text{ and } \mathcal{A} \text{ did not make a corruption query on index } c_\ell,$$
Let $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ be the public keys the challenger constructs during the challenge phase. If event NonCorrupt occurs, then the public key pk_ℓ was *honestly* sampled by the challenger in a key-registration query, and moreover, the adversary did *not* corrupt the key to learn its associated secret key. We write $\overline{\mathsf{NonCorrupt}}$ to denote the complement of event NonCorrupt. Now, we can write
$$\Pr\left[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] + \Pr\left[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right]$$
$$\Pr\left[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] + \Pr\left[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right].$$

It suffices then to show that
$$\left| \Pr\left[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] - \Pr\left[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] \right| = \mathsf{negl}(\lambda) \tag{5.8}$$
$$\left| \Pr\left[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] - \Pr\left[\mathsf{Hyb}_{2,\ell}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] \right| = \mathsf{negl}(\lambda). \tag{5.9}$$

Eq. (5.7) then follows by the triangle inequality.

General proof strategy. As surveyed in Section 2, the proof strategy for showing Eqs. (5.8) and (5.9) will construct a sequence of hybrid experiment culminating in an *information-theoretic* step that ensures the adversary cannot tell that ℓ^{th} slot has switched from normal mode to semi-functional mode. These two information-theoretic components critically relies on *different* admissibility properties on the adversary:

- If event NonCorrupt occurs, then the adversary does not know the secret key $\mathsf{sk}_\ell = r_\ell$ associated with slot ℓ (i.e., $r_\ell \xleftarrow{\text{R}} \mathbb{Z}_N$ is the secret exponent the challenger sampled when responding to the c_ℓ^{th} key-generation query). The final information-theoretic argument (Lemma 5.21) in the proof of Eq. (5.8) critically relies on the distribution of $r_\ell \bmod p_2$ being uniform and hidden from the view of the adversary. The full sequence of hybrids is described in the proof of Claim 5.17.
- If event $\overline{\mathsf{NonCorrupt}}$ occurs, then the adversary may know the secret key $\mathsf{sk}_\ell = r_\ell$ associated with slot ℓ, and as such, we cannot rely on the same information-theoretic argument as above. In this case, the admissibility requirement ensures that the set of attributes S_ℓ associated with slot ℓ do *not* satisfy the challenge policy. The final information-theoretic argument (Lemma 5.29) in the proof of Eq. (5.9) relies on information-theoretic security of the underlying linear secret sharing scheme. The full sequence of hybrids is described in the proof of Claim 5.26.

Analysis for the case where slot ℓ is not corrupted. We now show that Eq. (5.8) holds. As noted previously, when the public key pk_ℓ associated with slot ℓ is not corrupted, our analysis will (eventually) rely on the secret key $\mathsf{sk}_\ell = r_\ell$ associated with slot ℓ being hidden to argue that the semi-functional slot components look computationally indistinguishable from normal slot components. We state the precise claim below:

Claim 5.17. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*

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

Proof. To prove this claim, we introduce an additional sequence of (simpler) hybrid experiments:

- $\mathsf{ncHyb}_{\ell,0}^{(b)}$: Same as $\mathsf{iHyb}_\ell^{(b)}$ except during the challenge phase, the challenger constructs the challenge ciphertext as follows:
  - If event NonCorrupt did not occur, then the experiment halts with output 0.
  - Otherwise, if event NonCorrupt occurs, let pk_ℓ be the public key associated with slot ℓ. Since NonCorrupt occurs, the public key pk_ℓ was constructed by the challenger in response to the c_ℓ^{th} key-generation query the adversary made in the query phase. Let $r_\ell \in \mathbb{Z}_N$ be the randomness the challenger used to construct pk_ℓ (i.e., this is the secret key stored in $\mathsf{D}[c_\ell]$). Then, $\mathsf{pk}_\ell = \mathsf{KeyGen}(\mathsf{crs}, \ell; r_\ell)$. The challenger constructs the challenge ciphertext exactly as in $\mathsf{iHyb}_\ell^{(b)}$, except it computes C_4 as follows:

    $$C_4 \leftarrow (g_1 g_2)^{s\beta_1} (g_1 g_2)^{-s r_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right).$$

    The other components of the challenge ciphertext are constructed as in $\mathsf{iHyb}_\ell^{(b)}$. The output of the experiment is the output of $\mathcal{A}$, exactly as in $\mathsf{iHyb}_\ell^{(b)}$.

  Importantly, in this experiment, the only component that depends on the exponent $\delta_\ell \in \mathbb{Z}_N$ is P_ℓ. The challenge ciphertext no longer depends on δ_ℓ.
- $\mathsf{ncHyb}_{\ell,1}^{(b)}$: Same as $\mathsf{ncHyb}_{\ell,0}^{(b)}$, except the challenger samples $P_\ell \leftarrow g^{\delta_\ell}$ in the setup phase.
- $\mathsf{ncHyb}_{\ell,2}^{(b)}$: Same as $\mathsf{ncHyb}_{\ell,1}^{(b)}$ except the challenger samples $P_\ell \leftarrow (g_1 g_3)^{\delta_\ell}$ in the setup phase.

Hybrid	A_ℓ	B_ℓ	P_ℓ	**Justification**	
$\mathsf{Hyb}_{2,\ell-1}^{(b)}$	$(g_1g_3)^{t_\ell}$	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$((g_1g_2)^s g_3)^{\delta_\ell}$		
$\mathsf{iHyb}_\ell^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$((g_1g_2)^s g_3)^{\delta_\ell}$	Assumption 5.2b	Lemma 5.15
$\mathsf{ncHyb}_{\ell,0}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$((g_1g_2)^s g_3)^{\delta_\ell}$	Identical	Lemma 5.18
$\mathsf{ncHyb}_{\ell,1}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	g^{δ_ℓ}	Statistical	Lemma 5.19
$\mathsf{ncHyb}_{\ell,2}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$(g_1g_3)^{\delta_\ell}$	Assumption 5.2b	Lemma 5.20
$\mathsf{ncHyb}_{\ell,3}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$(g_1g_3)^{\delta_\ell}$	Statistical	Lemma 5.21
$\mathsf{ncHyb}_{\ell,4}^{(b)}$	$(g_1g_3)^{t_\ell}$	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$(g_1g_3)^{\delta_\ell}$	Assumption 5.2b	Lemma 5.22
$\mathsf{ncHyb}_{\ell,5}^{(b)}$	$(g_1g_3)^{t_\ell}$	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	g^{δ_ℓ}	Assumption 5.2b	Lemma 5.23
$\mathsf{ncHyb}_{\ell,6}^{(b)}$	$(g_1g_3)^{t_\ell}$	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$((g_1g_2)^s g_3)^{\delta_\ell}$	Statistical	Lemma 5.24
$\mathsf{Hyb}_{2,\ell}^{(b)}$	$(g_1g_3)^{t_\ell}$	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$((g_1g_2)^s g_3)^{\delta_\ell}$	Identical	Lemma 5.25

Table 1: Structure of slot parameters A_ℓ, B_ℓ, P_ℓ in the hybrid experiments for analyzing the NonCorrupt branch (Claim 5.17). For each pair of adjacent hybrids, we indicate whether they are identically distributed, statistically indistinguishable, or computationally indistinguishable. The highlighted row is the information-theoretic step that relies on event NonCorrupt occurring (i.e., that the adversary does not know the secret key for slot ℓ).

- $\mathsf{ncHyb}_{\ell,3}^{(b)}$: Same as $\mathsf{ncHyb}_{\ell,2}^{(b)}$ except the challenger samples $B_\ell \leftarrow g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$ in the setup phase.
- $\mathsf{ncHyb}_{\ell,4}^{(b)}$: Same as $\mathsf{ncHyb}_{\ell,3}^{(b)}$ except the challenger samples $A_\ell \leftarrow (g_1g_3)^{t_\ell}$ in the setup phase.
- $\mathsf{ncHyb}_{\ell,5}^{(b)}$: Same as $\mathsf{ncHyb}_{\ell,4}^{(b)}$ except the challenger samples $P_\ell \leftarrow g^{\delta_\ell}$ in the setup phase
- $\mathsf{ncHyb}_{\ell,6}^{(b)}$: Same as $\mathsf{ncHyb}_{\ell,5}^{(b)}$ except the challenger samples $P_\ell \leftarrow ((g_1g_2)^s g_3)^{\delta_\ell}$ in the setup phase

We provide a summary of the hybrid experiments in Table 1. We now show that each pair of adjacent hybrids are computationally indistinguishable.

Lemma 5.18. *For all adversaries $\mathcal{A}$ and $b \in \{0,1\}$,* $\Pr[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}] = \Pr[\mathsf{ncHyb}_{\ell,0}^{(b)}(\mathcal{A}) = 1]$.

Proof. By construction, the output of $\mathsf{ncHyb}_{\ell,0}^{(b)}(\mathcal{A})$ is 1 only if event NonCorrupt occurs. Then, $\mathsf{pk}_\ell = \big(T_\ell, Q_\ell, R_\ell, \{V_{j,\ell}\}_{j\neq\ell}\big) = \mathsf{KeyGen}(\mathsf{crs}, \ell; r_\ell)$. By construction of KeyGen, this means that

$$Q_\ell = P_\ell^{r_\ell} = ((g_1g_2)^s)g_3)^{\delta_\ell r_\ell}$$

and $R_\ell = g_3^{r_\ell}$. In particular, this means that

$$\frac{Q_\ell^{\delta_\ell^{-1}}}{R_\ell} = \frac{(g_1g_2)^{r_\ell s} g_3^{r_\ell}}{g_3^{r_\ell}} = (g_1g_2)^{r_\ell s}.$$

Thus, if event NonCorrupt occurs, then C_4 in $\mathsf{ncHyb}_{\ell,0}^{(b)}$ satisfies

$$C_4 = (g_1g_2)^{s\beta_1}(g_1g_2)^{-sr_\ell}\left(\prod_{i\in[L]\setminus\{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}}\right) = (g_1g_2)^{s\beta_1}\left(\frac{R_\ell}{Q_\ell^{\delta_\ell^{-1}}}\right)\left(\prod_{i\in[L]\setminus\{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}}\right) = (g_1g_2)^{s\beta_1}\left(\prod_{i\in[L]} \frac{R_i}{Q_i^{\delta_i^{-1}}}\right).$$

This is exactly the distribution of C_4 in $\mathsf{iHyb}_\ell(\mathcal{A})$. Therefore, conditioned on event NonCorrupt, the output distribution of $\mathsf{ncHyb}_{\ell,0}^{(b)}(\mathcal{A})$ is identical to the output distribution of $\mathsf{iHyb}_\ell(\mathcal{A})$. Correspondingly,

$$\begin{aligned}\Pr[\mathsf{ncHyb}_{\ell,0}^{(b)}(\mathcal{A}) = 1] &= \Pr[\mathsf{NonCorrupt}] \cdot \Pr[\mathsf{iHyb}_\ell(\mathcal{A}) = 1 \mid \mathsf{NonCorrupt}] \\ &= \Pr[\mathsf{iHyb}_\ell(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}],\end{aligned}$$

and the claim follows. □

Lemma 5.19. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$ and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\left|\Pr[\mathsf{ncHyb}_{\ell,0}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{ncHyb}_{\ell,1}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. The only difference between $\mathsf{ncHyb}_{\ell,0}^{(b)}$ and $\mathsf{ncHyb}_{\ell,1}^{(b)}$ is the distribution of P_ℓ. In $\mathsf{ncHyb}_{\ell,1}^{(b)}$, P_ℓ is uniform over $\mathbb{G}$. In $\mathsf{ncHyb}_{\ell,0}^{(b)}$, $P_\ell = g_1^{\delta_\ell s} g_2^{\delta_\ell s} g_3^{\delta_\ell}$. Since $\delta_\ell \xleftarrow{R} \mathbb{Z}_N$, as long as $s \bmod p_1$ and $s \bmod p_2$ are both non-zero, then the marginal distribution of P_ℓ is uniform over $\mathbb{G}$ (over the choice of δ_ℓ). Since $s \xleftarrow{R} \mathbb{Z}_N$, $s \bmod p_1$ and $s \bmod p_2$ are non-zero with probability at least $1 - 1/p_1 - 1/p_2 = 1 - \mathsf{negl}(\lambda)$. Thus, the marginal distribution of P_ℓ is statistically indistinguishable in $\mathsf{ncHyb}_{\ell,0}^{(b)}$ and $\mathsf{ncHyb}_{\ell,1}^{(b)}$. None of the other components in $\mathsf{ncHyb}_{\ell,0}^{(b)}$ and $\mathsf{ncHyb}_{\ell,1}^{(b)}$ depend on the exponent δ_ℓ, so the outputs of the two experiments are statistically indistinguishable. □

Lemma 5.20. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*

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

Proof. We use a similar argument as the proof of Lemma 5.15, except we use the challenge to program P_ℓ instead of A_ℓ. More formally, suppose there exists an efficient adversary $\mathcal{A}$ where

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

for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for Assumption 5.2b:

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

2. Algorithm $\mathcal{B}$ starts by sampling $\alpha, \beta_1, \beta_2 \xleftarrow{R} \mathbb{Z}_N$. It sets $Z \leftarrow e(g_1, g_1)^\alpha$, $\beta = \beta_1 + \beta_2$, and $h \leftarrow g_1^\beta$.

3. For each $i \in [L]$, algorithm $\mathcal{B}$ samples $t_i, \delta_i, \tau_i \xleftarrow{R} \mathbb{Z}_N$.
   - For $i < \ell$, algorithm $\mathcal{B}$ sets

   $$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^\alpha A_i^\beta Y^{\tau_i} \quad , \quad P_i \leftarrow (X g_3)^{\delta_i}.$$

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

   $$A_\ell \leftarrow g^{t_\ell} \quad , \quad B_\ell \leftarrow g_1^\alpha A_\ell^\beta g_3^{\tau_\ell} \quad , \quad P_\ell \leftarrow T.$$

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

   $$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^\alpha A_i^\beta g_3^{\tau_i} \quad , \quad P_i \leftarrow (X g_3)^{\delta_i}.$$

Then, for each attribute $w \in \mathcal{U}$ and slot $i \in [L]$, sample $u_{i,w} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and for each $j \neq i$, sample $\gamma_{i,j,w} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ then constructs the attribute-specific slot components $U_{i,w}$ and $W_{i,j,w}$ as in $\mathsf{Hyb}_{2,\ell-1,2}^{(b)}$ and $\mathsf{iHyb}_{\ell}^{(b)}$:

$$U_{i,w} = g_1^{u_{i,w}} \quad , \quad W_{i,j,w} = A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}} .$$

Algorithm $\mathcal{B}$ gives the common reference string

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, \{(A_i, B_i, P_i)\}_{i\in[L]}\,,\, \{U_{i,w}, W_{i,j,w}\}_{i\neq j, w\in\mathcal{U}}\right)$$

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.

4. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{ncHyb}_{\ell,1}^{(b)}$ and $\mathsf{ncHyb}_{\ell,2}^{(b)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} \leftarrow \mathsf{ctr}+1$ and samples $r_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. It then computes $T_i \leftarrow g_1^{r_i}$, $Q_i \leftarrow P_i^{r_i}$, $R_i \leftarrow g_3^{r_i}$, and $V_{j,i} \leftarrow A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j\neq i})$ 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 $1 \le i \le \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[i]$ and replies to $\mathcal{A}$ with sk'.

5. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy P^*, the messages μ_0^*, μ_1^*, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{ncHyb}_{\ell,1}^{(b)}$ and $\mathsf{ncHyb}_{\ell,2}^{(b)}$:

   - If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i \leftarrow \mathsf{pk}'$. Otherwise, algorithm $\mathcal{B}$ aborts with output 0.
   - If $c_i = \perp$, then algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, it aborts with output 0. Otherwise, it sets $\mathsf{pk}_i \leftarrow \mathsf{pk}_i^*$.

   Finally, for each $i \in [L]$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j\neq i})$.

6. Algorithm $\mathcal{B}$ parses the challenge policy as $P^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K\times n}$ and $\rho\colon [K] \to \mathcal{U}$. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:

   - **Message-embedding components:** Set $C_1 \leftarrow \mu_b^* \cdot e(g_1, X)^{\alpha}$ and $C_2 \leftarrow X$.
   - **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$. For each $k \in [K]$, set
   $$C_{3,k} \leftarrow X^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}} .$$
   - **Slot-specific component:** Set
   $$C_4 \leftarrow X^{\beta_1 - r_\ell} \left(\prod_{i\in[L]\setminus\{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

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

As in the proof of Lemma 5.15, the exponent $s_{12} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ plays the role of $s \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ in $\mathsf{ncHyb}_{\ell,1}^{(b)}$ and $\mathsf{ncHyb}_{\ell,2}^{(b)}$. Next, consider the distribution of B_i for $i < \ell$. 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 $s_{23} \xleftarrow{\mathrm{R}} \mathbb{Z}_N$), then the distributions

$$\{Y^{\tau_i} = (g_2 g_3)^{s_{23}\tau_i} : \tau_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N\} \quad \text{and} \quad \{(g_2 g_3)^{\tau_i} : \tau_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N\}$$

are identical. Consider now the distribution of P_ℓ:

- If $T = g^{\delta}$ for some $\delta \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, then the components coincide with the distribution in $\mathsf{ncHyb}_{\ell,1}^{(b)}$.
- If $T = (g_1 g_3)^{\delta}$ for some $\delta \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, then the components coincide with the distribution in $\mathsf{ncHyb}_{\ell,2}^{(b)}$.

Thus, we conclude that algorithm $\mathcal{B}$ breaks Assumption 5.2b with advantage at least $\varepsilon - \mathsf{negl}(\lambda)$. $\square$

Lemma 5.21. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{ncHyb}_{\ell,2}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{ncHyb}_{\ell,3}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. We show that the distributions $\mathsf{ncHyb}_{\ell,2}^{(b)}(\mathcal{A})$ and $\mathsf{ncHyb}_{\ell,3}^{(b)}(\mathcal{A})$ are statistically close. Let $(c_\ell, S_\ell, \mathsf{pk}_\ell^*)$ be the tuple the adversary chooses for slot ℓ during the challenge phase. Let $r_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ be the randomness the challenger used to answer the c_ℓ^{th} key-generation query. For either experiment to output 1, event NonCorrupt must occur, which means the adversary does *not* issue a corruption query on index c_ℓ. Correspondingly, the challenger *never* gives r_ℓ to the adversary. This property will be critical for arguing that the two distributions are statistically indistinguishable.

Consider the distributions $\mathsf{ncHyb}_{\ell,2}^{(b)}(\mathcal{A})$ and $\mathsf{ncHyb}_{\ell,3}^{(b)}(\mathcal{A})$. By construction, the only difference between them is the distribution of component B_ℓ in the $\mathbb{G}_2$ subgroup. In $\mathsf{ncHyb}_{\ell,2}^{(b)}$, $B_\ell = g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$ while in $\mathsf{ncHyb}_{\ell,3}^{(b)}$, $B_\ell = g_1^\alpha A_\ell^\beta (g_2 g_3)^{\tau_\ell}$, where $A_\ell = g^{t_\ell}$. Write $A_\ell = g_1^{t'_{\ell,1}} g_2^{t'_{\ell,2}} g_3^{t'_{\ell,3}}$ where $t'_{\ell,1} \in \mathbb{Z}_{p_1}$, $t'_{\ell,2} \in \mathbb{Z}_{p_2}$, and $t'_{\ell,3} \in \mathbb{Z}_{p_3}$. Suppose that $t'_{\ell,2} \neq 0$ and $s'_0 \neq 0 \bmod p_2$. Since $t_\ell, s'_0 \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, this holds with probability $1 - 2/p_2 = 1 - \mathsf{negl}(\lambda)$. Consider the following relabeling of the variables β_1 and r_ℓ in $\mathsf{ncHyb}_{\ell,2}^{(b)}$:

- Let $\sigma_1 \in \mathbb{Z}_N$ be the unique value where $\sigma_1 = 0 \bmod p_1 p_3$ and $\sigma_1 = (t'_{\ell,2})^{-1}\tau_\ell \bmod p_2$. Then, write $\beta_1 = \beta'_1 + \sigma_1$, for some $\beta'_1 \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.
- Let $\sigma_2 \in \mathbb{Z}_N$ be the unique value where $\sigma_2 = 0 \bmod p_1 p_3$ and $\sigma_2 = (s'_0)^{-1}\sigma_1 \bmod p_2$. Then write $r_\ell \leftarrow r'_\ell + \sigma_2$ for some $r'_\ell \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.

By construction, observe that the distributions of β_1 and r_ℓ remain uniform over $\mathbb{Z}_N$ in $\mathsf{ncHyb}_{\ell,2}^{(b)}$. Consider the other components in the adversary's view with the above relabeling. It suffices to only consider components that depend on either β_1 or r_ℓ since the other components are unchanged. Note also that by design, $\beta_1 = \beta'_1 \bmod p_1 p_3$ and $r_\ell = r'_\ell \bmod p_1 p_3$.

- Consider the components in the common reference string. First, $h = g_1^{\beta_1 + \beta_2} = g_1^{\beta'_1 + \beta_2}$. Next $A_i = (g_1 g_3)^{t_i}$ for all $i \neq \ell$ and $A_\ell = g_1^{t'_{\ell,1}} g_2^{t'_{\ell,2}} g_3^{t'_{\ell,3}}$. Consider the distribution of each B_i:
  - If $i < \ell$, then $B_i = g_1^\alpha A_i^{\beta_1+\beta_2} (g_2 g_3)^{\tau_i} = g_1^\alpha A_i^{\beta'_1+\beta_2} (g_2 g_3)^{\tau_i}$.
  - If $i = \ell$, then
  $$B_\ell = g_1^\alpha A_\ell^{\beta_1+\beta_2} g_3^{\tau_\ell} = g_1^\alpha g_1^{t'_{\ell,1}(\beta_1+\beta_2)} g_2^{t'_{\ell,2}(\beta_1+\beta_2)} g_3^{t'_{\ell,3}(\beta_1+\beta_2)} g_3^{\tau_\ell} = g_1^\alpha A_\ell^{\beta'_1+\beta_2} (g_2 g_3)^{\tau_\ell},$$
  since $\beta_1 = \beta'_1 \bmod p_1 p_3$ and $\beta_1 = \beta'_1 + (t'_{\ell,2})^{-1}\tau_\ell \bmod p_2$.
  - If $i > \ell$, then $B_i = g_1^\alpha A_i^{\beta_1+\beta_2} g_3^{\tau_i} = g_1^\alpha A_i^{\beta'_1+\beta_2} g_3^{\tau_i}$.

  The remaining components in the CRS do not depend on either β_1 or r_ℓ and are thus unchanged.
- Consider the components in the key-generation queries. The only key-generation query that is affected by this change of variables is the c_ℓ^{th} query. When the adversary makes the c_ℓ^{th} key-generation query, the challenger constructs the public key $\mathsf{pk}_\ell = (T_\ell, Q_\ell, R_\ell, \{V_{j,\ell}\}_{j \neq \ell})$ using randomness r_ℓ. Under the above substitution this means $T_\ell = g_1^{r_\ell} = g_1^{r'_\ell}$, $Q_\ell = P_\ell^{r_\ell} = P_\ell^{r'_\ell}$, $R_\ell = g_3^{r_\ell} = g_3^{r'_\ell}$, and $V_{j,\ell} = A_j^{r_\ell} = A_j^{r'_\ell}$ for all $j \neq \ell$ since $r_\ell = r'_\ell \bmod p_1 p_3$, and the components P_ℓ and A_j for $j \neq \ell$ do not contain any non-trivial components in the $\mathbb{G}_2$ subgroup. Here, it is critical that $P_\ell = (g_1 g_3)^{\delta_\ell}$ in $\mathsf{ncHyb}_{\ell,2}^{(b)}$ does *not* contain any components in $\mathbb{G}_2$.
- Finally, consider the components in the challenge ciphertext. The components $C_1, C_2, C_{3,k}$ for $k \in [K]$ are all unchanged (i.e., they are independent of β_1 and r_ℓ). Consider now ciphertext component C_4. In $\mathsf{ncHyb}_{\ell,2}^{(b)}$,
$$C_4 = (g_1 g_2)^{s\beta_1} (g_1 g_2)^{-s r_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right) = (g_1 g_2)^{s\beta'_1} (g_1 g_2)^{-s r'_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right),$$

since $\beta_1 = \beta_1' \bmod p_1$ and $r_\ell = r_\ell' \bmod p_1$, and

$$s\beta_1 - sr_\ell = s(\beta_1' + \sigma_1) - s(r_\ell' + \sigma_1)) = s\beta_1' - sr_\ell' \bmod p_2.$$

Observe now that this is precisely the distribution in $\mathsf{ncHyb}_{\ell,3}^{(b)}$ (with the relabeling $\beta_1 \mapsto \beta_1'$ and $r_\ell \mapsto r_\ell'$). Thus, whenever $t_{\ell,2}' \neq 0$ and $s_0' \neq 0 \bmod p_2$, hybrids $\mathsf{ncHyb}_{\ell,2}^{(b)}$ and $\mathsf{ncHyb}_{\ell,3}^{(b)}$ are identically distributed. Since this holds with probability $1 - \mathsf{negl}(\lambda)$ over the choice of t_ℓ and s_0', the claim holds. Note that this argument critically relies on the fact that r_ℓ is not given to the adversary in the game, as this allows us to reinterpret r_ℓ as $r_\ell' = r_\ell + \sigma_2$. □

Lemma 5.22. *Suppose Assumption 5.2b holds with respect of* 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 $\lambda \in \mathbb{N}$,*

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

Proof. This follows by a similar argument as the proof of Lemma 5.15. □

Lemma 5.23. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*

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

Proof. This follows by a similar argument as the proof of Lemma 5.20. □

Lemma 5.24. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\left|\Pr[\mathsf{ncHyb}_{\ell,5}^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{ncHyb}_{\ell,6}^{(b)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. This follows by the same argument as the proof of Lemma 5.19. □

Lemma 5.25. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$,*

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

Proof. This follows by the same argument as the proof of Lemma 5.18. □

Combining Lemmas 5.18 to 5.25, Claim 5.17 now follows by a hybrid argument. □

Analysis for the case where slot ℓ is corrupted. Next, we show that Eq. (5.9) holds. As noted previously, when slot ℓ is corrupted (and the adversary knows the associated secret key), we are guaranteed that the set of attributes S_ℓ associated with slot ℓ does *not* satisfy the challenge policy. Our analysis here will (eventually) rely on the security of the linear secret sharing scheme to argue that that the semi-functional slot components look computationally indistinguishable from normal slot components. We state the precise claim below:

Claim 5.26. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*

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

Proof. Similar to the proof of Claim 5.17, we introduce an additional sequence of hybrid experiments:

- $\mathsf{cHyb}_{\ell,0}^{(b)}$: Same as $\mathsf{iHyb}_{\ell}^{(b)}$ except during the challenge phase, when constructing the challenge ciphertext, the challenger performs several additional checks:
  - If event NonCorrupt occurs, then the experiment halts with output 0.

    - Let pk_ℓ be the public key associated with slot ℓ and $S_\ell \subseteq \mathcal{U}$ be the set of associated attributes. Let $P^* = (\mathbf{M}, \rho)$ be the challenge policy where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$ is an injective row-labeling function. Let $I = \{k \in [K] : \rho(k) \in S_\ell\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes in S_ℓ, and let $\mathbf{M}_I$ be the corresponding submatrix of $\mathbf{M}$. Since event NonCorrupt does not occur, this means that S_ℓ does not satisfy the policy $(\mathbf{M}, \rho)$, so $\mathbf{e}_1^{\mathsf{T}}$ is *not* in the row-span of $\mathbf{M}_I$. This means that there exists a vector $\mathbf{v}^* \in \mathbb{Z}_N^n$ such that $\mathbf{M}_I \mathbf{v}^* = \mathbf{0} \bmod N$ and $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* \neq 0 \bmod N$. In this experiment, the challenger computes $\mathbf{v}^* \in \mathbb{Z}_N^n$ using Gaussian elimination.
    - If $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$, the experiment halts with output 0.

  The rest of the experiment proceeds as in $\mathsf{iHyb}_\ell^{(b)}$.

- $\mathsf{cHyb}_{\ell,1}^{(b)}$: Same as $\mathsf{cHyb}_{\ell,0}^{(b)}$ except the challenger changes how it constructs the $C_{3,k}$ components in the challenger ciphertext:
    - Sample $\xi \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $\hat{v}'_2, \ldots, \hat{v}'_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $\hat{\mathbf{v}}' = [\beta_2 - \xi v_1^*, \hat{v}'_2, \ldots, \hat{v}'_n]^{\mathsf{T}}$.
    - For each $k \in [K]$, set $C_{3,k} \leftarrow ((g_1 g_2)^s)^{\mathbf{m}_k^{\mathsf{T}}(\hat{\mathbf{v}}' + \xi \mathbf{v}^*) - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}}$.

  All of the other components are constructed exactly as in $\mathsf{cHyb}_{\ell,0}^{(b)}$.

- $\mathsf{cHyb}_{\ell,2}^{(b)}$: Same as $\mathsf{cHyb}_{\ell,1}^{(b)}$ except the challenger samples $B_\ell \leftarrow g_1^\alpha A_\ell^\beta (g_2 g_3)^{\tau_\ell}$ in the setup phase.

- $\mathsf{cHyb}_{\ell,3}^{(b)}$: Same as $\mathsf{cHyb}_{\ell,2}^{(b)}$, except the challenger changes how it constructs the $C_{3,k}$ components in the challenge ciphertext:
    - Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$.
    - For each $k \in [K]$, set $C_{3,k} \leftarrow ((g_1 g_2)^s)^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}}$.

- $\mathsf{cHyb}_{\ell,4}^{(b)}$: Same as $\mathsf{cHyb}_{\ell,3}^{(b)}$, except the experiment no longer halts with output 0 if $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$.

We provide a summary of the hybrid experiments in Table 2. We now show that each pair of adjacent hybrids are computationally indistinguishable.

Lemma 5.27. *Suppose Assumption 5.2b 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*

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

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ where

$$\left| \Pr[\mathsf{iHyb}_\ell^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}] - \Pr[\mathsf{cHyb}_{\ell,0}^{(b)}(\mathcal{A}) = 1] \right| = \varepsilon$$

for some non-negligible ε. Since these two experiments are identical except the additional check of whether $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$, this means that with probability at least ε, algorithm $\mathcal{A}$ outputs a challenge $(\mathbf{M}, \rho)$ such that $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* \neq 0 \bmod N$ but $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$, where $\mathbf{v}^*$ is the vector derived from $(\mathbf{M}, \rho)$ according to the specification of $\mathsf{cHyb}_{\ell,0}^{(b)}$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2b via Lemma 5.3:

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

2. Algorithm $\mathcal{B}$ starts by sampling $\alpha, \beta_1, \beta_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. It sets $Z \leftarrow e(g_1, g_1)^\alpha$, $\beta = \beta_1 + \beta_2$, and $h \leftarrow g_1^\beta$.

3. For each $i \in [L]$, algorithm $\mathcal{B}$ samples $t_i, \delta_i, \tau_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$.

Hybrid	A_ℓ	B_ℓ	$C_{3,k}$	**Justification**	
$\mathsf{Hyb}_{2,\ell-1}^{(b)}$	$(g_1g_3)^{t_\ell}$	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$((g_1g_2)^s)^{\beta_2\mathbf{m}_k^\mathsf{T}\mathbf{v}' - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$		
$\mathsf{iHyb}_{\ell}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$((g_1g_2)^s)^{\beta_2\mathbf{m}_k^\mathsf{T}\mathbf{v}' - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$	Assumption 5.2b	Lemma 5.15
$\mathsf{cHyb}_{\ell,0}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$((g_1g_2)^s)^{\beta_2\mathbf{m}_k^\mathsf{T}\mathbf{v}' - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$	Assumption 5.2b	Lemma 5.27
$\mathsf{cHyb}_{\ell,1}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$	$((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}'+\xi\mathbf{v}^*) - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$	Identical	Lemma 5.28
$\mathsf{cHyb}_{\ell,2}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}'+\xi\mathbf{v}^*) - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$	Statistical	Lemma 5.29
$\mathsf{cHyb}_{\ell,3}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$((g_1g_2)^s)^{\beta_2\mathbf{m}_k^\mathsf{T}\mathbf{v}' - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$	Statistical	Lemma 5.30
$\mathsf{cHyb}_{\ell,4}^{(b)}$	g^{t_ℓ}	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$((g_1g_2)^s)^{\beta_2\mathbf{m}_k^\mathsf{T}\mathbf{v}' - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$	Assumption 5.2b	Lemma 5.31
$\mathsf{Hyb}_{2,\ell}^{(b)}$	$(g_1g_3)^{t_\ell}$	$g_1^\alpha A_\ell^\beta (g_2g_3)^{\tau_\ell}$	$((g_1g_2)^s)^{\beta_2\mathbf{m}_k^\mathsf{T}\mathbf{v}' - \sum_{i\in[L]:\rho(k)\notin S_i} u_{i,\rho(k)}}$	Assumption 5.2b	Lemma 5.32

Table 2: Structure of slot parameters A_ℓ, B_ℓ and challenge ciphertext component $C_{3,k}$ in the hybrid experiments for analyzing the $\overline{\mathsf{NonCorrupt}}$ branch (Claim 5.26). For each pair of adjacent hybrids, we indicate whether they are identically distributed, statistically indistinguishable, or computationally indistinguishable. The highlighted row is the information-theoretic step that relies on event $\overline{\mathsf{NonCorrupt}}$ occurring (i.e., that the set of attributes S_ℓ associated with slot ℓ does *not* satisfy the challenge policy P^*). Note that two of the hybrid experiments either introduce or remove an abort condition ($\mathsf{cHyb}_{\ell,0}^{(b)}$ and $\mathsf{cHyb}_{\ell,4}^{(b)}$) *without* changing the distribution of A_ℓ, B_ℓ, and $C_{3,k}$.

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

$$A_i \leftarrow (g_1g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^\alpha A_i^\beta Y^{\tau_i} \quad , \quad P_i \leftarrow (Xg_3)^{\delta_i}.$$

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

$$A_\ell \leftarrow g^{t_\ell} \quad , \quad B_\ell \leftarrow g_1^\alpha A_\ell^\beta g_3^{\tau_\ell} \quad , \quad P_\ell \leftarrow (Xg_3)^{\delta_\ell}.$$

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

$$A_i \leftarrow (g_1g_3)^{t_i} \quad , \quad B_i \leftarrow g_1^\alpha A_i^\beta g_3^{\tau_i} \quad , \quad P_i \leftarrow (Xg_3)^{\delta_i}.$$

Then, for each attribute $w \in \mathcal{U}$ and slot $i \in [L]$, sample $u_{i,w} \xleftarrow{R} \mathbb{Z}_N$ and for each $j \neq i$, sample $\gamma_{i,j,w} \xleftarrow{R} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ then constructs the attribute-specific slot components $U_{i,w}$ and $W_{i,j,w}$ as in $\mathsf{Hyb}_{2,\ell-1}^{(b)}$ and $\mathsf{iHyb}_\ell^{(b)}$:

$$U_{i,w} = g_1^{u_{i,w}} \quad , \quad W_{i,j,w} = A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}}.$$

Algorithm $\mathcal{B}$ gives the common reference string

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, \{(A_i, B_i, P_i)\}_{i\in[L]}\,,\, \{U_{i,w}, W_{i,j,w}\}_{i\neq j, w\in\mathcal{U}}\right)$$

to the adversary $\mathcal{A}$.

4. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries exactly as described in $\mathsf{iHyb}_\ell^{(b)}$ and $\mathsf{cHyb}_{\ell,0}^{(b)}$.
5. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy P^*, the messages μ_0^*, μ_1^*, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. Algorithm $\mathcal{B}$ parses $P^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K\times n}$ and $\rho\colon [K] \to \mathcal{U}$, and computes $\mathbf{v}^*$ as described in $\mathsf{cHyb}_{\ell,0}^{(b)}$, and outputs $\gcd(N, \mathbf{e}_1^\mathsf{T}\mathbf{v}^*)$.
6. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{iHyb}_\ell^{(b)}$ and $\mathsf{cHyb}_{\ell,0}^{(b)}$.

Like in the proof of Lemma 5.15, the exponent $s_{12} \xleftarrow{R} \mathbb{Z}_N$ plays the role of $s \xleftarrow{R} \mathbb{Z}_N$ in $\mathsf{iHyb}_\ell^{(b)}$ and $\mathsf{cHyb}_{\ell,0}^{(b)}$. Next, consider the distribution of B_i for $i < \ell$. 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 $s_{23} \xleftarrow{R} \mathbb{Z}_N$), then the distributions

$$\{Y^{\tau_i} = (g_2 g_3)^{s_{23}\tau_i} : \tau_i \xleftarrow{R} \mathbb{Z}_N\} \quad \text{and} \quad \{(g_2 g_3)^{\tau_i} : \tau_i \xleftarrow{R} \mathbb{Z}_N\}$$

are identical. The other components are simulated exactly as in $\mathsf{iHyb}_\ell^{(b)}$ and $\mathsf{cHyb}_{\ell,0}^{(b)}$, so with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, algorithm $\mathcal{A}$ outputs $(\mathbf{M}, \rho)$ such that $\mathbf{e}_1^\mathsf{T}\mathbf{v}^* \neq 0 \bmod N$ but $\mathbf{e}_1^\mathsf{T}\mathbf{v}^* = 0 \bmod p_2$. In this case, $\gcd(N, \mathbf{e}_1^\mathsf{T}\mathbf{v}^*)$ yields a non-trivial factor of N (and wins the game in Lemma 5.3). □

Lemma 5.28. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, $\Pr[\mathsf{cHyb}_{\ell,0}^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{cHyb}_{\ell,1}^{(b)}(\mathcal{A}) = 1]$.*

Proof. Without loss of generality, we can assume that NonCorrupt does not occur and moreover, $\mathbf{e}_1^\mathsf{T}\mathbf{v}^* \neq 0 \bmod p_2$. Otherwise, the output in both experiments is 0. The only difference between the two distributions is the distribution of the challenge ciphertext components $C_{3,k}$. In $\mathsf{cHyb}_{\ell,1}^{(b)}$, $C_{3,k} = ((g_1 g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi\mathbf{v}^*) - \sum_{i \in [L]:\rho(k) \notin S_i} u_{i,\rho(k)}}$. By definition,

$$\hat{\mathbf{v}}' + \xi\mathbf{v}^* = [\beta_2, \hat{v}_2' + \xi v_2^*, \ldots, \hat{v}_n' + \xi v_n^*] = \beta_2 \hat{\mathbf{v}}'',$$

where $\hat{\mathbf{v}}'' = [1, \hat{v}_2'', \ldots, \hat{v}_n'']$, and the distribution of $\hat{v}_2'', \ldots, \hat{v}_n''$ are independent and uniform over $\mathbb{Z}_N$ (since $\hat{v}_2', \ldots, \hat{v}_n' \xleftarrow{R} \mathbb{Z}_N$). Thus, we can equivalently write $C_{3,k}$ as

$$C_{3,k} = ((g_1 g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi\mathbf{v}^*) - \sum_{i \in [L]:\rho(k) \notin S_i} u_{i,\rho(k)}} = ((g_1 g_2)^s)^{\beta_2 \mathbf{m}_k^\mathsf{T}\hat{\mathbf{v}}'' - \sum_{i \in [L]:\rho(k) \notin S_i} u_{i,\rho(k)}}.$$

This is precisely the distribution of $C_{3,j}$ in $\mathsf{cHyb}_{\ell,0}^{(b)}$. □

Lemma 5.29. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. We show that the distributions $\mathsf{cHyb}_{\ell,1}^{(b)}(\mathcal{A})$ and $\mathsf{cHyb}_{\ell,2}^{(b)}(\mathcal{A})$ are statistically indistinguishable. This argument will rely on the fact that the attributes S_ℓ associated with slot ℓ do *not* satisfy the challenge policy. By construction, the only difference between the two experiments is the distribution of component B_ℓ in the $\mathbb{G}_2$ subgroup. In $\mathsf{cHyb}_{\ell,1}^{(b)}$, $B_\ell = g_1^\alpha A_\ell^\beta g_3^{\tau_\ell}$ while in $\mathsf{cHyb}_{\ell,2}^{(b)}$, $B_\ell = g_1^\alpha A_\ell^\beta (g_2 g_3)^{\tau_\ell}$, where $A_\ell = g^{t_\ell}$ in both experiments. We start by defining a few quantities that will be useful in our analysis:

- Write $A_\ell = g_1^{t'_{\ell,1}} g_2^{t'_{\ell,2}} g_3^{t'_{\ell,3}}$ where $t'_{\ell,1} \in \mathbb{Z}_{p_1}$, $t'_{\ell,2} \in \mathbb{Z}_{p_2}$, and $t'_{\ell,3} \in \mathbb{Z}_{p_3}$.
- Let $P^* = (\mathbf{M}, \rho)$ be the challenge policy where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$ is the row-labeling function.
- Let $S_\ell \subseteq \mathcal{U}$ be the set of attributes associated with slot ℓ, and let $I = \{k \in [K] : \rho(k) \in S_\ell\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes in S_ℓ. Let $\mathbf{M}_I$ be the corresponding submatrix of $\mathbf{M}$.
- Let $\mathbf{v}^* \in \mathbb{Z}_N^n$ be the vector where $\mathbf{M}_I \mathbf{v}^* = \mathbf{0} \bmod p_2$ and $\mathbf{e}_1^\mathsf{T}\mathbf{v}^* \neq 0 \bmod p_2$.

Suppose that $t'_{\ell,2} \neq 0 \bmod p_2$. Since $t_\ell \xleftarrow{R} \mathbb{Z}_N$, this holds with probability at least $1 - 1/p_2 = 1 - \mathsf{negl}(\lambda)$. Consider the following relabeling of the variables in $\mathsf{cHyb}_{\ell,1}^{(b)}$:

- Let $\sigma^{(\beta)} \in \mathbb{Z}_N$ be the unique value where $\sigma^{(\beta)} = 0 \bmod p_1 p_3$ and $\sigma^{(\beta)} = (t'_{\ell,2})^{-1}\tau_\ell \bmod p_2$. Suppose we write $\beta_2 = \beta_2' + \sigma^{(\beta)}$ for some $\beta_2' \xleftarrow{R} \mathbb{Z}_N$.
- Let $\sigma^{(\xi)} \in \mathbb{Z}_N$ be the unique value where $\sigma^{(\xi)} = 0 \bmod p_1 p_3$ and $\sigma^{(\xi)} = (v_1^*)^{-1}\sigma^{(\beta)} \bmod p_2$. Suppose we write $\xi = \xi' + \sigma^{(\xi)}$ for some $\xi' \xleftarrow{R} \mathbb{Z}_N$.

- For each $k \in [K]$ where $\rho(k) \notin S_\ell$, let $\sigma_k^{(u)}$ be the unique value where $\sigma_k^{(u)} = 0 \bmod p_1 p_3$ and $\sigma_k^{(u)} = \sigma^{(\xi)} \mathbf{m}_k^\top \mathbf{v}^* \bmod p_2$. Suppose we write $u_{\ell,\rho(k)} = u'_{\ell,\rho(k)} + \sigma_k^{(u)}$ for some $u'_{\ell,\rho(k)} \xleftarrow{\text{R}} \mathbb{Z}_N$.

By construction, observe that these substitutions preserve the distribution of β_2, ξ, and $u_{\ell,\rho(k)}$ in $\mathsf{cHyb}_{\ell,1}^{(b)}$. Consider the remaining components in the adversary's view with this variable substitution:

- Consider the components in the common reference string. First, $h = g_1^{\beta_1+\beta_2} = g_1^{\beta_1+\beta_2'}$. Next $A_i = (g_1 g_3)^{t_i}$ for all $i \neq \ell$ and $A_\ell = g_1^{t'_{\ell,1}} g_2^{t'_{\ell,2}} g_3^{t'_{\ell,3}}$. Consider the distribution of each B_i:

  - If $i < \ell$, then $B_i = g_1^\alpha A_i^{\beta_1+\beta_2} (g_2 g_3)^{\tau_i} = g_1^\alpha A_i^{\beta_1+\beta_2'} (g_2 g_3)^{\tau_i}$.
  - If $i = \ell$, then
  $$B_\ell = g_1^\alpha A_\ell^{\beta_1+\beta_2} g_3^{\tau_\ell} = g_1^\alpha g_1^{t'_{\ell,1}(\beta_1+\beta_2)} g_2^{t'_{\ell,2}(\beta_1+\beta_2)} g_3^{t'_{\ell,3}(\beta_1+\beta_2)} g_3^{\tau_\ell} = g_1^\alpha A_\ell^{\beta_1+\beta_2'} (g_2 g_3)^{\tau_\ell},$$
  since $\beta_2 = \beta_2' \bmod p_1 p_3$ and $\beta_2 = \beta_2' + (t'_{\ell,2})^{-1} \tau_\ell \bmod p_2$.
  - If $i > \ell$, then $B_i = g_1^\alpha A_i^{\beta_1+\beta_2} g_3^{\tau_i} = g_1^\alpha A_i^{\beta_1+\beta_2'} g_3^{\tau_i}$.

  Consider the slot components $U_{\ell,\rho(k)}$ and $W_{i,\ell,\rho(k)}$ for all $k \in [K]$ where $\rho(k) \notin S_\ell$ and $i \neq \ell$. By definition,
  $$\begin{aligned} U_{\ell,\rho(k)} &= g_1^{u_{\ell,\rho(k)}} = g_1^{u'_{\ell,\rho(k)}} \\ W_{i,\ell,\rho(k)} &= A_i^{u_{\ell,\rho(k)}} g_3^{\gamma_{i,\ell,\rho(k)}} = A_i^{u'_{\ell,w}} g_3^{\gamma_{i,\ell,\rho(k)}}, \end{aligned}$$
  since $A_i = (g_1 g_3)^{t_i}$ for all $i \neq \ell$. The remaining components in the CRS do not depend on β_2, ξ, or $u_{\ell,\rho(k)}$, and are thus unchanged.

- Next, the components the challenger constructs when responding to key-generation queries do not depend on the exponents β_2, ξ, or $u_{\ell,\rho(k)}$, so their distributions (given the components in the CRS) are unchanged with this substitution.

- Finally, consider the components in the challenge ciphertext. The components C_1, C_2, C_4 for $k \in [K]$ are all unchanged (i.e., they are independent of $\beta_2, \xi, u_{\ell,\rho(k)}$). It suffices to consider the ciphertext components $C_{3,k}$. First, since $\beta_2 = \beta_2' + \sigma^{(\beta)}$ and $\xi = \xi' + (v_1^*)^{-1} \sigma^{(\beta)}$, we have
  $$\hat{\mathbf{v}}' = [\beta_2 - \xi v_1^*, \hat{v}_2', \ldots, \hat{v}_n'] = [\beta_2' - \xi' v_1^*, \hat{v}_2', \ldots, \hat{v}_n'] \bmod N.$$
  We now consider two possibilities:

  - Suppose $\rho(k) \in S_\ell$. By definition, $\mathbf{m}_k^\top \mathbf{v}^* = 0$ in this case, so we can write $C_{3,k}$ as
  $$C_{3,k} = ((g_1 g_2)^s)^{\mathbf{m}_k^\top (\hat{\mathbf{v}}' + \xi \mathbf{v}^*) - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}} = ((g_1 g_2)^s)^{\mathbf{m}_k^\top \hat{\mathbf{v}}' - \sum_{i \neq \ell : \rho(k) \notin S_i} u_{i,\rho(k)}}.$$
  - Suppose $\rho(k) \notin S_\ell$. Then, we can write $C_{3,k}$ as
  $$\begin{aligned} C_{3,k} &= ((g_1 g_2)^s)^{\mathbf{m}_k^\top (\hat{\mathbf{v}}' + \xi \mathbf{v}^*) - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}} \\ &= ((g_1 g_2)^s)^{\mathbf{m}_k^\top (\hat{\mathbf{v}}' + \xi \mathbf{v}^*) - \sum_{i \neq \ell : \rho(k) \notin S_i} u_{i,\rho(k)} - u_{\ell,\rho(k)}} \\ &= ((g_1 g_2)^s)^{\mathbf{m}_k^\top (\hat{\mathbf{v}}' + \xi' \mathbf{v}^*) - \sum_{i \neq \ell : \rho(k) \notin S_i} u_{i,\rho(k)} - u'_{\ell,\rho(k)}}, \end{aligned}$$
  since
  $$\begin{aligned} \xi \mathbf{m}_k^\top \mathbf{v}^* - u_{\ell,\rho(k)} &= \xi' \mathbf{m}_k^\top \mathbf{v}^* - u'_{\ell,\rho(k)} \bmod p_1 p_3 \\ \xi \mathbf{m}_k^\top \mathbf{v}^* - u_{\ell,\rho(k)} &= \big(\xi' + \sigma^{(\xi)}\big) \mathbf{m}_k^\top \mathbf{v}^* - \big(u'_{\ell,\rho(k)} + \sigma^{(\xi)} \mathbf{m}_k^\top \mathbf{v}^*\big) \bmod p_2 \\ &= \xi' \mathbf{m}_k^\top \mathbf{v}^* - u'_{\ell,\rho(k)} \bmod p_2. \end{aligned}$$

Observe now with this relabeling of variables, we have recovered the ciphertext distribution in $\mathsf{cHyb}^{(b)}_{\ell,2}$ (with randomness β_2', ξ' and $u'_{\ell,\rho(k)}$). Thus, as long as $t'_{\ell,2} \neq 0$, the distributions $\mathsf{cHyb}^{(b)}_{\ell,2}$ and $\mathsf{cHyb}^{(b)}_{\ell,3}$ are identically distributed. This holds with probability $1 - \mathsf{negl}(\lambda)$ over the choice of t_ℓ. ◻

Lemma 5.30. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $\left|\Pr[\mathsf{cHyb}^{(b)}_{\ell,2}(\mathcal{A}) = 1] - \Pr[\mathsf{cHyb}^{(b)}_{\ell,3}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.*

Proof. This follows by the same argument as in the proof of Lemma 5.28. ◻

Lemma 5.31. *Suppose Assumption 5.2b 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*

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

Proof. This follows by a similar argument as in the proof of Lemma 5.27. ◻

Lemma 5.32. *Suppose Assumption 5.2b 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 $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{cHyb}^{(b)}_{\ell,4}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{2,\ell}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}]\right| = \mathsf{negl}(\lambda).$$

Proof. By construction, when event NonCorrupt does not occur, the only difference between $\mathsf{cHyb}^{(b)}_{\ell,4}$ and $\mathsf{Hyb}^{(b)}_{2,\ell}$ is the distribution of A_ℓ in the common reference string. Namely, in $\mathsf{cHyb}^{(b)}_{\ell,4}$, the challenger samples $A_\ell \leftarrow g^{t_\ell}$ while in $\mathsf{Hyb}^{(b)}_{2,\ell}$, the challenger samples $A_\ell \leftarrow (g_1 g_3)^{t_\ell}$. The argument now follows from a similar argument as in the proof of Lemma 5.15. ◻

Combining Lemmas 5.27 to 5.32, Claim 5.26 now follows by a hybrid argument. ◻

By Claim 5.17, we have that Eq. (5.8) holds (i.e., the case where slot ℓ was not corrupted). Similarly, by Claim 5.26, we have that Eq. (5.9) holds (i.e., the case where slot ℓ was corrupted but the attributes S_ℓ do not satisfy the challenge policy). The main claim (Eq. (5.7)) now follows by the triangle inequality. ◻

Lemma 5.14 now follows by combining Lemmas 5.15 and 5.16. ◻

Lemma 5.33. *Suppose Assumption 5.2c 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 $\lambda \in \mathbb{N}$,*

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

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ where $\left|\Pr[\mathsf{Hyb}^{(b)}_{2,L}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_{\mathsf{rand}}(\mathcal{A}) = 1]\right| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2c with the same advantage:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_2, g_3, X, Y, T)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, $g_3 \in \mathbb{G}_3$, $X = g_1^\alpha g_2^{\gamma_1}$, $Y = g_1^s g_2^{\gamma_2}$ for some $\alpha, \gamma_1, \gamma_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, and either $T = e(g_1, g_1)^{\alpha s}$ or $T = e(g,g)^r$, where $r \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. The components that depend on the challenge elements X, Y, T are colored for clarity.

2. Algorithm $\mathcal{B}$ starts by sampling $\beta_1, \beta_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and sets $Z \leftarrow e(g_1, X)$, $\beta \leftarrow \beta_1 + \beta_2$, and $h \leftarrow g_1^\beta$.

3. For each slot $i \in [L]$, sample $t_i, \delta_i, \tau_i \xleftarrow{R} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ constructs the (semi-functional) slot components as follows:
$$A_i \leftarrow (g_1 g_3)^{t_i} \quad , \quad B_i \leftarrow X A_i^{\beta} (g_2 g_3)^{\tau_i} \quad , \quad P_i \leftarrow (Y g_3)^{\delta_i}.$$
Then, for each attribute $w \in \mathcal{U}$ and each slot $i \in [L]$, algorithm $\mathcal{B}$ samples $u_{i,w} \xleftarrow{R} \mathbb{Z}_N$. In addition, for each $j \neq i$, it samples $\gamma_{i,j,w} \xleftarrow{R} \mathbb{Z}_N$. It then constructs the attribute components $U_{i,w}$ and $W_{i,j,w}$ as follows:
$$U_{i,w} \leftarrow g_1^{u_{i,w}} \quad , \quad W_{i,j,w} \leftarrow A_i^{u_{j,w}} g_3^{\gamma_{i,j,w}}.$$
Algorithm $\mathcal{B}$ gives the common reference string
$$\mathsf{crs} = \left(\mathcal{G}, Z, g_1, h, g_3, \{(A_i, B_i, P_i)\}_{i \in [L]}, \{U_{i,w}, W_{i,j,w}\}_{i \neq j, w \in \mathcal{U}}\right)$$
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.

4. 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 $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{R} \mathbb{Z}_N$. It then computes $T_i \leftarrow g_1^{r_i}$, $Q_i \leftarrow P_i^{r_i}$, $R_i \leftarrow g_3^{r_i}$, and $V_{j,i} \leftarrow A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ 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 $1 \leq i \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[i]$ and replies to $\mathcal{A}$ with sk'.

5. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy P^*, the messages μ_0^*, μ_1^*, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{ncHyb}_{\ell,1}^{(b)}$ and $\mathsf{ncHyb}_{\ell,2}^{(b)}$:
   - If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i \leftarrow \mathsf{pk}'$. Otherwise, algorithm $\mathcal{B}$ aborts with output 0.
   - If $c_i = \bot$, then algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, it aborts with output 0. Otherwise, it sets $\mathsf{pk}_i \leftarrow \mathsf{pk}_i^*$.

   Finally, for each $i \in [L]$, algorithm $\mathcal{B}$ parses $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$.

6. Algorithm $\mathcal{B}$ parses the challenge policy as $(\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}$. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:
   - **Message-embedding components:** Set $C_1 \leftarrow \mu_b^* \cdot T$ and $C_2 \leftarrow Y$.
   - **Attribute-specific component:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$. For each $k \in [K]$, set
   $$C_{3,k} \leftarrow Y^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - \sum_{i \in [L] : \rho(k) \notin S_i} u_{i,\rho(k)}}.$$
   - **Slot-specific component:** Set
   $$C_4 \leftarrow Y^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right).$$

7. 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 $\gamma_2 \bmod p_2$ plays the role of $s \bmod p_2$:

- First, $Z = e(g_1, X) = e(g_1, g_1^{\alpha} g_2^{\gamma_1}) = e(g_1, g_1)^{\alpha}$.

- Consider the components of the CRS:

$$\begin{aligned} A_i &= (g_1 g_3)^{t_i} \\ B_i &= X A_i^\beta (g_2 g_3)^{\tau_i} = g_1^\alpha g_2^{\gamma_1} A_i^\beta (g_2 g_3)^{\tau_i} = g_1^\alpha A_i^\beta g_2^{\gamma_1 + \tau_i} g_3^{\tau_i} \\ P_i &= (Y g_3)^{\delta_i} = ((g_1^s g_2^{\gamma_2}) g_3)^{\delta_i}, \end{aligned}$$

  where $\tau_i, \delta_i \xleftarrow{\text{R}} \mathbb{Z}_N$. Since $\tau_i \xleftarrow{\text{R}} \mathbb{Z}_N$, the distribution of $\tau_i + \gamma_1$ and τ_i is identical. Thus, we conclude that these components of the CRS are distributed identically to those in $\mathsf{Hyb}_{2,L}^{(b)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$, with $\gamma_2 \bmod p_2$ playing the role of $s \bmod p_2$ in $\mathsf{Hyb}_{2,L}^{(b)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$. The remaining CRS components are sampled as in $\mathsf{Hyb}_{2,L}^{(b)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$.

- 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 $C_2, C_{3,k}, C_4$ are distributed exactly as in $\mathsf{Hyb}_{2,L}^{(b)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$ where $\gamma_2 \bmod p_2$ plays the role of $s \bmod p_2$. Since $s \xleftarrow{\text{R}} \mathbb{Z}_N$ in $\mathsf{Hyb}_{2,L}^{(b)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$, the distribution of $s \bmod p_2$ and $\gamma_2 \bmod p_2$ 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 $\mathsf{Hyb}_{2,L}^{(b)}$.

- If $T = e(g_1, g_2)^r$, where $r \xleftarrow{\text{R}} \mathbb{Z}_N$, 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 5.2c with the same distinguishing advantage as $\mathcal{A}$ and the claim follows. □

By construction, the distribution $\mathsf{Hyb}_{\mathsf{rand}}^{(b)}$ is *independent* of the message μ_b^*. Thus, for all adversaries $\mathcal{A}$, $\mathsf{Hyb}_{\mathsf{rand}}^{(0)}(\mathcal{A}) \equiv \mathsf{Hyb}_{\mathsf{rand}}^{(1)}(\mathcal{A})$. Security now follows by combining Lemmas 5.10, 5.13, 5.14 and 5.33. □

6 From Slotted Registered ABE to Registered ABE

In this section, we show how to generically transform a slotted registered ABE scheme (Definition 4.9) to a standard registered ABE scheme (Definition 4.1). We refer to Section 2 for an overview of thhe construction.

Construction 6.1 (Slotted Registered ABE to Registered ABE)**.** Let λ be a security parameter. Let $\Pi_{\mathsf{sRBE}} = (\mathsf{sRBE.Setup}, \mathsf{sRBE.KeyGen}, \mathsf{sRBE.IsValid}, \mathsf{sRBE.Aggregate}, \mathsf{sRBE.Encrypt}, \mathsf{sRBE.Decrypt})$ be a slotted registered ABE scheme with attribute universe $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$, policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$, and message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda \in \mathbb{N}}$. We now construct a registered ABE scheme $\Pi_{\mathsf{R\text{-}ABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ that supports a bounded number of users and over the same attribute space $\mathcal{U}$, policy space $\mathcal{P}$, and message space $\mathcal{M}$ as follows. In the description, we adopt the following conventions:

- 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.

- The registered ABE scheme will internally maintain $\ell + 1$ slotted ABE schemes, where the k^{th} scheme is a slotted scheme with 2^k slots (for $k \in [0, \ell]$).

- The auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ consists of the following components:
  - A counter ctr that keeps track of the number of registered users in the system.
  - A dictionary D_1 that maps a scheme index $k \in [0, \ell]$ and a slot index $i \in [2^k]$ to a pair (pk, S) which specifies the public key and attribute set currently assigned to slot i of scheme k.
  - A dictionary D_2 that maps a scheme index $k \in [0, \ell]$ and a user index $i \in [L]$ to the helper decryption key associated with scheme k and user i.

   – The current master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$.

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

We construct our registered ABE scheme as follows:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$: On input the security parameter λ, the attribute universe $\mathcal{U}$, and a bound on number of registrants $L = 2^\ell$, the setup algorithm runs the setup algorithm for $\ell + 1$ copies of the slotted RBE scheme. Specifically, for each $k \in [0, \ell]$, it samples $\mathsf{crs}_k \leftarrow \mathsf{sRBE.Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^{2^k})$ and outputs $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{aux})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$ and the auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$, the key-generation algorithm generates a public/secret key-pair for each of the $\ell+1$ underlying schemes. Specifically, for each $k \in [0, \ell]$, let $i_k \leftarrow (\mathsf{ctr} \bmod 2^k) + 1 \in [2^k]$ be a slot index for the k^{th} scheme, and sample a key $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{sRBE.KeyGen}(\mathsf{crs}_k, i_k)$. Output $\mathsf{pk} = (\mathsf{ctr}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$ and $\mathsf{sk} = (\mathsf{ctr}, \mathsf{sk}_0, \ldots, \mathsf{sk}_\ell)$.
- $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S_{\mathsf{pk}})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$, the auxiliary data $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$, where $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$, a public key $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$, and an associated set of attributes S_{pk}, the registration algorithm proceeds as follows:
  – For each $k \in [0, \ell]$, let $i_k = (\mathsf{ctr}_{\mathsf{aux}} \bmod 2^k) + 1 \in [2^k]$ be the slot index for the k^{th} scheme.
  – For each $k \in [0, \ell]$, check that $\mathsf{sRBE.IsValid}(\mathsf{crs}_k, i_k, \mathsf{pk}_k) = 1$. In addition, check that $\mathsf{ctr}_{\mathsf{aux}} = \mathsf{ctr}_{\mathsf{pk}}$. If any check fails, the algorithm halts and outputs the current auxiliary data aux and master public key mpk.
  – Then for each $k \in [0, \ell]$, the registration algorithm updates $\mathsf{D}_1[k, i_k] \leftarrow (\mathsf{pk}, S_{\mathsf{pk}})$. In addition, if $i_k = 2^k$ (i.e., all of the slots in scheme k are filled), the registration algorithm additionally does the following:
    * Compute
$$\left(\mathsf{mpk}'_k, \mathsf{hsk}'_{k,1}, \ldots, \mathsf{hsk}'_{k,2^k}\right) \leftarrow \mathsf{sRBE.Aggregate}\left(\mathsf{crs}_k, \mathsf{D}_1[k, 1], \ldots, \mathsf{D}_1[k, 2^k]\right).$$
    * Update $\mathsf{D}_2[\mathsf{ctr} + 1 - 2^k + i, k] \leftarrow \mathsf{hsk}'_{k,i}$ for each $i \in [2^k]$.
    * If $i_k \neq 2^k$, $\mathsf{mpk}'_k = \mathsf{mpk}_k$ is unchanged.
  – Define the new master public key $\mathsf{mpk}' = (\mathsf{ctr}_{\mathsf{aux}} + 1, \mathsf{mpk}'_0, \ldots, \mathsf{mpk}'_\ell)$.
  – Finally, the registration algorithm outputs the new master public key mpk' and auxiliary data $\mathsf{aux}' = (\mathsf{ctr}_{\mathsf{aux}} + 1, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk}')$.
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu)$: On input the master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$, the access policy $P \in \mathcal{P}$, and a message $\mu \in \mathcal{M}$, the encryption algorithm computes $\mathsf{ct}_k \leftarrow \mathsf{sRBE.Encrypt}(\mathsf{mpk}_k, P, \mu)$ for each $k \in [0, \ell]$; if $\mathsf{mpk}_k = \perp$, then it sets $\mathsf{ct}_k \leftarrow \perp$. Then it outputs $\mathsf{ct} = (\mathsf{ctr}, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$.
- $\mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$, the auxiliary data $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$, and a public key $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$, the update algorithm outputs $\perp$ if $\mathsf{ctr}_{\mathsf{pk}} \geq \mathsf{ctr}_{\mathsf{aux}}$. Otherwise, for each $k \in [0, \ell]$, it sets $\mathsf{hsk}_k \leftarrow \mathsf{D}_2[\mathsf{ctr}_{\mathsf{pk}} + 1, k]$ and replies with $\mathsf{hsk} = (\mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell)$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input a secret key $\mathsf{sk} = (\mathsf{ctr}_{\mathsf{sk}}, \mathsf{sk}_0, \ldots, \mathsf{sk}_\ell)$, a helper key $\mathsf{hsk} = (\mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell)$, and a ciphertext $\mathsf{ct} = (\mathsf{ctr}_{\mathsf{ct}}, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$, the decryption algorithm outputs $\perp$ if $\mathsf{ctr}_{\mathsf{ct}} \leq \mathsf{ctr}_{\mathsf{sk}}$. Otherwise, it computes the largest index k on which ctr and ctr' differ (where bits are 0-indexed starting from the least significant bit). If $\mathsf{hsk}_k = \perp$, then the decryption algorithm outputs GetUpdate. Otherwise, it outputs $\mathsf{sRBE.Decrypt}(\mathsf{sk}_k, \mathsf{hsk}_k, \mathsf{ct}_k)$.

Correctness, compactness, and efficiency. Recall the correctness game from Definition 4.2. We will show that that perfect correctness, compactness, and efficiency of the slotted registered ABE scheme Π_{sRBE} implies perfect correctness, compactness, and efficiency of the registered ABE scheme from Construction 6.1.

Theorem 6.2 (Correctness). *Suppose Π_{sRBE} is complete and perfectly correct. Then Construction 6.1 is perfectly correct.*

Proof. Let $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell) \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$. Consider the challenger's behavior in the correctness game. Let $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ be the auxiliary data maintained by the challenger at some point during the correctness game. Here, $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{mpk}}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$. By design, the counter $\mathsf{ctr}_{\mathsf{mpk}}$ associated with the master public key mpk always coincides with the counter $\mathsf{ctr}_{\mathsf{aux}}$ embedded in aux. Thus, in the following description, we will often write $\mathsf{ctr}_{\mathsf{aux}}$ to denote both counters. Let $(\mathsf{pk}^*, \mathsf{sk}^*)$ be the target key sampled by the challenger in response to a target-key registration query. We start by showing the following invariant:

Lemma 6.3. *Let* $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ *be the auxiliary data (maintained by the challenger) 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* $\mathsf{pk}^* = (\mathsf{ctr}^*, \mathsf{pk}_0^*, \ldots, \mathsf{pk}_\ell^*)$ *be the target key the challenger sampled when responding to the target-key registration query. Let $k' \in [0, \ell]$ be the most significant bit on which the binary representations of* ctr^* *and* $\mathsf{ctr}_{\mathsf{aux}}$ *differ (indexed as in* $\mathsf{Decrypt}$*). Then the master public key* mpk_{k^*} *was the output of a call to* $\mathsf{sRBE.Aggregate}(\mathsf{crs}_{k'}, \cdot)$ *on a tuple of keys and attributes that included the target key* $(\mathsf{pk}_{k'}^*, S^*)$.

Proof. We start by showing the following simple observation on the position of the most significant differing bit between two integers.

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 $y + 1 = 0 \bmod 2^{k_{x,y+1}}$*

Proof. Let $y'_\ell \cdots y'_1 y'_0 = y + 1$ be the binary representation of $y' = y + 1$. We show the two properties individually:

- By construction, since $x < y$, we have $x_{k_{x,y}} = 0$ and $y_{k_{x,y}} = 1$. Assume for contradiction that $k_{x,y} > k_{x,y+1}$. Since $k_{x,y+1}$ is the most significant differing bit between x and $y + 1$, this means that $x_{k_{x,y+1}} = 0$ and $y'_{k_{x,y+1}} = 1$. Moreover, $k_{x,y+1}$ is the most significant differing bit between x and $y + 1$, so if $k_{x,y} > k_{x,y+1}$ and $x_{k_{x,y}} = 0$, then it must be the case that $y'_{k_{x,y}} = 0$. However, we also have that $y_{k_{x,y}} = 1$. Since $y' = y + 1$, the only way $y_{k_{x,y}} = 1$ and $y'_{k_{x,y}} = 0$ is if $y_k = 1$ for all $k \leq k_{x,y}$. This means that $y'_k = 0$ for all $k \leq k_{x,y}$. When $k_{x,y+1} < k_{x,y}$, this now contradicts the previous deduction that $y'_{k_{x,y+1}} = 1$.
- For the second part of the claim, suppose that $k_{x,y} < k_{x,y+1}$. By definition of $k_{x,y+1}$, this means $x_{k_{x,y+1}} = 0$ and $y'_{k_{x,y+1}} = 1$. Since $k_{x,y}$ is the most significant bit on which x and y differ and $k_{x,y} < k_{x,y+1}$, this means that $y_{k_{x,y+1}} = x_{k_{x,y+1}} = 0$. Otherwise $k_{x,y+1}$ is a more significant bit on which x and y differ. Since $y' = y + 1$ and y' and y differ on bit $k_{x,y+1}$, this means that for all $k < k_{x,y+1}$, $y_k = 1$. Correspondingly, for all $k < k_{x,y+1}$, $y'_k = 0$. This means that $y' = y + 1 = 0 \bmod 2^{k_{x,y+1}}$. □

We now prove Lemma 6.3 via induction. The base case corresponds to the state of the challenger immediately after the adversary registers the target key. Let $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ be the auxiliary data at the beginning of the adversary's first target key query. We start by showing the invariant holds immediately following the query:

- On a target-key registration query, algorithm $\mathcal{A}$ sends an attribute set $S^* \subseteq \mathcal{U}_\lambda$ to the challenger.
- The challenger samples $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{aux})$. By construction, for each $k \in [0, \ell]$, the challenger computes $i_k^* \leftarrow (\mathsf{ctr} \bmod 2^k) + 1$ and samples a key $(\mathsf{pk}_k^*, \mathsf{sk}_k^*) \leftarrow \mathsf{sRBE.KeyGen}(\mathsf{crs}_k, i_k^*)$. The public key is then $\mathsf{pk}^* = (\mathsf{ctr}, \mathsf{pk}_0^*, \ldots, \mathsf{pk}_\ell^*)$.
- Next, the challenger runs $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*, S^*)$. By completeness of Π_{sRBE}, we have that for all $k \in [0, \ell]$, $\mathsf{sRBE.IsValid}(\mathsf{crs}_k, i_k^*, \mathsf{pk}_k) = 1$. This means the challenger sets $\mathsf{D}_1[k, i_k^*] \leftarrow (\mathsf{pk}^*, S^*)$.

- For an index $k \in [0, \ell]$, let $\mathsf{ctr}_k \in \{0, 1\}$ denote the k^{th} bit of ctr (starting from 0 for the least significant bit and ℓ for the most significant bit). Let $k' \in [0, \ell]$ be the index of the most significant bit on which ctr and $\mathsf{ctr} + 1$ differ. By construction, this means that $\mathsf{ctr}_{k'} = 0$ and for all $k < k'$, $\mathsf{ctr}_k = 1$. Thus, $\mathsf{ctr} = 2^{k'} - 1 \bmod 2^{k'}$. In this case, $i_{k'} = (\mathsf{ctr} \bmod 2^{k'}) + 1 = 2^{k'}$ and the registration algorithm will compute
$$\left(\mathsf{mpk}'_{k'}, \mathsf{hsk}'_{k',1}, \ldots, \mathsf{hsk}'_{k',2^{k'}}\right) \leftarrow \mathsf{sRBE.Aggregate}\left(\mathsf{crs}_{k'}, \mathsf{D}_1[k', 1], \ldots, \mathsf{D}_1[k', 2^{k'}]\right).$$
By construction, the updated master public key mpk' contains $\mathsf{mpk}'_{k'}$.
- At the end of the target-key registration query, the counter $\mathsf{ctr}_{\mathsf{aux}}$ associated with aux satisfies $\mathsf{ctr}_{\mathsf{aux}} = \mathsf{ctr}^* + 1$. By construction, we have that ctr^* and $\mathsf{ctr}_{\mathsf{aux}} = \mathsf{ctr}^* + 1$ differ on position k'. But by construction, the master public key $\mathsf{mpk}'_{k'}$ was the output to $\mathsf{sRBE.Aggregate}(\mathsf{crs}_{k'}, \mathsf{D}_1[k', 1], \ldots, \mathsf{D}_1[k', 2^{k'}])$, and moreover,
$$\mathsf{D}_1\left[k', i^*_{k'}\right] = \mathsf{D}_1[k', 2^{k'}] = (\mathsf{pk}^*, S^*).$$
Thus, the challenge key pk^* was aggregated into $\mathsf{mpk}'_{k'}$, and the invariant holds.

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

- Let $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ and $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$ 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 aux.
- In a non-target-key registration query, algorithm $\mathcal{A}$ sends a public key $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$ and an associated attribute set S_{pk} to the challenger. The challenger then runs $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S_{\mathsf{pk}})$ and updates $\mathsf{aux} \leftarrow \mathsf{aux}'$, $\mathsf{mpk} \leftarrow \mathsf{mpk}'$. It replies to $\mathcal{A}$ with the updated parameters mpk' and aux'. In the following, we will write $\mathsf{mpk}' = (\mathsf{ctr}_{\mathsf{mpk}'}, \mathsf{mpk}'_0, \ldots, \mathsf{mpk}'_\ell)$.
- Let $i_k = (\mathsf{ctr}_{\mathsf{aux}} \bmod 2^k) + 1$ for each $k \in [0, \ell]$. First, if $\mathsf{ctr}_{\mathsf{pk}} \neq \mathsf{ctr}_{\mathsf{aux}}$ or $\mathsf{sRBE.IsValid}(\mathsf{crs}_k, i_k, \mathsf{pk}_k) = 0$, then the challenger does *not* update aux or mpk (i.e., $\mathsf{aux}' = \mathsf{aux}$ and $\mathsf{mpk}' = \mathsf{mpk}$). Since the invariant holds for aux, it also holds for aux'.
- Consider the case where $\mathsf{ctr}_{\mathsf{pk}} = \mathsf{ctr}_{\mathsf{aux}}$ and moreover, $\mathsf{sRBE.IsValid}(\mathsf{crs}_k, i_k, \mathsf{pk}_k) = 1$ for all $k \in [0, \ell]$. Let $k_{\mathsf{old}} \in [0, \ell]$ be the index of the most significant bit on which ctr^* and $\mathsf{ctr}_{\mathsf{aux}}$ differ, and let $k_{\mathsf{new}} \in [0, \ell]$ be the index of the most significant bit on which ctr^* and $\mathsf{ctr}_{\mathsf{aux}} + 1$ differ. By Claim 6.4, $k_{\mathsf{old}} \leq k_{\mathsf{new}}$. In addition, since $\mathsf{ctr}_{\mathsf{aux}} > \mathsf{ctr}^*$, this means $\mathsf{ctr}_{\mathsf{aux}, k_{\mathsf{old}}} = 1$. We now consider two possibilities:
  - Suppose $k_{\mathsf{old}} = k_{\mathsf{new}}$. This means that the $k_{\mathsf{old}}^{\text{th}}$ bit of ctr and $\mathsf{ctr} + 1$ are the same. Correspondingly, this means that $(\mathsf{ctr} \bmod 2^{k_{\mathsf{old}}}) + 1 < 2^{k_{\mathsf{old}}}$. This means that $\mathsf{mpk}'_{k_{\mathsf{old}}} = \mathsf{mpk}_{k_{\mathsf{old}}}$ is *unchanged* by the registration algorithm. By the inductive hypothesis, the challenge key pk^* was aggregated in $\mathsf{mpk}_{k_{\mathsf{old}}}$. Correspondingly, the challenge key pk^* is aggregated in $\mathsf{mpk}'_{k_{\mathsf{new}}}$, and the invariant holds.
  - Suppose $k_{\mathsf{old}} < k_{\mathsf{new}}$. By Claim 6.4, this means that $\mathsf{ctr}_{\mathsf{aux}} + 1 = 0 \bmod 2^{k_{\mathsf{new}}}$. This means that $i_{k_{\mathsf{new}}} = 2^{k_{\mathsf{new}}}$. In this case, the challenger updates $\mathsf{mpk}'_{k_{\mathsf{new}}} \leftarrow \mathsf{sRBE.Aggregate}(\mathsf{crs}_{k_{\mathsf{new}}}, \mathsf{D}_1[k_{\mathsf{new}}, 1], \ldots, \mathsf{D}_1[k_{\mathsf{new}}, 2^{k_{\mathsf{new}}}])$. By construction of the registration algorithm, the entries $\mathsf{D}_1[k_{\mathsf{new}}, 1], \ldots, \mathsf{D}_1[k_{\mathsf{new}}, 2^{k_{\mathsf{new}}}]$ correspond to (pk_i, S_i) for $i \in [\mathsf{ctr}_{\mathsf{aux}} - 2^{k_{\mathsf{new}}} + 1, \mathsf{ctr}_{\mathsf{aux}}]$, where (pk_i, S_i) denotes the key that was registered in the i^{th} successful invocation of RegPK (in response to either a target-key registration query or a non-target-key registration query). Since the most significant differing bit between $\mathsf{ctr}_{\mathsf{aux}}$ and ctr^* is k_{old}, we have that $\mathsf{ctr}_{\mathsf{aux}} - \mathsf{ctr}^* \leq 2^{k_{\mathsf{old}}+1} - 1 \leq 2^{k_{\mathsf{new}}} - 1$. This means that $\mathsf{ctr}^* \in [\mathsf{ctr}_{\mathsf{aux}} - 2^{k_{\mathsf{new}}} + 1, \mathsf{ctr}_{\mathsf{aux}}]$, and so (pk^*, S^*) is aggregated in $\mathsf{mpk}'_{k_{\mathsf{new}}}$. Finally, since $\mathsf{mpk}' = (\mathsf{ctr}_{\mathsf{aux}} + 1, \mathsf{mpk}'_0, \ldots, \mathsf{mpk}'_\ell)$ where $\mathsf{mpk}'_k = \mathsf{mpk}_k$ for all $k \neq k_{\mathsf{new}}$, the invariant again 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. The claim now follows by induction. □

To complete the proof, we now argue that the output on each of the decryption queries is correct. Let (i_t, m_t, P_t) be the t^{th} encryption query made by the adversary $\mathcal{A}$, and let $\mathsf{ct}_t \leftarrow \mathsf{Encrypt}(\mathsf{mpk}_{i_t}, P_t, m_t)$ be the resulting ciphertext. Consider a decryption query on any index $j \in [t]$. Here, the challenger computes $m'_j \leftarrow \mathsf{Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_j)$:

- By construction, we can write $\mathsf{sk}^* = (\mathsf{ctr}^*, \mathsf{sk}^*_0, \ldots, \mathsf{sk}^*_\ell)$, $\mathsf{ct}_j = (\mathsf{ctr}_{\mathsf{ct}}, \mathsf{ct}_{j,0}, \ldots, \mathsf{ct}_{j,\ell})$, $\mathsf{hsk}^* = (\mathsf{hsk}^*_0, \ldots, \mathsf{hsk}^*_\ell)$, and $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ where $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$.
- Let $k^* \in [0, \ell]$ be the index of the most significant bit on which which $\mathsf{ctr}_{\mathsf{ct}}$ and ctr^* differ.
- If $\mathsf{hsk}^*_{k^*} \neq \bot$, the challenger replies with $m'_j \leftarrow \mathsf{sRBE.Decrypt}(\mathsf{sk}_{k^*}, \mathsf{hsk}_{k^*}, \mathsf{ct}_{j,k^*})$.
- If $\mathsf{hsk}^*_{k^*} = \bot$, the challenger first computes $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$. By construction, this sets $\mathsf{hsk}^*_{k^*} \leftarrow \mathsf{D}_2[k^*, \mathsf{ctr}^* + 1]$. The challenger then replies with $m'_j \leftarrow \mathsf{sRBE.Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_{j,k^*})$.

We now show that $m'_j = m_j$. Recall that ct_j is the output of $\mathsf{Encrypt}(\mathsf{mpk}_{i_t}, P_t, m_t)$ from an encryption query. Let $\mathsf{mpk}_{i_t} = (\mathsf{ctr}_{i_t}, \mathsf{mpk}_{i_t,0}, \ldots, \mathsf{mpk}_{i_t,\ell})$. By definition of the encryption algorithm, it must be the case that $\mathsf{ctr}_{i_t} = \mathsf{ctr}_{\mathsf{ct}}$. The correctness game requires that the i_t^{th} master public key mpk_{i_t} is constructed *after* the target key pk^* is registered, so $\mathsf{ctr}_{\mathsf{ct}} \geq \mathsf{ctr}^*$. By Lemma 6.3, this means that the target key-attribute pair (pk^*, S^*) was aggregated in mpk_{i_t,k^*} via a call to sRBE.Aggregate. By construction of sRBE.Aggregate, this means that $\mathsf{D}_2[\mathsf{ctr}^* + 1, k^*] = \mathsf{hsk}_{k^*,\mathsf{ctr}^*}$. Moreover, by construction of RegPK, the value of $\mathsf{D}_2[\mathsf{ctr}^* + 1, k^*]$ will *never* be updated after the first time it is assigned in a call to RegPK (since the counter $\mathsf{ctr}_{\mathsf{aux}}$ is monotonically increasing). Now, by correctness of Π_{sRBE}, we have that $\mathsf{sRBE.Decrypt}(\mathsf{sk}_{k^*,\mathsf{ctr}^*}, \mathsf{hsk}_{k^*}, \mathsf{ct}_{j,k^*}) = m_t$, and correctness follows. □

Theorem 6.5 (Compactness). *Suppose Π_{sRBE} is a compact slotted registered ABE scheme. Then Construction 6.1 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 at most $L = 2^\ell$ slots, the length of each mpk_i is bounded by $\mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$ by compactness of Π_{sRBE}. Thus, the overall public parameters mpk have size at most $\mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$. □

Theorem 6.6 (Update Efficiency). *Suppose Π_{sRBE} is a compact slotted registered ABE scheme. Then, Construction 6.1 satisfies update efficiency.*

Proof. We consider each requirement separately:

- **Number of updates:** The number of updates is at most $\ell + 1 = \log L$ since each helper decryption key hsk contains at most $\ell + 1$ helper decryption keys $\mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell$, one for each of the underlying schemes. The Update algorithm is only invoked when one of the underlying helper decryption keys hsk_i is $\bot$, and after the update, the key is no longer $\bot$ in hsk.
- **Running time of update:** By construction, the GetUpdate operation simply looks up and updates the $\ell + 1$ helper decryption keys $\mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell$. By compactness of Π_{sRBE}, each helper decryption key hsk_i has size $\mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$. Since the auxiliary data maintains a dictionary D_2 mapping each index slot index k to its set of helper decryption keys, the update operation can be implemented in $\mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$ time in the RAM model of computation. □

Theorem 6.7 (Security). *Suppose Π_{sRBE} is a secure slotted registered ABE scheme. Then Construction 6.1 is secure.*

Proof. Let $\mathcal{A}$ be an adversary for the registered ABE scheme, and let $L = 2^\ell$ be a bound on the number of slots algorithm $\mathcal{A}$ selects. We start by defining a sequence of hybrid experiments, each parameterized by an index $k^* \in [0, \ell]$:

- Hyb_{k^*}: This is the (bounded) registered ABE security game, except when generating the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ctr}_{\mathsf{ct}^*}, \mathsf{ct}^*_0, \ldots, \mathsf{ct}^*_\ell)$, the first k^* ciphertexts $\mathsf{ct}^*_0, \ldots, \mathsf{ct}^*_{k^*-1}$ are encryptions of μ^*_1 while the remaining ciphertexts $\mathsf{ct}^*_{k^*}, \ldots, \mathsf{ct}^*_\ell$ are encryptions of μ^*_0. More specifically, the game proceeds as follows:

- **Setup phase:** At the beginning of the game, algorithm $\mathcal{A}$ chooses a bound 1^L and sends it to the challenger. The challenger then samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^L)$. It then initializes the auxiliary input $\mathsf{aux} \leftarrow \perp$, an initial master public key $\mathsf{mpk} \leftarrow \perp$, a counter $t \leftarrow 0$, an empty set of keys $C = \varnothing$, and an empty dictionary D mapping public keys to registered attribute sets (with default value $\varnothing$). It gives crs to $\mathcal{A}$.
- **Query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
  - **Register corrupt key:** In a corrupted-key-registration query, algorithm $\mathcal{A}$ specifies a public key pk and a set of attributes $S \subseteq \mathcal{U}_\lambda$. The challenger registers the key by computing $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S)$. The challenger updates its copy of the public key $\mathsf{mpk} \leftarrow \mathsf{mpk}'$ and its auxiliary data $\mathsf{aux} \leftarrow \mathsf{aux}'$, adds pk to C, and updates $\mathsf{D}[\mathsf{pk}] \leftarrow \mathsf{D}[\mathsf{pk}] \cup \{S\}$. It replies to $\mathcal{A}$ with $(\mathsf{mpk}', \mathsf{aux}')$.
  - **Register honest key:** In an honest-key-registration query, algorithm $\mathcal{A}$ specifies a set of attributes $S \subseteq \mathcal{U}_\lambda$. The challenger increments the counter $t \leftarrow t+1$ and samples $(\mathsf{pk}_t, \mathsf{sk}_t) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{aux})$ and $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}_t, S)$. The challenger updates its public key $\mathsf{mpk} \leftarrow \mathsf{mpk}'$, its auxiliary data $\mathsf{aux} \leftarrow \mathsf{aux}'$, and $\mathsf{D}[\mathsf{pk}_t] \leftarrow \mathsf{D}[\mathsf{pk}_t] \cup \{S\}$. Finally, it replies to $\mathcal{A}$ with $(t, \mathsf{mpk}', \mathsf{aux}', \mathsf{pk}_t)$.
  - **Corrupt honest key:** In a corrupt-honest-key query, algorithm $\mathcal{A}$ specifies an index $i \in [t]$. Let $(\mathsf{pk}_i, \mathsf{sk}_i)$ be the i^{th} public/secret key-pair the challenger sampled when responding to the i^{th} honest-key-registration query. The challenger adds pk_i to C and replies to $\mathcal{A}$ with sk_i.
- **Challenge phase:** In the challenge phase, algorithm $\mathcal{A}$ chooses two messages $\mu_0^*, \mu_1^* \in \mathcal{M}$ and a challenge policy P^*. The challenger then generates the challenge ciphertext ct^* as follows:
  - Let the current auxiliary data be $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ where $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{mpk}_0, \ldots \mathsf{mpk}_\ell)$.
  - For each $k \in [0, \ell]$, if $\mathsf{mpk}_k = \perp$, then set $\mathsf{ct}_k \leftarrow \perp$. Otherwise, if $k < k^*$, compute $\mathsf{ct}_k \leftarrow \mathsf{sRBE.Encrypt}(\mathsf{mpk}_k, P, \mu_1^*)$, and if $k \geq k^*$, compute $\mathsf{ct}_k \leftarrow \mathsf{sRBE.Encrypt}(\mathsf{mpk}_k, P, \mu_0^*)$.
  - The challenger replies to $\mathcal{A}$ with the ciphertext $\mathsf{ct} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$.
- **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is also the output of the experiment.

Lemma 6.8. *If Π_{sRBE} is secure, then for all efficient adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, all $k^* \in [0, \ell-1]$,* $|\Pr[\mathsf{Hyb}_{k^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{k^*+1}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda)$.

Proof. Suppose $\mathcal{A}$ is an efficient algorithm such that $|\Pr[\mathsf{Hyb}_{k^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{k^*+1}(\mathcal{A}) = 1]| = \varepsilon$ for some non-negligible ε. We use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ for the underlying slotted scheme Π_{sRBE}:

- **Setup phase:** Algorithm $\mathcal{B}$ starts running $\mathcal{A}$, who starts by outputting the number of slots 1^L. Algorithm $\mathcal{B}$ then proceeds as follows:
  - Algorithm $\mathcal{B}$ sends $1^{2^{k^*}}$ to the challenger, who replies with a common reference string crs_{k^*}. Note that by construction, $2^{k^*} \leq L$, so if L is polynomially-bounded, so is 2^{k^*}.[8]
  - Algorithm $\mathcal{B}$ internally initializes the auxiliary input $\mathsf{aux} = \perp$, the master public key $\mathsf{mpk} = \perp$, and an (initially empty) dictionary D to keep track of the secret keys associated with each key-generation query. In addition, algorithm $\mathcal{B}$ maintains two ordered lists $S_{\mathrm{cur}}, S_{\mathrm{new}}$ which will track the public keys and attribute sets aggregated as part of mpk_{k^*}. Initially, $S_{\mathrm{cur}} \leftarrow \perp$ and $S_{\mathrm{new}} = (\perp, \ldots, \perp)$ is an (uninitialized) list of length 2^{k^*}. For an index $i \in [2^{k^*}]$, we write $S_{\mathrm{new}}[i]$ to denote the i^{th} element of S_{new}.
  - Then, for each $k \in [0, \ell] \setminus \{k^*\}$, algorithm $\mathcal{B}$ samples a common reference string $\mathsf{crs}_k \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}|}, 1^{2^k})$.
  - Finally, algorithm $\mathcal{B}$ sets $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$ and gives crs to $\mathcal{A}$.

[8]Note that if the underlying slotted registered ABE scheme has an efficient setup that runs polylogarithmically in the number of slots L, then we can allow the adversary $\mathcal{A}$ to output the number of slots L encoded in *binary* instead. In this case, the reduction algorithm would also output 2^{k^*} in *binary*. In other words, if the underlying slotted scheme supports an arbitrary polynomial number of users, then the transformed scheme also does. We provide additional discussion on this in Remark 6.10.

- **Query phase:** In the query phase, algorithm $\mathcal{B}$ simulates the queries $\mathcal{A}$ makes as follows:
  - **Register corrupt key:** When algorithm $\mathcal{A}$ issues a corrupt-key-generation query on public key pk and attribute set S, let ctr be the current counter associated with aux. Let $i_{k^*} = (\text{ctr} \bmod 2^k) + 1$. Algorithm $\mathcal{B}$ first runs $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{aux}, \mathsf{mpk}, \mathsf{pk}, S)$ and replies to $\mathcal{A}$ with mpk′ and aux′. In addition, if $\mathsf{aux}' \neq \mathsf{aux}$ (i.e., the registration process updated the auxiliary input), then $\mathcal{B}$ updates $S_{\text{new}}[i_{k^*}] \leftarrow (\perp, S, \mathsf{pk})$. Moreover, if $i_{k^*} = 2^{k^*}$, then algorithm $\mathcal{B}$ sets $S_{\text{cur}} \leftarrow S_{\text{new}}$. Finally, algorithm $\mathcal{B}$ updates its local state by assigning $\mathsf{mpk}' \leftarrow \mathsf{mpk}$ and $\mathsf{aux}' \leftarrow \mathsf{aux}$.
  - **Register honest key:** When algorithm $\mathcal{A}$ makes an honest-key-registration query on a set of attributes S, algorithm $\mathcal{B}$ proceeds as follows:
    - Let ctr be the current counter in aux. For each $k \in [0, \ell]$, let $i_k \leftarrow (\text{ctr} \bmod 2^k) + 1$.
    - For each $k \neq k^*$, sample $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{sRBE.KeyGen}(\mathsf{crs}_k, i_k)$.
    - Next, algorithm $\mathcal{B}$ makes a key-generation query on slot i_{k^*} to obtain a public key (t, pk_{k^*}). It sets the public key to $\mathsf{pk} = (\mathsf{ctr}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$ and adds the mapping $t \mapsto (\mathsf{ctr}, \mathsf{sk}_0, \ldots, \mathsf{sk}_{k^*-1}, \mathsf{sk}_{k^*+1}, \ldots, \mathsf{sk}_\ell)$ to the dictionary D. Here, t is the counter on the number of honest-key-generation queries maintained by the challenger (which coincides with the number of honest-key-generation queries made by $\mathcal{A}$).

    Next, algorithm $\mathcal{B}$ runs $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S)$ and updates $\mathsf{mpk} \leftarrow \mathsf{mpk}'$ and $\mathsf{aux} = \mathsf{aux}'$. It replies with $(t, \mathsf{mpk}', \mathsf{aux}', \mathsf{pk})$ to $\mathcal{A}$. In addition, algorithm $\mathcal{B}$ updates $S_{\text{new}}[i_{k^*}] \leftarrow (t, S, \perp)$. Moreover, if $i_{k^*} = 2^{k^*}$, then algorithm $\mathcal{B}$ sets $S_{\text{cur}} \leftarrow S_{\text{new}}$. Finally, algorithm $\mathcal{B}$ updates its local state by assigning $\mathsf{mpk}' \leftarrow \mathsf{mpk}$ and $\mathsf{aux}' \leftarrow \mathsf{aux}$.
  - **Corrupt honest key:** When algorithm $\mathcal{A}$ makes a corruption query on index i, algorithm $\mathcal{B}$ first looks up $(\mathsf{ctr}_{\mathsf{sk}}, \mathsf{sk}_0, \ldots, \mathsf{sk}_{k^*-1}, \mathsf{sk}_{k^*+1}, \ldots, \mathsf{sk}_\ell) \leftarrow \mathsf{D}_1[i]$. Algorithm $\mathcal{B}$ makes a corruption query on index i to obtain a secret key sk_{k^*}. It replies with the secret key $\mathsf{sk} = (\mathsf{ctr}_{\mathsf{sk}}, \mathsf{sk}_0, \ldots, \mathsf{sk}_\ell)$.
- **Challenge phase:** After algorithm $\mathcal{A}$ outputs a pair of messages μ_0^*, μ_1^* along with the challenge policy P^*, algorithm $\mathcal{B}$ constructs the challenge ciphertext ct* as follows. Let $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$ be the current master public key. Algorithm $\mathcal{B}$ gives the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{ct}_0^*, \ldots, \mathsf{ct}_\ell^*)$ to $\mathcal{A}$ where the components ct_k^* are constructed as follows:
  - If $\mathsf{mpk}_k = \perp$, then $\mathsf{ct}_k^* \leftarrow \perp$.
  - If $\mathsf{mpk}_k \neq \perp$ and $k < k^*$, let $\mathsf{ct}_k^* \leftarrow \mathsf{sRBE.Encrypt}(\mathsf{mpk}_k, P^*, \mu_1^*)$.
  - If $\mathsf{mpk}_k \neq \perp$ and $k > k^*$, let $\mathsf{ct}_k^* \leftarrow \mathsf{sRBE.Encrypt}(\mathsf{mpk}_k, P^*, \mu_0^*)$.
  - If $\mathsf{mpk}_k \neq \perp$ and $k = k^*$, algorithm $\mathcal{B}$ makes a challenge query using the components of S_{cur} as the attribute/public-keys for the slots, P^* as the challenge attribute, and μ_0^*, μ_1^* as the pair of challenge messages. Algorithm $\mathcal{B}$ sets $\mathsf{ct}_{k^*}^*$ to be the challenger's response.
- **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$, which $\mathcal{B}$ also outputs.

First, we show that if $\mathcal{A}$ is an admissible adversary, then $\mathcal{B}$ is also admissible. By construction, the set S_{cur} exactly tracks the public keys currently aggregated in mpk_{k^*}. If $\mathsf{mpk}_{k^*} = \perp$, then $\mathcal{B}$ does not make a challenge query, and is admissible by definition. Suppose $\mathsf{mpk}_{k^*} \neq \perp$. In this case, $S_{\text{cur}} \neq \perp$. Consider each component $S_{\text{cur}}[i]$ in the challenge phase:

- Suppose $S_{\text{cur}}[i] = (j, S, \perp)$. By construction, this corresponds to the case where algorithm $\mathcal{A}$ made an honest-key-generation query with attribute S. Since $\mathcal{A}$ is admissible, either S does not satisfy the challenge policy P^*, or alternatively, if S satisfies P^*, then algorithm $\mathcal{A}$ did *not* make a corruption query on index j. Correspondingly, this means that either S does not satisfy P^* or algorithm $\mathcal{B}$ does not make a corruption query on index j. In both cases, this is an admissible input for slot i.
- Suppose $S_{\text{cur}}[i] = (\perp, S, \mathsf{pk})$. By construction, this happens if algorithm $\mathcal{A}$ made a corrupt-key-registration query with public key pk and attribute S. Since $\mathcal{A}$ is admissible, it must be the case S does not satisfy the challenge policy P^*.

Next, algorithm $\mathcal{B}$ perfectly simulates an execution of the registered ABE security game for $\mathcal{A}$. In the case where $\mathsf{ct}^*_{k^*} \leftarrow \mathsf{sRBE.Encrypt}(\mathsf{mpk}_{k^*}, P^*, \mu^*_0)$, algorithm $\mathcal{B}$ simulates an execution of Hyb_{k^*} whereas in the case where $\mathsf{ct}^*_{k^*} \leftarrow \mathsf{sRBE.Encrypt}(\mathsf{mpk}_{k^*}, P^*, \mu^*_1)$, algorithm $\mathcal{B}$ simulates an execution of Hyb_{k^*+1}. Correspondingly, algorithm $\mathcal{B}$ wins the registered ABE security game with the same non-negligible advantage ε. □

To conclude the proof, observe that in Hyb_0, the challenge ciphertext is an encryption of μ^*_0 and corresponds to the registered ABE security game with $b = 0$, while in Hyb_ℓ, the challenge ciphertext is an encryption of μ^*_1 and corresponds to the registered ABE security game with $b = 1$. Since $\ell = \mathsf{poly}(\lambda)$, the claim now follows by Lemma 6.8. □

Registered ABE from pairings. Combining Construction 6.1 with our slotted registered ABE scheme (Construction 5.4) from Section 5, we now obtain the following corollary:

Corollary 6.9 (Bounded Registered ABE from Pairings). *Let λ be a security parameter. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ be any (polynomial-size) attribute space, and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of policies that can be described by a one-use linear secret sharing scheme over $\mathcal{U}$. Then, under Assumption 5.2, for every polynomial $L = L(\lambda)$, there exists a bounded registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and supporting up to L users with the following properties:*

- *The size of the CRS and the size of the auxiliary data maintained by the key curator is $L^2 \cdot \mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$.*
- *The running time of key-generation and registration is $L \cdot \mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$.*
- *The size of the master public key and the helper decryption keys are both $|\mathcal{U}| \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The size of a ciphertext is $K \cdot \mathsf{poly}(\lambda, \log L)$, where K denotes the number of rows in the linear secret sharing matrix $\mathbf{M}$ associated with the access policy.*

Remark 6.10 (Efficiency Preserving). Our transformation in Construction 6.1 preserves the efficiency of the underlying slotted registered ABE scheme with respect to the following properties:

- **Large universe:** If the underlying slotted registered ABE scheme supports a large universe (i.e., $|\mathcal{U}| = 2^{\omega(\log \lambda)}$), then the transformed scheme also supports a large universe. As discussed in Remark 4.8, we would formally model this by having the Setup algorithm take as input the *bit-length* of the attributes rather than the size of the attribute space in both the slotted scheme and the full scheme. Our obfuscation-based construction in Section 7 (Construction 7.4) supports a large universe.

- **Arbitrary number of users:** If the running time of Setup in the underlying slotted scheme is *polylogarithmic* in the bound on the number of users L, then the running time of Setup in the transformed scheme is also polylogarithmic in the number of users L. Note that if Setup runs in time that is polylogarithmic in L, the size of the CRS must also be polylogarithmic in L. In this case, we can set $L = 2^\lambda$ to support an arbitrary polynomial number of users. Formally, we would model this setting by having Setup take the bound L in *binary* rather than unary in both the slotted scheme and the full registered ABE scheme. While our pairing-based construction (Construction 5.4) does not support this notion, our obfuscation-based construction (Construction 7.4) does.

7 Registered ABE from Indistinguishability Obfuscation

In this section, we show how to build a registered ABE scheme that does *not* impose an a priori bound on the number of users in the system (in contrast to the pairing-based construction from Section 5 (Corollaries 5.5 and 6.9)) using indistinguishability obfuscation ($i\mathcal{O}$) [BGI+12, GGH+13], a somewhere statistically binding (SSB) hash function [HW15] and a pseudorandom generator (PRG). Our approach is similar to but generalizes the RBE construction of Garg et al. [GHMR18] which uses $i\mathcal{O}$, SSB hash functions and public-key encryption.

7.1 Construction Building Blocks

Our construction uses three main building blocks: an indistinguishability obfuscation scheme, a pseudorandom generator, and an SSB hash function [HW15]. We specifically require SSB hash functions that satisfy a *local opening* property; which can be built from standard number-theoretic assumption including DDH, ϕ-Hiding, DCR, and LWE [HW15, OPWW15]. We review each of these notions below:

Definition 7.1 (Indistinguishability Obfuscation [BGI⁺12, GGH⁺13])**.** Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ 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 $C \in \mathcal{C}_\lambda$ 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 $C \in \mathcal{C}_\lambda$, 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 (Pseudorandom Generator)**.** Let λ be a security parameter, $n = n(\lambda)$ be a seed length, and $\ell = \ell(\lambda)$ be an output length. A pseudorandom generator $\mathsf{PRG} \colon \{0,1\}^n \to \{0,1\}^\ell$ is an efficiently-computable function such that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\left| \Pr\left[\mathcal{A}(\mathsf{PRG}(s)) = 1 : s \xleftarrow{\mathrm{R}} \{0,1\}^n\right] - \Pr\left[\mathcal{A}(r) = 1 : r \xleftarrow{\mathrm{R}} \{0,1\}^\ell\right] \right| = \mathsf{negl}(\lambda).$$

Definition 7.3 (Somewhere Statistically Binding Hash Function [HW15, OPWW15])**.** Let λ be a security parameter. A somewhere statistically binding (SSB) hash function with block length $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, output length $\ell_{\mathsf{hash}} = \ell_{\mathsf{hash}}(\lambda)$, and opening length $\ell_{\mathsf{open}} = \ell_{\mathsf{open}}(\lambda)$ is a tuple of efficient algorithms $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i^*) \to \mathsf{hk}$: On input the security parameter λ, the block size ℓ_{blk}, the message length $N \le 2^\lambda$, and an index $i^* \in [N]$, the setup algorithm outputs a hash key hk. Both N and i^* are encoded in *binary*; in particular, this means that $|\mathsf{hk}| = \mathrm{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$. We let $\Sigma = \{0,1\}^{\ell_{\mathsf{blk}}}$ denote the block alphabet.
- $\mathsf{Hash}(\mathsf{hk}, \mathbf{x}) \to h$: On input the hash key hk and a message $\mathbf{x} \in \Sigma^N$, 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 $\mathbf{x} \in \Sigma^N$, and an index $i \in [L]$, the open algorithm outputs an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$.
- $\mathsf{Verify}(\mathsf{hk}, h, i, x_i, \pi_i) \to \{0,1\}$: On input the hash key hk, a hash value $h \in \{0,1\}^{\ell_{\mathsf{hash}}}$, an index $i \in [N]$, a value $x_i \in \Sigma$, and an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$, 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 $N \le 2^\lambda$, all indices $i, i^* \in [N]$, and any $\mathbf{x} \in \Sigma^N$,

$$\Pr\left[\mathsf{Verify}(\mathsf{hk}, h, i, x_i, \pi_i) = 1 : \begin{array}{c} \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i^*); \\ h \leftarrow \mathsf{Hash}(\mathsf{hk}, \mathbf{x}); \pi_i \leftarrow \mathsf{Open}(\mathsf{hk}, \mathbf{x}, i) \end{array}\right] = 1.$$

- **Index hiding:** For a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, define the index hiding game $\mathsf{ExptIH}_{\mathcal{A}}(\lambda, b)$ as follows:

1. Algorithm $\mathcal{A}(1^\lambda)$ chooses an integer N and two indices $i_0, i_1 \in [N]$.
2. The challenger samples $\mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i_b)$ and gives hk 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 polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ and all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

- **Somewhere statistically binding:** We say that a hash key hk is statistically binding for an index $i^* \in [N]$ if there does not exist $h \in \{0,1\}^{\ell_{\mathsf{hash}}}$, $x \neq x' \in \Sigma$, and π, π' where $\mathsf{Verify}(\mathsf{hk}, h, i^*, x, \pi) = 1 = \mathsf{Verify}(\mathsf{hk}, h, i^*, x', \pi')$. We require that for all polynomials $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$ and all $N \leq 2^\lambda$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$ and all $i \in [N]$,

$$\Pr[\mathsf{hk} \text{ is statistically binding for index } i : \mathsf{hk} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i)] = 1 - \mathsf{negl}(\lambda).$$

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

7.2 Registered ABE from Indistinguishability Obfuscation

Our construction follows the slotted RBE specification from Definition 4.9. It can be compiled into a standard RBE scheme using the generic transformation in Section 6 (Construction 6.1). Our slotted construction here supports a large universe (Remark 4.8) and moreover, the running time of the Setup algorithm is polylogarithmic in the number of users L. Formally, the Setup algorithm in the following slotted construction takes only the *bit-length* of the attributes and the bit-length of L as input. As such, when we apply Construction 6.1 to the construction, we obtain a large-universe registered ABE for general circuit policies that supports an arbitrary polynomial of users (Remark 6.10). Our construction below supports arbitrary (possibly non-monotone) policies that can be computed by a Boolean circuit. Following the convention for circuit ABE [GVW13, BGG+14], we model the attributes as an arbitrary bit-string of length $\ell_c = \ell_c(\lambda)$ and the policy as a Boolean circuit $C \colon \{0,1\}^{\ell_c} \to \{0,1\}$. In particular, instead of associating users with a *set* of attributes $S \subseteq \mathcal{U} = \{0,1\}^{\ell_c}$, in the circuit-based setting below, we associate each user with a bit-string $S \in \mathcal{U} = \{0,1\}^{\ell_c}$.

Construction 7.4 (Slotted Registered ABE from iO)**.** Let λ be a security parameter. Let $\mathsf{PRG} \colon \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$ be a length-doubling pseudorandom function. Let $\ell_c = \ell_c(\lambda)$ be the attribute length and let $\mathcal{U} = \{0,1\}^{\ell_c}$ be the attribute space. Let $\mathcal{P} = \{\mathcal{P}_\lambda\}$ be a family of Boolean circuits on inputs of length ℓ_c. Let $\Pi_{\mathsf{SSB}} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$ be a somewhere statistically binding hash function. We construct a slotted registered attribute-based encryption scheme $\Pi_{\mathsf{sRBE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with message space $\mathcal{M} = \{0,1\}^\lambda$, attribute space $\mathcal{U}$, and policy space $\mathcal{P}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^{\ell_c}, L)$: On input the security parameter λ, the bit-length ℓ_c of the attributes, and the number of users L (in *binary*), the setup algorithm sets $\ell_{\mathsf{blk}} = 2\lambda + \ell_c$ and samples a hash key $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$. It outputs $\mathsf{crs} \leftarrow \mathsf{hk}$.

- $\mathsf{KeyGen}(\mathsf{crs}, i)$: On input the common reference string $\mathsf{crs} = \mathsf{hk}$, the key-generation algorithm samples a random seed $s \leftarrow \{0,1\}^\lambda$. It outputs the public key $\mathsf{pk} = \mathsf{PRG}(s)$ and the secret key $\mathsf{sk} = s$.

- $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$: On input the common reference string crs, an index i, and a public key pk, the validation algorithm outputs 1 if $\mathsf{pk} \in \{0,1\}^{2\lambda}$.

- $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1) \ldots, (\mathsf{pk}_L, S_L))$: On input the common reference string $\mathsf{crs} = \mathsf{hk}$ and a collection of public keys pk_i along with their associated attributes $S_i \in \{0,1\}^{\ell_c}$, the aggregation algorithm computes the master public key

$$\mathsf{mpk} \leftarrow \big(\mathsf{hk}, \mathsf{SSB.Hash}\big(\mathsf{hk}, \big((\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)\big)\big)\big).$$

Here we treat each pair (pk_i, S_i) as a binary string of length $\{0,1\}^{2\lambda+\ell_c}$, which is the length of an SSB hash block. Then, for each user $i \in [L]$, the aggregate algorithm computes

$$\pi_i \leftarrow \mathsf{SSB.Open}\big(\mathsf{hk}, \big((\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)\big), i\big),$$

which is the local opening of the SSB hash for index i, and sets the helper secret key to $\mathsf{hsk}_i \leftarrow (i, \mathsf{pk}_i, S_i, \pi_i)$. Finally, it outputs mpk and hsk_i for all $i \in [L]$.

- $\mathsf{Encrypt}(\mathsf{mpk}, C, \mu)$: On input the master public key $\mathsf{mpk} = (\mathsf{hk}, h)$, the ciphertext policy $C \in \mathcal{P}$ and a message $\mu \in \{0,1\}^\lambda$, the encryption algorithm sets $j = 0$ and defines the following program:

Constants: $\mathsf{mpk} = (\mathsf{hk}, h)$, Boolean circuit $C \colon \{0,1\}^{\ell_c} \to \{0,1\}$, message $\mu \in \{0,1\}^\lambda$, index $j \in [0, L+1]$
Inputs: index $i \in [L]$, public key $\mathsf{pk}_i \in \{0,1\}^{2\lambda}$, attribute $S_i \in \{0,1\}^{\ell_c}$, opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$, and secret key $\mathsf{sk}_i \in \{0,1\}^\lambda$.

1. If $\mathsf{SSB.Verify}(\mathsf{hk}, h, i, (\mathsf{pk}_i, S_i), \pi_i) = 1$ and $C(S_i) = 1$ and $\mathsf{pk}_i = \mathsf{PRG}(\mathsf{sk}_i)$ and $i > j$, output μ.
2. Otherwise, output $\perp$.

Figure 1: Program $\mathsf{Embed}[\mathsf{mpk}, C, \mu, j]$.

Here we assume that the circuit $\mathsf{Embed}[\mathsf{mpk}, C, \mu, j]$ is padded to the maximum size of any program appearing in the proof of Theorem 7.6. The encryption algorithm then computes the obfuscated program $C' \leftarrow i\mathcal{O}(\mathsf{Embed}[\mathsf{mpk}, C, \mu, j])$ and outputs $\mathsf{ct} = C'$.

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input the the secret key sk, the helper secret key $\mathsf{hsk} = (i, \mathsf{pk}_i, S_i, \pi_i)$, and a ciphertext $\mathsf{ct} = C'$, the decryption algorithm outputs $C'(i, \mathsf{pk}_i, S_i, \pi_i, \mathsf{sk})$.

Theorem 7.5 (Completeness, Correctness, and Compactness). *Suppose $i\mathcal{O}$ is functionality-preserving and Π_{SSB} is correct and succinct. Then, Construction 7.4 is complete, correct and compact.*

Proof. We consider each property separately:

- **Completeness:** Completeness holds since RBE.IsValid always outputs 1 on all public keys $\mathsf{pk} \in \{0,1\}^{2\lambda}$, and by construction, the public keys output by KeyGen are 2λ-bit strings.
- **Correctness:** For correctness, take any security parameter $\lambda \in \mathbb{N}$, any attribute length $\ell_c = \ell_c(\lambda)$, any number of slots $L \in \mathbb{N}$, and index $i \in [L]$. Consider the following components in the correctness experiment:
  - Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{\ell_c}, L)$. In this case $\mathsf{crs} = \mathsf{hk}$, where $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$.
  - Let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. Then $\mathsf{sk}_i = s \in \{0,1\}^\lambda$ and $\mathsf{pk}_i = \mathsf{PRG}(s)$.
  - Take any set of public keys $\{\mathsf{pk}_j\}_{j \neq i}$ where each $\mathsf{pk}_j \in \{0,1\}^{2\lambda}$. For each $j \in [L]$, let $S_j \in \{0,1\}^{\ell_c}$ be the attribute associated with pk_j.
  - Let $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}\big(\mathsf{crs}, \big((\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)\big)\big)$. By construction, $\mathsf{mpk} = (\mathsf{hk}, h)$ and $\mathsf{hsk}_i = (i, \mathsf{pk}_i, S_i, \pi_i)$ where
  $$h = \mathsf{SSB.Hash}(\mathsf{hk}, ((\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)))$$
  $$\pi_i = \mathsf{SSB.Open}(\mathsf{hk}, ((\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)), i).$$
  - Take any Boolean circuit $C \colon \{0,1\}^{\ell_c} \to \{0,1\}$ where $C(S_i) = 1$ and message $\mu \in \{0,1\}$. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, C, \mu)$, and consider $\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{hsk}_i, \mathsf{ct})$. By construction $\mathsf{ct} = C'$ where C' is an obfuscation of the program $\mathsf{Embed}[\mathsf{mpk}, C, \mu, 0]$. By correction of the obfuscation scheme, $C'(i, \mathsf{pk}_i, S_i, \pi_i, \mathsf{sk}) = \mathsf{Embed}[\mathsf{mpk}, C, \mu, 0](i, \mathsf{pk}_i, S_i, \pi_i, \mathsf{sk})$. We consider each of the conditions:
    * By correctness of the SSB scheme, $\mathsf{SSB.Verify}(\mathsf{hk}, h, (\mathsf{pk}_i, S_i), \pi_i) = 1$.
    * By assumption, $C(S_i) = 1$. By construction, $\mathsf{pk}_i = \mathsf{PRG}(s) = \mathsf{PRG}(\mathsf{sk}_i)$.

    * Since $j = 0$ and $i \in [L]$, $i > j$.

  By construction, Embed outputs μ, and decryption succeeds.

- **Compactness:** Consider the length of the master public key mpk and helper decryption keys hsk_i output by RBE.Aggregate. The mpk consists of the SSB hash key hk and the SSB hash h. Since SSB.Setup is an efficient algorithm it must be the case that $|\mathsf{hk}| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log L)$. Moreover, by succinctness of the SSB hash function, the size of the SSB hash h is $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$. Finally $\ell_{\mathsf{blk}} = 2\lambda + \ell_c = 2\lambda + \log|\mathcal{U}|$. Correspondingly, $|\mathsf{mpk}| = \mathsf{poly}(\lambda, \log|\mathcal{U}|, \log L)$. Next, $\mathsf{hsk}_i = (i, \mathsf{pk}_i, S_i, \pi_i)$. Again by succinctness of the SSB hash function, $|\pi_i| = \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}) = \mathsf{poly}(\lambda, \log|\mathcal{U}|)$. Thus, $|\mathsf{hsk}_i| = \mathsf{poly}(\lambda, \log|\mathcal{U}|, \log L)$, as required. □

Theorem 7.6 (Security). *If iO is secure and Π_{SSB} is correct and secure, then Construction 7.4 is secure.*

Proof. We prove this theorem via a sequence of games (parameterized by a bit $b \in \{0,1\}$ and an index $i \in [0, L+1]$):

- $\mathsf{Hyb}_0^{(b)}$: This is the real security experiment where the challenger encrypts message μ_b^*. We recall the main steps here:
  - **Setup phase:** In the setup phase, the adversary $\mathcal{A}$ chooses a slot count L who then samples $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$, where $\ell_{\mathsf{blk}} = 2\lambda + \ell_c$. The challenger gives $\mathsf{crs} = \mathsf{hk}$ to $\mathcal{A}$. The challenger also initializes a counter $\mathsf{ctr} \leftarrow 0$ and an (empty) dictionary D to keep track of the key-generation queries.
  - **Query phase:** In the query phase, the challenger responds to queries as follows:
    * **Key-generation query:** When $\mathcal{A}$ makes a key-generation query on a slot i, the challenger increments its counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and samples a seed $s \xleftarrow{\mathrm{R}} \{0,1\}^\lambda$. It responds with the counter value ctr and the public key $\mathsf{pk} = \mathsf{PRG}(s)$. The challenger adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}, s)$ to D.
    * **Corruption query:** Whenever the adversary makes a corruption query on an index $c \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the tuple $(i, \mathsf{pk}, s) \leftarrow \mathsf{D}[c]$ and replies to $\mathcal{A}$ with s.
  - **Challenge phase:** In the challenge phase, the adversary $\mathcal{A}$ outputs a tuple $(c_i, S_i, \mathsf{pk}_i^*)$ for each slot $i \in [L]$, a challenge policy $P^* = C \colon \{0,1\}^{\ell_c} \to \{0,1\}$ and two messages $\mu_0^*, \mu_1^* \in \{0,1\}^\lambda$. The challenger now proceeds as follows:
    * If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') \leftarrow \mathsf{D}[c_i]$. If $i \neq i'$, then the challenger aborts. Otherwise, the challenger sets $\mathsf{pk}_i \leftarrow \mathsf{pk}'$.
    * If $c_i = \bot$, then the challenger checks that $\mathsf{pk}_i^* \in \{0,1\}^{2\lambda}$. If not, the challenger aborts. Otherwise, the challenger sets $\mathsf{pk}_i \leftarrow \mathsf{pk}_i^*$.

    The challenger then sets $j = 0$ and constructs the obfuscated program $C' \leftarrow i\mathcal{O}(\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, j])$, where Embed is the program in Fig. 1. The challenger gives $\mathsf{ct} = C'$ to the adversary $\mathcal{A}$.
  - **Output phase:** At the end of the game, the adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

- $\mathsf{Hyb}_i^{(b)}$: Same as $\mathsf{Hyb}_{i-1}^{(b)}$ except when constructing the challenge ciphertext, the challenger sets $j = i$ and computes $C' \leftarrow i\mathcal{O}(\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, j])$ as before.

- $\mathsf{Hyb}_{L+1}^{(b)}$: Same as $\mathsf{Hyb}_L^{(b)}$, except when constructing the challenge ciphertext, the challenger computes $C' \leftarrow i\mathcal{O}(C_{\mathsf{reject}})$, where C_{reject} is the program that takes an index $i \in [L]$, a public key $\mathsf{pk}_i \in \{0,1\}^{2\lambda}$, an attribute $S_i \in \{0,1\}^{\ell_c}$, an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$, and a secret key $\mathsf{sk}_i \in \{0,1\}^\lambda$, and outputs $\bot$. We assume that the circuit C_{reject} is padded to the maximum size of any program appearing in the proof of Theorem 7.6.

For an adversary $\mathcal{A}$, we write $\mathsf{Hyb}_i^{(b)}(\mathcal{A})$ to denote the output of game $\mathsf{Hyb}_i^{(b)}$ with adversary $\mathcal{A}$. We now show that the outputs of each adjacent pair of hybrid experiments are computationally indistinguishable.

Lemma 7.7. *If iO is secure,* PRG *is secure, and Π_{SSB} satisfies index hiding and somewhere statistical binding, then for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$, and $i \in [0, L-1]$,*

$$|\Pr[\mathsf{Hyb}_i^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i+1}^{(b)}(\mathcal{A}) = 1]| = \mathsf{negl}(\lambda).$$

Proof. We begin by introducing an intermediate experiment:

- $\mathsf{iHyb}_i^{(b)}$: Same as $\mathsf{Hyb}_i^{(b)}$, except during the setup phase, the challenger samples $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i+1)$ (i.e., the hash key binds on index $i+1$ instead of index 1).

We now show that $\mathsf{Hyb}_i^{(b)}$ and $\mathsf{iHyb}_i^{(b)}$ are computationally indistinguishable (Claim 7.8) and that $\mathsf{iHyb}_i^{(b)}$ and $\mathsf{iHyb}_{i+1}^{(b)}$ are computationally indistinguishable (Claim 7.9) for all $b \in \{0,1\}$ and $i \in [0, L-1]$.

Claim 7.8. *If Π_{SSB} satisfies index hiding, then for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$, and $i \in [0, L-1]$,*

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

Proof. The only difference between $\mathsf{Hyb}_i^{(b)}$ and $\mathsf{iHyb}_i^{(b)}$ is the hash key hk is binding on index 0 in $\mathsf{Hyb}_i^{(b)}$ and it is binding on index $i+1$ in $\mathsf{iHyb}_i^{(b)}$. The claim thus holds by index hiding of Π_{SSB}. Formally, suppose there exists an efficient and admissible adversary $\mathcal{A}$ such that $\left|\Pr\left[\mathsf{Hyb}_i^{(b)}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1\right]\right| = \varepsilon$. for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks index hiding of Π_{SSB}:

1. Algorithm $\mathcal{B}$ starts by running algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs the number of slots $L \in \mathbb{N}$.
2. Algorithm $\mathcal{B}$ outputs the message length L and $(1, i+1)$ as its challenge indices. 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 $\mathsf{Hyb}_i^{(b)}$ and $\mathsf{Hyb}_{i,1}^{(b)}$ 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 $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_i^{(b)}$ for $\mathcal{A}$. Likewise, if $\mathsf{hk} \leftarrow \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i+1)$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{iHyb}_i^{(b)}$ for $\mathcal{A}$. Thus, algorithm $\mathcal{B}$ succeeds with the same advantage ε, and the claim follows. □

Claim 7.9. *If $i\mathcal{O}$ is secure,* PRG *is secure, and Π_{SSB} is somewhere statistically binding, then for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$, and $i \in [0, L-1]$,*

$$\left|\Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1\right]\right| = \mathsf{negl}(\lambda). \tag{7.1}$$

Proof. As in Lemma 5.16, our analysis will depend on whether the adversary knows the secret key associated with slot $i+1$ or not. Let $(c_{i+1}, S_{i+1}, \mathsf{pk}_{i+1}^*)$ be the tuple the adversary provided for slot $i+1$ in the challenge phase. We say that event $\mathsf{NonCorrupt}$ occurs if both of the following conditions hold:

- The index c_{i+1} satisfies $c_{i+1} \in \{1, \ldots, \mathsf{ctr}\}$. This means that pk_{i+1} was generated by the challenger on the c_{i+1}^{th} key-generation query.
- Adversary $\mathcal{A}$ does *not* make a corruption query on index c_{i+1}.

We write $\overline{\mathsf{NonCorrupt}}$ to denote the complement of event $\mathsf{NonCorrupt}$. Now, we can write

$$\begin{aligned}
\Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1\right] &= \Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] + \Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] \\
\Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1\right] &= \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] + \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right]
\end{aligned}$$

It suffices then to show that

$$\left|\Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] - \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right]\right| = \mathsf{negl}(\lambda) \tag{7.2}$$

$$\left|\Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] - \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right]\right| = \mathsf{negl}(\lambda). \tag{7.3}$$

Eq. (7.1) then follows by the triangle inequality. We now show Eqs. (7.2) and (7.3) in Claims 7.10 and 7.16, respectively.

Analysis for the case where slot $i+1$ is not corrupted. We now show that Eq. (7.2) holds. Our analysis relies on the secret key s_{i+1} associated with slot $i+1$ being hidden (and uniform) from the view of the adversary. We state the precise claim below:

Claim 7.10. *Suppose $i\mathcal{O}$ is secure, PRG is secure, and Π_{SSB} is somewhere statistically binding. Then, for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$, and $i \in [0, L-1]$, we have that*

$$\left|\Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \land \mathsf{NonCorrupt}\right] - \Pr\left[\mathsf{iHyb}_{i+1}^{(b)} = 1 \land \mathsf{NonCorrupt}\right]\right| = \mathsf{negl}(\lambda).$$

Proof. We proceed via a sequence of games:

- $\mathsf{ncHyb}_{i,1}^{(b)}$: Same as $\mathsf{iHyb}_i^{(b)}$ except at the beginning of the game, the challenger samples $k \xleftarrow{\text{R}} [K]$, where $K = K(\lambda)$ is a bound on the the number of key-generation queries algorithm $\mathcal{A}$ makes during the query phase. Let pk_k be the public key the challenger samples in response to the k^{th} key-generation query (if there is one). The challenger now aborts with output 0 if either of the following events occurs:
  - In the challenge phase, the tuple $(c_{i+1}, S_{i+1}, \mathsf{pk}_{i+1}^*)$ the adversary provides for slot $i+1$ satisfies $c_{i+1} \neq k$.
  - The adversary makes a corruption query on index k.

  Otherwise, the experiment proceeds exactly as in $\mathsf{iHyb}_i^{(b)}$.
- $\mathsf{ncHyb}_{i,2}^{(b)}$: Same as $\mathsf{ncHyb}_{i,1}^{(b)}$ except on the k^{th} key-generation query, the challenger samples $\mathsf{pk}_k \xleftarrow{\text{R}} \{0,1\}^{2\lambda}$. Note that in this experiment, the challenger does *not* need to answer a corrupt query on index k (since the challenger immediately aborts if the adversary were to make such a query).
- $\mathsf{ncHyb}_{i,3}^{(b)}$: Same as $\mathsf{ncHyb}_{i,2}^{(b)}$, except during the challenge phase, the challenger sets $j = i+1$ instead of $j = i$ when constructing the challenge ciphertext.
- $\mathsf{ncHyb}_{i,4}^{(b)}$: Same as $\mathsf{ncHyb}_{i,3}^{(b)}$ except on the k^{th} key-generation query, the challenger reverts to sampling $s_k \xleftarrow{\text{R}} \{0,1\}^{\lambda}$ and sets $\mathsf{pk}_k \leftarrow \mathsf{PRG}(s_k)$.

As usual, for an adversary $\mathcal{A}$, we write $\mathsf{ncHyb}^{(b)}(\mathcal{A})$ to denote the output of an execution of $\mathsf{ncHyb}^{(b)}(\mathcal{A})$ with adversary $\mathcal{A}$. We now show that the outputs of each adjacent pair of hybrid experiments are computationally indistinguishable.

Lemma 7.11. *For all admissible adversaries $\mathcal{A}$ and all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$, and $i \in [0, L-1]$,*

$$\Pr\left[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \land \mathsf{NonCorrupt}\right] = K \cdot \Pr\left[\mathsf{ncHyb}_{i,1}^{(b)}(\mathcal{A}) = 1\right].$$

Proof. By construction, $\mathsf{iHyb}_i^{(b)}$ and $\mathsf{ncHyb}_{i,1}^{(b)}$ are identical experiments except for the additional abort condition in $\mathsf{ncHyb}_{i,1}^{(b)}$. If $\mathsf{ncHyb}_{i,1}^{(b)}$ outputs 1, then it must be the case that the output in $\mathsf{iHyb}_i^{(b)}$ is also 1, and moreover, $c_{i+1} = k$, and the adversary does *not* make a corruption query on index $k = c_{i+1}$. By definition, this means event $\mathsf{NonCorrupt}$ must also occur. Thus, we can write

$$\begin{aligned}\Pr[\mathsf{ncHyb}_{i,1}^{(b)}(\mathcal{A}) = 1] &= \Pr[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \land \mathsf{NonCorrupt} \land k = c_{i+1}] \\ &= \Pr[k = c_{i+1} \mid \mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \land \mathsf{NonCorrupt}] \cdot \Pr[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \land \mathsf{NonCorrupt}] \\ &= 1/K \cdot \Pr[\mathsf{iHyb}_i^{(b)}(\mathcal{A}) = 1 \land \mathsf{NonCorrupt}],\end{aligned}$$

since if event $\mathsf{NonCorrupt}$ occurs, then $c_{i+1} \in \{1, \ldots, \mathsf{ctr}\} \subseteq [K]$, and the challenger in $\mathsf{iHyb}_i^{(b)}$ samples $k \xleftarrow{\text{R}} [K]$. □

Lemma 7.12. *If PRG is secure, then for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$, and $i \in [0, L-1]$,*

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

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ such that

$$\left| \Pr\left[\mathsf{ncHyb}_{i,1}^{(b)}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{ncHyb}_{i,2}^{(b)}(\mathcal{A}) = 1\right] \right| \geq \varepsilon,$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks security of PRG:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a PRG challenge $t \in \{0,1\}^{2\lambda}$.
2. Algorithm $\mathcal{B}$ now starts to simulate an execution of $\mathsf{ncHyb}_{i,1}^{(b)}$ and $\mathsf{ncHyb}_{i,2}^{(b)}$ for $\mathcal{A}$. In particular, at the beginning of the game, algorithm $\mathcal{B}$ samples an index $k \xleftarrow{\text{R}} [K]$. It then simulates the setup phase and the query phase exactly as prescribed in $\mathsf{ncHyb}_{i,1}^{(b)}$ and $\mathsf{ncHyb}_{i,2}^{(b)}$. On the k^{th} key-generation query, algorithm $\mathcal{B}$ responds with $\mathsf{pk}_k = t$. If algorithm $\mathcal{A}$ ever makes a corruption query on index k, algorithm $\mathcal{B}$ aborts with output 0 as in $\mathsf{ncHyb}_{i,1}^{(b)}$ and $\mathsf{ncHyb}_{i,2}^{(b)}$.
3. When $\mathcal{A}$ enters the challenge phase with output $(c_i, S_i, \mathsf{pk}_i^*)$ for each $i \in [L]$, algorithm $\mathcal{B}$ aborts with output 0 if $c_{i+1} \neq k$. Otherwise, algorithm constructs $\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$, as well as the challenge ciphertext ct exactly according to the specification of $\mathsf{ncHyb}_{i,1}^{(b)}$ and $\mathsf{ncHyb}_{i,2}^{(b)}$.
4. At the end of the game, algorithm $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

By construction, algorithm $\mathcal{B}$ does not need to know the PRG seed $s \in \{0,1\}^{\lambda}$ (the only quantity in the game that would depend on s is the response to a corruption query on index k, but if $\mathcal{A}$ makes such a query, the output in both experiments is 0). By construction, if $t = G(s)$ where $s \xleftarrow{\text{R}} \{0,1\}^{\lambda}$, then algorithm $\mathcal{B}$ perfectly simulates $\mathsf{ncHyb}_{i,0}^{(b)}$ whereas if $t \xleftarrow{\text{R}} \{0,1\}^{2\lambda}$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{ncHyb}_{i,1}^{(b)}$ for $\mathcal{A}$. Thus algorithm $\mathcal{B}$ breaks pseudorandomness of PRG with the same advantage ε. □

Lemma 7.13. *Suppose iO is secure and Π_{SSB} is somewhere statistically binding. Then, for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$, and $i \in [0, L-1]$,*

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

Proof. The only difference between $\mathsf{ncHyb}_{i,2}^{(b)}$ and $\mathsf{ncHyb}_{i,3}^{(b)}$ is that during the challenge phase, the challenger sets the challenge ciphertext to be an obfuscation of $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i+1]$ instead of $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i]$. We argue that with overwhelming probability over the choice of hk and the public key pk_k, these two programs have identical input/output behavior. The claim then follows by *iO* security. Take any input $(x, \mathsf{pk}_x, S_x, \pi_x, \mathsf{sk}_x)$ and consider the behavior of the two programs on this input:

- Suppose $x \neq i+1$. Then the logic in the two programs is identical (i.e., $x > i$ if and only if $x > i+1$ when $x \neq i+1$ and x is an integer).
- Suppose $x = i+1$ but $(\mathsf{pk}_x, S_x) \neq (\mathsf{pk}_{i+1}, S_{i+1})$. In $\mathsf{ncHyb}_{i,2}^{(b)}$ and $\mathsf{ncHyb}_{i,3}^{(b)}$, the hash key hk is sampled to be binding on index $i+1$. Moreover, in both experiments, the challenger computes the hash value h as

  $$h \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, ((\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))).$$

  Since Π_{SSB} is somewhere statistically binding, with overwhelming probability over the choice of hk, there does not exist any $(\mathsf{pk}^*, S^*) \neq (\mathsf{pk}_{i+1}, S_{i+1})$ and π^* where $\mathsf{SSB.Verify}(\mathsf{hk}, h, i+1, (\mathsf{pk}^*, S^*), \pi^*) = 1$. In particular, this means that $\mathsf{SSB.Verify}(\mathsf{hk}, h, x, (\mathsf{pk}_x, S_x), \pi_x) = 0$, and both programs output $\perp$.
- Suppose $x = i+1$ and $(\mathsf{pk}_x, S_x) = (\mathsf{pk}_{i+1}, S_{i+1})$. Assuming $\mathsf{ncHyb}_{i,2}^{(b)}$ and $\mathsf{ncHyb}_{i,3}^{(b)}$ does not abort, this means that $\mathsf{pk}_x = \mathsf{pk}_{i+1} = \mathsf{pk}_k$, where $\mathsf{pk}_k \xleftarrow{\text{R}} \{0,1\}^{2\lambda}$ is the public key sampled on the k^{th} key-generation query. Over the randomness of pk_k, the probability that there exists $\mathsf{sk}^* \in \{0,1\}^{\lambda}$ such that $\mathsf{PRG}(\mathsf{sk}^*) = \mathsf{pk}_k$ is at most $2^{\lambda}/2^{2\lambda} = 2^{-\lambda}$. Thus, with overwhelming probability over the choice of pk_k, for *all* inputs $\mathsf{sk}^* \in \{0,1\}^{\lambda}$, $\mathsf{PRG}(\mathsf{sk}^*) \neq \mathsf{pk}_k$. Correspondingly, this means that $\mathsf{PRG}(\mathsf{sk}_x) \neq \mathsf{pk}_x$, and once more, both programs output $\perp$.

Thus, with overwhelming probability over the choice of hk and pk_k, the input/output behavior of $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i]$ or $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i+1]$ is identical. The claim now follows by $i\mathcal{O}$ security. ◻

Lemma 7.14. *If* PRG *is secure, then for all efficient and admissible adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $b \in \{0,1\}$*, and* $i \in [0, L-1]$*,*

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

Proof. This follows by the same argument as the proof of Lemma 7.12. ◻

Lemma 7.15. *For all admissible adversaries* $\mathcal{A}$ *and all* $\lambda \in \mathbb{N}$*,* $b \in \{0,1\}$*, and* $i \in [0, L-1]$*,*

$$\Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] = K \cdot \Pr\left[\mathsf{ncHyb}_{i,4}^{(b)}(\mathcal{A}) = 1\right].$$

Proof. This follows by the same argument as the proof of Lemma 7.11. ◻

Combining Lemmas 7.11 to 7.15, we now have the following relations:

$$\begin{aligned} \Pr\left[\mathsf{iHyb}_{i}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] &= K \cdot \Pr\left[\mathsf{ncHyb}_{i,1}^{(b)}(\mathcal{A}) = 1\right] \\ \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] &= K \cdot \Pr\left[\mathsf{ncHyb}_{i,4}^{(b)}(\mathcal{A}) = 1\right] \\ \left|\Pr\left[\mathsf{ncHyb}_{i,4}^{(b)}(\mathcal{A}) = 1\right] - \Pr\left[\mathsf{ncHyb}_{i,1}^{(b)}(\mathcal{A}) = 1\right]\right| &= \mathsf{negl}(\lambda). \end{aligned}$$

Thus, we conclude that

$$\left|\Pr\left[\mathsf{iHyb}_{i}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] - \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right]\right| \le K \cdot \mathsf{negl}(\lambda),$$

which is negligible since the adversary can make at most $K = \mathsf{poly}(\lambda)$ queries. Eq. (7.2) holds. ◻

Analysis for the case where slot $i+1$ is corrupted. We now show that Eq. (7.3) holds. Our analysis here will require that the set of attributes S_{i+1} do *not* satisfy the challenge policy. We state the precise claim below:

Claim 7.16. *Suppose* $i\mathcal{O}$ *is secure and* Π_{SSB} *function is somewhere statistically binding. Then, for all efficient and admissible adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$*,* $b \in \{0,1\}$*, and* $i \in [0, L-1]$*,*

$$\left|\Pr\left[\mathsf{iHyb}_{i}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] - \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right]\right| = \mathsf{negl}(\lambda).$$

Proof. We proceed via a sequence of games:

- $\mathsf{cHyb}_{i,1}^{(b)}$: Same as $\mathsf{iHyb}_{i}^{(b)}$ except the challenger aborts with output 0 if event $\mathsf{NonCorrupt}$ occurs. For an admissible adversary, this condition ensures that the experiment outputs 1 only if the attribute S_{i+1} chosen by the adversary for slot $i+1$ does *not* satisfy the challenge policy.

- $\mathsf{cHyb}_{i,2}^{(b)}$: Same as $\mathsf{cHyb}_{i,1}^{(b)}$ except during the challenge phase, the challenger sets $j = i+1$ instead of $j = i$ when constructing the challenge ciphertext.

By definition, we have that

$$\begin{aligned} \Pr\left[\mathsf{iHyb}_{i}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] &= \Pr[\mathsf{cHyb}_{i,1}^{(b)}(\mathcal{A}) = 1] \\ \Pr\left[\mathsf{iHyb}_{i+1}^{(b)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] &= \Pr[\mathsf{cHyb}_{i,2}^{(b)}(\mathcal{A}) = 1]. \end{aligned}$$

Thus, it suffices to argue that for all efficient and admissible adversaries $\mathcal{A}$,

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

This follows by security of $i\mathcal{O}$ and the somewhere statistically binding property of Π_{SSB}. The only difference between $\mathsf{cHyb}_{i,1}^{(b)}$ and $\mathsf{cHyb}_{i,2}^{(b)}$ is that during the challenge phase, the challenger sets the challenge ciphertext to be an obfuscation of $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i+1]$ instead of $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i]$. We argue that with overwhelming probability over the choice of hk, these two programs have identical input/output behavior. Then the claim follows by $i\mathcal{O}$ security. Take any input $(x, \mathsf{pk}_x, S_x, \pi_x, \mathsf{sk}_x)$ and consider the behavior of the two programs on this input:

- Suppose $x \neq i+1$. Then the logic in the two programs is identical (i.e., $x > i$ if and only if $x > i+1$ when $x \neq i+1$ and x is an integer).

- Suppose $x = i+1$ but $(\mathsf{pk}_x, S_x) \neq (\mathsf{pk}_{i+1}, S_{i+1})$. In $\mathsf{cHyb}_{i,1}^{(b)}$ and $\mathsf{cHyb}_{i,2}^{(b)}$, the hash key hk is sampled to be binding on index $i+1$. Moreover, in both experiments, the challenger computes the hash value h as

$$h \leftarrow \mathsf{SSB.Hash}(\mathsf{hk}, ((\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))).$$

  Since Π_{SSB} is somewhere statistically binding, with overwhelming probability over the choice of hk, there does not exist any $(\mathsf{pk}^*, S^*) \neq (\mathsf{pk}_{i+1}, S_{i+1})$ and π^* where $\mathsf{SSB.Verify}(\mathsf{hk}, h, i+1, (\mathsf{pk}^*, S^*), \pi^*) = 1$. In particular, this means that $\mathsf{SSB.Verify}(\mathsf{hk}, h, x, (\mathsf{pk}_x, S_x), \pi_x) = 0$, and both programs output $\perp$.

- Suppose $x = i+1$ and $(\mathsf{pk}_x, S_x) = (\mathsf{pk}_{i+1}, S_{i+1})$. Assuming $\mathsf{cHyb}_{i,1}^{(b)}$ and $\mathsf{cHyb}_{i,2}^{(b)}$ does not abort (so $\overline{\mathsf{NonCorrupt}}$ occurs) and $\mathcal{A}$ is admissible, this means that S_x does not satisfy the challenge policy C. In other words, $C(S_x) = 0$. Once more, both programs output $\perp$.

Thus, with overwhelming probability over the choice of hk and pk_k, the input/output behavior of $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i]$ or $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, i+1]$ is identical. The claim now follows by $i\mathcal{O}$ security. □

Combining Claims 7.10 and 7.16 both Eqs. (7.2) and (7.3) hold. Then, Eq. (7.1) follows by the triangle inequality, and Claim 7.9 holds. □

Combining Claims 7.8 and 7.9, Lemma 7.7 follows. □

Lemma 7.17. *Assuming $i\mathcal{O}$ is secure, then for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$, $b \in \{0,1\}$,*

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

Proof. The only difference between these two experiments is the obfuscated program used to construct the challenge ciphertext. In $\mathsf{Hyb}_L^{(b)}$, the challenge ciphertext is an obfuscation of the program $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, L]$ whereas in Hyb_{L+1}, the challenge ciphertext consists of an obfuscation of the program C_{reject}. It suffices to argue that these two programs have identical behavior. The claim then follows by $i\mathcal{O}$ security. Take any input $(x, \mathsf{pk}_x, S_x, \pi_x, \mathsf{sk}_x)$ and consider the behavior of the programs on this input:

- The program C_{reject} always outputs $\perp$ by construction.

- Since $x \in [L]$, it will never be the case that $x > L$. Thus, $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, L]$ also outputs $\perp$ by construction.

We conclude that the programs $\mathsf{Embed}[\mathsf{mpk}, C, \mu_b^*, L]$ and C_{reject} have identical input/output behavior. The claim now follows by $i\mathcal{O}$ security. □

Lemma 7.18. *For all adversaries $\mathcal{A}$, $\Pr\left[\mathsf{Hyb}_{L+1}^{(0)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{Hyb}_{L+1}^{(1)}(\mathcal{A}) = 1\right]$.*

Proof. In Hyb_{L+1}, the challenger's behavior and correspondingly, the adversary's view, is independent of the bit b. □

Since the adversary is efficient (and thus, can only output a collection of polynomial number of public keys and attribute sets), we have that $L = \mathsf{poly}(\lambda)$. Security now follows by combining Lemmas 7.7, 7.17 and 7.18. □

Acknowledgments

We thank the Eurocrypt reviewers for helpful feedback on the presentation. Susan Hohenberger is supported by NSF CNS-1908181, the Office of Naval Research N00014-19-1-2294, and a Packard Foundation Subaward via UT Austin. Brent Waters is supported by NSF CNS-1908611, a Simons Investigator award, and the Packard Foundation Fellowship. David J. Wu is supported by NSF CNS-2151131, NSF CNS-2140975, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[AP03] Sattam S. Al-Riyami and Kenneth G. Paterson. Certificateless public key cryptography. In *ASIACRYPT*, pages 452–473, 2003.

[Bei96] Amos Beimel. *Secure Schemes for Secret Sharing and Key Distribution.* PhD thesis, Technion, 1996.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO*, pages 213–229, 2001.

[BGG$^+$14] Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In *EUROCRYPT*, pages 533–556, 2014.

[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*, pages 1–18, 2001.

[BGI$^+$12] Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. *J. ACM*, 59(2):6:1–6:48, 2012.

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In *TCC*, pages 325–341, 2005.

[BWY11] Mihir Bellare, Brent Waters, and Scott Yilek. Identity-based encryption secure against selective opening attack. In *TCC*, pages 235–252, 2011.

[CC09] Melissa Chase and Sherman S. M. Chow. Improving privacy and security in multi-authority attribute-based encryption. In *ACM CCS*, pages 121–130, 2009.

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In *IMACC*, pages 129–157, 2021.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In *PKC*, 2013.

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In *TCC*, pages 515–534, 2007.

[CHSS02] Liqun Chen, Keith Harrison, David Soldera, and Nigel P. Smart. Applications of multiple trust authorities in pairing based cryptosystems. In *InfraSec*, pages 260–275, 2002.

[Coc01] Clifford C. Cocks. An identity based encryption scheme based on quadratic residues. In *Cryptography and Coding*, pages 360–363, 2001.

[DKW21a] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority ABE for DNFs from LWE. In *EUROCRYPT*, pages 177–209, 2021.

[DKW21b] Pratish Datta, Ilan Komargodski, and Brent Waters. Decentralized multi-authority ABE for ncˆ1 from computational-bdh. *IACR Cryptol. ePrint Arch.*, page 1325, 2021.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *CRYPTO*, pages 186–194, 1986.

[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*, pages 40–49, 2013.

[GHM$^+$19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In *PKC*, pages 63–93, 2019.

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In *TCC*, pages 689–718, 2018.

[GLSW08] Vipul Goyal, Steve Lu, Amit Sahai, and Brent Waters. Black-box accountable authority identity-based encryption. In *ACM CCS*, pages 427–436, 2008.

[Goy07] Vipul Goyal. Reducing trust in the PKG in identity based cryptosystems. In *CRYPTO*, pages 430–447, 2007.

[GPSW06] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In *ACM CCS*, pages 89–98, 2006.

[GV20] Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In *CRYPTO*, pages 621–651, 2020.

[GVW13] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Attribute-based encryption for circuits. In *STOC*, pages 545–554, 2013.

[HW15] Pavel Hubácek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In *ITCS*, pages 163–172, 2015.

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, pages 60–73, 2021.

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb {F}_p$, dlin, and prgs in nc^0. In *EUROCRYPT*, pages 670–699, 2022.

[KG10] Aniket Kate and Ian Goldberg. Distributed private-key generators for identity-based cryptography. In *SCN*, pages 436–453, 2010.

[LCLS08] Huang Lin, Zhenfu Cao, Xiaohui Liang, and Jun Shao. Secure threshold multi authority attribute based encryption without a central authority. In *INDOCRYPT*, pages 426–436, 2008.

[LM19] Russell W. F. Lai and Giulio Malavolta. Subvector commitments with application to succinct arguments. In *CRYPTO*, pages 530–560, 2019.

[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*, pages 62–91, 2010.

[LW10] Allison B. Lewko and Brent Waters. New techniques for dual system encryption and fully secure HIBE with short ciphertexts. In *TCC*, pages 455–479, 2010.

[LW11] Allison B. Lewko and Brent Waters. Decentralizing attribute-based encryption. In *EUROCRYPT*, pages 568–588, 2011.

[MKE08] Sascha Müller, Stefan Katzenbeisser, and Claudia Eckert. Distributed attribute-based encryption. In *ICISC*, pages 20–36, 2008.

[OPWW15] Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In *ASIACRYPT*, pages 121–145, 2015.

[PS08] Kenneth G. Paterson and Sriramkrishnan Srinivasan. Security and anonymity of identity-based encryption with multiple trusted authorities. In *Pairing*, pages 354–375, 2008.

[RW15] Yannis Rouselakis and Brent Waters. Efficient statically-secure large-universe multi-authority attribute-based encryption. In *Financial Cryptography and Data Security*, pages 315–332, 2015.

[Sha84] Adi Shamir. Identity-based cryptosystems and signature schemes. In *CRYPTO*, pages 47–53, 1984.

[SW05] Amit Sahai and Brent Waters. Fuzzy identity-based encryption. In *EUROCRYPT*, pages 457–473, 2005.

[Wat09] Brent Waters. Dual system encryption: Realizing fully secure IBE and HIBE under simple assumptions. In *CRYPTO*, pages 619–636, 2009.

[WW22] Brent Waters and David J. Wu. Batch arguments for NP and more from standard bilinear group assumptions. In *CRYPTO*, pages 433–463, 2022.

[WWW22] Brent Waters, Hoeteck Wee, and David J. Wu. Multi-authority ABE from lattices without random oracles. In *TCC*, 2022.