Pairing-Based Registered ABE for Boolean Formulas with a Linear-Size CRS

Roy Stracovsky[*] Brent Waters[†] David J. Wu[‡]

Abstract

Registered attribute-based encryption (ABE) is a generalization of ABE that replaces the central trusted key-issuer with an untrusted key curator. In registered (ciphertext-policy) ABE, users generate their own public keys and there is a transparent aggregation process that takes the public keys of the users together with their attributes and aggregates them into a short master public key that functions as the public key for a standard ABE scheme.

A sequence of works has focused on improving the efficiency and expressivity of pairing-based registered ABE. Today, all constructions of pairing-based registered ABE rely on a structured common reference string (CRS) whose size scales with the total number of users in the system N. While the first pairing-based constructions needed a CRS of size $O(N^2)$, a recent line of work has shown how to reduce it to $N^{1+o(1)}$ in the case of general policies (albeit with extremely large constant factors), and to $O(N)$ if we restrict the policy family to conjunctions and DNFs (earlier schemes could support general monotone Boolean formulas) and if we analyze security in the generic group model (earlier schemes could be proven secure in the plain model).

In this work, we give the first pairing-based registered ABE scheme with a linear-size CRS that supports general policies (i.e., monotone span programs which include monotone Boolean formulas as well as threshold policies). We can show static security based on a q-type assumption in the plain model and adaptive security if we instead work in the random oracle model. Our scheme is also the first pairing-based construction where users can be identified by arbitrary strings (e.g., identities) rather than by integers from a polynomial-size range. This directly enables registered ABE with *stateless* key-generation. Namely, users in our system can sample their key independently of the current state of the system. Previous approaches require users either to first retrieve the current state of the system before they could generate their key or to generate multiple public keys to avoid collisions.

1 Introduction

Registration-based cryptography [GHMR18] is a paradigm for building advanced cryptographic notions without having to rely on any central trusted authority. In this work, we focus on ciphertext-policy attribute-based encryption (ABE) [SW05, GPSW06] where each secret key is associated with an attribute x and each ciphertext is associated with an access policy P. Decryption is possible whenever the attribute satisfies the policy (e.g., when $P(x) = 1$). Traditionally, in an ABE scheme, a trusted authority is responsible for generating and issuing the decryption keys. If the authority is compromised, then the security of every user in the system is also compromised. Registered ABE [HLWW23] provides one solution to this problem. In registered ABE, users independently generate their own public/secret key-pair and then "register" the public key with a "key curator." The key curator can then aggregate the individual public keys along with their respective access policies to obtain a succinct master public key mpk. Importantly, this aggregation process is deterministic and transparent (i.e., does not depend on any secrets). The aggregated master public key mpk then functions as the master public key for a standard ABE scheme; namely, anyone can encrypt a message with respect to a policy such that only registered users with attributes that satisfy the policy can decrypt. Registered ABE allows us to combine the fine-grained decryption capabilities of attribute-based encryption with the trustless model of classic public-key encryption where users retain full control of their key material.

[*]Georgia Institute of Technology. Email: rstracovsky3@gatech.edu
[†]UT Austin and NTT Research. Email: bwaters@cs.utexas.edu
[‡]UT Austin. Email: dwu4@cs.utexas.edu

Our focus: pairing-based registered ABE. In the last few years, many works have studied registered ABE from a broad range of cryptographic assumptions, including pairing-based assumptions [HLWW23, ZZGQ23, AT24, GLWW24, GHK+25] and lattice-based assumptions [CHW25, WW25, ZZC+25, YZGC25, PST25], as well as constructions from general-purpose primitives such as witness encryption [FWW23] or indistinguishability obfuscation [HLWW23]. In this work, we focus on pairing-based constructions. The main advantage of the pairing-based approaches is that they are concretely efficient. The concrete efficiency of pairing-based registered ABE schemes is comparable to that of pairing-based centralized ABE schemes, which have been successfully implemented [AC17, RW22] and even deployed [Ver23]. A number of recent works have also implemented and used similar pairing-based primitives like registration-based encryption [GKMR23], distributed broadcast encryption [GLWW23], and threshold encryption with silent setup [GKPW24] to construct privacy-preserving systems with minimal trust assumptions. This is in contrast to lattice-based schemes, which all rely on the homomorphic evaluation procedures from [BGG+14] that, to our knowledge, have not been successfully implemented. Constructions based on tools like witness encryption or indistinguishability obfuscation are even less practical than lattice-based ABE. Moreover, schemes based on these general-purpose tools can only be instantiated assuming a combination of multiple cryptographic assumptions or by relying on new non-standard assumptions. This is also the case for the lattice-based constructions where security relies on either new assumptions like succinct LWE (*and* the random oracle model) or non-falsifiable assumptions like evasive LWE. An appealing property of pairing-based schemes is that we can prove security from simple assumptions in the plain model.

The CRS size in pairing-based registered ABE schemes. All existing pairing-based registered ABE schemes rely on a structured common reference string (CRS) whose size grows with the maximum number of users N in the system. The first constructions of pairing-based registered ABE [HLWW23, ZZGQ23, AT24] required a CRS whose size scales *quadratically* with N. This makes these schemes suitable only for a small number of users. Subsequently, the work of [GLWW24] showed how to reduce the CRS size to be *nearly linear* in the number of users (i.e., $N^{1+o(1)}$) using combinatorial techniques. However, the authors note that the asymptotic improvement only translates to an efficiency improvement for extremely large values of N, and for practical parameters of interest, the size of the CRS in their approach scales with $N^{\log_2 3} \approx N^{1.6}$. The main open problem in this line of research is to build a registered ABE scheme with a linear-size CRS. The recent work of [GHK+25] gives a partial answer to this problem by constructing a pairing-based registered ABE scheme with a linear-size CRS. However, their construction comes at the expense of expressivity: the scheme can only support conjunction policies (and more generally, DNF policies via concatenation). In contrast, all of the aforementioned works on registered ABE support monotone span programs (or more), which in particular includes the class of monotone Boolean formulas and threshold policies. In addition, the security of the construction in [GHK+25] critically relies on the generic group model, whereas earlier pairing-based schemes were shown secure in the plain model.

This work. Our main contribution in this work is the first pairing-based registered ABE scheme with a linear-size CRS that supports all monotone span programs. We prove static security of our scheme under a q-type assumption in the plain model (which holds unconditionally in the generic bilinear group model of [Sho97, BBG05]) and adaptive security in the random oracle model. We provide a comparison of our scheme to previous schemes in Table 1.

An additional feature: registered ABE with large index space. An appealing property of our registered ABE scheme is that it natively supports a large index space. In more detail, in all pairing-based registered ABE schemes, when a user generates their public key, they also associate it with an index $i \in [N]$ where N is the bound on the number of users. Correctness only holds if every user in the system associates a different index with their public key. Thus, when a user joins the system, they need to first choose an index $i \in [N]$ that is not already taken. In practice, this means that users would interact with a key curator to agree on an index i and then generate their key for the particular index. Alternatively, a user could generate public keys for multiple distinct indices and have the key curator associate a distinct index with each user [GLWW23]. The first approach requires interaction while the second increases the size of each user's public key. Our construction in this work natively supports a "large" index space where the index associated with a key can be an *arbitrary* string (or "identity"). We retain the restriction that every user in the system has a distinct index. However, since the indices can be drawn from an exponential-size universe, individual users can simply pick a random index when generating their key, and with overwhelming probability over the choice of the random index,

Scheme	|crs|	|pk|	|mpk|	|ct|	Policy	Assumption	AD	LI
[HLWW23]	1	1	1	$p(\|\mathcal{P}\|)$	circuits	$i\mathcal{O}$ + OWF	✓	✓
[FWW23]	$p(s)$	1	$p(s)$	$p(s)$	size-s circuits	witness encryption + LWE	✗	✓
[CHW25]*	$N^2 \cdot p(d)$	$N \cdot p(d)$	$p(d)$	$p(d, \|a\|)$	depth-d circuits	succinct LWE + ROM	✗	✗
[WW25]*	$p(d)$	$p(d)$	$p(d)$	$p(d)$	depth-d circuits	succinct LWE + ROM	✗	✓
[ZZC+25]*	$p(d)$	$p(d)$	$p(d)$	$p(d, \|a\|)$	depth-d circuits	evasive LWE	✗	✓
[YZGC25]*	$N \cdot p(d)$	$p(d)$	$p(d)$	$p(d, \|a\|)$	depth-d circuits	evasive LWE	✗	✗
[PST25]	1	1	1	1	circuits	evasive LWE	✗	✓
[HLWW23]	$\|\mathcal{U}\| \cdot N^2$	N	$\|\mathcal{U}\|$	$\|\mathcal{P}\|$	MSP	composite order (static)	✓	✗
[ZZGQ23]	$\|\mathcal{U}\| \cdot N^2$	N	$\|\mathcal{U}\|$	$\|\mathcal{P}\|$	ABP	prime order (static)	✓	✗
[GLWW24]	$N^{1+o(1)}$	N	$\|\mathcal{U}\|$	$\|\mathcal{P}\|$	MSP	prime order (q-type)	✗	✗
[AT24]†	N^2	N	$\|a\|^2$	$\|a\| \cdot \|\mathcal{P}\|$	NM-MSP	prime order (static)	✓	✗
[GHK+25]	N	N	$\|\mathcal{U}\|$	$\|\mathcal{P}\|$	DNF formulas	generic group model	✓	✗
[RT26]†	$N^{1+o(1)}$	N	$\|a\|^2$	$\|\mathcal{P}\|$	NM-MSP	generic group model + ROM	✓	✗
Cor. 4.13	N	N	$\|\mathcal{U}\|$	$\|\mathcal{P}\|$	MSP	prime order (q-type)	✗	✓
Cor. 4.14‡	$N \cdot p(K)$	$N \cdot p(K)$	$\|\mathcal{U}\| \cdot p(K)$	$p(K)$	MSP	prime order (q-type) + ROM	✓	✗

*These schemes are *key-policy* schemes and we write |a| to denote the length of the attribute in the ciphertext.
†For these schemes, we write |a| to denote the maximum number of attributes associated with a single user in the system.
‡This scheme assumes an a priori bound on the size K of a policy and relies on subexponential hardness of the underlying assumption.

Table 1: Comparison with previous registered ABE schemes. Here, crs denotes the common reference string, pk denotes an individual user's public key, mpk denotes the master public key, and ct denotes the ciphertext. We consider a system with N users, an attribute universe $\mathcal{U}$, and a policy $\mathcal{P}$. We write MSP to denote monotone span programs (which includes monotone Boolean formulas and threshold policies), ABP to denote arithmetic branching programs (which generalize MSPs), NM-MSP to denote non-monotone span programs, and DNF to denote formulas in disjunctive normal form ("an OR of ANDs"). We write $i\mathcal{O}$ to denote indistinguishability obfuscation, OWF to denote one-way functions, and ROM to denote the random oracle model. For the pairing-based schemes, we indicate whether they are based on static assumptions or q-type assumptions over composite-order or prime-order groups, or if they rely on the generic bilinear group model. We write $p(\cdot)$ to denote a fixed polynomial and suppress all polynomials in the security parameter and all polylogarithmic terms. For each scheme, we also indicate whether it can be shown to be adaptively secure ("AD") under the stated assumption and whether it supports a large index space ("LI") where user keys can be associated with an arbitrary bitstring (rather than an index from a polynomial-size range).

no two users will share the same index. This way, we obtain a scheme where users can sample their keys completely independently of all other users. In particular, this immediately gives a registered ABE scheme with "stateless key generation"—namely, a scheme where users can generate their key independently of the current state of the system.

1.1 Technical Overview

Like nearly all constructions of registered ABE, we focus on constructing a simpler primitive called a *slotted* registered ABE scheme, which was formally introduced in the work of [HLWW23]. We start by recalling the general syntax:

- **Common reference string:** The CRS for a slotted registered ABE scheme defines an a priori fixed number of users N (also called "slots").
- **Key generation:** When a user joins the system, they use the CRS to sample a public/private key-pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ associated with a specific slot index $i \in [N]$.
- **Aggregation:** Given a collection of exactly N public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$, where pk_i is a public key for slot i, along with their respective sets of attributes $S_1, \ldots, S_N$, the aggregation algorithm outputs a succinct master public key mpk together with a *public* decryption hint hsk_i for each user $i \in [N]$. We refer to hsk_i as the

"helper decryption key" for user i. Importantly, in registered ABE, the aggregation process is deterministic and requires no secrets. Everyone can compute and verify the output of the aggregation algorithm.

- **Encryption:** Using the master public key mpk, anyone can encrypt a message μ with respect to a policy P, just like in a standard ciphertext-policy ABE scheme.
- **Decryption:** Any user $i \in [N]$ associated with attributes S_i that satisfy the policy P can use their secret key sk_i together with their public decryption hint hsk_i to recover the message μ.

The main difference between slotted registered ABE and the standard notion of registered ABE is the lack of support for dynamic registration. In the slotted setting, *all* of the public keys and attribute sets must be provided together in order to generate the master public key. The standard notion of registered ABE allows users to join the system one at a time, and each time a user joins, the master public key and helper decryption keys are updated to reflect the current set of registered users. The abstraction of a slotted registered ABE scheme is simpler to work with since we only need to support one-shot aggregation as opposed to dynamic aggregation. The work of [HLWW23] shows how to lift the slotted scheme into the standard setting (with dynamic registration) with only logarithmic overhead. For this reason, we focus on the simpler setting of slotted registered ABE throughout this overview (and for much of the technical sections).

Starting point: a variant of the [HLWW23] registered ABE scheme. Next, we start with the general template from [HLWW23] for constructing a slotted registered ABE scheme. The cross-term cancellation structure introduced in that work is the cryptographic core of many pairing-based registered ABE schemes [ZZGQ23, GLWW24, AT24, RT26]. Specifically, we describe a slimmed-down variant of the prime-order scheme from [GLWW24, Construction B.3], which can be viewed as a prime-order analog of [HLWW23]. For compatibility with our subsequent exposition, we describe the scheme over an asymmetric pairing group (as opposed to a symmetric pairing group).

Let $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g_1, g_2, e)$ be an asymmetric pairing group where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order p, g_1 is a generator of $\mathbb{G}_1$, g_2 is a generator of $\mathbb{G}_2$, and $e\colon \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an efficiently computable non-degenerate bilinear map. We write group elements using implicit notation [EHK+13]. In particular, for a vector $\mathbf{v} \in \mathbb{Z}_p^n$, we write $[\mathbf{v}]_1, [\mathbf{v}]_2, [\mathbf{v}]_T$ to denote $g_1^{\mathbf{v}}, g_2^{\mathbf{v}}$, and $e(g_1, g_2)^{\mathbf{v}}$, respectively, where exponentiation is defined component-wise. The slotted registered ABE scheme now works as follows:

- **Common reference string:** Let N be the number of slots. Sample $\alpha, y \xleftarrow{R} \mathbb{Z}_p$. For each slot $i \in [N]$, sample exponents $t_i, u_i \xleftarrow{R} \mathbb{Z}_p$. The CRS consists of the following set of terms (which we highlight in green):
  - the encryption components $[\alpha]_T$ and $[y]_1$,
  - the slot components $([t_i]_2, [u_i]_1, [\alpha + y t_i]_2)$ for all $i \in [N]$, and
  - the cross terms $[t_i u_j]_1$ for all $i \neq j$.
- **Key generation:** To generate a key for slot i, the user samples $r_i \xleftarrow{R} \mathbb{Z}_p$ and sets their public key to be $\mathsf{pk}_i = [r_i]_1$ and the secret key to be $\mathsf{sk}_i = r_i$. In addition, they also include the cross terms $[t_j r_i]_2 = r_i [t_j]_2$ for all $j \neq i$ as part of their public key. The cross terms are used to construct the decryption hints.
- **Aggregation:** The aggregation algorithm takes public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ where $\mathsf{pk}_i = ([r_i]_1, \{[t_j r_i]_2\}_{j \neq i})$ and the attributes $S_1, \ldots, S_N$ and computes the aggregated master public key and the helper decryption key as follows (which we highlight in blue):
  - **Attribute-independent public key:** Compute $[\hat{r}]_1 = \sum_{i \in [N]} [r_i]_1$. In addition, for each $i \in [N]$, compute the cross terms $[\hat{v}_i]_2 = \sum_{j \neq i} [t_i r_j]_2$.
  - **Attribute-specific public key:** For each attribute a, let $[\hat{u}_a]_1 = \sum_{i \in [N] : a \notin S_i} [u_i]_1$. In addition, for each $i \in [N]$, compute the cross terms $[\hat{w}_{a,i}]_1 = \sum_{j \neq i : a \notin S_j} [t_i u_j]_1$.

  Let $\mathcal{U} = \bigcup_{i \in [N]} S_i$ be the set of attributes associated with the users. The aggregated master public key is $\mathsf{mpk} = ([\hat{r}]_1, \{[\hat{u}_a]_1\}_{a \in \mathcal{U}})$. The helper decryption key hsk_i for user i contains the slot components associated with slot i from the CRS as well as the cross terms:

$$\mathsf{hsk}_i = ([t_i]_2, [\alpha + y t_i]_2, [\hat{v}_i]_2, \{[\hat{w}_{a,i}]_1\}_{a \in \mathcal{U}}).$$

- **Encryption:** Let P be a policy over a set of attributes $A = (a_1, \ldots, a_K) \subseteq \mathcal{U}$. We assume that P has a linear secret sharing scheme over $\mathbb{Z}_p$ where each attribute is associated with a single share, which is an element of $\mathbb{Z}_p$.[1] To encrypt a message $[\mu]_T$ (i.e., we take the message to be an element of the target group $\mathbb{G}_T$) with respect to the policy P, the encrypter proceeds as follows:
  - Sample the encryption randomness $s \xleftarrow{R} \mathbb{Z}_p$ and $s_1, s_2 \xleftarrow{R} \mathbb{Z}_p$ such that $s_1 + s_2 = s$.[2]
  - Secret share s_2 according to the policy P over $\mathbb{Z}_p$. Let $s_{2,1}, \ldots, s_{2,K} \in \mathbb{Z}_p$ be the shares of s_2 associated with the attributes $a_1, \ldots, a_K$ respectively. Recall that for ease of exposition, we are assuming that P has a linear secret sharing scheme where each attribute is associated with a single share (i.e., a single element of $\mathbb{Z}_p$).
  - Compute the attribute-independent component $s_1[y]_1 - s[\hat{r}]_1 = [s_1 y - s\hat{r}]_1$. For each attribute $a_k \in A$, sample a blinding term $s'_k \xleftarrow{R} \mathbb{Z}_p$ and compute the attribute-specific component
$$\left(s_{2,k}[y]_1 - s'_k[\hat{u}_{a_k}]_1 \, , [s'_k]_2\right) = \left([s_{2,k} y - s'_k \hat{u}_{a_k}]_1 \, , \ [s'_k]_2\right).$$
  - The ciphertext then consists of the following group elements (which we highlight in purple):
$$\mathsf{ct} = \left([s\alpha]_T + [\mu]_T, [s]_1, [s_1 y - s\hat{r}]_1, \{[s_{2,k} y - s'_k \hat{u}_{a_k}]_1, [s'_k]_2\}_{k \in [K]}\right).$$
- **Decryption:** Suppose a user $i \in [N]$ has a set of attributes $S_i \subseteq \mathcal{U}$ that satisfies the policy P. The user's goal is to compute the blinding factor $[s\alpha]_T$. To do so, the user proceeds as follows:
  - **Slot check:** First, the user uses their secret key r_i together with the helper decryption components $[t_i]_2$ and $[\hat{v}_i]_2$ to compute
$$\begin{aligned}[s_1 y - s\hat{r}]_1 \cdot [t_i]_2 + [s]_1 \cdot [\hat{v}_i]_2 + r_i \cdot [s]_1 \cdot [t_i]_2 &= [s_1 y t_i - \textstyle\sum_{j \in [N]} s t_i r_j + \sum_{j \neq i} s t_i r_j + s t_i r_i]_T \\ &= [s_1 y t_i]_T.\end{aligned}$$
  - **Attribute check:** For each attribute a_k that the user possesses, they use the slot component $[t_i]_2$ and the cross term $[\hat{w}_{a_k,i}]_1$ from their helper decryption key to compute
$$\begin{aligned}[s_{2,k} y - s'_k \hat{u}_{a_k}]_1 \cdot [t_i]_2 + [\hat{w}_{a_k,i}]_1 \cdot [s'_k]_2 &= [s_{2,k} y t_i - \textstyle\sum_{j \in [N] : a_k \notin S_j} s'_k t_i u_j + \sum_{j \neq i : a_k \notin S_j} s'_k t_i u_j]_T \\ &= [s_{2,k} y t_i]_T.\end{aligned}$$
    Note that this equality relies on the fact that $a_k \in S_i$, which means
$$\{j \in [N] : a_k \notin S_j\} = \{j \in [N] \setminus \{i\} : a_k \notin S_j\}.$$
    Since $s_{2,1}, \ldots, s_{2,K}$ is a linear secret sharing of s_2, we can view each $[s_{2,k} y t_i]_T$ as a share of $[s_2 y t_i]_T$. If S_i satisfies the policy, then the user can take a linear combination of these shares to reconstruct $[s_2 y t_i]_T$.

  Since $s_1 + s_2 = s$, the user can now compute $[s_1 y t_i]_T + [s_2 y t_i]_T = [s y t_i]_T$. Finally, the user computes
$$[s]_1 \cdot [\alpha + y t_i]_2 - [s y t_i]_T = [s\alpha]_T,$$
  which is sufficient to recover the message $[\mu]_T$ from $[s\alpha]_T + [\mu]_T$.

[1] This restriction is for ease of exposition. Our actual construction (Construction 4.1) does not have this restriction.

[2] The schemes from [HLWW23, GLWW24] secret share the element $[y]_1$ from the CRS instead of the encryption randomness s. However, since the exponents s and y play similar roles in the ciphertext, we can interchange them. The variant we describe here is more amenable to our strategy for obtaining a scheme with a linear-size CRS.

Reducing the CRS size. The CRS size in the above scheme is quadratic due to the presence of the cross terms $[t_i u_j]_1$ for all $i \neq j$. These cross terms are needed to compute the attribute-specific component $\hat{w}_{a_k,i}$ of each user's helper decryption key. As shown above, the key relationship needed for decryption is that whenever $a_k \in S_i$, it holds that

$$[s'_k \hat{u}_{a_k}]_1 \cdot [t_i]_2 = \left[\sum_{j \in [N]: a_k \notin S_j} s'_k t_i u_j\right]_T = [\hat{w}_{a_k,i}]_1 \cdot [s'_k]_2. \tag{1.1}$$

The work of [GLWW24] shows how to compress the CRS by choosing the exponents t_i and u_i in a *structured* way so that the set of cross terms $[t_i u_j]_1$ have a succinct description. Specifically, they take t_i and u_i to be powers. To illustrate, suppose $t_i = \tau^i$ and $u_i = \beta\tau^i$, where $\tau, \beta \xleftarrow{R} \mathbb{Z}_p$. In this way, the cross terms $t_i u_j$ can be written as $t_i u_j = \beta\tau^{i+j}$, which can be described compactly using just $2N$ group elements $[\beta\tau]_1, \ldots, [\beta\tau^{2N}]_1$. The savings come from the fact that multiple distinct pairs of (i, j) can now share the *same* cross term. This seemingly already yields a scheme with a linear-size CRS (since there are now only $2N$ cross terms in the CRS).

The problem with this approach is that if the CRS contains $[\beta\tau]_1, \ldots, [\beta\tau^{2N}]_1$, it is simultaneously giving out the non-cross-terms $[t_i u_i]_1 = [\beta\tau^{2i}]_1$. This compromises security of the above construction since it would allow a user who does *not* have an attribute a_k to nonetheless compute the term $[s'_k \hat{u}_{a_k} t_i]_T$ in Eq. (1.1). To preserve security, the work of [GLWW24] chooses t_i, u_i so that the cross-terms have a compact description without simultaneously giving out the non-cross-terms. They achieve this by taking the powers from a "progression-free set" [ET36] instead of the integers $\{1, \ldots, N\}$. This leads to a scheme where the size of the CRS is $N^{1+o(1)}$, though the asymptotic improvement is only meaningful for very large values of N (cf. [GGK25, Table 8]). For typical values of N, the size of the CRS would scale with $N^{\log_2 3} \approx N^{1.6}$.

Using the polynomial basis. In this work, we show that working in the Lagrange interpolation basis rather than the power basis from [GLWW24] enables a construction with a linear-size CRS. This idea of using the Lagrange interpolation basis for aggregation[3] has been used in a number of previous settings, including distributed threshold encryption [DP08], identity-based broadcast encryption [DPP07, Del07], polynomial commitments [KZG10], locally verifiable signatures [GV20], and batch arguments for NP [CEW25]. Our goal is still to design a mechanism such that the decrypter can satisfy a relation analogous to the attribute check shown in Eq. (1.1) whenever they have the attribute a_k. Conversely, they should not be able to combine components from the public parameters and their helper decryption key to satisfy the relation if they do not have the attribute. We now describe our approach to implement this:

- Let $Z(X) = \prod_{i \in [N]} (X - i)$ be the polynomial that vanishes on the indices $[N]$. All polynomials are defined over $\mathbb{Z}_p$ where p is prime (i.e., $\mathbb{Z}_p$ is a field).
- For each $i \in [N]$, we define

$$L_i(X) = \prod_{j \in [N] \setminus \{i\}} (X - j) = \frac{Z(X)}{X - i} \tag{1.2}$$

  to be the basis polynomial that vanishes on the indices $[N] \setminus \{i\}$ and is non-zero at index i. Note that we do *not* normalize $L_i(X)$ to have value 1 at index i.
- In the scheme, we set the slot exponents to be $t_i = L_i(\tau)$ where $\tau \xleftarrow{R} \mathbb{Z}_p$. If we consider a parallel to pairing-based identity-based broadcast encryption schemes [DPP07, Del07], the slot exponents in our scheme share the same structure as the secret keys in the broadcast encryption schemes, but scaled up by the vanishing polynomial $Z(X)$. The scaling will be helpful for deriving the analogous attribute check from Eq. (1.1).
- Next, we modify how we compute the attribute-specific public keys at aggregation time. Let $S_1, \ldots, S_N$ be the attributes associated with the N users. Specifically, for each attribute a, we define the polynomial

$$F_a(X) \coloneqq \prod_{j \in [N]: a \notin S_j} (X - j). \tag{1.3}$$

  This is the polynomial whose roots correspond to the indices of users that do *not* possess the attribute. We take the attribute-specific public key to be $[\hat{u}_a]_2 = [F_a(\tau)]_2$.

[3] Specifically, we can view registered ABE as needing to devise a mechanism to *aggregate* user public keys into a short master public key (and then support encryption with respect to the aggregated key).

- Observe now that for all indices $i \in [N]$ and all attributes a, the polynomial $F_a(X)$ divides $L_i(X)$ if and only if $a \in S_i$. Moreover, when $a \in S_i$, the quotient polynomial $F_{a,i}(X) \coloneqq L_i(X)/F_a(X)$ has degree at most $N-1$.
- We can now write down a similar relation for the attribute check from Eq. (1.1):

$$[s'_k]_1 \cdot [t_i]_2 = [s'_k L_i(\tau)]_T = [s'_k F_{a_k}(\tau) F_{a_k,i}(\tau)]_T = [F_{a_k,i}(\tau)]_1 \cdot [s'_k \hat{u}_{a_k}]_2.$$

  In particular, for each attribute a and each user $i \in [N]$, we now define their attribute-specific helper decryption key $[\hat{w}_{a,i}]_1$ to be $\hat{w}_{a,i} = F_{a,i}(\tau)$. Since each $F_{a,i}$ is a polynomial of degree at most $N-1$, it suffices to publish encodings of the powers of τ in the CRS. That is, the CRS includes $[\tau]_1, [\tau^2]_1, \ldots, [\tau^{N-1}]_1$, and the aggregator can use these terms to construct the helper decryption terms $[\hat{w}_{a,i}]_1 = [F_{a,i}(\tau)]_1$ for all attributes a and indices $i \in [N]$.

Putting all the pieces together, we now arrive at our new registered ABE scheme with a linear-size CRS:

- **Common reference string:** Let N be the number of slots and sample $\alpha, \tau, y \xleftarrow{R} \mathbb{Z}_p$. The CRS now includes the following components:
  - the powers of τ used for key-generation and aggregation $[\boldsymbol{\tau}]_1$ and $[\boldsymbol{\tau}]_2$ where $\boldsymbol{\tau} = (\tau, \tau^2, \ldots, \tau^{N-1})$,
  - the encryption components $[\alpha]_T$ and $[y]_1$, and
  - the slot components $[y]_2, [y\tau]_2, \ldots, [y\tau^{N-2}]_2, [\alpha + y\tau^{N-1}]_2$.

  By construction, this consists of exactly $3N$ group elements.
- **Key-generation:** To generate a key for slot $i \in [N]$, the user samples $r_i \xleftarrow{R} \mathbb{Z}_p$. The public key consists of $[r_i]_1$ together with the cross terms $[r_i(\tau - i)]_2, [r_i\tau(\tau - i)]_2, \ldots, [r_i\tau^{N-2}(\tau - i)]_2$, which the user can compute from $[\boldsymbol{\tau}]_2$ published in the CRS. The secret key is $\mathsf{sk}_i = r_i$.
- **Aggregation:** Given public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_N$ with $\mathsf{pk}_i = ([r_i]_1, [r_i(\tau - i)]_2, [r_i\tau(\tau - i)]_2, \ldots, [r_i\tau^{N-2}(\tau - i)]_2)$, as well as their respective attribute sets $S_1, \ldots, S_N$, we compute the aggregated master public key and helper decryption components as follows:
  - **Pre-computed decryption components:** For each $i \in [N]$, compute $[t_i]_2 = [L_i(\tau)]_2$ and $[\alpha + yt_i]_2 = [\alpha + yL_i(\tau)]_2$, where L_i is the polynomial from Eq. (1.2). These can be computed from $[\boldsymbol{\tau}]_2$ and the slot components $[y]_2, [y\tau]_2, \ldots, [y\tau^{N-2}]_2, [\alpha + y\tau^{N-1}]_2$ published in the CRS.
  - **Attribute-independent public key:** Compute $[\hat{r}]_1 = \sum_{i \in [N]} [r_i]_1$ as before. Then, for each $i \in [N]$, compute the cross terms $[\hat{v}_i]_2 = \sum_{j \neq i} [r_j L_i(\tau)]_2 = \sum_{j \neq i} [t_i r_j]_2$. Note that $[r_j L_i(\tau)]_2$ can be computed using the $[r_j(\tau - j)]_2, [r_j\tau(\tau - j)]_2, \ldots, [r_j\tau^{N-2}(\tau - j)]_2$ components from pk_j. Here, we use the property that for all $j \neq i$, the polynomial $(X - j)$ is a factor of $L_i(X)$.
  - **Attribute-specific public key:** Let $\mathcal{U} = \bigcup_{i \in [N]} S_i$ be the set of attributes associated with the users. For each attribute $a \in \mathcal{U}$, define the attribute-specific public key to be $[\hat{u}_a]_2 = [F_a(\tau)]_2$, where F_a is the polynomial from Eq. (1.3). In addition, for each $i \in [N]$, compute the attribute-specific helper decryption key $[\hat{w}_{a,i}]_1 = [F_{a,i}(\tau)]_1$, where $F_{a,i}(\tau) = L_i(\tau)/F_a(\tau)$. As argued above, these terms can all be computed from $[\boldsymbol{\tau}]_1$ and $[\boldsymbol{\tau}]_2$ published in the CRS.

  The aggregated master public key is $\mathsf{mpk} = ([\hat{r}]_1, \{[\hat{u}_a]_2\}_{a \in \mathcal{U}})$. The helper decryption key hsk_i for user i consists of the pre-computed decryption components and the cross terms: $\mathsf{hsk}_i = ([t_i]_2, [\alpha + yt_i]_2, [\hat{v}_i]_2, \{[\hat{w}_{a,i}]_1\}_{a \in \mathcal{U}})$.
- **Encryption:** Let P be a policy over attributes $(a_1, \ldots, a_K)$ and as before, suppose P has a linear secret sharing scheme over $\mathbb{Z}_p$. To encrypt a message $[\mu]_T$, the encrypter proceeds as follows:
  - Sample $s \xleftarrow{R} \mathbb{Z}_p$ and $s_1, s_2 \xleftarrow{R} \mathbb{Z}_p$ such that $s = s_1 + s_2$.
  - Secret share s_2 according to the policy P. Let $s_{2,1}, \ldots, s_{2,K} \in \mathbb{Z}_p$ be the shares associated with $a_1, \ldots, a_K$.
  - The ciphertext is then $\mathsf{ct} = \left([s\alpha]_T + [\mu]_T, [s]_1, [s_1 y - s\hat{r}]_1, \{[s_{2,k} y - s'_k]_1, [s'_k \hat{u}_{a_k}]_2\}_{k \in [K]}\right)$, where again, the encrypter samples $s'_k \xleftarrow{R} \mathbb{Z}_p$ for all $k \in [K]$.

- **Decryption:** Decryption proceeds similarly to before. Suppose a user $i \in [N]$ has a set of attributes S_i that satisfies the policy P. The user proceeds as follows:
  - **Slot check:** This is the same as before. Namely, the user uses their secret key r_i together with the helper decryption components $[t_i]_2$ and $[\hat{v}_i]_2$ to compute

$$[s_1 y - s\hat{r}]_1 \cdot [t_i]_2 + [s]_1 \cdot [\hat{v}_i]_2 + r_i \cdot [s]_1 \cdot [t_i]_2 = [s_1 y t_i - \textstyle\sum_{j\in[N]} s r_j t_i + \sum_{j\neq i} s r_j t_i + s r_i t_i]_T$$
$$= [s_1 y t_i]_T,$$

    where we have used the fact that $\hat{r} = \sum_{j\in[N]} r_j$ and $\hat{v}_i = \sum_{j\neq i} r_j t_i$.
  - **Attribute check:** For each attribute a_k that the user possesses, they use the cross term $[\hat{w}_{a_k,i}]_1$ and the slot component $[t_i]_2$ from their helper decryption key hsk_i to compute

$$[s_{2,k} y - s'_k]_1 \cdot [t_i]_2 + [\hat{w}_{a_k,i}]_1 \cdot [s'_k \hat{u}_{a_k}]_2 = [s_{2,k} y t_i - s'_k t_i + s'_k F_{a_k,i}(\tau) \cdot F_{a_k}(\tau)]_T$$
$$= [s_{2,k} y t_i]_T,$$

    where the final equality uses the fact that $F_{a_k,i}(\tau) \cdot F_{a_k}(\tau) = L_i(\tau) = t_i$ whenever $a_k \in S_i$. Once again, the decrypter recovers a share of $[s_2 y t_i]_T$. If the decrypter has a set of attributes that satisfy the policy P, then they can interpolate to fully obtain $[s_2 y t_i]_T$.

  The rest of the decryption process proceeds exactly as before. The decrypter computes $[s y t_i]_T = [s_1 y t_i]_T + [s_2 y t_i]_T$ and $[s\alpha]_T = [s]_1 \cdot [\alpha + y t_i]_2 - [s y t_i]_T$. This can then be used to recover the message from $[s\alpha]_T + [\mu]_T$.

We provide the formal description in Construction 4.1.

Security under a q-type assumption. We prove that our scheme satisfies security under a q-type assumption. Here, we sketch our proof strategy against an adversary that is not allowed to make any corruption queries in the security game (i.e., request the secret key for an honest user). For ease of exposition in the overview, we will also assume here that the adversary declares the slots associated with corrupted users at the beginning of the security game (though this relaxation is not required in our actual analysis). Even with this restriction, the adversary can still adaptively choose the public keys for the corrupted users and the attribute set for all users *after* seeing the scheme parameters. Similar to [GLWW24], we rely on a partitioning proof strategy. To carry out this proof strategy, we rely on a q-type assumption that captures the polynomial structure we embed. The inverse structure used in this assumption can be viewed as a strengthening of the strong Diffie-Hellman assumption previously used to analyze the soundness of the polynomial commitment scheme of [KZG10]. We state the assumption below:

- The assumption is parameterized by an integer N.
- Sample $\gamma \xleftarrow{R} \mathbb{Z}_p$ and $\tau \xleftarrow{R} \mathbb{Z}_p \setminus [N]$. Let $\beta = \sum_{i\in[N]} \frac{1}{\tau - i} \in \mathbb{Z}_p$.
- Given

$$\mathsf{params} = \left(\begin{array}{c} [\tau]_1, \ldots, [\tau^{N-1}]_1, [\tau]_2, \ldots, [\tau^{N-1}]_2, \\ [\beta]_1, [\frac{\gamma}{\tau-1}]_1, \ldots, [\frac{\gamma}{\tau-N}]_1, \\ [\gamma]_2, [\gamma\tau]_2, \ldots, [\gamma\tau^{N-2}]_2, [\gamma\beta]_2, [\gamma\beta\tau]_2, \ldots, [\gamma\beta\tau^{N-2}]_2 \end{array} \right),$$

  the goal is to distinguish the group element $[\gamma\beta^2\tau^{N-1}]_T$ from a random element of $\mathbb{G}_T$.

We show that this assumption holds unconditionally in the generic bilinear group model in Appendix C. We now provide a sketch of the partitioning argument underlying our security analysis. In the sketch, we focus on the core cancellations needed to simulate all of the components arising in the security analysis. We omit the additional randomization needed to generate the correct distribution. The cancellations illustrate the main idea underlying our reduction strategy while the additional randomization needed to obtain the right distribution is standard. We refer to the proof of Theorem 4.6 for the formal proof (with proper re-randomization).

- **Setup:** Let N be the number of users. We consider an instance of the assumption with parameter N. For ease of exposition, recall that we also assume that the adversary declares the indices of the corrupted users $C \subseteq [N]$. Let $\mathcal{H} = [N] \setminus C$ be the indices of the honest users.
- **Simulating the CRS:** First, the reduction needs to simulate the components of the CRS from the components given in the assumption.
  - The reduction sets $[\boldsymbol{\tau}]_1$ and $[\boldsymbol{\tau}]_2$ using the corresponding terms from the assumption.
  - Next, the reduction implicitly defines $y = \gamma\beta$. This allows the reduction to simulate the slot components $[y]_2, [y\tau]_2, \ldots, [y\tau^{N-2}]_2$ using the corresponding terms from the assumption. Since $\beta = \sum_{i\in[N]} \frac{1}{\tau - i}$, the reduction can also simulate $[y]_1 = [\gamma\beta]_1 = \sum_{i\in[N]} [\frac{\gamma}{\tau-i}]_1$.
  - Finally, the reduction implicitly sets $\alpha = -y\tau^{N-1} = -\gamma\beta\tau^{N-1}$. Then it can compute $[\alpha]_T = [-\tau^{N-1}]_1 \cdot [\gamma\beta]_2$. Observe now that we have the cancellation $\alpha + y\tau^{N-1} = 0$, which allows the reduction to simulate the CRS component $[\alpha + y\tau^{N-1}]_2$ even though $[\gamma\beta\tau^{N-1}]_2$ is not (and cannot be) given out in the assumption.
- **Key-generation queries:** For the honest users $i \in \mathcal{H}$, the reduction will implicitly set the user's secret key to be $r_i = \frac{\gamma}{\tau-i}$. In this case, the reduction can simulate the public key $[r_i]_1 = [\frac{\gamma}{\tau-i}]_1$ using the term from the assumption. For the cross terms $[r_i(\tau - i)\tau^j]_2$ for $j = 0, 1, \ldots, N-2$ in the public key, we can rely on the cancellation $r_i(\tau - i) = \frac{\gamma}{\tau-i} \cdot (\tau - i) = \gamma$. Thus, the cross terms can be simulated using $[\gamma]_2, \ldots, [\gamma\tau^{N-2}]_2$.
- **Challenge ciphertext:** For each corrupted user $i \in C$, the adversary needs to specify the public key $[r_i]_1$. To simplify the analysis, we work in the registered-key model [RY07] where we additionally require the adversary to provide the associated secret key $r_i \in \mathbb{Z}_p$ when registering. A scheme with security in the registered-key model can be generically lifted to a scheme in the standard definition simply by having each user include a non-interactive zero-knowledge proof of knowledge of the secret key at registration time; we refer to [LWW25] for a formal proof of this statement.

  Let P be the challenge policy and $[\mu]_T$ be the challenge message. When simulating the challenge ciphertext, the reduction will implicitly set $s_1 = \sum_{i\in\mathcal{H}} \frac{1}{\tau-i}$ and $s_2 = \sum_{i\in C} \frac{1}{\tau-i}$. Recall from the construction that s_1 is the randomness associated with the slot check while s_2 is the randomness associated with the attribute check. For the honest users $i \in \mathcal{H}$, the adversary does not know the associated secret key r_i whereas for the corrupted users $i \in C$, their set of attributes S_i does not satisfy the challenge policy P. We now describe how the reduction simulates the components of the challenge ciphertext for this choice of s_1 and s_2:
  - Since $\mathcal{H}$ and C form a partition of $[N]$ and $s = s_1 + s_2$, this means $[s]_1 = \sum_{i\in[N]} [\frac{1}{\tau-i}]_1 = [\beta]_1$, which the reduction has from the assumption.
  - Next, for the attribute-independent component, we use the fact that for $i \in \mathcal{H}$, the reduction implicitly set $r_i = \frac{\gamma}{\tau-i}$. Since $y = \gamma\beta$, we have

    $$s_1 y = \gamma\beta \sum_{i\in\mathcal{H}} \frac{1}{\tau - i} = \beta \sum_{i\in\mathcal{H}} r_i = s \sum_{i\in\mathcal{H}} r_i.$$

    In this case,

    $$s_1 y - s\hat{r} = s \sum_{i\in\mathcal{H}} r_i - s\left(\sum_{i\in\mathcal{H}} r_i + \sum_{i\in C} r_i\right) = -s \sum_{i\in C} r_i.$$

    Since the reduction knows $r_i \in \mathbb{Z}_p$ for all $i \in C$ (recall we are working in the registered-key model), it can simulate the attribute-independent ciphertext component as $[s_1 y - s\hat{r}]_1 = -\sum_{i\in C} r_i [s]_1 = -\sum_{i\in C} r_i [\beta]_1$.
  - For the attribute-specific components, we first have that

    $$\hat{u}_{a_k} = F_{a_k}(\tau) = \prod_{i\in[N]: a_k \notin S_i} (\tau - i).$$

Now, by the (information-theoretic) security of the secret sharing scheme, for every corrupted index $i \in C$, there exists a secret share $(\tilde{s}_{i,1}, \ldots, \tilde{s}_{i,K})$ of 1 with respect to the policy P where $\tilde{s}_{i,k} = 0$ whenever $a_k \in S_i$. This is because S_i does not satisfy the policy so the subset of shares where $a_k \in S_i$ must be consistent with *every* possible secret. The reduction now implicitly sets the shares of $(s_{2,1}, \ldots, s_{2,K})$ to be

$$s_{2,k} = \sum_{i \in C} \frac{\tilde{s}_{i,k}}{\tau - i}.$$

Since each $(\tilde{s}_{i,1}, \ldots, \tilde{s}_{i,K})$ is a secret share of 1, we can appeal to linearity of the secret sharing scheme to conclude that $(s_{2,1}, \ldots, s_{2,K})$ is a valid secret sharing of $\sum_{i \in C} \frac{1}{\tau - i} = s_2$, as required. Finally, to complete the proof, the reduction sets $s'_k = \gamma\beta \sum_{i \in C : a_k \notin S_i} \frac{\tilde{s}_{i,k}}{\tau - i}$. Since $\tilde{s}_{i,k} = 0$ whenever $a_k \in S_i$ (and $y = \gamma\beta$), this means

$$s'_k = \gamma\beta \sum_{i \in C : a_k \notin S_i} \frac{\tilde{s}_{i,k}}{\tau - i} = \gamma\beta \sum_{i \in C} \frac{\tilde{s}_{i,k}}{\tau - i} = s_{2,k} y,$$

which now yields our desired cancellation: $s_{2,k} y - s'_k = 0$. This allows the reduction to simulate $[s_{2,k} y - s'_k]_1$. For the remaining attribute-specific component, we have

$$s'_k \hat{u}_{a_k} = \gamma\beta \sum_{i \in C : a_k \notin S_i} \frac{\tilde{s}_{i,k}}{\tau - i} \cdot \prod_{j \in [N] : a_k \notin S_j} (\tau - j) = \gamma\beta \sum_{i \in C : a_k \notin S_i} \tilde{s}_{i,k} \cdot f_{i,k}(\tau),$$

where each $f_{i,k}$ is a polynomial of degree at most $N - 2$. This is because $\hat{u}_{a_k}$ has degree at most $N - 1$ since the aggregation algorithm constructs $\hat{u}_{a_k}$ only if there is at least one index $i \in [N]$ such that $a_k \in S_i$. Thus, the reduction can use the components $\{[\gamma\beta\tau^j]_2\}_{j \in [0, N-2]}$ to simulate $[s'_k \hat{u}_{a_k}]_2$.

– Let $[z]_T$ be the challenge element. The reduction sets the encryption of $[\mu]_T$ to be $[\mu]_T - [z]_T$. Observe that if $z = \gamma\beta^2\tau^{N-1}$, then for the above choice of variables ($s = \beta$ and $\alpha = -\gamma\beta\tau^{N-1}$), this means $\mu - z = \mu + s\alpha$, and the ciphertext is distributed exactly as in the real scheme. Conversely if $z \xleftarrow{R} \mathbb{Z}_p$, then the ciphertext perfectly hides the message.

The above sketch illustrates how to use the components from the assumption to simulate the corresponding components in the scheme. We refer to the proof of Theorem 4.6 for the formal description (that properly re-randomizes the CRS, public keys, and ciphertexts).

Adaptive security in the random oracle model. Technically, our reduction strategy described above proves a slightly stronger notion of security: adaptive security where the adversary cannot make any corruption queries (i.e., ask to see the secret keys for a user with an honestly-generated key). Previously, the work of [FWW23] described a generic transformation that takes any registered ABE scheme that is secure in a model without corruptions and transforms it into a scheme that is adaptively secure in the random oracle model. Strictly speaking, the [FWW23] transformation only yields a scheme that is selectively secure in the policy. When the size of the policy is *a priori* bounded, we can use standard complexity leveraging to obtain a scheme that is adaptively secure (see Corollary 4.14). Taken together, our work also yields the first pairing-based adaptively secure registered ABE scheme with a linear-size CRS that supports monotone Boolean formulas.

Extending to a large index space and its benefits. In the scheme described above, each user's public key is associated with a slot index $i \in [N]$. However, there is no restriction that the indices have to be drawn from the set $\{1, \ldots, N\}$. Indeed, each user could have chosen an *arbitrary* index $i \in \mathbb{Z}_p$, and as long as every user picks a distinct index, aggregation is successful. If $I \subseteq \mathbb{Z}_p$ represents the set of identities associated with the different users, then we would modify the above aggregation algorithm to set $Z(X) = \prod_{i \in I}(X - i)$ as the polynomial that vanishes on the users' indices. The user polynomial $L_i(X)$ from Eq. (1.2) and the attribute polynomial $F_a(X)$ from Eq. (1.3) would be defined analogously with respect to I instead. Namely, $L_i(X) = \prod_{j \in I \setminus \{i\}}(X - j)$ and $F_a(X) = \prod_{j \in I : a \notin S_j}(X - j)$. Essentially, the basic version described above corresponds to the special case where $I = [N]$. When proving security (for a specific set

of identities I), we consider a modified version of the assumption where $\beta = \sum_{i \in I} \frac{1}{\tau - i}$ and the assumption gives out $\left[\frac{\gamma}{\tau - i_1}\right]_1, \ldots, \left[\frac{\gamma}{\tau - i_N}\right]_1$ where $i_1, \ldots, i_N \in I$ are the elements of I. We state the formal version in Assumption 4.4. Since the index set I is hardcoded into the assumption, we only prove security against adversaries that commit to the index set in advance. For this reason, our scheme for a large index space (Corollary 4.13) does not satisfy full adaptive security.

As noted earlier, the advantage of a slotted registered ABE scheme with a large index space is that users do not have to coordinate when generating their public key. They can simply pick a random slot index $i \xleftarrow{R} \mathbb{Z}_p$ and generate keys with respect to index i. With overwhelming probability, their chosen indices will be distinct from those chosen by other users. In previous schemes with a smaller index space, if users chose random indices, there would be a good chance that their choices would collide and their keys could not be aggregated together. One way around this is to have some coordination with the key curator when generating keys to avoid such collisions. However, this has limitations. For example, a user might want to generate a single public key and have it be used across multiple registered ABE systems run by different key curators. To get around this coordination, prior works like that of [GLWW23] developed a transformation where a public key would consist of multiple public keys for different random indices in the base system. At aggregation time, the key curator would run a matching algorithm to associate each user with a distinct index (from the ones they made available) so that none collide. This incurs a logarithmic blowup in key sizes and has the added complexity of needing to implement the matching algorithm. In contrast, if the scheme supports a large index space, users can simply choose a random index and know with very high probability that there will be no collisions.

For completeness, we show in Appendix B that the same powers-of-two-style transformation from [GHMR18, HLWW23] suffices to lift a slotted registered ABE scheme with large indices to a standard registered ABE scheme (that supports dynamic registration). The transformed scheme automatically supports stateless key-generation as described above. We note that our transformation from the statically secure scheme to the adaptively secure one only applies in the setting with a small index space. Achieving adaptive security with a large index space (while retaining our other efficiency properties) is an interesting challenge.

1.2 Additional Related Work

Registration-based cryptography. The goal of registration-based cryptography is to enable expressive encryption capabilities without needing to rely on any central authority. This paradigm has been successfully integrated into a multitude of settings, including notions like registration-based encryption (i.e., registered identity-based encryption) [GHMR18, GHM+19, GV20, CES21, GKMR23, DKL+23, FKdP23], registered ABE (see Table 1), registered functional encryption [FFM+23, DPY24, PST25], distributed broadcast encryption [WQZD10, BZ14, KMW23, FWW23, GKPW24, CW24, CHW25, WW25, AMR26, GY26b, GY26a], threshold encryption with silent setup [GKPW24, GHK+25, WW26, GWWW26], and distributed monotone-policy encryption [ADM+24, DJWW25, CW26, AGY26].

Cross-term cancellation and polynomials. The cross-term cancellation technique underlying pairing-based registered ABE schemes (e.g., [HLWW23]) has facilitated many other pairing-based cryptographic primitives, including vector commitments [CF13], batch arguments [WW22, GLWW24, CEW25], and aggregate signatures [HWW25]. Because of the need to publish cross terms, many of these constructions (e.g., [CF13, WW22, HLWW23, HWW25]) require a quadratic-size CRS. A common theme in many of these settings is that subsequent constructions show that by encoding things as polynomials it is possible to avoid needing to publish quadratically-many cross terms. This idea of working in the Lagrange polynomial basis to enable more efficient aggregation has been used in a variety of settings, including distributed threshold encryption [DP08], identity-based broadcast encryption [DPP07, Del07], polynomial commitments [KZG10], locally-verifiable signatures [GV22], and batch arguments for NP [CEW25]. We expect these ideas to provide a general template for reducing the CRS size in other cryptographic schemes that currently rely on a cross-term cancellation technique (which is the case for many pairing-based registration-based cryptographic schemes).

Comparison with [GHK+25]. The work of [GHK+25] proposes a modular framework for building registration-based encryption schemes by designing special-purpose witness encryption schemes for specific relations. Notably, their work provides the first registered ABE scheme with a linear-size CRS that supports the class of conjunctions (and by concatenation, the class of DNF policies). To obtain the registered ABE scheme for conjunctions, they rely on a special-purpose witness encryption for a set membership policy, and they show how to express a conjunction

policy as a set membership policy. Since it is unclear how to encode more complex policies like a monotone Boolean formula as a set membership policy, we cannot directly apply their framework to obtain registered ABE for policies beyond conjunctions (and DNFs). Moreover, the security analysis in their framework critically relies on extractable witness encryption, which in turn necessitates the use of the generic group model. In this work, we show how to directly obtain an algebraic construction of registered ABE that can support general policies (i.e., monotone span programs) and whose security can be based on a q-type assumption in the plain model.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. For integers $m, n \in \mathbb{Z}$ with $m \leq n$, we write $[m, n] \coloneqq \{m, \ldots, n\}$ and, provided $n \in \mathbb{N}$, $[n] \coloneqq [1, n]$. For a modulus $p \in \mathbb{N}$, we write $\mathbb{Z}_p$ to denote the ring of integers modulo p. We let $\{0,1\}^k$ denote bit-strings of length k and assume they can be implicitly interpreted as elements of $\{0, 1, \ldots, 2^k - 1\}$. For a set S, we write $\{x_i\}_{i \in S}$ to denote the set of *pairs* $\{(i, x_i) : i \in S\}$. In particular, $\{x_i\}_{i \in S}$ specifies both the set S and the association between each index $i \in S$ and the corresponding element x_i. We call $\{x_i\}_{i \in S}$ a sequence if either $S = [n]$ for some $n \in \mathbb{N}$ or $S = \mathbb{N}$. When S is a finite set, we write $x \xleftarrow{R} S$ to denote a uniform random sample from S. When $\mathcal{D}$ is a distribution (or a probabilistic algorithm), we write $x \leftarrow \mathcal{D}$ to denote a random draw from $\mathcal{D}$. For a polynomial $F(X)$ of degree at most N over $\mathbb{Z}_p$, we define its coefficient vector $\mathbf{f} = (f_1, f_2, \ldots, f_{N+1}) \in \mathbb{Z}_p^{N+1}$ such that $F(X) = \sum_{i \in [N+1]} f_i X^{i-1}$.

We use boldface lowercase letters (e.g., $\mathbf{u}, \mathbf{v}$) to denote vectors and boldface uppercase letters (e.g., $\mathbf{A}, \mathbf{B}$) to denote matrices. We use non-boldface letters to denote their components; namely, we write $\mathbf{u} = (u_1, \ldots, u_n)$. We write $\mathsf{poly}(\lambda)$ to denote a function that is upper bounded by a fixed polynomial in λ and $\mathsf{negl}(\lambda)$ to denote a function that is $o(\lambda^{-c})$ for all constants $c \in \mathbb{N}$. We say an algorithm is efficient if it runs in probabilistic polynomial time in the length of its input. We say two (ensembles of) distributions $\mathcal{D}_0 = \{\mathcal{D}_{0,\lambda}\}_{\lambda \in \mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{1,\lambda}\}_{\lambda \in \mathbb{N}}$ are computationally indistinguishable if, for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{0,\lambda}] - \Pr[\mathcal{A}(1^\lambda, x) = 1 : x \leftarrow \mathcal{D}_{1,\lambda}]| = \mathsf{negl}(\lambda).$$

Prime-order pairing groups. Throughout this work, we use (asymmetric) prime-order pairing groups.

Definition 2.1 (Prime-Order Pairing Group)**.** An (asymmetric) prime-order pairing group consists of an efficient algorithm GroupGen that takes as input the security parameter 1^λ and outputs a description of a pairing group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e)$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ are cyclic groups of prime order p satisfying $2^{2\lambda} < p = 2^{\Theta(\lambda)}$, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$ is an efficiently computable non-degenerate bilinear map, and g_1, g_2 are generators of $\mathbb{G}_1, \mathbb{G}_2$, respectively. We require that the group operations in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ are efficiently computable. For ease of exposition, we assume that there is a fixed function $p(\lambda)$ such that for each security parameter λ, $\mathsf{GroupGen}(1^\lambda)$ outputs a group of order $p = p(\lambda)$.

Implicit notation. Throughout this work, we will describe group elements using implicit notation [EHK+13]. Specifically, fix a pair of generators $\hat{g}_1$ and $\hat{g}_2$ for $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively. For a matrix $\mathbf{A} \in \mathbb{Z}_p^{n \times m}$, we write $[\mathbf{A}]_1, [\mathbf{A}]_2, [\mathbf{A}]_\mathsf{T}$ to denote $\hat{g}_1^\mathbf{A}, \hat{g}_2^\mathbf{A}, e(\hat{g}_1, \hat{g}_2)^\mathbf{A}$, respectively, where exponentiation is defined component-wise. For matrices $\mathbf{A}, \mathbf{B}$ with compatible dimensions, we define $[\mathbf{A}]_1 \cdot [\mathbf{B}]_2 \coloneqq [\mathbf{AB}]_\mathsf{T}$, where the pairing is used to compute the pairwise products between the elements in $[\mathbf{A}]_1$ and $[\mathbf{B}]_2$. We will often work relative to a *random* choice of generators $\hat{g}_1 \in \mathbb{G}_1, \hat{g}_2 \in \mathbb{G}_2$ (as opposed to the generators g_1, g_2 output by GroupGen). We denote this by writing $[1]_1 \coloneqq \hat{g}_1$ and $[1]_2 \coloneqq \hat{g}_2$.

Linear secret sharing schemes. Let S be a set of parties and let 2^S denote the power set of S. An access policy $P : 2^S \to \{0,1\}$ is a function that specifies for each set $T \subseteq S$ whether it satisfies the policy (i.e., $P(T) = 1$) or not (i.e., $P(T) = 0$). We now recall the notion of a linear secret sharing scheme that realizes an access policy.

Definition 2.2 (Linear Secret Sharing Scheme [Bei96, adapted])**.** Let $p \in \mathbb{N}$ be a prime and S be a set of parties. A linear secret sharing scheme for S over the field $\mathbb{Z}_p$ is a pair $(\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_p^{\ell \times n}$ is a "share-generating" matrix and $\rho : [\ell] \to S$ is a "row-labeling" function. For a set $T \subseteq S$, let $I_T \coloneqq \{i \in [\ell] : \rho(i) \in T\}$ be the row indices associated with T and let $\mathbf{M}_T \in \mathbb{Z}_p^{|I_T| \times n}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I_T. The pair $(\mathbf{M}, \rho)$ defines an access policy as follows:

- **Authorized sets:** We say a subset $T \subseteq S$ satisfies the policy if there exists a vector $\boldsymbol{\omega}_T \in \mathbb{Z}_p^{|I_T|}$ such that $\boldsymbol{\omega}_T^\intercal \mathbf{M}_T = \mathbf{e}_1^\intercal$, where $\mathbf{e}_1 \in \mathbb{Z}_p^n$ is the first elementary basis vector.
- **Unauthorized sets:** We say a subset $T \subseteq S$ does not satisfy the policy if $\mathbf{e}_1^\intercal$ is not in the row-span of $\mathbf{M}_T$. Equivalently, there exists a vector $\mathbf{v}^* \in \mathbb{Z}_p^n$ with $v_1^* = 1$ such that $\mathbf{M}_T \mathbf{v}^* = \mathbf{0}$ (i.e., the vector $\mathbf{v}^*$ is orthogonal to the rows of $\mathbf{M}_T$ associated with the parties in T).

To share a value $s \in \mathbb{Z}_p$ with respect to a policy $(\mathbf{M}, \rho)$, sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_p$ and define the vector $\mathbf{v} = (s, v_2, \ldots, v_n)^\intercal$. Then, $\mathbf{u} = \mathbf{M}\mathbf{v}$ is the vector of shares where $u_i \in \mathbb{Z}_p$ belongs to party $\rho(i) \in S$ for each $i \in [\ell]$.

In this work, we consider access policies that can be described by a linear secret sharing scheme. Formally, this is equivalent to the class of policies that can be described by a monotone span program. In particular, this includes the class of threshold policies [Sha79] as well as the class of monotone Boolean formulas [BL88, LW11].

3 Registered Attribute-Based Encryption

Following [HLWW23], we focus on a simpler notion of *slotted* registered ABE, where each user in the system is assigned a *slot index* used for key generation, and the key curator runs an *aggregation* algorithm that takes all users' slot indices, public keys, and attributes and outputs a master public key. This can be compiled into a standard registered ABE scheme that supports dynamic registration [HLWW23]. For completeness, we include the usual definition of registered ABE in Appendix A. Then, in Appendix B, we show how to adapt the transformation of [HLWW23] from slotted registered ABE to standard registered ABE in the large-index setting. As we show there, the ability to support large indices immediately implies support for stateless key generation, in which users can sample their public key without knowledge of the current state of the system.

3.1 Slotted Registered Attribute-Based Encryption

We recall the syntax and definitions for slotted registered ABE. Our definition is adapted from the corresponding definition in [HLWW23], but generalized to support an arbitrary index space (e.g., an index space of exponential size). Specifically, in a slotted registered ABE scheme, each public key is associated with an index $\mathsf{id} \in \mathcal{I}$ from an index space $\mathcal{I}$, and we only support aggregation of public keys associated with distinct indices. In [HLWW23], the index space was fixed to the integers $\mathcal{I} = \{1, \ldots, N\}$, where N is the total number of users in the system. This means that users must either coordinate to generate their keys with respect to distinct indices, or alternatively, users have to generate keys for multiple indices so that there is a way to associate each user with a unique index. In this work, we consider a more general setting where the index space can have exponential size, and we support aggregating any collection of N public keys as long as each one is still associated with a distinct index. Observe that in this setting, each of the $N = \mathsf{poly}(\lambda)$ users can generate their key with respect to a *random* index, and with overwhelming probability, all of the keys will have a distinct index. Supporting a large index space is thus useful for enabling stateless key generation, in which users can generate their keys independently of all other users in the system. We now give the formal definition:

Definition 3.1 (Slotted Registered ABE)**.** Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ be an attribute universe and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of policies over $\mathcal{U}$. Let $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda \in \mathbb{N}}$ be an index space. A slotted registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and index space $\mathcal{I}$ is a tuple of algorithms $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{crs}$: On input the security parameter λ and a number of slots N, the setup algorithm outputs a common reference string crs. We assume that crs implicitly contains the security parameter 1^λ and a description of the message space $\mathcal{M}_\lambda$ associated with the scheme.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}) \to (\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$: On input the common reference string crs and an index $\mathsf{id} \in \mathcal{I}_\lambda$, the key-generation algorithm outputs a public key $\mathsf{pk}_{\mathsf{id}}$ and a secret decryption key $\mathsf{sk}_{\mathsf{id}}$ for index id.
- $\mathsf{IsValid}(\mathsf{crs}, \mathsf{id}, \mathsf{pk}_{\mathsf{id}}) \to b$: On input the common reference string crs, an index $\mathsf{id} \in \mathcal{I}_\lambda$, and a public key $\mathsf{pk}_{\mathsf{id}}$, the key-validation algorithm outputs a bit $b \in \{0, 1\}$ indicating whether or not $\mathsf{pk}_{\mathsf{id}}$ is valid. This algorithm is deterministic. In certain instances, this algorithm may be omitted.

- $\mathsf{Aggregate}(\mathsf{crs}, \{(\mathsf{pk}_{\mathsf{id}}, S_{\mathsf{id}})\}_{\mathsf{id}\in I}) \to (\mathsf{mpk}, \{\mathsf{hsk}_{\mathsf{id}}\}_{\mathsf{id}\in I})$: On input the common reference string crs and a collection of public keys and associated attributes $\{(\mathsf{pk}_{\mathsf{id}}, S_{\mathsf{id}})\}_{\mathsf{id}\in I}$ where $|I| = N$, the aggregation algorithm outputs the master public key mpk and a collection of helper decryption keys $\{\mathsf{hsk}_{\mathsf{id}}\}_{\mathsf{id}\in I}$. This algorithm is deterministic. We assume that the master public key mpk and the helper decryption keys $\mathsf{hsk}_{\mathsf{id}}$ implicitly contain (from crs) the security parameter 1^λ and a description of the message space $\mathcal{M}_\lambda$ associated with the scheme.
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu) \to \mathsf{ct}$: On input the master public key mpk, an access policy $P \in \mathcal{P}_\lambda$, and a message $\mu \in \mathcal{M}_\lambda$, the encryption algorithm outputs a ciphertext ct.
- $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{hsk}_{\mathsf{id}}, \mathsf{ct}) \to \mu$: On input a secret key $\mathsf{sk}_{\mathsf{id}}$, a helper decryption key $\mathsf{hsk}_{\mathsf{id}}$, and a ciphertext ct, the decryption algorithm outputs either a message $\mu \in \mathcal{M}_\lambda$ or a special symbol $\mu = \perp$ to indicate decryption failure.

Definition 3.2 (Completeness)**.** Let $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a slotted registered ABE scheme with index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. We say it is *complete* if, for all parameters $\lambda, N \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, and all indices $\mathsf{id} \in \mathcal{I}_\lambda$, the following holds:

$$\Pr\left[\mathsf{IsValid}(\mathsf{crs}, \mathsf{id}, \mathsf{pk}_{\mathsf{id}}) = 1 : (\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id})\right] = 1.$$

Definition 3.3 (Correctness)**.** Let $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a slotted registered ABE scheme with index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. We say it is *correct* if, for all parameters $\lambda, N \in \mathbb{N}$, all but a negligible fraction of crs in the support of $\mathsf{Setup}(1^\lambda, 1^N)$, all index sets $I \subseteq \mathcal{I}_\lambda$ where $|I| = N$, all indices $\mathsf{id} \in I$, $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id})$, all collections of public keys $\{\mathsf{pk}_{\mathsf{id}'} : \mathsf{IsValid}(\mathsf{crs}, \mathsf{id}', \mathsf{pk}_{\mathsf{id}'}) = 1\}_{\mathsf{id}'\in I\setminus\{\mathsf{id}\}}$, all messages $\mu \in \mathcal{M}_\lambda$ where $\mathcal{M}_\lambda$ is the message space associated with crs, all attribute sets $\{S_{\mathsf{id}'}\}_{\mathsf{id}'\in I}$ with each $S_{\mathsf{id}'} \subseteq \mathcal{U}_\lambda$, and all policies $P \in \mathcal{P}_\lambda$ such that S_{id} satisfies P, the following holds:

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{hsk}_{\mathsf{id}}, \mathsf{ct}) = \mu : \begin{array}{c}(\mathsf{mpk}, \{\mathsf{hsk}_{\mathsf{id}'}\}_{\mathsf{id}'\in I}) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, \{(\mathsf{pk}_{\mathsf{id}'}, S_{\mathsf{id}'})\}_{\mathsf{id}'\in I}) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P, \mu)\end{array}\right] = 1.$$

Definition 3.4 (Compactness)**.** Let $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ be a slotted registered ABE scheme. We say it is *compact* if there exists a universal polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that the master public key mpk and each helper decryption key hsk output by Aggregate are $\mathsf{poly}(\lambda, |A|, \log N)$ size, where A is the union of all attribute sets input to aggregation.

Definition 3.5 (Adaptive Security)**.** Let $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ 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 index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. The adaptive security game between an adversary $\mathcal{A}$ and a challenger is parameterized by a positive integer-valued polynomial $N = N(\lambda)$, a bit $b \in \{0,1\}$, and a security parameter λ; it consists of the following phases:

- **Setup phase:** The challenger samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$, which it provides to the adversary $\mathcal{A}$. It additionally initializes a counter $\mathsf{ctr} = 0$, an empty set C to keep track of corrupted users, and an empty dictionary D to keep track of key-generation queries.
- **Query phase:** During the query phase, the adversary $\mathcal{A}$ can make the following queries:
  - **Key-generation query:** In a key-generation query, the adversary specifies an index $\mathsf{id} \in \mathcal{I}_\lambda$. In response, the challenger generates $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id})$ and provides $\mathsf{pk}_{\mathsf{id}}$ to $\mathcal{A}$. It also increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and adds the mapping $\mathsf{D}[\mathsf{ctr}] = (\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$ to the dictionary.
  - **Corruption query:** In a corruption query, the adversary specifies a counter value $c \in [\mathsf{ctr}]$. In response, the challenger looks up the tuple $(\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}) = \mathsf{D}[c]$ and gives $\mathsf{sk}_{\mathsf{id}}$ to the adversary.
- **Challenge phase:** For each $i \in [N]$, the adversary $\mathcal{A}$ specifies a tuple $(c_i, \mathsf{id}_i^*, S_i^*, \mathsf{pk}_i^*)$ where either $c_i \in [\mathsf{ctr}]$ to reference a challenger-generated key or $c_i = \perp$ to reference a key outside this set. The adversary additionally submits a challenge policy $P^* \in \mathcal{P}_\lambda$ and messages $\mu_0^*, \mu_1^* \in \mathcal{M}_\lambda$ where $\mathcal{M}_\lambda$ is the message space associated with crs. First, the challenger defines $I = \{\mathsf{id}_i^* : i \in [N]\}$. If $|I| \neq N$ (i.e., the adversary specified two public keys for the same slot index), the challenger halts with output 0. Next, for each $i \in [N]$, the challenger proceeds as follows:

- If $c_i \in [\mathsf{ctr}]$, the challenger retrieves $(\mathsf{id}_i, \mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{D}[c_i]$. If $\mathsf{id}_i \neq \mathsf{id}_i^*$, the challenger halts with output 0. Otherwise, the challenger sets $\mathsf{pk}_{\mathsf{id}_i} = \mathsf{pk}_i$ and $S_{\mathsf{id}_i} = S_i^*$. If the adversary previously issued a corruption query on counter c_i, then the challenger adds the index id_i to C.
- If $c_i = \perp$, then the challenger sets $\mathsf{id}_i = \mathsf{id}_i^*$, $\mathsf{pk}_{\mathsf{id}_i} = \mathsf{pk}_i^*$, and $S_{\mathsf{id}_i} = S_i^*$. It then adds id_i to C.

For each index $\mathsf{id} \in I$, the challenger also checks that $\mathsf{IsValid}(\mathsf{crs}, \mathsf{id}, \mathsf{pk}_{\mathsf{id}}) = 1$ and halts with output 0 if any check fails. If all checks pass, the challenger computes $(\mathsf{mpk}, \{\mathsf{hsk}_{\mathsf{id}}\}_{\mathsf{id}\in I}) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, \{(\mathsf{pk}_{\mathsf{id}}, S_{\mathsf{id}})\}_{\mathsf{id}\in I})$ and a challenge ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P^*, \mu_b^*)$. The challenger provides ct to $\mathcal{A}$.

- **Output phase:** At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit b^*, which is the output of the game.

We say an adversary $\mathcal{A}$ is admissible for the adaptive security game if, for all $\mathsf{id} \in C$, the set S_{id} does not satisfy P^* (i.e., the attributes associated with corrupted indices do not satisfy the challenge policy). Next, we define the adversary's advantage in the adaptive security game with parameter N to be

$$\mathsf{Adv}_{\mathcal{A},N}(\lambda) \coloneqq |\Pr[b^* = 1 : b = 0] - \Pr[b^* = 1 : b = 1]|.$$

Finally, we say that Π_{sRABE} is adaptively secure if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all efficient admissible adversaries $\mathcal{A}$ and all positive integer-valued polynomials $N = N(\lambda)$, there exists a positive polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\mathsf{Adv}_{\mathcal{A},N}(\lambda) \leq \mathsf{poly}(\lambda) \cdot \mathsf{negl}(\lambda). \tag{3.1}$$

Definition 3.6 (Relaxed Notions of Security)**.** Let Π_{sRABE} be a slotted registered ABE scheme with policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ and index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. We will also consider the following relaxed variants of adaptive security:

- **Index-set-selective security:** For a positive integer-valued polynomial $N = N(\lambda)$, a collection of index sets $I = \{I_\lambda\}_{\lambda\in\mathbb{N}}$ where each $I_\lambda \subseteq \mathcal{I}_\lambda$ and $|I_\lambda| = N(\lambda)$, a bit $b \in \{0, 1\}$, and a security parameter λ, we define the index-set-selective security game exactly as the adaptive security game in Definition 3.5, except we impose the following additional restrictions on the adversary $\mathcal{A}$:
  - The adversary can only make key-generation queries on indices $\mathsf{id} \in I_\lambda$.
  - In the challenge phase, the adversary can only specify tuples $(c_i, \mathsf{id}_i^*, S_i^*, \mathsf{pk}_i^*)$ where $\mathsf{id}_i^* \in I_\lambda$.

  In other words, the set of indices is fixed to be I_λ in the index-set-selective security game. Then, we say that Π_{sRABE} satisfies index-set-selective security if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all polynomials $N = N(\lambda)$, all collections of index sets $I = \{I_\lambda\}_{\lambda\in\mathbb{N}}$ where $|I_\lambda| = N(\lambda)$, and all efficient admissible adversaries $\mathcal{A}$, there exists a positive polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

  $$\mathsf{Adv}_{\mathcal{A},N,I}(\lambda) \coloneqq |\Pr[b^* = 1 : b = 0] - \Pr[b^* = 1 : b = 1]| \leq \mathsf{poly}(\lambda) \cdot \mathsf{negl}(\lambda)$$

  in the index-set-selective security game with parameters (N, I).

- **Policy-selective security:** We define the policy-selective security game to be the adaptive security game from Definition 3.5, except we require that the adversary $\mathcal{A}$ declare the challenge policy $P^* \in \mathcal{P}_\lambda$ at the beginning of the security game (before the setup phase). We say Π_{sRABE} satisfies policy-selective security if Eq. (3.1) holds for all efficient admissible adversaries $\mathcal{A}$ and all polynomials $N = N(\lambda)$ in the policy-selective security game.

- **Adaptive security without corruptions:** We define the adaptive security without corruptions game to be the adaptive security game except we do not allow the adversary to make any corruption queries during the query phase. We then say Π_{sRABE} satisfies adaptive security without corruptions if Eq. (3.1) holds for all efficient admissible adversaries $\mathcal{A}$ and all polynomials $N = N(\lambda)$ in the adaptive security without corruptions game.

Combinations of these relaxations are defined in the natural way. For instance, in analyzing a scheme that satisfies index-set-selective security without corruptions, we fix the index set $I = \{I_\lambda\}_{\lambda\in\mathbb{N}}$ in the security game and disallow the adversary from making any corruption queries.

Definition 3.7 (Large Index Space)**.** Let Π_{sRABE} be a slotted registered ABE scheme with index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. We say that Π_{sRABE} has a *large index space* if $1/|\mathcal{I}_\lambda| = \mathsf{negl}(\lambda)$.

Remark 3.8 (Amplifying Security)**.** We now describe two known techniques for amplifying the security of a (slotted) registered ABE scheme:

- **Security without corruptions to security with corruptions.** The work of [FWW23] describes a generic transformation that compiles a slotted registered ABE scheme with policy-selective security *without* corruptions into a slotted registered ABE scheme with policy-selective security that supports corruptions in the random oracle model. The transformation of [FWW23] only incurs constant overhead in the scheme parameters.
- **Policy-selective security to adaptive security.** If the policy size is *a priori* bounded, and we additionally have a policy-selective slotted registered ABE scheme with sub-exponential security, then using standard complexity leveraging (where the reduction guesses the adversary's policy at the beginning of the security game), we can obtain an adaptively secure slotted registered ABE scheme. Complexity leveraging will scale up all of the scheme parameters by a $\mathsf{poly}(K)$ factor, where K is the bound on the policy size.

By composing the two transformations, we can generically lift a slotted registered ABE scheme that is secure without corruptions into an adaptively secure slotted registered ABE scheme in the random oracle model. Note that we cannot directly apply complexity leveraging to go from a scheme without corruptions into a scheme with corruptions; this is because the naïve complexity leveraging strategy of guessing the indices of the corrupted users would incur a 2^N loss in the security reduction, where N is the number of users. As a result, complexity leveraging would incur a $\mathsf{poly}(N)$ overhead in the scheme parameters, and the resulting scheme no longer satisfies the efficiency requirements of slotted registered ABE. For this reason, we need both transformations to lift a scheme that has security without corruptions into one that is adaptively secure.

Registered-key model. To simplify the exposition, we analyze correctness and security of our construction in the *registered-key model* [RY07], which assumes that all of the public keys appearing in the correctness and security games are in the support of the key-generation algorithm. It is straightforward to compile a scheme with correctness and security in the registered-key model into a scheme that satisfies the full correctness and security requirements in Definitions 3.3 and 3.5 by having the user include a (simulation-sound) non-interactive zero-knowledge (NIZK) proof of knowledge of the associated secret key. Aggregation would only proceed if all of the public keys come with an accepting NIZK proof. The work of [LWW25] provides a formal proof of this transformation in the setting of multi-authority registered ABE. Specializing their analysis to a single authority yields a proof for standard registered ABE (in the registered-key model). We note that previous registered ABE schemes [HLWW23, ZZGQ23, GLWW24] essentially incorporated an algebraic NIZK proof of knowledge as part of the scheme itself; we believe a similar mechanism could be integrated into our scheme as well to avoid reliance on the registered-key model. For ease of exposition, we elide this extra complication and instead focus on the core ideas that enable a registered ABE scheme with a linear-size CRS. We now give the correctness and security definitions in the registered-key model:

Definition 3.9 (Correctness in the Registered-Key Model)**.** We say a slotted registered ABE scheme Π_{sRABE} satisfies correctness in the registered-key model if, in the correctness definition (Definition 3.3), we replace the quantification over collections of public keys $\{\mathsf{pk}_{\mathsf{id}'} : \mathsf{IsValid}(\mathsf{crs}, \mathsf{id}', \mathsf{pk}_{\mathsf{id}'}) = 1\}_{\mathsf{id}'\in I\setminus\{\mathsf{id}\}}$ with quantification over collections $\{\mathsf{pk}_{\mathsf{id}'} : \mathsf{pk}_{\mathsf{id}'} \text{ is in the support of } \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}')\}_{\mathsf{id}'\in I\setminus\{\mathsf{id}\}}$.

Definition 3.10 (Adaptive Security in the Registered-Key Model)**.** We say a slotted registered ABE scheme Π_{sRABE} satisfies adaptive security in the registered-key model if, in the adaptive security definition (Definition 3.5), during the challenge phase, for every $i \in [N]$ for which the adversary's tuple $(c_i, \mathsf{id}_i^*, S_i^*, \mathsf{pk}_i^*)$ has $c_i = \bot$, the adversary must additionally specify the secret key sk_i^* and the randomness r_i^* such that $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) = \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}_i^*; r_i^*)$. Since all of the public keys in this experiment are in the support of the honest key-generation algorithm, the challenger no longer runs the IsValid predicate in the adaptive security game. We define the relaxed variants from Definition 3.6 in the registered-key model in an analogous manner.

Remark 3.11 (Removing the IsValid Algorithm in the Registered-Key Model)**.** When we work in the registered-key model, we will drop the IsValid algorithm from the description of the slotted registered ABE scheme. This is because,

per Definitions 3.9 and 3.10, we only need to consider correctness and security in the setting where all public keys are in the support of KeyGen, and thus are always valid.

4 Slotted Registered ABE with Linear-Size CRS

In this section, we show how to use asymmetric prime-order pairing groups to construct a slotted registered ABE scheme with a linear-size CRS in the registered-key model. Our scheme supports a large index space (i.e., $\{0,1\}^\lambda$) and any policy that can be described by a linear secret sharing scheme. We refer to Section 1.1 for an overview of our construction.

Construction 4.1 (Slotted Registered ABE)**.** Let λ be a security parameter. Let GroupGen be a prime-order asymmetric pairing-group generator that outputs groups of order $p = p(\lambda)$. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be an attribute universe and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be an ensemble where each $\mathcal{P}_\lambda$ is a collection of policies that can be described by a linear secret sharing scheme for $\mathcal{U}$ over $\mathbb{Z}_p$. We construct a slotted registered ABE scheme $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and index space $\mathcal{I} = \{\{0,1\}^\lambda\}_{\lambda\in\mathbb{N}}$ in the registered-key model as follows:

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and number of slots N, the setup algorithm proceeds as follows:
  - Sample a prime-order pairing group $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ as well as random generators $\hat{g}_1 \xleftarrow{\mathrm{R}} \mathbb{G}_1 \setminus \{g_1^0\}$ and $\hat{g}_2 \xleftarrow{\mathrm{R}} \mathbb{G}_2 \setminus \{g_2^0\}$. Define $[1]_1 \coloneqq \hat{g}_1$ and $[1]_2 \coloneqq \hat{g}_2$.
  - Sample random exponents $\alpha, \tau, y \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
  - Let $\boldsymbol{\tau} = (1, \tau, \tau^2, \ldots, \tau^{N-1}) \in \mathbb{Z}_p^N$ and $\mathbf{y} = (y, y\tau, y\tau^2, \ldots, y\tau^{N-2}, \alpha + y\tau^{N-1}) \in \mathbb{Z}_p^N$.

  Output the common reference string $\mathsf{crs} = (\mathcal{G}, [\alpha]_\mathsf{T}, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [y]_1, [\mathbf{y}]_2)$. The message space is $\mathcal{M}_\lambda \coloneqq \mathbb{G}_\mathsf{T}$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{id})$: On input the common reference string crs where $\mathsf{crs} = (\mathcal{G}, [\alpha]_\mathsf{T}, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [y]_1, [\mathbf{y}]_2)$ and an index $\mathsf{id} \in \{0,1\}^\lambda$, the key-generation algorithm first samples a random exponent $r_\mathsf{id} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Interpret id as an integer in $[0, 2^\lambda - 1]$ and parse $[\boldsymbol{\tau}]_2 = ([1]_2, [\tau]_2, [\tau^2]_2, \ldots, [\tau^{N-1}]_2)$. The algorithm computes

$$\begin{aligned}[\mathbf{r}'_\mathsf{id}]_2 &= \big(r_\mathsf{id}([\tau]_2 - \mathsf{id}[1]_2), r_\mathsf{id}([\tau^2]_2 - \mathsf{id}[\tau]_2), \ldots, r_\mathsf{id}([\tau^{N-1}]_2 - \mathsf{id}[\tau^{N-2}]_2)\big) \\ &= \big([r_\mathsf{id}(\tau - \mathsf{id})]_2, [r_\mathsf{id}(\tau - \mathsf{id})\tau]_2, \ldots, [r_\mathsf{id}(\tau - \mathsf{id})\tau^{N-2}]_2\big).\end{aligned}$$

  Finally, it outputs the public key $\mathsf{pk}_\mathsf{id} = ([r_\mathsf{id}]_1, [\mathbf{r}'_\mathsf{id}]_2)$ and the secret key $\mathsf{sk}_\mathsf{id} = r_\mathsf{id}$.
- $\mathsf{Aggregate}(\mathsf{crs}, \{(\mathsf{pk}_\mathsf{id}, S_\mathsf{id})\}_{\mathsf{id}\in I})$: On input the common reference string $\mathsf{crs} = (\mathcal{G}, [\alpha]_\mathsf{T}, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [y]_1, [\mathbf{y}]_2)$ and a collection of public keys $\mathsf{pk}_\mathsf{id} = ([r_\mathsf{id}]_1, [\mathbf{r}'_\mathsf{id}]_2)$ and associated attributes $S_\mathsf{id} \subseteq \mathcal{U}_\lambda$ for indices $\mathsf{id} \in I$ (where $|I| = N$), the aggregation algorithm proceeds as follows:
  - Compute the attribute-independent public key $[\hat{r}]_1 = \sum_{\mathsf{id}\in I}[r_\mathsf{id}]_1$.
  - Next, let $A = \bigcup_{\mathsf{id}\in I} S_\mathsf{id}$. For each attribute $a \in A$, define the polynomial

$$F_a(X) \coloneqq \prod_{\mathsf{id}\in I : a\notin S_\mathsf{id}} (X - \mathsf{id}).$$

    By construction, F_a is a polynomial of degree at most $N-1$ because a must be in at least one S_id. Let $\mathbf{f}_a \in \mathbb{Z}_p^N$ be its associated coefficient vector. Then, compute the attribute-specific public key

$$[\hat{u}_a]_2 = \mathbf{f}_a^\mathsf{T}[\boldsymbol{\tau}]_2 = [F_a(\tau)]_2.$$

  - Define the master public key to be $\mathsf{mpk} = (\mathcal{G}, [\alpha]_\mathsf{T}, [y]_1, [\hat{r}]_1, \{[\hat{u}_a]_2\}_{a\in A})$.

  Next, to compute the helper decryption keys, the aggregation algorithm proceeds as follows:
  - Define the vanishing polynomial $Z_I(X) = \prod_{\mathsf{id}\in I}(X - \mathsf{id})$ associated with I.

– For each $\mathsf{id} \in I$, define $L_{\mathsf{id}}(X) \coloneqq Z_I(X)/(X-\mathsf{id}) = \prod_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} (X - \mathsf{id}')$. By construction, L_{id} is a monic polynomial of degree at most $N-1$. Let $\boldsymbol{\ell}_{\mathsf{id}} \in \mathbb{Z}_p^N$ be its associated coefficient vector.
– For each distinct $\mathsf{id}, \mathsf{id}' \in I$, define $L_{\mathsf{id},\mathsf{id}'}(X) \coloneqq L_{\mathsf{id}}(X)/(X-\mathsf{id}') = \prod_{\mathsf{id}'' \in I \setminus \{\mathsf{id},\mathsf{id}'\}} (X - \mathsf{id}'')$. By construction, $L_{\mathsf{id},\mathsf{id}'}$ is a polynomial of degree at most $N-2$. Let $\boldsymbol{\ell}_{\mathsf{id},\mathsf{id}'} \in \mathbb{Z}_p^{N-1}$ be its associated coefficient vector.
– For each $\mathsf{id} \in I$, compute the attribute-independent decryption components

$$
\begin{aligned}
[\hat{v}_{\mathsf{id},1}]_2 &= \boldsymbol{\ell}_{\mathsf{id}}^\top [\boldsymbol{\tau}]_2 = [L_{\mathsf{id}}(\tau)]_2, \\
[\hat{v}_{\mathsf{id},2}]_2 &= \boldsymbol{\ell}_{\mathsf{id}}^\top [\mathbf{y}]_2 = [\alpha + y \cdot L_{\mathsf{id}}(\tau)]_2, \\
[\hat{v}_{\mathsf{id},3}]_2 &= \sum_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} \boldsymbol{\ell}_{\mathsf{id},\mathsf{id}'}^\top [\mathbf{r}'_{\mathsf{id}'}]_2 = \sum_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} [r_{\mathsf{id}'}(\tau - \mathsf{id}') \cdot L_{\mathsf{id},\mathsf{id}'}(\tau)]_2 = \sum_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} [r_{\mathsf{id}'} \cdot L_{\mathsf{id}}(\tau)]_2.
\end{aligned}
$$

– For each attribute $a \in S_{\mathsf{id}}$, define the polynomial $F_{a,\mathsf{id}}(X) \coloneqq L_{\mathsf{id}}(X)/F_a(X) = \prod_{\mathsf{id}' \in I \setminus \{\mathsf{id}\} : a \in S_{\mathsf{id}'}} (X - \mathsf{id}')$. By construction, $F_{a,\mathsf{id}}$ has degree at most $N-1$. Let $\mathbf{f}_{a,\mathsf{id}} \in \mathbb{Z}_p^N$ be its associated coefficient vector. Define the attribute-specific decryption component

$$
[\hat{w}_{a,\mathsf{id}}]_1 = \mathbf{f}_{a,\mathsf{id}}^\top [\boldsymbol{\tau}]_1 = [F_{a,\mathsf{id}}(\tau)]_1.
$$

– For each index $\mathsf{id} \in I$, let the helper decryption key be $\mathsf{hsk}_{\mathsf{id}} = (\mathcal{G}, [\hat{v}_{\mathsf{id},1}]_2, [\hat{v}_{\mathsf{id},2}]_2, [\hat{v}_{\mathsf{id},3}]_2, \{[\hat{w}_{a,\mathsf{id}}]_1\}_{a \in S_{\mathsf{id}}})$.

Output the master public key mpk and the helper decryption key $\mathsf{hsk}_{\mathsf{id}}$ for each $\mathsf{id} \in I$.

- $\mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{M}, \rho), [\mu]_\mathsf{T})$: On input the master public key $\mathsf{mpk} = (\mathcal{G}, [\alpha]_\mathsf{T}, [y]_1, [\hat{r}]_1, \{[\hat{u}_a]_2\}_{a \in A})$, a policy $(\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{\ell \times n}$ and $\rho \colon [\ell] \to A$ is a row-labeling function, and a message $[\mu]_\mathsf{T} \in \mathbb{G}_\mathsf{T}$, the encryption algorithm starts by sampling $s_1, s_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and sets $s = s_1 + s_2$. Then, it constructs the ciphertext components as follows:
  – **Message-embedding components:** Let $[c_1]_\mathsf{T} = s[\alpha]_\mathsf{T} + [\mu]_\mathsf{T}$ and $[c_2]_1 = [s]_1$.
  – **Index-specific component:** Let $[c_3]_1 = s_1[y]_1 - s[\hat{r}]_1$.
  – **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and define $\mathbf{v} = (1, v_2, \ldots, v_n)^\top$. For each $k \in [\ell]$, sample $s'_k \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Let $[c_{4,k}]_1 = (s_2 \mathbf{m}_k^\top \mathbf{v})[y]_1 - [s'_k]_1$ and $[c_{5,k}]_2 = s'_k [\hat{u}_{\rho(k)}]_2$ where $\mathbf{m}_k^\top$ is the k^{th} row of $\mathbf{M}$.

  Finally, output the ciphertext as follows:

$$
\mathsf{ct} = ((\mathbf{M}, \rho), [c_1]_\mathsf{T}, [c_2]_1, [c_3]_1, \{([c_{4,k}]_1, [c_{5,k}]_2)\}_{k \in [\ell]}). \tag{4.1}
$$

- $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{hsk}_{\mathsf{id}}, \mathsf{ct})$: On input a secret key $\mathsf{sk}_{\mathsf{id}} = r_{\mathsf{id}}$, a helper decryption key $\mathsf{hsk}_{\mathsf{id}}$, and a ciphertext ct where

$$
\begin{aligned}
\mathsf{hsk}_{\mathsf{id}} &= (\mathcal{G}, [\hat{v}_{\mathsf{id},1}]_2, [\hat{v}_{\mathsf{id},2}]_2, [\hat{v}_{\mathsf{id},3}]_2, \{[\hat{w}_{a,\mathsf{id}}]_1\}_{a \in S_{\mathsf{id}}}), \text{ and} \\
\mathsf{ct} &= ((\mathbf{M}, \rho), [c_1]_\mathsf{T}, [c_2]_1, [c_3]_1, \{([c_{4,k}]_1, [c_{5,k}]_2)\}_{k \in [\ell]}),
\end{aligned}
$$

  the decryption algorithm proceeds as follows:
  – If the user's set of attributes S_{id} is not authorized by $(\mathbf{M}, \rho)$, output $\perp$.
  – Otherwise, let $J = \{k \in [\ell] : \rho(k) \in S_{\mathsf{id}}\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes in the user's attribute set S_{id}. Write $J = \{k_1, \ldots, k_{|J|}\}$.
  – Now, let $\mathbf{M}_J$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by J. Since S_{id} is authorized, there exists a vector $\boldsymbol{\omega}_{S_{\mathsf{id}}} = (\omega_{S_{\mathsf{id}},1}, \ldots, \omega_{S_{\mathsf{id}},|J|}) \in \mathbb{Z}_p^{|J|}$ such that $\boldsymbol{\omega}_{S_{\mathsf{id}}}^\top \mathbf{M}_J = \mathbf{e}_1^\top$.
  – Compute the following values:

$$
\begin{aligned}
[d_{\mathsf{id}}]_\mathsf{T} &= \left(r_{\mathsf{id}}[c_2]_1 + [c_3]_1\right) \cdot [\hat{v}_{\mathsf{id},1}]_2 + [c_2]_1 \cdot [\hat{v}_{\mathsf{id},3}]_2, \\
[d_{\mathsf{attrib}}]_\mathsf{T} &= \sum_{j \in [|J|]} \omega_{S_{\mathsf{id}},j} \cdot \left([c_{4,k_j}]_1 \cdot [\hat{v}_{\mathsf{id},1}]_2 + [\hat{w}_{\rho(k_j),\mathsf{id}}]_1 \cdot [c_{5,k_j}]_2\right).
\end{aligned}
$$

  – Finally, compute and output

$$
[\mu]_\mathsf{T} = [c_1]_\mathsf{T} - [c_2]_1 \cdot [\hat{v}_{\mathsf{id},2}]_2 + [d_{\mathsf{id}}]_\mathsf{T} + [d_{\mathsf{attrib}}]_\mathsf{T}. \tag{4.2}
$$

Correctness and compactness. We now prove that Construction 4.1 satisfies correctness and compactness.

Theorem 4.2 (Correctness). *Construction 4.1 is correct in the registered-key model.*

Proof. Take any $\lambda, N \in \mathbb{N}$ and let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$. By construction, $\mathsf{crs} = (\mathcal{G}, [\alpha]_T, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [y]_1, [\mathbf{y}]_2)$ where $\boldsymbol{\tau} = (1, \tau, \tau^2, \ldots, \tau^{N-1}) \in \mathbb{Z}_p^N$, $\mathbf{y} = (y, y\tau, y\tau^2, \ldots, y\tau^{N-2}, \alpha + y\tau^{N-1}) \in \mathbb{Z}_p^N$, and $\mathcal{G}$ specifies a group of prime order $p \geq 2^{2\lambda}$. Now, suppose that $\tau \geq 2^\lambda$ (which occurs with probability $1 - 2^\lambda/p \geq 1 - 2^{-\lambda}$) and take any set of indices $I \subseteq \{0,1\}^\lambda$ with $|I| = N$. By our supposition, we have that $\tau \notin I$.

For each $\mathsf{id} \in I$, let $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$ be any public/secret key-pair in the support of $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{id})$. Then, we can write

$$\mathsf{pk}_{\mathsf{id}} = ([r_{\mathsf{id}}]_1, [\mathbf{r}'_{\mathsf{id}}]_2) \quad \text{and} \quad \mathsf{sk}_{\mathsf{id}} = r_{\mathsf{id}},$$

where $\mathbf{r}'_{\mathsf{id}} = (r_{\mathsf{id}}(\tau - \mathsf{id}), r_{\mathsf{id}}(\tau - \mathsf{id})\tau, \ldots, r_{\mathsf{id}}(\tau - \mathsf{id})\tau^{N-2})$. Let $\{S_{\mathsf{id}}\}_{\mathsf{id} \in I}$ be arbitrary attribute sets associated with the indices in I, where each $S_{\mathsf{id}} \subseteq \mathcal{U}_\lambda$, and let $A = \bigcup_{\mathsf{id} \in I} S_{\mathsf{id}}$. Now, let $(\mathsf{mpk}, \{\mathsf{hsk}_{\mathsf{id}}\}_{\mathsf{id} \in I}) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, \{(\mathsf{pk}_{\mathsf{id}}, S_{\mathsf{id}})\}_{\mathsf{id} \in I})$. By construction, the following holds:

- First, the master public key $\mathsf{mpk} = (\mathcal{G}, [\alpha]_T, [y]_1, [\hat{r}]_1, \{[\hat{u}_a]_2\}_{a \in A})$, where

$$\hat{r} = \sum_{\mathsf{id} \in I} r_{\mathsf{id}} \quad \text{and} \quad \hat{u}_a = \mathbf{f}_a^\top \boldsymbol{\tau} = \sum_{j \in [N]} f_{a,j} \tau^{j-1} = F_a(\tau),$$

  since $\mathbf{f}_a$ is the coefficient vector of the polynomial $F_a(X)$.

- Next, each helper decryption key $\mathsf{hsk}_{\mathsf{id}} = (\mathcal{G}, [\hat{v}_{\mathsf{id},1}]_2, [\hat{v}_{\mathsf{id},2}]_2, [\hat{v}_{\mathsf{id},3}]_2, \{[\hat{w}_{a,\mathsf{id}}]_1\}_{a \in S_{\mathsf{id}}})$. Because $\boldsymbol{\ell}_{\mathsf{id}}$ is the coefficient vector of the polynomial $L_{\mathsf{id}}(X) = \prod_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} (X - \mathsf{id}')$, this means that

$$\begin{aligned} \hat{v}_{\mathsf{id},1} &= \boldsymbol{\ell}_{\mathsf{id}}^\top \boldsymbol{\tau} = \sum_{j \in [N]} \ell_{\mathsf{id},j} \tau^{j-1} = L_{\mathsf{id}}(\tau), \text{ and} \\ \hat{v}_{\mathsf{id},2} &= \boldsymbol{\ell}_{\mathsf{id}}^\top \mathbf{y} = \ell_{\mathsf{id},N} \cdot (\alpha + y\tau^{N-1}) + \sum_{j \in [N-1]} \ell_{\mathsf{id},j} y \tau^{j-1} = \alpha + y \sum_{j \in [N]} \ell_{\mathsf{id},j} \tau^{j-1} = \alpha + y \cdot L_{\mathsf{id}}(\tau), \end{aligned} \tag{4.3}$$

  where we have also used the fact that $L_{\mathsf{id}}(X) = \prod_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} (X - \mathsf{id}')$ is a monic polynomial of degree $N - 1$ (i.e., the coefficient of X^{N-1} is 1).

- Now, for each $\mathsf{id}' \in I \setminus \{\mathsf{id}\}$, consider the value of $\boldsymbol{\ell}_{\mathsf{id},\mathsf{id}'}^\top \mathbf{r}'_{\mathsf{id}'}$. Using the fact that $\boldsymbol{\ell}_{\mathsf{id},\mathsf{id}'} \in \mathbb{Z}_p^{N-1}$ is the coefficient vector of the polynomial $L_{\mathsf{id},\mathsf{id}'}(X) = \prod_{\mathsf{id}'' \in I \setminus \{\mathsf{id},\mathsf{id}'\}} (X - \mathsf{id}'')$, this means that

$$\begin{aligned} \boldsymbol{\ell}_{\mathsf{id},\mathsf{id}'}^\top \mathbf{r}'_{\mathsf{id}'} = \sum_{j \in [N-1]} \ell_{\mathsf{id},\mathsf{id}',j} r_{\mathsf{id}'} (\tau - \mathsf{id}') \tau^{j-1} &= r_{\mathsf{id}'}(\tau - \mathsf{id}') \cdot L_{\mathsf{id},\mathsf{id}'}(\tau) \\ &= r_{\mathsf{id}'}(\tau - \mathsf{id}') \cdot \prod_{\mathsf{id}'' \in I \setminus \{\mathsf{id},\mathsf{id}'\}} (\tau - \mathsf{id}'') \\ &= r_{\mathsf{id}'} \cdot L_{\mathsf{id}}(\tau). \end{aligned}$$

  Correspondingly, we have that

$$\hat{v}_{\mathsf{id},3} = \sum_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} \boldsymbol{\ell}_{\mathsf{id},\mathsf{id}'}^\top \mathbf{r}'_{\mathsf{id}'} = \sum_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} r_{\mathsf{id}'} \cdot L_{\mathsf{id}}(\tau). \tag{4.4}$$

  Finally, consider the value of $\hat{w}_{a,\mathsf{id}}$ for $a \in S_{\mathsf{id}}$. By construction, it holds that

$$\hat{w}_{a,\mathsf{id}} = \sum_{j \in [N]} f_{a,\mathsf{id},j} \tau^{j-1} = F_{a,\mathsf{id}}(\tau) = \frac{L_{\mathsf{id}}(\tau)}{F_a(\tau)}. \tag{4.5}$$

First, we note that the quotient polynomial $F_{a,\mathsf{id}}(X) \coloneqq L_{\mathsf{id}}(X)/F_a(X)$ is well-defined as a polynomial of degree at most $N-1$: specifically, since $a \in S_{\mathsf{id}}$, the index id is not a root of $F_a(X) = \prod_{\mathsf{id}' \in I : a \notin S_{\mathsf{id}'}} (X - \mathsf{id}')$, so the roots of F_a are contained in $I \setminus \{\mathsf{id}\}$. The roots of L_{id} are precisely $I \setminus \{\mathsf{id}\}$, so we conclude that F_a divides L_{id}. Moreover, the value of the denominator $F_a(\tau)$ is non-zero since all of the roots of F_a are contained in I (by construction), and we are working from the assumption that $\tau \notin I$.

Take any message $[\mu]_{\mathsf{T}} \in \mathbb{G}_{\mathsf{T}}$, any index $\mathsf{id} \in I$, and any policy $(\mathbf{M}, \rho)$ for $\mathbf{M} \in \mathbb{Z}_p^{\ell \times n}$ and $\rho\colon [\ell] \to A$ such that the attribute set S_{id} satisfies the policy. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{M}, \rho), [\mu]_{\mathsf{T}})$. We can write that

$$\mathsf{ct} = ((\mathbf{M}, \rho), [c_1]_{\mathsf{T}}, [c_2]_1, [c_3]_1, \{([c_{4,k}]_1, [c_{5,k}]_2)\}_{k \in [\ell]}),$$

where

$$\begin{aligned} c_1 &= s\alpha + \mu & c_{4,k} &= s_2 y \mathbf{m}_k^{\mathsf{T}} \mathbf{v} - s'_k \\ c_2 &= s & c_{5,k} &= s'_k \hat{u}_{\rho(k)} = s'_k \cdot F_{\rho(k)}(\tau), \\ c_3 &= s_1 y - s\hat{r} = s_1 y - s \textstyle\sum_{\mathsf{id}' \in I} r_{\mathsf{id}'} \end{aligned} \tag{4.6}$$

for $\mathbf{v} = (1, v_2, \ldots, v_n) \in \mathbb{Z}_p^n$ and $s_1, s_2, s'_k \in \mathbb{Z}_p$ (for each $k \in [\ell]$) where $s = s_1 + s_2$. Finally, consider $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{hsk}_{\mathsf{id}}, \mathsf{ct})$ for the user with index id:

- Let $J = \{k \in [\ell] : \rho(k) \in S_{\mathsf{id}}\}$ be the indices of $\mathbf{M}$ associated with the attributes in S_{id}. Write $J = \{k_1, \ldots, k_{|J|}\}$. Now, let $\boldsymbol{\omega} = (\omega_1, \omega_2, \ldots, \omega_{|J|}) \in \mathbb{Z}_p^{|J|}$ be a vector such that $\boldsymbol{\omega}^{\mathsf{T}} \mathbf{M}_J = \mathbf{e}_1^{\mathsf{T}}$, where $\mathbf{M}_J$ is the matrix obtained by taking only the rows of $\mathbf{M}$ indexed by J. Recall that such a vector exists because S_{id} is authorized by $(\mathbf{M}, \rho)$.
- We now consider the d_{id} and d_{attrib} values computed by the decryption algorithm. First, using Eqs. (4.3), (4.4) and (4.6), we have that

$$d_{\mathsf{id}} = (r_{\mathsf{id}} c_2 + c_3) \cdot \hat{v}_{\mathsf{id},1} + c_2 \cdot \hat{v}_{\mathsf{id},3} = \left(s r_{\mathsf{id}} + s_1 y - s \sum_{\mathsf{id}' \in I} r_{\mathsf{id}'} \right) \cdot L_{\mathsf{id}}(\tau) + s \sum_{\mathsf{id}' \in I \setminus \{\mathsf{id}\}} r_{\mathsf{id}'} \cdot L_{\mathsf{id}}(\tau) = s_1 y \cdot L_{\mathsf{id}}(\tau). \tag{4.7}$$

  Next, take any $k \in J$. By definition, this means $\rho(k) \in S_{\mathsf{id}}$. By Eqs. (4.3), (4.5) and (4.6), we thus have that

$$c_{4,k} \cdot \hat{v}_{\mathsf{id},1} + \hat{w}_{\rho(k),\mathsf{id}} \cdot c_{5,k} = (s_2 y \mathbf{m}_k^{\mathsf{T}} \mathbf{v} - s'_k) \cdot L_{\mathsf{id}}(\tau) + s'_k \cdot F_{\rho(k)}(\tau) \cdot \frac{L_{\mathsf{id}}(\tau)}{F_{\rho(k)}(\tau)} = s_2 y \mathbf{m}_k^{\mathsf{T}} \mathbf{v} \cdot L_{\mathsf{id}}(\tau).$$

  This means that

$$d_{\mathsf{attrib}} = \sum_{j \in [|J|]} \omega_j \cdot \left(c_{4,k_j} \cdot \hat{v}_{\mathsf{id},1} + \hat{w}_{\rho(k_j),\mathsf{id}} \cdot c_{5,k_j} \right) = s_2 y \cdot L_{\mathsf{id}}(\tau) \cdot \boldsymbol{\omega}^{\mathsf{T}} \mathbf{M}_J \mathbf{v} = s_2 y \cdot L_{\mathsf{id}}(\tau), \tag{4.8}$$

  where we have used the fact that $\boldsymbol{\omega}^{\mathsf{T}} \mathbf{M}_J \mathbf{v} = \mathbf{e}_1^{\mathsf{T}} \mathbf{v} = v_1 = 1$.

Combining Eqs. (4.3), (4.7) and (4.8) and using the fact that $s_1 + s_2 = s$, we have

$$\begin{aligned} c_1 - c_2 \cdot \hat{v}_{\mathsf{id},2} + d_{\mathsf{id}} + d_{\mathsf{attrib}} &= (s\alpha + \mu) - s(\alpha + y \cdot L_{\mathsf{id}}(\tau)) + s_1 y \cdot L_{\mathsf{id}}(\tau) + s_2 y \cdot L_{\mathsf{id}}(\tau) \\ &= \mu - (s - s_1 - s_2) y \cdot L_{\mathsf{id}}(\tau) \\ &= \mu. \end{aligned}$$

We conclude that $\mathsf{Decrypt}(\mathsf{sk}_{\mathsf{id}}, \mathsf{hsk}_{\mathsf{id}}, \mathsf{ct})$ outputs $[\mu]_{\mathsf{T}}$, and correctness holds. □

Theorem 4.3 (Compactness). *Construction 4.1 is compact.*

Proof. This follows by inspection, as each group element can be represented in $\mathsf{poly}(\lambda)$ size. In particular, the master public key mpk consists of the group description, 3 group elements, and 1 group element for each attribute in the set A (i.e., the universe of attributes associated with the users). Each helper decryption key $\mathsf{hsk}_{\mathsf{id}}$ (associated with an index id and an attribute set S) consists of 3 group elements and 1 group element for each attribute in S. There are at most $|A|$ such attributes. □

4.1 Security Analysis

We prove that our slotted scheme achieves index-set-selective security without corruptions in the registered-key model from the following assumption. In Appendix C, we show this assumption holds unconditionally in the generic bilinear group model [Sho97, BBG05].

Assumption 4.4 (Main Assumption)**.** Let GroupGen be an asymmetric prime-order pairing-group generator. For a security parameter λ, a positive integer-valued polynomial $N = N(\lambda)$, a set sequence $S = \{S_\lambda\}_{\lambda\in\mathbb{N}}$ where $S_\lambda \subseteq \{0,1,\ldots,2^\lambda-1\}$ and $|S_\lambda| = N(\lambda)$, and a bit $b \in \{0,1\}$, we define the distribution $\mathcal{D}_{\lambda,N,S,b}$ as follows:

- Sample $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$.
- Sample random generators $\hat{g}_1 \xleftarrow{\mathrm{R}} \mathbb{G}_1 \setminus \{g_1^0\}$ and $\hat{g}_2 \xleftarrow{\mathrm{R}} \mathbb{G}_2 \setminus \{g_2^0\}$. Define $[1]_1 \coloneqq \hat{g}_1$ and $[1]_2 \coloneqq \hat{g}_2$.
- Sample $\gamma \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $\tau \xleftarrow{\mathrm{R}} \mathbb{Z}_p \setminus S_\lambda$. Let $\beta = \sum_{i\in[N]} \frac{1}{\tau - s_i} \in \mathbb{Z}_p$ where $S_\lambda = \{s_1,\ldots,s_N\}$, and define

$$\mathsf{params} = \left(\begin{array}{c} [1]_1, [\tau]_1, \ldots, [\tau^{N-1}]_1, [1]_2, [\tau]_2, \ldots, [\tau^{N-1}]_2, \\ [\beta]_1, [\frac{\gamma}{\tau-s_1}]_1, \ldots, [\frac{\gamma}{\tau-s_N}]_1, \\ [\gamma]_2, [\gamma\tau]_2, \ldots, [\gamma\tau^{N-2}]_2, [\gamma\beta]_2, [\gamma\beta\tau]_2, \ldots, [\gamma\beta\tau^{N-2}]_2 \end{array} \right).$$

- If $b = 0$, let $T = [\gamma\beta^2\tau^{N-1}]_\mathsf{T}$, and if $b = 1$, let $T \xleftarrow{\mathrm{R}} \mathbb{G}_\mathsf{T}$. Output $(\mathcal{G}, \mathsf{params}, T)$.

We say Assumption 4.4 holds with respect to GroupGen, a positive integer-valued function $N = N(\lambda)$, and a set sequence S if the distributions $\mathcal{D}_0 = \{\mathcal{D}_{\lambda,N,S,0}\}_{\lambda\in\mathbb{N}}$ and $\mathcal{D}_1 = \{\mathcal{D}_{\lambda,N,S,1}\}_{\lambda\in\mathbb{N}}$ are computationally indistinguishable. We further say that Assumption 4.4 holds with respect to GroupGen if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all efficient adversaries $\mathcal{A}$, all polynomials $N = N(\lambda)$, and all set sequences $S = \{S_\lambda\}_{\lambda\in\mathbb{N}}$ where $S_\lambda \subseteq \{0,1,\ldots,2^\lambda-1\}$ and $|S_\lambda| = N(\lambda)$, there exists a polynomial $\mathsf{poly}(\lambda) \geq 1$ such that for all $\lambda \in \mathbb{N}$,

$$|\Pr[\mathcal{A}(1^\lambda, \mathsf{params}) = 1 : \mathsf{params} \leftarrow \mathcal{D}_{\lambda,N,S,0}] - \Pr[\mathcal{A}(1^\lambda, \mathsf{params}) = 1 : \mathsf{params} \leftarrow \mathcal{D}_{\lambda,N,S,1}]| \leq \mathsf{poly}(\lambda)\cdot\mathsf{negl}(\lambda).$$

Remark 4.5 (Structure of Assumption 4.4)**.** Before stating the security analysis, we first discuss the structure of Assumption 4.4. In particular, we note that the assumption does *not* hold if we set $\gamma = 1$. This is because, for all $i \in [N]$, we can write

$$\frac{\tau}{\tau - s_i} = \frac{\tau - s_i}{\tau - s_i} + \frac{s_i}{\tau - s_i} = 1 + \frac{s_i}{\tau - s_i}.$$

Since the elements s_i are fixed (and known to the adversary), the adversary can now compute (when $\gamma = 1$)

$$\sum_{i\in[N]} [1]_1 + s_i \cdot \left[\frac{1}{\tau - s_i}\right]_1 = \sum_{i\in[N]} \left[\frac{\tau}{\tau - s_i}\right]_1 = [\beta\tau]_1.$$

Now, the adversary can pair $[\tau\beta]_1$ with $[\beta\tau^{N-2}]_2$ and obtain $[\beta^2\tau^{N-1}]_\mathsf{T}$, which completely breaks the assumption (when $\gamma = 1$). If we introduce the randomizing exponent γ, then the above procedure would allow the adversary to compute $[\gamma\beta\tau]_1$.[4] However, since the other $\mathbb{G}_2$ components either lack β or also include γ, the adversary cannot simply pair this element with another term in the assumption to reach the challenge $[\gamma\beta^2\tau^{N-1}]_\mathsf{T}$. Essentially, the extra γ term is introduced to prevent this attack. As we show in Appendix C, adding γ is sufficient to prove the assumption holds unconditionally in the generic bilinear group model [Sho97, BBG05].

Theorem 4.6 (Index-Set-Selective Security without Corruptions)**.** *Suppose Assumption 4.4 holds with respect to* GroupGen. *Then Construction 4.1 is index-set-selectively secure without corruptions in the registered-key model.*

Proof. Take any positive integer-valued polynomial $N = N(\lambda)$ and any collection of index sets $I = \{I_\lambda\}_{\lambda\in\mathbb{N}}$ where each $I_\lambda \subseteq \{0,1\}^\lambda$ and $|I_\lambda| = N(\lambda)$. Consider the following hybrids parameterized by a bit b and, implicitly, a security parameter λ and an efficient admissible adversary $\mathcal{A}$ that only submits indices in I_λ. We use green to denote changes between hybrids.

[4] Strictly speaking, computing this term would also require giving out $[\gamma]_1$, which the assumption does not do. However, even if the assumption gave out $[\gamma]_1$, the attack would not apply.

- **Hybrid** $\mathsf{Hyb}_0^{(b)}$: This is the index-set-selective security without corruptions game in the registered-key model where the challenger encrypts challenge message μ_b^*. We recall the main steps here:

  - **Setup phase:** The challenger generates the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ which it then provides to the adversary. By construction, we have that $\mathsf{crs} = (\mathcal{G}, [\alpha]_\mathsf{T}, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [\mathbf{y}]_1, [\mathbf{y}]_2)$ where $\boldsymbol{\tau} = (1, \tau, \tau^2, \ldots, \tau^{N-1}) \in \mathbb{Z}_p^N$ and $\mathbf{y} = (y, y\tau, y\tau^2, \ldots, y\tau^{N-2}, \alpha + y\tau^{N-1}) \in \mathbb{Z}_p^N$ for uniform $\tau, \alpha, y \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. The challenger additionally initializes $\mathsf{ctr} = 0$, an empty set C, and an empty dictionary D.
  - **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following query:
    - **Key-generation query:** The adversary submits an index $\mathsf{id} \in I_\lambda$. In response, the challenger generates $(\mathsf{pk}_\mathsf{id}, \mathsf{sk}_\mathsf{id}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id})$ and provides pk_id to $\mathcal{A}$. By construction, we have that

      $$\mathsf{pk}_\mathsf{id} = ([r_\mathsf{id}]_1, [\mathbf{r}'_\mathsf{id}]_2) \quad \text{and} \quad \mathsf{sk}_\mathsf{id} = r_\mathsf{id},$$

      where $r_\mathsf{id} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ is uniform and $\mathbf{r}'_\mathsf{id} = (r_\mathsf{id}(\tau - \mathsf{id}), r_\mathsf{id}(\tau - \mathsf{id})\tau, \ldots, r_\mathsf{id}(\tau - \mathsf{id})\tau^{N-2})$. The challenger also increments $\mathsf{ctr} = \mathsf{ctr} + 1$ and adds the mapping $\mathsf{D}[\mathsf{ctr}] = (\mathsf{id}, \mathsf{pk}_\mathsf{id}, \mathsf{sk}_\mathsf{id})$ to the dictionary.
  - **Challenge phase:** For $i \in [N]$, $\mathcal{A}$ specifies a tuple $(c_i, \mathsf{id}_i^*, S_i^*, \mathsf{pk}_i^*, \mathsf{sk}_i^*, r_i^*)$ where $c_i \in [\mathsf{ctr}] \cup \{\perp\}$ and $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}_i^*; r_i^*)$. If $\{\mathsf{id}_i^*\}_{i\in[N]} \neq I_\lambda$, the challenger halts with output 0. Otherwise, it proceeds as follows:
    - If $c_i \in [\mathsf{ctr}]$, the challenger sets $(\mathsf{id}_i, \mathsf{pk}_i, \mathsf{sk}_i) = \mathsf{D}[c_i]$. If $\mathsf{id}_i \neq \mathsf{id}_i^*$, the challenger halts with output 0. Otherwise, the challenger sets $\mathsf{pk}_{\mathsf{id}_i} = \mathsf{pk}_i$ and $S_{\mathsf{id}_i} = S_i^*$.
    - If $c_i = \perp$, then the challenger sets $\mathsf{id}_i = \mathsf{id}_i^*$, $\mathsf{pk}_{\mathsf{id}_i} = \mathsf{pk}_i^*$, $r_{\mathsf{id}_i} = r_i^*$, and $S_{\mathsf{id}_i} = S_i^*$. It then adds id_i to C. As in the honest case, we have that

      $$\mathsf{pk}_{\mathsf{id}_i} = ([r_{\mathsf{id}_i}]_1, [\mathbf{r}'_{\mathsf{id}_i}]_2) \quad \text{and} \quad \mathsf{sk}_{\mathsf{id}_i} = r_{\mathsf{id}_i},$$

      where $\mathbf{r}'_{\mathsf{id}_i} = (r_{\mathsf{id}_i}(\tau - \mathsf{id}_i), r_{\mathsf{id}_i}(\tau - \mathsf{id}_i)\tau, \ldots, r_{\mathsf{id}_i}(\tau - \mathsf{id}_i)\tau^{N-2})$.

    The adversary also submits a challenge policy $P^* = (\mathbf{M}^*, \rho^*)$ where $\mathbf{M}^* \in \mathbb{Z}_p^{\ell \times n}$ and $\rho^* \colon [\ell] \to A$ where $A = \bigcup_{\mathsf{id}\in I_\lambda} S_\mathsf{id}$, as well as messages $[\mu_0^*]_\mathsf{T}, [\mu_1^*]_\mathsf{T}$. The challenger now performs the aggregation procedure by computing $(\mathsf{mpk}, \{\mathsf{hsk}_\mathsf{id}\}_{\mathsf{id}\in I_\lambda}) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, \{(\mathsf{pk}_\mathsf{id}, S_\mathsf{id})\}_{\mathsf{id}\in I_\lambda})$. By construction, we can write $\mathsf{mpk} = (\mathcal{G}, [\alpha]_\mathsf{T}, [y]_1, [\hat{r}]_1, \{[\hat{u}_a]_2\}_{a\in A})$ where

    $$[\hat{r}]_1 = \sum_{\mathsf{id}\in I_\lambda} [r_\mathsf{id}]_1 \quad \text{and} \quad [\hat{u}_a]_2 = \mathbf{f}_a^\mathsf{T}[\boldsymbol{\tau}]_2 = [F_a(\tau)]_2,$$

    and $F_a(X) = \prod_{\mathsf{id}\in I_\lambda : a\notin S_\mathsf{id}} (X - \mathsf{id})$ is a polynomial of degree at most $N-1$ with $\mathbf{f}_a$ being the associated coefficient vector. Because the helper decryption keys hsk_id are unused during encryption, we omit them here.

    Finally, the challenger constructs the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P^*, [\mu_b^*]_\mathsf{T})$. By construction, we can write that $\mathsf{ct}^* = ((\mathbf{M}^*, \rho^*), [c_1^*]_\mathsf{T}, [c_2^*]_1, [c_3^*]_1, \{([c_{4,k}^*]_1, [c_{5,k}^*]_2)\}_{k\in[\ell]})$. The message-embedding and index-specific ciphertext components are computed by sampling uniform $s_1, s_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, letting $s = s_1 + s_2$, and assigning

    $$[c_1^*]_\mathsf{T} = s[\alpha]_\mathsf{T} + [\mu_b^*]_\mathsf{T} \quad , \quad [c_2^*]_1 = [s]_1 \quad , \quad [c_3^*]_1 = s_1[y]_1 - s[\hat{r}]_1.$$

    The attribute-specific components are computed by sampling uniform $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and defining the vector $\mathbf{v} = (1, v_2, \ldots, v_n)^\mathsf{T}$, then sampling $s'_k \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ for $k \in [\ell]$ and assigning

    $$[c_{4,k}^*]_1 = (s_2\mathbf{m}_k^\mathsf{T}\mathbf{v})[y]_1 - [s'_k]_1 \quad \text{and} \quad [c_{5,k}^*]_2 = s'_k[\hat{u}_{\rho^*(k)}]_2.$$

    Here, $\mathbf{m}_k^\mathsf{T}$ denotes the k^{th} row of $\mathbf{M}^*$.

– **Output phase:** At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit b^* which is the output of the experiment.

- **Hybrid** $\mathsf{Hyb}_1^{(b)}$: Same as $\mathsf{Hyb}_0^{(b)}$, except the challenger instead samples $\tau \xleftarrow{R} \mathbb{Z}_p \setminus I_\lambda$ during the setup phase.
- **Hybrid** $\mathsf{Hyb}_2^{(b)}$: Same as $\mathsf{Hyb}_1^{(b)}$ except the challenger halts with output 0 if either of the following events occurs:
  - If $\sum_{\mathsf{id} \in I_\lambda} \frac{1}{\tau - \mathsf{id}} = 0$ during the setup phase, or
  - If $s_2 = 0$ during the challenge phase.
- **Hybrid** $\mathsf{Hyb}_3^{(b)}$: Same as $\mathsf{Hyb}_2^{(b)}$ except we modify how the challenger's random coins are generated, as highlighted below:
  - **Setup phase:** When the challenger runs Setup to generate $\mathsf{crs} = (\mathcal{G}, [\alpha]_T, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [y]_1, [\mathbf{y}]_2)$, it first samples $\tau \xleftarrow{R} \mathbb{Z}_p \setminus I_\lambda$ and then samples $\gamma, \tilde{\alpha} \xleftarrow{R} \mathbb{Z}_p$. Then, it assigns $\beta = \sum_{\mathsf{id} \in I_\lambda} \frac{1}{\tau - \mathsf{id}}$ and halts with output 0 if $\beta = 0$. Otherwise, the challenger defines $\alpha = \tilde{\alpha} - \gamma\beta\tau^{N-1}$ and $y = \gamma\beta$ and constructs $\boldsymbol{\tau} = (1, \tau, \tau^2, \ldots, \tau^{N-1}) \in \mathbb{Z}_p^N$ and $\mathbf{y} = (y, y\tau, y\tau^2, \ldots, y\tau^{N-2}, \alpha + y\tau^{N-1})$ as before.
  - **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following query:
    - **Key-generation query:** When the adversary submits $\mathsf{id} \in I_\lambda$, the challenger generates

      $$\mathsf{pk}_{\mathsf{id}} = ([r_{\mathsf{id}}]_1, [\mathbf{r}'_{\mathsf{id}}]_2) \quad \text{and} \quad \mathsf{sk}_{\mathsf{id}} = r_{\mathsf{id}},$$

      where r_{id} is instead generated by sampling $\tilde{r}_{\mathsf{id}} \xleftarrow{R} \mathbb{Z}_p$ and letting $r_{\mathsf{id}} = \tilde{r}_{\mathsf{id}} + \frac{\gamma}{\tau - \mathsf{id}}$. The challenger then adds the mapping $\mathsf{D}[\mathsf{ctr}] = (\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}}, \tilde{r}_{\mathsf{id}})$ to the dictionary.
  - **Challenge phase:** When generating $\mathsf{ct}^* = ((\mathbf{M}^*, \rho^*), [c_1^*]_T, [c_2^*]_1, [c_3^*]_1, \{([c_{4,k}^*]_1, [c_{5,k}^*]_2)\}_{k \in [\ell]})$, the challenger generates s_1, s_2 used in the message-embedding and index-specific components by instead sampling $\tilde{s}_1, \tilde{s}_2 \xleftarrow{R} \mathbb{Z}_p$ and setting $s_1 = \tilde{s}_1 + \sum_{\mathsf{id} \in \mathcal{H}} \frac{1}{\tau - \mathsf{id}}$ (where $\mathcal{H} = I_\lambda \setminus C$) and $s_2 = \tilde{s}_2 + \sum_{\mathsf{id} \in C} \frac{1}{\tau - \mathsf{id}}$. If $s_2 = 0$, the hybrid halts with output 0; otherwise, it defines $\tilde{\mathbf{s}}_2 = (\tilde{s}_2, \ldots, \tilde{s}_2)$.

    For the attribute-specific components, it generates $\mathbf{v}$ by first letting $\tilde{\mathbf{v}} = (0, \tilde{v}_2, \ldots, \tilde{v}_n)$ for $\tilde{v}_2, \ldots, \tilde{v}_n \xleftarrow{R} \mathbb{Z}_p$ and setting $\mathbf{v} = \frac{1}{s_2}(\tilde{\mathbf{v}} + \tilde{\mathbf{s}}_2 + \sum_{\mathsf{id} \in C} \frac{1}{\tau - \mathsf{id}} \mathbf{v}_{\mathsf{id}}^*)$ where $\mathbf{v}_{\mathsf{id}}^*$ is the special vector from Definition 2.2 with first component 1 and $\mathbf{m}_k^\top \mathbf{v}_{\mathsf{id}}^* = 0$ whenever $\rho^*(k) \in S_{\mathsf{id}}$. Here, we are using the fact that S_{id} does not satisfy the policy P^*, so such a vector $\mathbf{v}_{\mathsf{id}}^*$ is guaranteed to exist by Definition 2.2. Now, for each $k \in [\ell]$, the challenger generates s'_k by instead sampling $\tilde{s}'_k \xleftarrow{R} \mathbb{Z}_p$ and letting $s'_k = \tilde{s}'_k + \gamma\beta \sum_{\mathsf{id} \in C : \rho^*(k) \notin S_{\mathsf{id}}} \frac{1}{\tau - \mathsf{id}} \mathbf{m}_k^\top \mathbf{v}_{\mathsf{id}}^*$.
- **Hybrid** $\mathsf{Hyb}_4^{(b)}$: Same as $\mathsf{Hyb}_3^{(b)}$, except the challenger now samples $[c_1^*]_T \xleftarrow{R} \mathbb{G}_T$. In particular, the message-embedding component $[c_1^*]_T$ of the challenge ciphertext is replaced with a random group element (i.e., the challenge ciphertext in this experiment no longer depends on the bit b).

We now argue that each pair of hybrids is indistinguishable provided the adversary $\mathcal{A}$ is efficient and admissible. In particular, we bound the difference in the output distribution of each pair of consecutive hybrids in the lemmas below.

Lemma 4.7. *There exists a negligible function* $\mathsf{negl}_0(\cdot)$ *independent of* (N, I) *and a positive polynomial* $\mathsf{poly}_0(\lambda)$ *such that* $|\Pr[\mathsf{Hyb}_0^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1]| \leq \mathsf{poly}_0(\lambda) \cdot \mathsf{negl}_0(\lambda)$.

Proof. The only difference between the hybrids is that $\tau \xleftarrow{R} \mathbb{Z}_p$ in hybrid $\mathsf{Hyb}_0^{(b)}$ and $\tau \xleftarrow{R} \mathbb{Z}_p \setminus I_\lambda$ in hybrid $\mathsf{Hyb}_1^{(b)}$. The statistical distance between the uniform distributions on these two sets is bounded by N/p. The lemma follows from the fact that $N = N(\lambda)$ is positive (integer-valued) and $p \geq 2^{2\lambda}$ per Definition 2.1. □

Lemma 4.8. *There exists a negligible function* $\mathsf{negl}_1(\cdot)$ *independent of* (N, I) *and a positive polynomial* $\mathsf{poly}_1(\lambda)$ *such that* $|\Pr[\mathsf{Hyb}_1^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1]| \leq \mathsf{poly}_1(\lambda) \cdot \mathsf{negl}_1(\lambda)$.

Proof. The two hybrids are identical until the event that $\sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau-\mathsf{id}} = 0$ in the setup phase or the event that $s_2 = 0$ occurs in the challenge phase. To analyze the former, we first note that $\prod_{\mathsf{id}\in I_\lambda}(\tau - \mathsf{id}) \neq 0$ if $\tau \notin I_\lambda$. Then we can write

$$\begin{aligned}
\Pr\left[\sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau-\mathsf{id}} = 0 : \tau \xleftarrow{\text{R}} \mathbb{Z}_p \setminus I_\lambda\right] &= \Pr\left[\left(\prod_{\mathsf{id}\in I_\lambda}(\tau-\mathsf{id})\right)\left(\sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau-\mathsf{id}}\right) = 0 : \tau \xleftarrow{\text{R}} \mathbb{Z}_p \setminus I_\lambda\right] \\
&= \Pr\left[\sum_{\mathsf{id}\in I_\lambda}\prod_{\mathsf{id}'\in I_\lambda\setminus\{\mathsf{id}\}} (\tau-\mathsf{id}') = 0 : \tau \xleftarrow{\text{R}} \mathbb{Z}_p \setminus I_\lambda\right] \\
&\leq \frac{N-1}{p-N}.
\end{aligned}$$

The inequality follows from the Schwartz–Zippel lemma and the fact that $T(X) = \sum_{\mathsf{id}\in I_\lambda}\prod_{\mathsf{id}'\in I_\lambda\setminus\{\mathsf{id}\}}(X-\mathsf{id}')$ is a non-zero polynomial of degree at most $N-1$. To analyze the latter event, since the challenger samples $s_2 \xleftarrow{\text{R}} \mathbb{Z}_p$, this event occurs with probability $1/p$. The lemma now follows from the fundamental lemma of game playing [BR06, Lemma 2], the fact that $N = N(\lambda)$ is positive, and $p \geq 2^{2\lambda}$. □

Lemma 4.9. *It holds that* $\Pr[\mathsf{Hyb}_2^{(b)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1]$.

Proof. Both hybrids generate identical distributions. The only difference is in how these distributions are sampled. Below, we analyze each individual phase of both hybrids.

- **Setup phase and key-generation queries:** Consider the randomness used to generate the common reference string and the key-generation queries. For the former, the challenger samples $\tau, y, \alpha \xleftarrow{\text{R}} \mathbb{Z}_p$ to construct the crs, while for the latter, the only randomness is $r_{\mathsf{id}} \xleftarrow{\text{R}} \mathbb{Z}_p$ which is used to construct $(\mathsf{pk}_{\mathsf{id}}, \mathsf{sk}_{\mathsf{id}})$ for each $\mathsf{id} \in \mathcal{H}$ where $\mathcal{H} = I_\lambda \setminus C$. We specify the distributions of these elements in the two hybrids below:

	Hybrid $\mathsf{Hyb}_2^{(b)}$	**Hybrid** $\mathsf{Hyb}_3^{(b)}$	
	$\tau \xleftarrow{\text{R}} \mathbb{Z}_p$	$\tau \xleftarrow{\text{R}} \mathbb{Z}_p$	
	$y \xleftarrow{\text{R}} \mathbb{Z}_p$	$y \leftarrow \gamma\beta$	where $\gamma \xleftarrow{\text{R}} \mathbb{Z}_p$
	$\alpha \xleftarrow{\text{R}} \mathbb{Z}_p$	$\alpha \leftarrow \tilde{\alpha} - \gamma\beta\tau^{N-1}$	where $\tilde{\alpha} \xleftarrow{\text{R}} \mathbb{Z}_p$
$\mathsf{id} \in \mathcal{H}$:	$r_{\mathsf{id}} \xleftarrow{\text{R}} \mathbb{Z}_p$	$r_{\mathsf{id}} \leftarrow \tilde{r}_{\mathsf{id}} + \frac{\gamma}{\tau-\mathsf{id}}$	where $\tilde{r}_{\mathsf{id}} \xleftarrow{\text{R}} \mathbb{Z}_p$,

  where $\beta = \sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau-\mathsf{id}}$. If $\beta = 0$, then both experiments output 0. If $\beta \neq 0$, then the marginal distribution of y in both experiments is uniform over $\mathbb{Z}_p$. Furthermore, α and each r_{id} are generated by masking constants with fresh uniform samples used nowhere else (including later phases, as can be seen below). Consequently, the distributions of the elements in this phase are identical between the two hybrids.

- **Challenge phase:** As aggregation is deterministic, the only random coins in this phase are those used in

generating the challenge ciphertext ct^*, which we detail below.

	Hybrid $\mathsf{Hyb}_2^{(b)}$	**Hybrid** $\mathsf{Hyb}_3^{(b)}$	
	$s_1 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$	$s_1 \leftarrow \tilde{s}_1 + \sum_{\mathsf{id}\in\mathcal{H}} \frac{1}{\tau - \mathsf{id}}$	where $\tilde{s}_1 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$
	$s_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$	$s_2 \leftarrow \tilde{s}_2 + \sum_{\mathsf{id}\in C} \frac{1}{\tau - \mathsf{id}}$	where $\tilde{s}_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$
	$v_1 = 1$	$v_1 \leftarrow \frac{1}{s_2}\left(\tilde{v}_1 + \tilde{s}_2 + \sum_{\mathsf{id}\in C} \frac{1}{\tau - \mathsf{id}} v^*_{\mathsf{id},1}\right)$	where $\tilde{v}_1 = 0, v^*_{\mathsf{id},1} = 1$
$i \in [2,n]:$	$v_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$	$v_i \leftarrow \frac{1}{s_2}\left(\tilde{v}_i + \tilde{s}_2 + \sum_{\mathsf{id}\in C} \frac{1}{\tau - \mathsf{id}} v^*_{\mathsf{id},i}\right)$	where $\tilde{v}_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$
$k \in [\ell]:$	$s'_k \xleftarrow{\mathrm{R}} \mathbb{Z}_p$	$s'_k \leftarrow \tilde{s}'_k + \gamma\beta \sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}} \frac{1}{\tau - \mathsf{id}} \mathbf{m}_k^\top \mathbf{v}^*_{\mathsf{id}}$	where $\tilde{s}'_k \xleftarrow{\mathrm{R}} \mathbb{Z}_p,$

where $\mathbf{v}^*_{\mathsf{id}} = (1, v^*_{\mathsf{id},2}, \ldots, v^*_{\mathsf{id},n}) \in \mathbb{Z}_p^n$ for each $\mathsf{id} \in C$. To analyze this phase, we can imagine an intermediate hybrid with two transitions: the first modifies the generation of s_1, s_2, and each s'_k, while the second modifies the generation of $\mathbf{v} = (v_1, \ldots, v_n)$. In the first transition, the distributions are identical because s_1, s_2, and s'_k are generated by masking constants with samples used nowhere else (as $\mathbf{v}$ yet remains unchanged). For the second transition, substituting $\tilde{v}_1 = 0$ and $v^*_{\mathsf{id},1} = 1$ directly yields $v_1 = 1$. For $i \in [2,n]$, since $\tilde{v}_i$ is uniform over $\mathbb{Z}_p$, the term $\tilde{v}_i/s_2$ is also uniform and masks the remaining constants (which can now be dependent on the previous samples).

We conclude that the two distributions are identically distributed. □

Lemma 4.10. *Suppose Assumption 4.4 holds with respect to* GroupGen. *Then there exists a negligible function* $\mathsf{negl}_3(\cdot)$ *independent of* (N, I) *such that for all efficient admissible adversaries* $\mathcal{A}$*, there exists a positive polynomial* $\mathsf{poly}_3(\cdot)$ *such that* $|\Pr[\mathsf{Hyb}_3^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_4^{(b)}(\mathcal{A}) = 1]| \le \mathsf{poly}_3(\lambda) \cdot \mathsf{negl}_3(\lambda)$.

Proof. The lemma follows by constructing an efficient reduction $\mathcal{B}_b$ against Assumption 4.4 with respect to GroupGen and the $N = N(\lambda)$ and $I = \{I_\lambda\}_{\lambda\in\mathbb{N}}$ fixed above. We describe the reduction below:

- **Setup phase:** The reduction $\mathcal{B}_b$ receives $(\mathcal{G}, \mathsf{params}, T)$ where $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, p, g_1, g_2, e) \leftarrow \mathsf{GroupGen}(1^\lambda)$ and the parameters

$$\mathsf{params} = \begin{pmatrix} [1]_1, [\tau]_1, \ldots, [\tau^{N-1}]_1, [1]_2, [\tau]_2, \ldots, [\tau^{N-1}]_2, \\ [\beta]_1, \{[\frac{\gamma}{\tau - \mathsf{id}}]_1\}_{\mathsf{id}\in I_\lambda} \\ [\gamma]_2, [\gamma\tau]_2, \ldots, [\gamma\tau^{N-2}]_2, [\gamma\beta]_2, [\gamma\beta\tau]_2, \ldots, [\gamma\beta\tau^{N-2}]_2 \end{pmatrix},$$

  for $\tau \xleftarrow{\mathrm{R}} \mathbb{Z}_p \setminus I_\lambda$, $\gamma \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, and $\beta = \sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau - \mathsf{id}}$, as well as $T = [\gamma\beta^2\tau^{N-1}]_\mathsf{T}$ or $T \xleftarrow{\mathrm{R}} \mathbb{G}_\mathsf{T}$. Recall that $[1]_1 \coloneqq \hat{g}_1$ and $[1]_2 \coloneqq \hat{g}_2$ denote random generators $\hat{g}_1 \xleftarrow{\mathrm{R}} \mathbb{G}_1 \setminus \{g_1^0\}$ and $\hat{g}_2 \xleftarrow{\mathrm{R}} \mathbb{G}_2 \setminus \{g_2^0\}$. In what follows, we will use blue to denote the challenge terms the reduction has received.

  First, the reduction checks if $[\beta]_1 = [0]_1$. If so, then the reduction halts with output 0. Otherwise, the reduction defines and computes the following values:

  - Let $[\gamma\beta]_1 = \sum_{\mathsf{id}\in I_\lambda} \left[\frac{\gamma}{\tau - \mathsf{id}}\right]_1$.
  - Let $[\boldsymbol{\tau}]_1 = ([1]_1, [\tau]_1, [\tau^2]_1, \ldots, [\tau^{N-1}]_1)$ and $[\boldsymbol{\tau}]_2 = ([1]_2, [\tau]_2, [\tau^2]_2, \ldots, [\tau^{N-1}]_2)$.
  - Let $[\boldsymbol{\chi}]_2 = ([\gamma\beta]_2, [\gamma\beta\tau]_2, \ldots, [\gamma\beta\tau^{N-2}]_2)$.

The reduction $\mathcal{B}_b$ first samples $\tilde{\alpha} \xleftarrow{R} \mathbb{Z}_p$ then generates $\mathsf{crs} = (\mathcal{G}, [\alpha]_T, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [y]_1, [\mathbf{y}]_2)$ by computing the following:

$$[\alpha]_T = [\tilde{\alpha}]_T - [\tau^{N-1}]_1 \cdot [\gamma\beta]_2, \qquad [y_i]_2 = [\gamma\beta\tau^{i-1}]_2 \quad \text{for } i \in [N-1],$$
$$[y]_1 = [\gamma\beta]_1, \qquad [y_N]_2 = [\tilde{\alpha}]_2.$$

The reduction $\mathcal{B}_b$ starts running the adversary $\mathcal{A}$ on input $(1^\lambda, 1^N, I_\lambda)$ and provides it crs. The reduction also initializes $\mathsf{ctr} = 0$ and an empty dictionary D.

- **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following query:
  - **Key-generation query:** When the adversary submits an index $\mathsf{id} \in I_\lambda$, the reduction $\mathcal{B}_b$ generates a public key $\mathsf{pk}_{\mathsf{id}} = ([r_{\mathsf{id}}]_1, [\mathbf{r}'_{\mathsf{id}}]_2)$ by sampling $\tilde{r}_{\mathsf{id}} \xleftarrow{R} \mathbb{Z}_p$ and computing

$$[r_{\mathsf{id}}]_1 = [\tilde{r}_{\mathsf{id}}]_1 + \left[\frac{\gamma}{\tau - \mathsf{id}}\right]_1 \quad \text{and} \quad [r'_{\mathsf{id},i}]_2 = \tilde{r}_{\mathsf{id}}([\tau^i]_2 - \mathsf{id}[\tau^{i-1}]_2) + [\gamma\tau^{i-1}]_2 \quad \text{for } i \in [N-1].$$

    The reduction sends $\mathsf{pk}_{\mathsf{id}}$ to the adversary $\mathcal{A}$. It also increments the counter $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$ and adds the mapping $\mathsf{D}[\mathsf{ctr}] = (\mathsf{id}, \mathsf{pk}_{\mathsf{id}}, \tilde{r}_{\mathsf{id}})$ to the dictionary.

- **Challenge phase:** For each $i \in [N]$, the adversary $\mathcal{A}$ specifies a tuple $(c_i, \mathsf{id}_i^*, S_i^*, \mathsf{pk}_i^*, \mathsf{sk}_i^*, r_i^*)$ where $c_i \in [\mathsf{ctr}] \cup \{\perp\}$ and $(\mathsf{pk}_i^*, \mathsf{sk}_i^*) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}_i^*; r_i^*)$. The reduction $\mathcal{B}_b$ halts with output 0 if $\{\mathsf{id}_i^*\}_{i\in[N]} \neq I_\lambda$. Otherwise, it initializes empty sets $\mathcal{H}$ and C and then proceeds as follows:
  - If $c_i \in [\mathsf{ctr}]$, the reduction $\mathcal{B}_b$ sets $(\mathsf{id}_i, \mathsf{pk}_i, \tilde{r}_i) = \mathsf{D}[c_i]$. If $\mathsf{id}_i \neq \mathsf{id}_i^*$, the reduction $\mathcal{B}_b$ halts with output 0. Otherwise, it sets $\mathsf{pk}_{\mathsf{id}_i} = \mathsf{pk}_i$, $S_{\mathsf{id}_i} = S_i^*$, and $\tilde{r}_{\mathsf{id}_i} = \tilde{r}_i$. It also adds id_i to $\mathcal{H}$.
  - If $c_i = \perp$, the reduction $\mathcal{B}_b$ sets $\mathsf{id}_i = \mathsf{id}_i^*$, $\mathsf{pk}_{\mathsf{id}_i} = \mathsf{pk}_i^*$, $r_{\mathsf{id}_i} = r_i^*$, and $S_{\mathsf{id}_i} = S_i^*$. With this, $\mathcal{B}_b$ computes

$$\mathsf{pk}_{\mathsf{id}_i} = ([r_{\mathsf{id}_i}]_1, [\mathbf{r}'_{\mathsf{id}_i}]_2) \quad \text{and} \quad \mathsf{sk}_{\mathsf{id}_i} = r_{\mathsf{id}_i},$$

    where $\mathbf{r}'_{\mathsf{id}_i} = (r_{\mathsf{id}_i}(\tau - \mathsf{id}_i), r_{\mathsf{id}_i}(\tau - \mathsf{id}_i)\tau, \ldots, r_{\mathsf{id}_i}(\tau - \mathsf{id}_i)\tau^{N-2})$. It then adds id_i to C.

Note that the reduction $\mathcal{B}_b$ has now generated $\mathsf{pk}_{\mathsf{id}}$ for all indices $\mathsf{id} \in I_\lambda$ and can hence directly compute $(\mathsf{mpk}, \{\mathsf{hsk}_{\mathsf{id}}\}_{\mathsf{id}\in I_\lambda}) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, \{(\mathsf{pk}_{\mathsf{id}}, S_{\mathsf{id}})\}_{\mathsf{id}\in I_\lambda})$. By construction, we have that the master public key $\mathsf{mpk} = (\mathcal{G}, [\alpha]_T, [y]_1, [\hat{r}]_1, \{[\hat{u}_a]_2\}_{a\in A})$ for $A = \bigcup_{\mathsf{id}\in I_\lambda} S_{\mathsf{id}}$.

Now consider the adversary-submitted challenge policy $(\mathbf{M}^*, \rho^*)$ where $\mathbf{M}^* \in \mathbb{Z}_p^{\ell\times n}$ and $\rho^* : [\ell] \to A$, as well as the challenge message $[\mu_b^*]_T$. The reduction begins by sampling $\tilde{s}_1, \tilde{s}_2 \xleftarrow{R} \mathbb{Z}_p$ and assigning $\tilde{\mathbf{s}}_2 = (\tilde{s}_2, \ldots, \tilde{s}_2) \in \mathbb{Z}_p^n$. It then generates the message-embedding and index-specific challenge ciphertext components by computing

$$[c_1^*]_T = [\tilde{\alpha}(\tilde{s}_1 + \tilde{s}_2)]_T + [\beta]_1 \cdot [\tilde{\alpha}]_2 - (\tilde{s}_1 + \tilde{s}_2)[\tau^{N-1}]_1 \cdot [\gamma\beta]_2 - T + [\mu_b^*]_T,$$
$$[c_2^*]_1 = [\tilde{s}_1 + \tilde{s}_2]_1 + [\beta]_1,$$
$$[c_3^*]_1 = \tilde{s}_1[\gamma\beta]_1 - (\tilde{s}_1 + \tilde{s}_2)\sum_{\mathsf{id}\in\mathcal{H}}\left[\frac{\gamma}{\tau - \mathsf{id}}\right]_1 - \left(\sum_{\mathsf{id}\in\mathcal{H}}\tilde{r}_{\mathsf{id}} + \sum_{\mathsf{id}\in C} r_{\mathsf{id}}\right)([\tilde{s}_1 + \tilde{s}_2]_1 + [\beta]_1).$$

For the attribute-specific components, the reduction $\mathcal{B}_b$ samples $\tilde{v}_2, \ldots, \tilde{v}_n \xleftarrow{R} \mathbb{Z}_p$ and lets $\tilde{\mathbf{v}} = (0, \tilde{v}_2, \ldots, \tilde{v}_n)$. Next, for each $\mathsf{id} \in C$, it lets $\mathbf{v}_{\mathsf{id}}^*$ be the vector with first component 1 where $\mathbf{m}_k^\top \mathbf{v}_{\mathsf{id}}^* = 0$ for all $k \in [\ell]$ such that $\rho^*(k) \in S_{\mathsf{id}}$. Such vectors exist because the attribute sets for corrupted users do not satisfy challenge policies output by an admissible adversary. Finally, for each $k \in [\ell]$, the reduction samples $\tilde{s}'_k \xleftarrow{R} \mathbb{Z}_p$ and assigns the remaining challenge ciphertext components by computing

$$[c_{4,k}^*]_1 = \mathbf{m}_k^\top(\tilde{\mathbf{v}} + \tilde{\mathbf{s}}_2)[\gamma\beta]_1 - [\tilde{s}'_k]_1,$$
$$[c_{5,k}^*]_2 = \tilde{s}'_k \mathbf{f}_{\rho^*(k)}^\top [\boldsymbol{\tau}]_2 + \sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}} \mathbf{m}_k^\top \mathbf{v}_{\mathsf{id}}^* \mathbf{h}_{\rho^*(k),\mathsf{id}}^\top [\boldsymbol{\chi}]_2.$$

Here, $\mathbf{f}_{\rho^*(k)}$ is the coefficient vector associated with the polynomial $F_{\rho^*(k)}(X)$ of degree $N-1$ (used to generate mpk) and $\mathbf{h}_{\rho^*(k),\mathsf{id}}$ is the coefficient vector associated with the polynomial $H_{\rho^*(k),\mathsf{id}}(X) \coloneqq F_{\rho^*(k)}(X)/(X-\mathsf{id})$ of degree $N-2$ (used only by the reduction), which is well-defined as $\mathcal{B}_b$ only computes it for $\mathsf{id} \in C$ such that $\rho^*(k) \notin S_{\mathsf{id}}$. The reduction provides $\mathsf{ct}^* = ((\mathbf{M}^*, \rho^*), [c_1^*]_T, [c_2^*]_1, [c_3^*]_1, \{([c_{4,k}^*]_1, [c_{5,k}^*]_2)\}_{k\in[\ell]})$ to the adversary.

- **Output phase:** When the adversary $\mathcal{A}$ outputs a bit b^*, the reduction $\mathcal{B}_b$ forwards this as its own answer.

We now argue that conditioned on $s_2 = \tilde{s}_2 + \sum_{\mathsf{id}\in C} \frac{1}{\tau-\mathsf{id}} \neq 0$, the reduction algorithm $\mathcal{B}_b$ perfectly simulates $\mathsf{Hyb}_3^{(b)}$ to $\mathcal{A}$ when it receives $T = [\gamma\beta^2\tau^{N-1}]_T$ and perfectly simulates $\mathsf{Hyb}_4^{(b)}$ to $\mathcal{A}$ when $T \xleftarrow{R} \mathbb{G}_T$. We first analyze the components that are identical between the two hybrids.

- **Setup phase:** Consider now the common reference string $\mathsf{crs} = (\mathcal{G}, [\alpha]_T, [\boldsymbol{\tau}]_1, [\boldsymbol{\tau}]_2, [y]_1, [\mathbf{y}]_2)$, recalling that $y = \gamma\beta$ and $\alpha = \tilde{\alpha} - \gamma\beta\tau^{N-1}$ in both hybrids. We have that

$$\begin{aligned}
[\alpha]_T &= [\tilde{\alpha} - \gamma\beta\tau^{N-1}]_T = [\tilde{\alpha}]_T - [\tau^{N-1}]_1 \cdot [\gamma\beta]_2,\\
[y_i]_2 &= [y\tau^{i-1}]_2 = [\gamma\beta\tau^{i-1}]_2 \quad \text{for } i \in [N-1],\\
[y_N]_2 &= [\alpha + y\tau^{N-1}]_2 = [\tilde{\alpha} - \gamma\beta\tau^{N-1} + \gamma\beta\tau^{N-1}]_2 = [\tilde{\alpha}]_2.
\end{aligned}$$

  Finally, recalling that the reduction computes $\sum_{\mathsf{id}\in I_\lambda} [\frac{\gamma}{\tau-\mathsf{id}}]_1 = [\gamma \sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau-\mathsf{id}}]_1 = [\gamma\beta]_1$, we have that $[y]_1 = [\gamma\beta]_1$. This matches the specification in $\mathsf{Hyb}_3^{(b)}$ and $\mathsf{Hyb}_4^{(b)}$. Note that if $\beta = 0$, then the reduction outputs 0, which matches the behavior in $\mathsf{Hyb}_3^{(b)}$ and $\mathsf{Hyb}_4^{(b)}$.

- **Honest registration queries:** Moving on to the public keys $\mathsf{pk}_{\mathsf{id}} = ([r_{\mathsf{id}}]_1, [\mathbf{r}'_{\mathsf{id}}]_2)$ generated during honest registration queries, we can see that

$$\begin{aligned}
[r_{\mathsf{id}}]_1 &= \left[\tilde{r}_{\mathsf{id}} + \frac{\gamma}{\tau-\mathsf{id}}\right]_1 = [\tilde{r}_{\mathsf{id}}]_1 + \left[\frac{\gamma}{\tau-\mathsf{id}}\right]_1,\\
[r'_{\mathsf{id},i}]_2 &= [r_{\mathsf{id}}(\tau-\mathsf{id})\tau^{i-1}]_2\\
&= \left[\left(\tilde{r}_{\mathsf{id}} + \frac{\gamma}{\tau-\mathsf{id}}\right)(\tau-\mathsf{id})\tau^{i-1}\right]_2\\
&= \left[\tilde{r}_{\mathsf{id}}(\tau-\mathsf{id})\tau^{i-1} + \frac{\gamma}{\tau-\mathsf{id}}(\tau-\mathsf{id})\tau^{i-1}\right]_2\\
&= \tilde{r}_{\mathsf{id}}\big([\tau^i]_2 - \mathsf{id}[\tau^{i-1}]_2\big) + [\gamma\tau^{i-1}]_2 \quad \text{for } i \in [N-1].
\end{aligned}$$

  Again, this matches the specification in $\mathsf{Hyb}_3^{(b)}$ and $\mathsf{Hyb}_4^{(b)}$.

- **Challenge phase:** We now consider the components of the challenge ciphertext. Recall first that $\beta = \sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau-\mathsf{id}}$ per Assumption 4.4. Next, both hybrids assign $s_1 = \tilde{s}_1 + \sum_{\mathsf{id}\in\mathcal{H}} \frac{1}{\tau-\mathsf{id}}$ and $s_2 = \tilde{s}_2 + \sum_{\mathsf{id}\in C} \frac{1}{\tau-\mathsf{id}}$, and $s = s_1 + s_2$ by construction. Here, $\mathcal{H}$ and C are the honest index and corrupted index sets the reduction creates based on the adversary's submitted tuples. Consider first the message-embedding component $[c_2^*]_1$.

$$\begin{aligned}
[c_2^*]_1 = [s]_1 &= \left[\tilde{s}_1 + \sum_{\mathsf{id}\in\mathcal{H}} \frac{1}{\tau-\mathsf{id}} + \tilde{s}_2 + \sum_{\mathsf{id}\in C} \frac{1}{\tau-\mathsf{id}}\right]_1\\
&= \left[\tilde{s}_1 + \tilde{s}_2 + \sum_{\mathsf{id}\in I_\lambda} \frac{1}{\tau-\mathsf{id}}\right]_1 = [\tilde{s}_1 + \tilde{s}_2]_1 + [\beta]_1,
\end{aligned}$$

  which coincides with the how the reduction simulates $[c_2^*]_1$. Next, we consider the index-based component $[c_3^*]_1 = s_1[y]_1 - s[\hat{r}]_1$. Here, it is simpler to consider the terms $[s_1 y]_1$ and $[s\hat{r}]_1$ separately. For the former, we have that

$$[s_1 y]_1 = \left[\left(\tilde{s}_1 + \sum_{\mathsf{id}\in\mathcal{H}} \frac{1}{\tau-\mathsf{id}}\right)\gamma\beta\right]_1 = \tilde{s}_1[\gamma\beta]_1 + \left[\gamma\beta \sum_{\mathsf{id}\in\mathcal{H}} \frac{1}{\tau-\mathsf{id}}\right]_1,$$

recalling that the hybrids define $s_1 = \tilde{s}_1 + \sum_{\mathsf{id}\in\mathcal{H}} \frac{1}{\tau-\mathsf{id}}$ and $y = \gamma\beta$. For the latter, we have that

$$\begin{aligned}
[s\hat{r}]_1 &= \left[(\tilde{s}_1+\tilde{s}_2+\beta)\left(\sum_{\mathsf{id}\in\mathcal{H}}\left(\tilde{r}_{\mathsf{id}}+\frac{\gamma}{\tau-\mathsf{id}}\right)+\sum_{\mathsf{id}\in C} r_{\mathsf{id}}\right)\right]_1 \\
&= \left[(\tilde{s}_1+\tilde{s}_2+\beta)\left(\sum_{\mathsf{id}\in\mathcal{H}}\frac{\gamma}{\tau-\mathsf{id}}+\sum_{\mathsf{id}\in\mathcal{H}}\tilde{r}_{\mathsf{id}}+\sum_{\mathsf{id}\in C} r_{\mathsf{id}}\right)\right]_1 \\
&= (\tilde{s}_1+\tilde{s}_2)\sum_{\mathsf{id}\in\mathcal{H}}\left[\frac{\gamma}{\tau-\mathsf{id}}\right]_1 + \left[\gamma\beta\sum_{\mathsf{id}\in\mathcal{H}}\frac{1}{\tau-\mathsf{id}}\right]_1 + \left(\sum_{\mathsf{id}\in\mathcal{H}}\tilde{r}_{\mathsf{id}}+\sum_{\mathsf{id}\in C} r_{\mathsf{id}}\right)([\tilde{s}_1+\tilde{s}_2]_1+[\beta]_1),
\end{aligned}$$

using the fact that $s = \tilde{s}_1+\tilde{s}_2+\beta$ (as shown above), $\hat{r} = \sum_{\mathsf{id}\in I_\lambda} r_{\mathsf{id}}$ by construction, and that $r_{\mathsf{id}} = \tilde{r}_{\mathsf{id}} + \frac{\gamma}{\tau-\mathsf{id}}$ for $\mathsf{id}\in\mathcal{H}$. Combining the two terms, we have that

$$\begin{aligned}
[c_3^*]_1 &= s_1[y]_1 - s[\hat{r}]_1 = [s_1 y]_1 - [s\hat{r}]_1 \\
&= \tilde{s}_1[\gamma\beta]_1 - (\tilde{s}_1+\tilde{s}_2)\sum_{\mathsf{id}\in\mathcal{H}}\left[\frac{\gamma}{\tau-\mathsf{id}}\right]_1 - \left(\sum_{\mathsf{id}\in\mathcal{H}}\tilde{r}_{\mathsf{id}}+\sum_{\mathsf{id}\in C} r_{\mathsf{id}}\right)([\tilde{s}_1+\tilde{s}_2]_1+[\beta]_1).
\end{aligned}$$

For the attribute-based components in the ciphertext, we first consider $[c_{4,k}^*]_1 = (s_2\mathbf{m}_k^\top\mathbf{v})[y]_1 - [s_k']_1$ for $k\in[\ell]$, and, similar to the previous step, we consider terms $[s_2 y\mathbf{m}_k^\top\mathbf{v}]_1$ and $[s_k']_1$ separately. Suppose for now that $s_2\neq 0$. Then, for the term $[s_2 y\mathbf{m}_k^\top\mathbf{v}]_1$, we have

$$\begin{aligned}
[s_2 y\mathbf{m}_k^\top\mathbf{v}]_1 &= \left[s_2\gamma\beta\mathbf{m}_k^\top\frac{1}{s_2}\left(\tilde{\mathbf{v}}+\tilde{\mathbf{s}}_2+\sum_{\mathsf{id}\in C}\frac{1}{\tau-\mathsf{id}}\mathbf{v}_{\mathsf{id}}^*\right)\right]_1 \\
&= [\gamma\beta\mathbf{m}_k^\top(\tilde{\mathbf{v}}+\tilde{\mathbf{s}}_2)]_1 + \left[\gamma\beta\mathbf{m}_k^\top\sum_{\mathsf{id}\in C}\frac{1}{\tau-\mathsf{id}}\mathbf{v}_{\mathsf{id}}^*\right]_1 \\
&= [\gamma\beta\mathbf{m}_k^\top(\tilde{\mathbf{v}}+\tilde{\mathbf{s}}_2)]_1 + \left[\gamma\beta\left(\sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}}\frac{1}{\tau-\mathsf{id}}\mathbf{m}_k^\top\mathbf{v}_{\mathsf{id}}^* + \sum_{\mathsf{id}\in C:\rho^*(k)\in S_{\mathsf{id}}}\frac{1}{\tau-\mathsf{id}}\mathbf{m}_k^\top\mathbf{v}_{\mathsf{id}}^*\right)\right]_1 \\
&= \mathbf{m}_k^\top(\tilde{\mathbf{v}}+\tilde{\mathbf{s}}_2)[\gamma\beta]_1 + \left[\gamma\beta\sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}}\frac{1}{\tau-\mathsf{id}}\mathbf{m}_k^\top\mathbf{v}_{\mathsf{id}}^*\right]_1,
\end{aligned}$$

where we have used the fact that the hybrids define $\mathbf{v} = \frac{1}{s_2}(\tilde{\mathbf{v}}+\tilde{\mathbf{s}}_2+\sum_{\mathsf{id}\in C}\frac{1}{\tau-\mathsf{id}}\mathbf{v}_{\mathsf{id}}^*)$ where $\mathbf{v}_{\mathsf{id}}^*$ is a vector with first component 1 where $\mathbf{m}_k^\top\mathbf{v}_{\mathsf{id}}^* = 0$ for all $k\in[\ell]$ such that $\rho^*(k)\in S_{\mathsf{id}}$. For the latter term, it follows by definition that

$$[s_k']_1 = \left[\tilde{s}_k' + \gamma\beta\sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}}\frac{1}{\tau-\mathsf{id}}\mathbf{m}_k^\top\mathbf{v}_{\mathsf{id}}^*\right]_1 = [\tilde{s}_k']_1 + \left[\gamma\beta\sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}}\frac{1}{\tau-\mathsf{id}}\mathbf{m}_k^\top\mathbf{v}_{\mathsf{id}}^*\right]_1.$$

Hence, combining the two terms gives us

$$[c_{4,k}^*]_1 = (s_2\mathbf{m}_k^\top\mathbf{v})[y]_1 - [s_k']_1 = \mathbf{m}_k^\top(\tilde{\mathbf{v}}+\tilde{\mathbf{s}}_2)[\gamma\beta]_1 - [\tilde{s}_k']_1.$$

Finally, we consider the $[c^*_{5,k}]_2$ component of the ciphertext. Substituting in the definition of s'_k, we have that

$$\begin{aligned}
[c^*_{5,k}]_2 &= [s'_k \cdot F_{\rho^*(k)}(\tau)]_2 = \left[\left(\tilde{s}'_k + \gamma\beta \sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}} \frac{1}{\tau-\mathsf{id}} \mathbf{m}_k^\top \mathbf{v}^*_{\mathsf{id}}\right) \cdot F_{\rho^*(k)}(\tau)\right]_2 \\
&= [\tilde{s}'_k \cdot F_{\rho^*(k)}(\tau)]_2 + \left[\gamma\beta \sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}} \frac{1}{\tau-\mathsf{id}} \mathbf{m}_k^\top \mathbf{v}^*_{\mathsf{id}} \cdot F_{\rho^*(k)}(\tau)\right]_2 \\
&= [\tilde{s}'_k \cdot F_{\rho^*(k)}(\tau)]_2 + \left[\gamma\beta \sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}} \mathbf{m}_k^\top \mathbf{v}^*_{\mathsf{id}} \cdot H_{\rho^*(k),\mathsf{id}}(\tau)\right]_2 \\
&= \tilde{s}'_k \mathbf{f}^\top_{\rho^*(k)} [\boldsymbol{\tau}]_2 + \sum_{\mathsf{id}\in C:\rho^*(k)\notin S_{\mathsf{id}}} \mathbf{m}_k^\top \mathbf{v}^*_{\mathsf{id}} \mathbf{h}^\top_{\rho^*(k),\mathsf{id}} [\boldsymbol{\chi}]_2,
\end{aligned}$$

where $\mathbf{f}_{\rho^*(k)}$ is the coefficient vector associated with the polynomial $F_{\rho^*(k)}(X)$ of degree $N-1$ and $\mathbf{h}_{\rho^*(k),\mathsf{id}}$ is the coefficient vector associated with the polynomial $H_{\rho^*(k),\mathsf{id}}(X) = F_{\rho^*(k)}(X)/(X-\mathsf{id})$ of degree $N-2$.

Overall, we can see that the common reference string, honest keys, and challenge ciphertext components which are defined identically in the two hybrids are equivalent to what the reduction $\mathcal{B}_b$ simulates from the challenge terms, provided that $s_2 \neq 0$. We now consider the message-dependent ciphertext component $[c^*_1]_1$, which the reduction $\mathcal{B}_b$ simulates using the challenge term T it receives. When $T = [\gamma\beta^2\tau^{N-1}]_\mathsf{T}$, we have that the reduction computes the component as

$$\begin{aligned}
[c^*_1]_\mathsf{T} &= [\tilde{\alpha}(\tilde{s}_1+\tilde{s}_2)]_\mathsf{T} + [\beta]_1 \cdot [\tilde{\alpha}]_2 - (\tilde{s}_1+\tilde{s}_2)[\tau^{N-1}]_1 \cdot [\gamma\beta]_2 - T + [\mu^*_b]_\mathsf{T} \\
&= [\tilde{\alpha}(\tilde{s}_1+\tilde{s}_2) + \tilde{\alpha}\beta - \gamma\beta\tau^{N-1}(\tilde{s}_1+\tilde{s}_2) - \gamma\beta^2\tau^{N-1} + \mu^*_b]_\mathsf{T} \\
&= [(\tilde{\alpha} - \gamma\beta\tau^{N-1})(\tilde{s}_1+\tilde{s}_2+\beta) + \mu^*_b]_\mathsf{T} \\
&= [s\alpha + \mu^*_b]_\mathsf{T} \\
&= s[\alpha]_\mathsf{T} + [\mu^*_b]_\mathsf{T}.
\end{aligned}$$

Thus, the reduction $\mathcal{B}_b$ simulates an execution of hybrid $\mathsf{Hyb}^{(b)}_3$ to $\mathcal{A}$ in this case. When $T \xleftarrow{\mathrm{R}} \mathbb{G}_\mathsf{T}$, we have that $[c^*_1]_\mathsf{T}$ is additively masked by T and hence is uniform (and independent of all other components). In this case, the reduction $\mathcal{B}_b$ simulates an execution of hybrid $\mathsf{Hyb}^{(b)}_4$ to $\mathcal{A}$. Thus, we can now write

$$\begin{aligned}
&\Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T = [\gamma\beta^2\tau^{N-1}]_\mathsf{T}] = \\
&\quad \Pr[s_2 \neq 0] \cdot \Pr[\mathsf{Hyb}^{(b)}_3(\mathcal{A}) = 1] + \Pr[s_2 = 0] \cdot \Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T = [\gamma\beta^2\tau^{N-1}]_\mathsf{T} \wedge s_2 = 0] \\
&\Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T \xleftarrow{\mathrm{R}} \mathbb{G}_\mathsf{T}] = \\
&\quad \Pr[s_2 \neq 0] \cdot \Pr[\mathsf{Hyb}^{(b)}_4(\mathcal{A}) = 1] + \Pr[s_2 = 0] \cdot \Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T \xleftarrow{\mathrm{R}} \mathbb{G}_\mathsf{T} \wedge s_2 = 0].
\end{aligned}$$

Now, in the experiment, the exponent s_2 is (implicitly) set to $s_2 = \tilde{s}_2 + \sum_{\mathsf{id}\in C} \frac{1}{\tau-\mathsf{id}}$, where $\tilde{s}_2 \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Thus, the probability that $s_2 = 0$ is at most $1/p$. We conclude then that

$$\begin{aligned}
&|\Pr[\mathsf{Hyb}^{(b)}_3(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}^{(b)}_4(\mathcal{A}) = 1]| \\
&\quad \leq |\Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T = [\gamma\beta^2\tau^{N-1}]_\mathsf{T}] - \Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T \xleftarrow{\mathrm{R}} \mathbb{G}_\mathsf{T}]| + O(1/p).
\end{aligned}$$

Since $\mathcal{A}$ is efficient, the reduction algorithm $\mathcal{B}_b$ is also efficient. From Assumption 4.4, we know there exists a (universal) negligible function $\mathsf{negl}(\cdot)$ independent of (N, I) and $\mathcal{B}_b$ and a polynomial $\mathsf{poly}(\cdot)$ such that

$$|\Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T = [\gamma\beta^2\tau^{N-1}]_\mathsf{T}] - \Pr[\mathcal{B}_b(1^\lambda, \mathsf{params}) = 1 : T \xleftarrow{\mathrm{R}} \mathbb{G}_\mathsf{T}]| = \mathsf{poly}(\lambda) \cdot \mathsf{negl}(\lambda).$$

Since $p \geq 2^{2\lambda}$, the claim follows. □

Lemma 4.11. $\Pr[\mathsf{Hyb}_4^{(0)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_4^{(1)}(\mathcal{A}) = 1]$.

Proof. This follows from the fact that the adversary's view is independent of the bit b. □

Theorem 4.6 now follows by combining Lemmas 4.7 to 4.11 using a hybrid argument as follows:

$$\begin{aligned}
\mathsf{Adv}_{\mathcal{A},N,I}(\lambda) &= |\Pr[\mathsf{Hyb}_0^{(0)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_0^{(1)}(\mathcal{A}) = 1]| \\
&\leq \sum_{b\in\{0,1\}} \sum_{i=0}^{3} |\Pr[\mathsf{Hyb}_i^{(b)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{i+1}^{(b)}(\mathcal{A}) = 1]| \\
&\leq 2\left(\mathsf{poly}_0(\lambda)\cdot\mathsf{negl}_0(\lambda) + \mathsf{poly}_1(\lambda)\cdot\mathsf{negl}_1(\lambda) + \mathsf{poly}_3(\lambda)\cdot\mathsf{negl}_3(\lambda)\right) \\
&\leq \left(2\mathsf{poly}_0(\lambda) + 2\mathsf{poly}_1(\lambda) + 2\mathsf{poly}_3(\lambda)\right)\left(\mathsf{negl}_0(\lambda) + \mathsf{negl}_1(\lambda) + \mathsf{negl}_3(\lambda)\right).
\end{aligned}$$

The last inequality holds because $\mathsf{negl}_0, \mathsf{negl}_1, \mathsf{negl}_3 \geq 0$. The theorem follows by setting $\mathsf{poly}(\lambda) = 2\mathsf{poly}_0(\lambda) + 2\mathsf{poly}_1(\lambda) + 2\mathsf{poly}_3(\lambda)$ and $\mathsf{negl}(\lambda) = \mathsf{negl}_0(\lambda) + \mathsf{negl}_1(\lambda) + \mathsf{negl}_3(\lambda)$, and observing that all of the negligible functions are independent of (N, I). □

Corollary 4.12 (Slotted Registered ABE from Pairings)**.** *Let λ be a security parameter and* GroupGen *be an asymmetric prime-order pairing-group generator that outputs groups of order $p = p(\lambda)$. Suppose Assumption 4.4 holds with respect to* GroupGen*. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be any attribute universe and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies that can be described by a linear secret sharing scheme with attribute universe $\mathcal{U}$ over $\mathbb{Z}_p$. Then there exists a slotted registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and index space $\mathcal{I} = \{\{0,1\}^\lambda\}_{\lambda\in\mathbb{N}}$. The scheme is index-set-selectively secure without corruptions in the registered-key model. To support N users, the scheme has the following efficiency properties:*

- *The common reference string* crs *contains $N+1$ elements in $\mathbb{G}_1$, $2N$ elements in $\mathbb{G}_2$, and 1 element in $\mathbb{G}_T$.*
- *Each user public key* $\mathsf{pk}_{\mathsf{id}}$ *contains* 1 *element in $\mathbb{G}_1$ and $N-1$ elements in $\mathbb{G}_2$.*
- *The master public key* mpk *contains* 2 *elements in $\mathbb{G}_1$, $|A|$ elements in $\mathbb{G}_2$, and 1 element in $\mathbb{G}_T$. Here, A denotes the union of all users' sets of attributes.*
- *Each helper decryption key* $\mathsf{hsk}_{\mathsf{id}}$ *contains $|S_{\mathsf{id}}|$ elements in $\mathbb{G}_1$ and* 3 *elements in $\mathbb{G}_2$. Here, S_{id} is the attribute set of the user with index* id*.*
- *A ciphertext* ct *contains $|P|+2$ elements in $\mathbb{G}_1$, $|P|$ elements in $\mathbb{G}_2$, and* 1 *element in $\mathbb{G}_T$, where P is the policy.*

Lifting to standard registered ABE. As discussed in Section 3.1, we can lift Corollary 4.12 from the registered-key model to the plain model using a (simulation-sound) NIZK proof of knowledge (cf. [LWW25, Appendix A]). Then, we can apply the powers-of-two transformation from [HLWW23] to obtain a full-fledged registered ABE scheme in the plain model (that supports dynamic registration). Technically, the transformation from [HLWW23] only considers a fixed-size index space; for completeness, we show in Appendix B that the transformation still applies when we consider a large index space. Taken together, we obtain the following corollary:

Corollary 4.13 (Statically-Secure Registered ABE from Pairings)**.** *Let λ be a security parameter and* GroupGen *be an asymmetric prime-order pairing-group generator that outputs groups of order $p = p(\lambda)$. Suppose Assumption 4.4 holds with respect to* GroupGen*. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be any attribute universe and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies that can be described by a linear secret sharing scheme with attribute universe $\mathcal{U}$ over $\mathbb{Z}_p$. Then there exists a registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and index space $\mathcal{I} = \{\{0,1\}^\lambda\}_{\lambda\in\mathbb{N}}$. The scheme satisfies static security (i.e., where the adversary has to declare the index set upfront and is not allowed to corrupt any users). To support up to N users, the scheme has the following efficiency properties:*

- *The size of the common reference string* crs *is $N\cdot\mathsf{poly}(\lambda, \log N)$.*
- *The size of each user public key* $\mathsf{pk}_{\mathsf{id}}$ *is $N\cdot\mathsf{poly}(\lambda, \log N)$.*

- *The size of the master public key* mpk *is* $|A| \cdot \mathsf{poly}(\lambda, \log N)$ *where* A *is the attribute set for all aggregated users.*
- *The size of each helper decryption key* $\mathsf{hsk}_{\mathsf{id}}$ *is* $|S_{\mathsf{id}}| \cdot \mathsf{poly}(\lambda, \log N)$ *where* S_{id} *is the attribute set for the user with index* id.
- *The size of a ciphertext* ct *is* $|P| \cdot \mathsf{poly}(\lambda, \log N)$ *where* P *is the policy.*

Adaptive security in the random oracle model. We can also consider Construction 4.1 in the setting where we fix the index space to be $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda \in \mathbb{N}}$ where $\mathcal{I}_\lambda = \{1, \ldots, N(\lambda)\}$. This coincides with the setting considered in previous pairing-based registered ABE schemes [HLWW23, ZZGQ23, GLWW23, AT24, LWW25, GHK+25]. Since the index set is completely fixed in this case, Corollary 4.12 gives a slotted registered ABE scheme with index space $\mathcal{I}$ that is *adaptively secure without corruptions* in the registered-key model. We can now appeal to the transformations in Remark 3.8 to obtain an *adaptively secure* registered ABE for the same attribute universe and policy family in the random oracle model. We summarize this instantiation below:

Corollary 4.14 (Adaptively-Secure Registered ABE from Pairings in the Random Oracle Model)**.** *Let* λ *be a security parameter and* $\mathsf{GroupGen}$ *be an asymmetric prime-order pairing-group generator that outputs groups of order* $p = p(\lambda)$. *Suppose Assumption 4.4 holds with respect to* $\mathsf{GroupGen}$ *with sub-exponential security.*[5] *Let* $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ *be any attribute universe and let* $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ *be a set of policies where each policy is of size at most* $|\mathcal{P}|$ *and can be described by a linear secret sharing scheme with attribute universe* $\mathcal{U}$ *over* $\mathbb{Z}_p$. *Then there exists an adaptively secure registered ABE scheme with attribute universe* $\mathcal{U}$, *policy space* $\mathcal{P}$, *and index set* $\mathcal{I} = \{[N(\lambda)]\}_{\lambda \in \mathbb{N}}$ *in the random oracle model. To support up to* N *users, the scheme has the following efficiency properties:*

- *The size of the common reference string* crs *is* $N \cdot \mathsf{poly}(\lambda, |\mathcal{P}|, \log N)$.
- *The size of each user's public key* $\mathsf{pk}_{\mathsf{id}}$ *is* $N \cdot \mathsf{poly}(\lambda, |\mathcal{P}|, \log N)$.
- *The size of the master public key* mpk *is* $|A| \cdot \mathsf{poly}(\lambda, |\mathcal{P}|, \log N)$ *where* A *is the attribute set for all aggregated users.*
- *The size of each helper decryption key* $\mathsf{hsk}_{\mathsf{id}}$ *is* $|S_{\mathsf{id}}| \cdot \mathsf{poly}(\lambda, |\mathcal{P}|, \log N)$ *where* S_{id} *is the attribute set for the user with index* id.
- *The size of a ciphertext* ct *is* $\mathsf{poly}(\lambda, |\mathcal{P}|, \log N)$.

Acknowledgments

We thank Shota Yamada for helpful discussions and pointers to related work. Brent Waters is supported by NSF CNS-2318701 and a Simons Investigator Award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Sloan Fellowship, a Microsoft Research Faculty Fellowship, a Google Research Scholar Award, an Amazon Research Award, and a gift from the Stellar Development Foundation.

References

[AC17] Shashank Agrawal and Melissa Chase. FAME: fast attribute-based message encryption. In *ACM CCS*, 2017.

[ADM+24] Gennaro Avitabile, Nico Döttling, Bernardo Magri, Christos Sakkas, and Stella Wohnig. Signature-based witness encryption with compact ciphertext. In *ASIACRYPT*, 2024.

[AGY26] Abtin Afshar, Rishab Goyal, and Saikumar Yadugiri. Distributed monotone-policy encryption with silent setup from lattices. *IACR Cryptol. ePrint Arch.*, 2026.

[5] Namely, there is a constant $\delta > 0$ such that for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the distinguishing advantage of $\mathcal{A}$ is at most $2^{-\lambda^\delta}$.

[AMR26] Damiano Abram, Giulio Malavolta, and Lawrence Roy. How to authenticate a non-deterministic computation: Shift-hiding functions, compressed LWE sampling, broadcast encryption, and obfuscation. *IACR Cryptol. ePrint Arch.*, 2026.

[AT24] Nuttapong Attrapadung and Junichi Tomida. A modular approach to registered ABE for unbounded predicates. In *CRYPTO*, 2024.

[BBG05] Dan Boneh, Xavier Boyen, and Eu-Jin Goh. Hierarchical identity based encryption with constant size ciphertext. In *EUROCRYPT*, 2005.

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

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

[BL88] Josh Cohen Benaloh and Jerry Leichter. Generalized secret sharing and monotone functions. In *CRYPTO*, 1988.

[BR06] Mihir Bellare and Phillip Rogaway. Code-based game-playing proofs and the security of triple encryption. In *EUROCRYPT*, 2006.

[BZ14] Dan Boneh and Mark Zhandry. Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation. In *CRYPTO*, 2014.

[CES21] Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In *Cryptography and Coding*, 2021.

[CEW25] Binyi Chen, Noel Elias, and David J. Wu. Pairing-based batch arguments for NP with a linear-size CRS. In *ASIACRYPT*, 2025.

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

[CHW25] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In *CRYPTO*, 2025.

[CW24] Jeffrey Champion and David J. Wu. Distributed broadcast encryption from lattices. In *TCC*, 2024.

[CW26] Jeffrey Champion and David J. Wu. Distributed monotone-policy encryption for DNFs from lattices. In *EUROCRYPT*, 2026.

[Del07] Cécile Delerablée. Identity-based broadcast encryption with constant size ciphertexts and private keys. In *ASIACRYPT*, 2007.

[DJWW25] Lalita Devadas, Abhishek Jain, Brent Waters, and David J. Wu. Succinct witness encryption for batch languages and applications. In *ASIACRYPT*, 2025.

[DKL+23] Nico Döttling, Dimitris Kolonelos, Russell W. F. Lai, Chuanwei Lin, Giulio Malavolta, and Ahmadreza Rahimi. Efficient laconic cryptography from learning with errors. In *EUROCRYPT*, 2023.

[DP08] Cécile Delerablée and David Pointcheval. Dynamic threshold public-key encryption. In *CRYPTO*, 2008.

[DPP07] Cécile Delerablée, Pascal Paillier, and David Pointcheval. Fully collusion secure dynamic broadcast encryption with constant-size ciphertexts or decryption keys. In *Pairing-Based Cryptography*, 2007.

[DPY24] Pratish Datta, Tapas Pal, and Shota Yamada. Registered FE beyond predicates: (attribute-based) linear functions and more. In *ASIACRYPT*, 2024.

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge L. Villar. An algebraic framework for Diffie-Hellman assumptions. In *CRYPTO*, 2013.

[ET36] Paul Erdös and Paul Turán. On some sequences of integers. *Journal of the London Mathematical Society*, 1(4), 1936.

[FFM+23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. In *ASIACRYPT*, 2023.

[FKdP23] Dario Fiore, Dimitris Kolonelos, and Paola de Perthuis. Cuckoo commitments: Registration-based encryption and key-value map commitments for large spaces. In *ASIACRYPT*, 2023.

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In *CRYPTO*, 2023.

[GGK25] William Gasarch, James Glenn, and Clyde Kruskal. Finding large sets without arithmetic progressions of length three: An empirical view and survey ii. *arXiv preprint arXiv:2501.01634*, 2025.

[GHK+25] Sanjam Garg, Mohammad Hajiabadi, Dimitris Kolonelos, Abhiram Kothapalli, and Guru-Vamsi Policharla. A framework for witness encryption from linearly verifiable SNARKs and applications. In *CRYPTO*, 2025.

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

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

[GKMR23] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. In *ACM CCS*, 2023.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *CRYPTO*, 2024.

[GLWW23] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Realizing flexible broadcast encryption: How to broadcast to a public-key directory. In *ACM CCS*, 2023.

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In *CRYPTO*, 2024.

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

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

[GV22] Rishab Goyal and Vinod Vaikuntanathan. Locally verifiable signature and key aggregation. In *CRYPTO*, 2022.

[GWWW26] Junqing Gong, Brent Waters, Hoeteck Wee, and David J. Wu. Threshold batched identity-based encryption from pairings in the plain model. In *EUROCRYPT*, 2026.

[GY26a] Rishab Goyal and Saikumar Yadugiri. Adaptively-secure flexible and identity-based broadcast encryption from decomposed LWE. *IACR Cryptol. ePrint Arch.*, 2026.

[GY26b] Rishab Goyal and Saikumar Yadugiri. Equivocal broadcast encryption: Adaptively-secure optimal distributed broadcast encryption from lattices. In *CRYPTO*, 2026.

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In *EUROCRYPT*, 2023.

[HWW25] Susan Hohenberger, Brent Waters, and David J. Wu. Pairing-based aggregate signatures without random oracles. In *ASIACRYPT*, 2025.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, 2023.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.

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

[LWW25] George Lu, Brent Waters, and David J. Wu. Multi-authority registered attribute-based encryption. In *EUROCRYPT*, 2025.

[PST25] Tapas Pal, Robert Schädlich, and Erkan Tairi. Registered functional encryption for pseudorandom functionalities from lattices: Registered ABE for unbounded depth circuits and turing machines, and more. *IACR Cryptol. ePrint Arch.*, 2025.

[RT26] Yannis Rouselakis and Junichi Tomida. Towards practical registered ABE: More efficient, non-monotone, and CCA-secure. *IACR Cryptol. ePrint Arch.*, 2026.

[RW22] Doreen Riepel and Hoeteck Wee. FABEO: fast attribute-based encryption with optimal security. In *ACM CCS*, 2022.

[RY07] Thomas Ristenpart and Scott Yilek. The power of proofs-of-possession: Securing multiparty signatures against rogue-key attacks. In *EUROCRYPT*, 2007.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11), 1979.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, 1997.

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

[Ver23] Tanya Verma. Inside Geo Key Manager v2: re-imagining access control for distributed systems. The CloudFlare Blog, 2023. Available at https://blog.cloudflare.com/inside-geo-key-manager-v2/.

[WQZD10] Qianhong Wu, Bo Qin, Lei Zhang, and Josep Domingo-Ferrer. Ad hoc broadcast encryption. In *ACM CCS*, 2010.

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

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In *CRYPTO*, 2025.

[WW26] Brent Waters and David J. Wu. Silent threshold cryptography from pairings: Expressive policies in the plain model. In *EUROCRYPT*, 2026.

[YZGC25] Xinrui Yang, Yijian Zhang, Ying Gao, and Jie Chen. Registered ABE for circuits from evasive lattice assumptions. *IACR Cryptol. ePrint Arch.*, 2025.

[ZZC+25] Ziqi Zhu, Kai Zhang, Zhili Chen, Junqing Gong, and Haifeng Qian. Black-box registered ABE from lattices. *IACR Cryptol. ePrint Arch.*, 2025.

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In *ASIACRYPT*, 2023.

A Registered Attribute-Based Encryption

We recall the syntax and definitions for registered attribute-based encryption (registered ABE). The preliminaries below are directly adapted from [HLWW23] with changes that reflect specific properties of our construction. We discuss these changes in Remarks A.5 to A.8.

Definition A.1 (Registered ABE)**.** Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be an attribute universe and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies over $\mathcal{U}$. Let $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$ be a message space and $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$ be an index space. A registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, message space $\mathcal{M}$, and index space $\mathcal{I}$ is a tuple of algorithms $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^N) \to \mathsf{crs}$: On input the security parameter λ and a maximum number of users N, the setup algorithm outputs a common reference string crs. We assume that crs implicitly contains the security parameter 1^λ.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}) \to (\mathsf{pk}, \mathsf{sk})$: On input the common reference string crs and an index $\mathsf{id} \in \mathcal{I}_\lambda$, the key-generation algorithm outputs a public key pk and a secret decryption key sk.
- $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{id}, \mathsf{pk}, S) \to (\mathsf{mpk}, \mathsf{aux}')$: On input the common reference string crs, a curator state aux (initially set to $\perp$), an index $\mathsf{id} \in \mathcal{I}_\lambda$, a public key pk, and a set of attributes $S \subseteq \mathcal{U}_\lambda$, the registration algorithm outputs the master public key mpk and an updated curator state aux′. This algorithm is deterministic.
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu) \to \mathsf{ct}$: On input the master public key mpk, an access policy $P \in \mathcal{P}_\lambda$, and a message $\mu \in \mathcal{M}_\lambda$, 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 curator state aux, and a public key pk, the update algorithm outputs a helper decryption key hsk. This algorithm is deterministic.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct}) \to \mu$: On input a decryption key sk, a helper decryption key hsk, and a ciphertext ct, the decryption algorithm outputs either a message $\mu \in \mathcal{M}_\lambda$, a special symbol $\mu = \perp$ to indicate a decryption failure, or a special flag $\mu = \mathsf{GetUpdate}$ that indicates an updated helper decryption key is needed to decrypt.

Definition A.2 (Correctness, Compactness, and Update Efficiency)**.** Let $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ be a registered ABE scheme with attribute universe $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$, policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$, message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$, and index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. Consider a game between an adversary $\mathcal{A}$ and a challenger that is parameterized by a security parameter λ and consists of the following phases:

- **Setup phase:** On input the security parameter 1^λ, the adversary $\mathcal{A}$ begins by declaring a bound 1^N on the maximum number of users. The challenger responds to $\mathcal{A}$ with $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and additionally initializes a curator state $\mathsf{aux} = \perp$, a master public key $\mathsf{mpk}_0 = \perp$, registration and encryption counters $\mathsf{ctr}[\mathsf{reg}] = 0$ and $\mathsf{ctr}[\mathsf{enc}] = 0$, respectively, and a challenge registration index $\mathsf{ctr}^*[\mathsf{reg}] = \infty$.
- **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following queries:
  - **Non-challenge registration query:** The adversary submits an index id, a public key pk, and a set of attributes $S \subseteq \mathcal{U}_\lambda$. If the index has already been registered, the challenger returns $\perp$. Otherwise, it increments the counter $\mathsf{ctr}[\mathsf{reg}] \leftarrow \mathsf{ctr}[\mathsf{reg}] + 1$, registers the public key by computing $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{id}, \mathsf{pk}, S)$, and then updates the curator state by setting $\mathsf{aux} \leftarrow \mathsf{aux}'$.
  - **Challenge registration query:** The adversary submits a challenge index $\mathsf{id}^* \in \mathcal{I}_\lambda$ and challenge set of attributes $S^* \subseteq \mathcal{U}_\lambda$. If the adversary has previously made a challenge registration query or if the index has previously been registered, then the challenger replies with $\perp$. Otherwise, it increments the counter $\mathsf{ctr}[\mathsf{reg}] \leftarrow \mathsf{ctr}[\mathsf{reg}] + 1$, generates the challenge key $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}^*)$, registers by computing $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{id}^*, \mathsf{pk}^*, S^*)$, and computes the helper decryption key $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$. The challenger then updates $\mathsf{aux} \leftarrow \mathsf{aux}'$, stores the index of the challenge index $\mathsf{ctr}^*[\mathsf{reg}] \leftarrow \mathsf{ctr}[\mathsf{reg}]$, and sends $(\mathsf{pk}^*, \mathsf{sk}^*)$ to $\mathcal{A}$.

- **Encryption query:** The adversary submits an index i with $\mathsf{ctr}^*[\mathsf{reg}] \le i \le \mathsf{ctr}[\mathsf{reg}]$, a policy $P \in \mathcal{P}_\lambda$, and a message $\mu \in \mathcal{M}_\lambda$. If the adversary has not yet performed a challenge registration query or if the challenge set of attributes S^* does not satisfy the policy P, the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr}[\mathsf{enc}] \leftarrow \mathsf{ctr}[\mathsf{enc}] + 1$, responds to $\mathcal{A}$ with $\mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}_i, P, \mu)$, and stores the message $\mu_{\mathsf{ctr}[\mathsf{enc}]} \leftarrow \mu$.
- **Decryption query:** The adversary submits a ciphertext index $1 \le j \le \mathsf{ctr}[\mathsf{enc}]$. The challenger computes $\mu'_j \leftarrow \mathsf{Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_j)$. If $\mu'_j = \mathsf{GetUpdate}$, then the challenger computes an updated helper decryption key $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$ and recomputes $\mu'_j \leftarrow \mathsf{Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_j)$. If $\mu'_j \neq \mu_j$, the experiment halts with output $b = 1$.

- **Output phase:** If the adversary $\mathcal{A}$ finishes making its queries and the experiment has not halted (as a result of a decryption query), then the experiment outputs $b = 0$.

We now define correctness, compactness, and update efficiency for Π_{RABE} as follows:

- **Correctness:** We say that Π_{RABE} is *correct* if, for all (possibly unbounded) adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ where b is the output of the above game.

- **Compactness:** We say that Π_{RABE} is *compact* if there exists a universal polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for all $i \in [N]$ (where N is the user bound declared by $\mathcal{A}$ at the start of the game), the master public key mpk_i has size $\mathsf{poly}(\lambda, |A_i|, \log i)$. We additionally require that the helper decryption key hsk^* always has size $\mathsf{poly}(\lambda, |A|, \log N)$. Here, A_i is the union of all attribute sets registered up through registration query i and A is the union of all registered attribute sets throughout the game.

- **Update efficiency:** We say that Π_{RABE} is *update-efficient* if the challenger invokes the update algorithm Update at most $O(\log N)$ times, and each invocation runs in $\mathsf{poly}(\lambda, |A|, \log N)$ time in the RAM model of computation.

Remark A.3 (Malicious Correctness)**.** Following [HLWW23], we define correctness so that, once a (non-challenge) honest user generates their key, it is immediately registered. One could consider a stronger correctness notion where a "rushing" adversary is allowed to ask an honest user to generate a key, then later on decide whether to register it. Under this notion, the adversary can mount a "stealing" attack against the powers-of-two transform of [HLWW23] by claiming slots associated with an honest user. Large index spaces natively prevent this kind of attack. For instance, users could choose their indices to be verification keys for a one-time signature scheme, which is used to sign their public keys. The key curator would then reject all public keys with an invalid signature. The resulting registered ABE scheme would then satisfy a computational notion of correctness where a computationally bounded adversary is unable to register keys in a way that violates correctness for honest users.

Our construction of slotted registered ABE with a large index space is adaptively secure without corruptions, which directly implies static security of the slotted scheme. In Appendix B, we show that the powers-of-two-style transformation from [GHMR18, HLWW23] lifts statically secure slotted registered ABE schemes with large indices to statically secure standard registered ABE schemes with stateless key generation (that support dynamic registration). We thus define static security of registered ABE here.

Definition A.4 (Static Security)**.** Let $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ be a registered ABE scheme with attribute universe $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$, policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$, message space $\mathcal{M} = \{\mathcal{M}_\lambda\}_{\lambda\in\mathbb{N}}$, and index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. The static security game between an adversary $\mathcal{A}$ and a challenger is parameterized by positive integer-valued polynomials $N = N(\lambda)$ and $R = R(\lambda)$ such that $R(\lambda) \le N(\lambda)$ for all $\lambda \in \mathbb{N}$, a collection of index sequences $I = \{I_\lambda\}_{\lambda\in\mathbb{N}}$ where each $I_\lambda = \{\mathsf{id}_i\}_{i\in[R]} \subseteq \mathcal{I}_\lambda$, a bit $b \in \{0,1\}$, and a security parameter λ and consists of the following phases:

- **Setup phase:** On input the security parameter 1^λ, number of slots 1^N, and the sequence of indices I_λ, the adversary $\mathcal{A}$ begins by declaring a set $C \subseteq [0, R-1]$ where each $i \in C$ indicates that the $(i+1)^{\text{th}}$ registration query will be a corrupted registration query. The challenger responds to $\mathcal{A}$ with $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$ and additionally initializes a curator state $\mathsf{aux} = \perp$, a master public key $\mathsf{mpk} = \perp$, and a counter $\mathsf{ctr} = 0$.

- **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following queries:
  - **Corrupted registration query:** The adversary submits a public key pk and a set of attributes $S \subseteq \mathcal{U}_\lambda$. If $\mathsf{ctr} \notin C$, then the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr} \leftarrow \mathsf{ctr}+1$ and registers the key by computing $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{id}_{\mathsf{ctr}}, \mathsf{pk}, S)$. Finally, it updates the master public key and curator state by setting $\mathsf{mpk} \leftarrow \mathsf{mpk}'$ and $\mathsf{aux} \leftarrow \mathsf{aux}'$, respectively.
  - **Honest registration query:** The adversary submits a set of attributes $S \subseteq \mathcal{U}_\lambda$. If $\mathsf{ctr} \in C$, then the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr} \leftarrow \mathsf{ctr}+1$, generates $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, \mathsf{id}_{\mathsf{ctr}})$, and registers the public key by computing $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{id}_{\mathsf{ctr}}, \mathsf{pk}, S)$. Finally, it updates the master public key $\mathsf{mpk} \leftarrow \mathsf{mpk}'$ and curator state $\mathsf{aux} \leftarrow \mathsf{aux}'$, and it sends pk to $\mathcal{A}$.
- **Challenge phase:** The adversary submits a challenge policy $P^* \in \mathcal{P}_\lambda$ and messages $\mu_0^*, \mu_1^* \in \mathcal{M}_\lambda$. The challenger computes a challenge ciphertext $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P^*, \mu_b^*)$ which it provides to $\mathcal{A}$.
- **Output phase:** At the end of the experiment, the adversary $\mathcal{A}$ outputs a bit b^*, which is the output of the game.

We say that an adversary $\mathcal{A}$ is admissible for the static security game if, for every corrupted registration query, the set S submitted by $\mathcal{A}$ does not satisfy P^* (i.e., attributes associated with corrupted users do not satisfy the challenge policy). Next, we define the adversary's advantage in the static security game with parameters (N, R, I) to be

$$\mathsf{StaticAdv}_{\mathcal{A},N,R,I}(\lambda) \coloneqq |\Pr[b^* = 1 : b = 0] - \Pr[b^* = 1 : b = 1]|.$$

We say that Π_{RABE} is *statically secure* for a fixed choice of (N, R, I) if, for all efficient admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

$$\mathsf{StaticAdv}_{\mathcal{A},N,R,I}(\lambda) \leq \mathsf{negl}(\lambda).$$

Finally, we say that Π_{RABE} is statically secure if it is statically secure for all positive integer-valued polynomials $N = N(\lambda)$ and $R = R(\lambda)$ and all collections of index sequences $I = \{I_\lambda\}_{\lambda \in \mathbb{N}}$ where each $I_\lambda = \{\mathsf{id}_i\}_{i \in [R]} \subseteq \mathcal{I}_\lambda$.

Remark A.5 (Bounded Registered ABE)**.** The work of [HLWW23] defines *bounded* and *unbounded* registered ABE, where the latter does not require an a priori bound on the maximum number of users that can register in the system. As we only consider bounded registered ABE, we directly integrate the 1^N bound into the syntax and definitions above.

Remark A.6 (Attribute-Universe-Independent Setup)**.** Some registered ABE constructions allow the size of the common reference string to scale with the size of the attribute universe, which must therefore be provided as input $1^{|\mathcal{U}|}$ to the setup procedure. We omit this since our construction does not require such a bound.

Remark A.7 (Index Space)**.** Our formulation of registered ABE introduces indices from an index space, which may be exponentially large. This serves as a generalization of the standard formulation, which can be recovered as a special case by letting the index space be the set of integers $\{1, \ldots, N\}$.

Remark A.8 (Stateless Key Generation)**.** The work of [FWW23] defines *stateless* key generation, where users need only the long-term common reference string to generate keys. This is in contrast to *stateful* key generation, where users must first obtain an up-to-date state from the key curator, which they use (in addition to the CRS) during key generation. The syntax above reflects that our construction has stateless key generation.

B From Slotted Registered ABE with Large Index Space to Registered ABE with Stateless Key Generation

The work of [HLWW23] constructs registered ABE via a generic transformation applied to the simpler notion of slotted registered ABE, where each user in the system is assigned a slot index used for key generation, and the key curator runs an *aggregation* algorithm that takes all users' slot indices, public keys, and attributes simultaneously and outputs a master public key. A drawback of existing pairing-based constructions of slotted registered ABE is that the user

slot indices are integers $i \in [N]$, where N is the bound on the number of users. The [HLWW23] transformation thus requires the key curator to assign to every user in the system a different slot index, which they will use to generate their public key. The [HLWW23] transform handles this by requiring users to first obtain an index i from the key curator's current state. Alternatively, each user can generate public keys for multiple distinct indices and have the key curator select indices that are still unassigned during registration. This comes at the cost of increasing the size of the public key.

In this section, we show that (a small modification to) the [HLWW23] transform applied to statically-secure slotted registered ABE with large index space directly results in a statically-secure standard registered ABE scheme with stateless key generation (i.e., users can generate their keys using a fixed index independent from the state of the system). Applied to our slotted registered ABE construction (Construction 4.1), we immediately obtain a registered ABE scheme with stateless key generation. The transformation and proofs are nearly identical to those of the [HLWW23] compiler, except that each user has a unique, static index they use to generate keys instead of obtaining up-to-date unused slot indices from the key curator. Furthermore, in each user's helper decryption key, the key curator provides registration ordering information used to determine when to get updates (in [HLWW23], the user obtains this in the curator state that the user obtains prior to key generation).

Construction B.1 (Slotted Registered ABE to Registered ABE)**.** Let $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}', \mathsf{KeyGen}', \mathsf{IsValid}', \mathsf{Aggregate}', \mathsf{Encrypt}', \mathsf{Decrypt}')$ 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 large index space $\mathcal{I} = \{\mathcal{I}_\lambda\}_{\lambda\in\mathbb{N}}$. We now construct a registered ABE scheme $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ over the same attribute space $\mathcal{U}$, policy space $\mathcal{P}$, and index space $\mathcal{I}$ as follows.

- $\mathsf{Setup}(1^\lambda, 1^N)$: On input the security parameter λ and the number of slots $N = 2^n$ (assumed for simplicity to be a power of two), the setup algorithm runs the setup algorithm for $n+1$ copies of the slotted registered ABE scheme. Specifically, for each $k \in [0, n]$ it samples $\mathsf{crs}_k \leftarrow \mathsf{Setup}'(1^\lambda, 1^{2^k})$, then it returns $\mathsf{crs} = (\mathsf{crs}_0, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$.
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{id})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$ and an index $\mathsf{id} \in \mathcal{I}_\lambda$, the key generation algorithm generates a public/secret key-pair for each of the $n+1$ underlying schemes. Specifically, for each $k \in [0, n]$ it generates $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{KeyGen}'(\mathsf{crs}_k, \mathsf{id})$, then it outputs the public key $\mathsf{pk} = (\mathsf{id}, \mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and the secret decryption key $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
- $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{id}, \mathsf{pk}, S)$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$, the curator state aux, an index $\mathsf{id} \in \mathcal{I}_\lambda$, a public key $\mathsf{pk} = (\mathsf{id}, \mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and a set of attributes $S \subseteq \mathcal{U}_\lambda$, the registration algorithm proceeds as follows:
  - If $\mathsf{aux} = \bot$, the algorithm initializes a counter $\mathsf{ctr} = 0$, empty dictionaries $\mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3 = \bot$, an empty index set $I = \emptyset$, and $\mathsf{mpk} = (\mathsf{ctr}, \bot, \ldots, \bot)$ then sets $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$. Otherwise, it parses $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$ where $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$.
  - If $\mathsf{ctr} \geq N = 2^n$, the algorithm halts and outputs the current master public key mpk and curator state aux.
  - For each $k \in [0, n]$, the algorithm checks that $\mathsf{IsValid}'(\mathsf{crs}_k, \mathsf{id}, \mathsf{pk}_k) = 1$. If any of the checks fail, the algorithm halts and outputs the current master public key mpk and curator state aux. Otherwise, it updates the dictionary $\mathsf{D}_1[\mathsf{pk}] \leftarrow \mathsf{ctr} + 1$.
  - For each $k \in [0, n]$, the algorithm sets $i_k \leftarrow (\mathsf{ctr} \bmod 2^k) + 1$ and updates $\mathsf{D}_2[k, i_k] \leftarrow (\mathsf{pk}, S)$. It then performs the following steps (for each given k):
    - If $i_k = 2^k$, the registration algorithm computes
    $$(\mathsf{mpk}'_k, \mathsf{hsk}_{k,1}, \ldots, \mathsf{hsk}_{k,2^k}) \leftarrow \mathsf{Aggregate}'(\mathsf{crs}_k, \mathsf{D}_2[k, 1], \ldots, \mathsf{D}_2[k, 2^k])$$
    and updates $\mathsf{D}_3[\mathsf{ctr} + 1 - 2^k + i, k] \leftarrow \mathsf{hsk}_{k,i}$ for each $i \in [2^k]$.
    - If $i_k \neq 2^k$, the registration algorithm sets $\mathsf{mpk}'_k \leftarrow \mathsf{mpk}_k$.
  - Finally, output the updated master public key $\mathsf{mpk} = (\mathsf{ctr}+1, \mathsf{mpk}'_0, \mathsf{mpk}'_1, \ldots, \mathsf{mpk}'_n)$ and updated curator state $\mathsf{aux}' = (\mathsf{ctr} + 1, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$.
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu)$: On input the master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$, an access policy $P \in \mathcal{P}_\lambda$, and a message $\mu \in \mathcal{M}_\lambda$, the encryption algorithm does the following for each $k \in [0, n]$:

- If $\mathsf{mpk}_k = \bot$, then let $\mathsf{ct}_k = \bot$.
- Otherwise, compute $\mathsf{ct}_k \leftarrow \mathsf{Encrypt}'(\mathsf{mpk}_k, P, \mu)$.

and outputs $\mathsf{ct} = (\mathsf{ctr}, \mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$.

- $\mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$, the curator state $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$, and a user's public key $\mathsf{pk} = (\mathsf{id}, \mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, the algorithm retrieves $\mathsf{ctr}_{\mathsf{pk}} \leftarrow \mathsf{D}_1[\mathsf{pk}]$, sets $\mathsf{hsk}_k \leftarrow \mathsf{D}_3[\mathsf{ctr}_{\mathsf{pk}}, k]$ for $k \in [0, n]$, and outputs $\mathsf{hsk} = (\mathsf{ctr}_{\mathsf{pk}} - 1, \mathsf{hsk}_0, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_n)$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input a secret decryption key $\mathsf{sk} = (\mathsf{sk}_0, \mathsf{sk}_1, \ldots, \mathsf{sk}_n)$, a helper decryption key $\mathsf{hsk} = (\mathsf{ctr}_{\mathsf{hsk}}, \mathsf{hsk}_0, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_n)$, and a ciphertext $\mathsf{ct} = (\mathsf{ctr}_{\mathsf{ct}}, \mathsf{ct}_0, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$, the decryption algorithm outputs $\bot$ if $\mathsf{ctr}_{\mathsf{ct}} \leq \mathsf{ctr}_{\mathsf{hsk}}$. Otherwise, it computes the largest index k on which $\mathsf{ctr}_{\mathsf{ct}}$ and $\mathsf{ctr}_{\mathsf{hsk}}$ differ (where bits are 0-indexed starting from the least significant bit). If $\mathsf{hsk}_k = \bot$, then the decryption algorithm outputs GetUpdate. Otherwise, it computes and outputs $\mu = \mathsf{Decrypt}'(\mathsf{sk}_k, \mathsf{hsk}_k, \mathsf{ct}_k)$. Note that μ may be $\bot$.

Remark B.2 (Common Message Space)**.** Construction B.1 implicitly requires the message space for all $n+1$ underlying slotted registered ABE schemes to be the same (recall that technically, the message space is determined during setup). This is without loss of generality since we can take any scheme with arbitrary message space and convert it into a scheme with message space $\{0, 1\}$. Then, we apply the transformation to the bit encryption schemes.

Theorem B.3 (Correctness)**.** *Suppose Π_{sRABE} is complete and correct. Then Construction B.1 is correct.*

Proof. To argue that an efficient adversary $\mathcal{A}$ cannot win the correctness game except with negligible probability, we will require the following observation that is proven in [HLWW23].

Claim B.4 ([HLWW23, Claim 6.4])**.** *Let $0 \leq x, y < 2^{n+1} - 1$ be integers with binary representations $x = x_n \ldots x_1 x_0$ and $y = y_n \ldots y_1 y_0$. Suppose $x < y$ and let $k_{x,y}$ be the index of the most significant bit on which x and y differ (where the bits are 0-indexed starting at the least significant bit). Then $k_{x,y} \leq k_{x,y+1}$. Moreover, if $k_{x,y} < k_{x,y+1}$, then $y + 1 = 0 \bmod 2^{k_{x,y}} + 1$.*

Continuing with the proof, suppose the adversary $\mathcal{A}$ declares the bound $N = 2^n$ and let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$. By construction, we have that $\mathsf{crs} = (\mathsf{crs}_0, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$ where $\mathsf{crs}_k \leftarrow \mathsf{Setup}'(1^\lambda, 1^{2^k})$ for $k \in [0, n]$. Now suppose that none of the crs_k are in the negligible fraction of "bad" common reference strings for which the correctness of Π_{sRABE} is not guaranteed to hold. Let $\mathsf{negl}'(\cdot)$ be a negligible bound on the fraction of "bad" common reference strings. Then the supposition must occur with probability at least $1 - n \cdot \mathsf{negl}'(\lambda)$, which is overwhelming as n is polylogarithmic in λ.

Consider the curator state aux at some arbitrary point in the game. We have that $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$ where $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$. We will show that the following invariant holds for mpk.

Lemma B.5. *Let $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$ for $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$ be the curator state maintained by the challenger at any point in the correctness game after the adversary has made a challenge registration query. Let $\mathsf{pk}^* = (\mathsf{id}^*, \mathsf{pk}_0^*, \mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*)$ be the challenge public key the challenger sampled when responding to the challenge registration query, and let $\mathsf{hsk}^* = (\mathsf{ctr}^*, \mathsf{hsk}_0^*, \mathsf{hsk}_1^*, \ldots, \mathsf{hsk}_n^*)$ be the (current) associated helper secret key. Let k' be the most significant bit on which the binary representations of ctr^* and ctr differ (where the bits are 0-indexed starting at the least significant bit). Then the master public key $\mathsf{mpk}_{k'}$ was the output of a call to $\mathsf{Aggregate}'(\mathsf{crs}_{k'}, \cdot)$ on a set of key-pair/attribute tuples that included the challenge (pk^*, S^*).*

Proof. The invariant follows by induction. We begin with the base case corresponding to the state of the challenger immediately after the adversary's target key registration query. Let $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$ be the curator state prior to this query, where $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$. Now, when $\mathcal{A}$ submits S^*,

- The challenger generates $(\mathsf{pk}^*, \mathsf{sk}^*)$ for a new index $\mathsf{id}^* \in \mathcal{I}_\lambda$ by computing $(\mathsf{pk}_k^*, \mathsf{sk}_k^*) \leftarrow \mathsf{KeyGen}'(\mathsf{crs}_k, \mathsf{id}^*)$ for each $k \in [0, n]$ and setting $\mathsf{pk}^* = (\mathsf{id}^*, \mathsf{pk}_0^*, \mathsf{pk}_1^*, \ldots, \mathsf{pk}_n^*)$ and $\mathsf{sk}^* = (\mathsf{sk}_0^*, \mathsf{sk}_1^*, \ldots, \mathsf{sk}_n^*)$.
- By the completeness of Π_{sRABE}, $\mathsf{IsValid}'(\mathsf{crs}_k, \mathsf{id}^*, \mathsf{pk}_k^*) = 1$ for all $k \in [0, n]$, so registration proceeds.
- The challenger begins registration by recording $\mathsf{D}_1[\mathsf{pk}^*] \leftarrow \mathsf{ctr} + 1$. Then, for each $k \in [0, n]$, it computes $i_k^* \leftarrow (\mathsf{ctr} \bmod 2^k) + 1$ and records $\mathsf{D}_2[k, i_k^*] = (\mathsf{pk}^*, S^*)$.

- Let k' be the index of the most significant bit on which ctr and ctr + 1 differ (where bits are 0-indexed starting from the least significant bit). Then $(\mathsf{ctr} \bmod 2^{k'}) = 2^{k'} - 1$ (because the k'^{th} bit of ctr is 0 and all less significant bits of ctr are 1) hence $i^*_{k'} = (\mathsf{ctr} \bmod 2^{k'}) + 1 = 2^{k'}$. The registration algorithm thus aggregates by computing

$$(\mathsf{mpk}'_{k'}, \mathsf{hsk}_{k',1}, \ldots, \mathsf{hsk}_{k',2^{k'}}) \leftarrow \mathsf{Aggregate}'(\mathsf{crs}_{k'}, \{\mathsf{D}_2[k', i]\}_{i \in [1,2^{k'}]}),$$

  and it records $\mathsf{D}_3[\mathsf{ctr} + 1 - 2^{k'} + i, k'] \leftarrow \mathsf{hsk}_{k',i}$ for each $i \in [2^{k'}]$. The registration algorithm creates a new master public key mpk containing $\mathsf{mpk}'_{k'}$ (along with the other existing or newly aggregated mpk'_k) and a new curator state aux′ that contains $\mathsf{ctr}' = \mathsf{ctr} + 1$.

- The challenger finally computes the target helper decryption key by retrieving $\mathsf{hsk}^*_k \leftarrow \mathsf{D}_3[\mathsf{ctr}', k]$ for each $k \in [0, n]$ and setting $\mathsf{hsk}^* = (\mathsf{ctr}^*, \mathsf{hsk}^*_0, \mathsf{hsk}^*_1, \ldots, \mathsf{hsk}^*_n)$ where $\mathsf{ctr}^* = \mathsf{ctr}' - 1 = \mathsf{ctr}$. Furthermore, it updates $\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]} \leftarrow \mathsf{mpk}$ and $\mathsf{aux} \leftarrow \mathsf{aux}'$, the latter of which includes $\mathsf{ctr} \leftarrow \mathsf{ctr}'$.

- We thus have that k' is exactly the index of the most significant bit on which ctr^* and ctr differ, and $\mathsf{mpk}'_{k'}$ contains (pk^*, S^*). We have shown that the invariant holds for our base case.

Now, suppose the invariant holds, and the adversary $\mathcal{A}$ submits a non-challenge registration query consisting of a public key $\mathsf{pk} = (\mathsf{id}, \mathsf{pk}_0, \ldots, \mathsf{pk}_n)$ and an attribute set S to the challenger.

- For each $k \in [0, n]$, if $\mathsf{IsValid}'(\mathsf{crs}_k, \mathsf{id}, \mathsf{pk}_k) = 0$ then the registration algorithm directly outputs the current master public key mpk and curator state $\mathsf{aux}' = \mathsf{aux}$. The invariant trivially holds in this case.

- If the above checks pass, the challenger begins registration by recording $\mathsf{D}_1[\mathsf{pk}] \leftarrow \mathsf{ctr} + 1$. Then, for each $k \in [0, n]$, it computes $i_k \leftarrow (\mathsf{ctr} \bmod 2^k) + 1$ and records $\mathsf{D}_2[k, i_k] = (\mathsf{pk}, S)$.

- Let k'_{old} be the index of the most significant bit on which ctr^* and ctr differ and k'_{new} be the index of the most significant bit on which ctr^* and ctr+1 differ. By Claim B.4, we have that $k'_{\text{old}} \leq k'_{\text{new}}$. Consider the following two cases.

  - If $k'_{\text{old}} = k'_{\text{new}}$ $(= k')$, then the k'^{th} bit of ctr and ctr + 1 are equal and $(\mathsf{ctr} \bmod 2^{k'}) + 1 < 2^{k'}$. It follows that the registration procedure does not update $\mathsf{mpk}_{k'}$, i.e., it sets $\mathsf{mpk}'_{k'} \leftarrow \mathsf{mpk}_{k'}$. By the inductive hypothesis, the challenge key pk^* was aggregated in $\mathsf{mpk}_{k'}$ and is thus aggregated in the $\mathsf{mpk}'_{k'}$ in mpk output by registration. The invariant holds in this case.
  - If $k'_{\text{old}} < k'_{\text{new}}$, then $\mathsf{ctr} + 1 = 0 \pmod{2^{k'_{\text{new}}}}$ per Claim B.4 so $i_{k'_{\text{new}}} = 2^{k'_{\text{new}}}$. The challenger thus computes

$$(\mathsf{mpk}'_{k'_{\text{new}}}, \mathsf{hsk}_{k'_{\text{new}},1}, \ldots, \mathsf{hsk}_{k'_{\text{new}},2^{k'_{\text{new}}}}) \leftarrow \mathsf{Aggregate}'(\mathsf{crs}_{k'_{\text{new}}}, \{\mathsf{D}_2[k'_{\text{new}}, i]\}_{i \in [1,2^{k'_{\text{new}}}]}).$$

    Observe that $\mathsf{D}_2[k'_{\text{new}}, 1], \ldots, \mathsf{D}_2[k'_{\text{new}}, 2^{k'_{\text{new}}}]$ store (pk_i, S_i) for $i \in [\mathsf{ctr} - 2^{k'_{\text{new}}} + 1, \mathsf{ctr}]$, where (pk_i, S_i) denotes the i^{th} successful (challenge or non-challenge) registration query. Since the most significant differing bit between ctr and ctr^* is k'_{old}, we have that $\mathsf{ctr} - \mathsf{ctr}^* \leq 2^{k'_{\text{old}}+1} - 1 \leq 2^{k'_{\text{new}}} - 1$. It follows that $\mathsf{ctr}^* \in [\mathsf{ctr} - 2^{k'_{\text{new}}} + 1, \mathsf{ctr}]$, which implies that (pk^*, S^*) was aggregated into $\mathsf{mpk}'_{k'_{\text{new}}}$. Now, when the challenger updates $\mathsf{mpk} \leftarrow \mathsf{mpk}'$ and $\mathsf{aux} \leftarrow \mathsf{aux}'$ (including $\mathsf{ctr} \leftarrow \mathsf{ctr} + 1$), we have that k'_{new} is exactly the index of the most significant bit on which ctr^* and ctr differ, and (pk^*, S^*) was aggregated into $\mathsf{mpk}'_{k'_{\text{new}}}$ which is contained in the mpk output by registration. We have shown that the invariant holds in this case.

The above shows that if the invariant holds before a given non-challenge registration query (after the target query), then it also holds after that query. Furthermore, because encryption and decryption queries do not modify the curator state aux (and hence ctr) nor does update (following the challenge registration query) change the ctr^* stored in hsk^*, the invariant holds after those queries as well (provided it holds before). The claim now follows by induction. □

We finally analyze encryption and decryption queries. For each encryption query on registration (query) index i, policy P, and message μ, recall that the challenger increments the counter $\mathsf{ctr}[\mathsf{enc}] \leftarrow \mathsf{ctr}[\mathsf{enc}] + 1$ and generates $\mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]} = (\mathsf{ctr}_{\mathsf{ctr}[\mathsf{enc}],\mathsf{ct}}, \mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}],0}, \mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}],1}, \ldots, \mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}],n})$ where $\mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}],k} \leftarrow \mathsf{Encrypt}'(\mathsf{mpk}_{i,k}, P, \mu)$ for $k \in [0, n]$. Now, consider a decryption query on encryption (query) index j, i.e., of the ciphertext ct_j output by encryption on some

registration (query) index i. First observe that encryption queries must occur after the challenge registration query. Hence $\mathsf{ctr}_{j,\mathsf{ct}} > \mathsf{ctr}^*$ for the challenge helper decryption key $\mathsf{hsk}^* = (\mathsf{ctr}^*, \mathsf{hsk}_0^*, \mathsf{hsk}_1^*, \ldots, \mathsf{hsk}_n^*)$. Now, the decryption algorithm begins by computing the largest index k_j on which $\mathsf{ctr}_{j,\mathsf{ct}}$ and ctr^* differ (where the bits are 0-indexed starting from the least significant bit). If $\mathsf{hsk}_{k_j}^* \neq \bot$, then the decryption algorithm outputs $\mu_j' = \mathsf{Decrypt}'(\mathsf{sk}_{k_j}^*, \mathsf{hsk}_{k_j}^*, \mathsf{ct}_{j,k_j})$ where $\mathsf{sk}^* = (\mathsf{sk}_0^*, \mathsf{sk}_1^*, \ldots, \mathsf{sk}_n^*)$ is the challenge secret key. If $\mathsf{hsk}_{k_j}^* = \bot$, then the challenger performs the update procedure for hsk^*, which sets $\mathsf{hsk}_{k_j}^* \leftarrow \mathsf{D}_3[\mathsf{ctr}^* + 1, k_j]$ (because the registration algorithm set $\mathsf{D}_1[\mathsf{pk}^*] \leftarrow \mathsf{ctr}^* + 1$). The decryption algorithm then outputs $\mu_j' = \mathsf{Decrypt}'(\mathsf{sk}_{k_j}^*, \mathsf{hsk}_{k_j}^*, \mathsf{ct}_{j,k_j})$. By the invariant in Lemma B.5, we have that (pk^*, S^*) was aggregated into the master public key mpk_{i,k_j} used to encrypt ct_{j,k_j}. Furthermore, $\mathsf{hsk}_{k_j}^* = \mathsf{hsk}_{k_j,\mathsf{ctr}^*}$, and hence $\mu_j' = \mu$ by the correctness of Π_{sRABE} (recalling our supposition on the crs). □

Theorem B.6 (Compactness). *Suppose Π_{sRABE} is compact. Then Construction B.1 is compact.*

Proof. Suppose the adversary has made i registration queries. Then $\mathsf{mpk}_i = (\mathsf{ctr}_i, \mathsf{mpk}_{i,0}, \mathsf{mpk}_{i,1}, \ldots, \mathsf{mpk}_{i,n})$ where $N = 2^n$ is the bound on the number of users (declared by the adversary), $\mathsf{ctr}_i = i$ by construction, and for each $k \in [0, n]$, $\mathsf{mpk}_{i,k}$ is either an aggregated public key for the underlying slotted registered ABE scheme Π_{sRABE} or it is uninitialized and hence set to $\bot$. Now let us consider which $\mathsf{mpk}_{i,k}$ have been aggregated, recalling that this only occurs if $(\mathsf{ctr} \bmod 2^k) + 1 = 2^k$ where ctr is the counter contained in the master public key at any given point. We thus have that, up to and including the i^{th} registration query, the largest $k' \in [0, n]$ such that $(\mathsf{ctr}_j \bmod 2^{k'}) + 1 = 2^{k'}$ for $\mathsf{ctr}_j \leq i$ is simply $k' = \lfloor \log i \rfloor$. It follows that $\mathsf{mpk}_{i,k}$ has only been aggregated for the underlying Π_{sRABE} schemes supporting at most $2^{\lfloor \log i \rfloor} \leq i$ slots. By the compactness of Π_{sRABE}, it follows that the size of mpk_i is bounded by $\log i \cdot \mathsf{poly}'(\lambda, |A_i|, \log i) = \mathsf{poly}(\lambda, |A_i|, \log i)$. Here, $\mathsf{poly}'(\cdot, \cdot, \cdot)$ denotes the universal polynomial that bounds the size of $\mathsf{mpk}_{i,k}$ output by aggregation, and A_i is the union of all attribute sets registered through the i^{th} registration query.

Now consider the helper decryption key $\mathsf{hsk}^* = (\mathsf{ctr}^*, \mathsf{hsk}_0^*, \mathsf{hsk}_1^*, \ldots, \mathsf{hsk}_n^*)$ at any point in the game. By the compactness of Π_{sRABE}, we have that the size of hsk^* is bounded by $\log N \cdot \mathsf{poly}'(\lambda, |A|, \log N) = \mathsf{poly}(\lambda, |A|, \log N)$, where A is the union of all attribute sets registered by the adversary throughout the game. □

Theorem B.7 (Update Efficiency). *Suppose Π_{sRABE} is compact. Then Construction B.1 is update-efficient.*

Proof. Recall that the challenger invokes the update algorithm every time decryption outputs $\mu_j' = \mathsf{GetUpdate}$. Now consider the helper decryption key $\mathsf{hsk}^* = (\mathsf{ctr}^*, \mathsf{hsk}_0^*, \mathsf{hsk}_1^*, \ldots, \mathsf{hsk}_n^*)$. By construction, decryption only outputs GetUpdate when a particular $\mathsf{hsk}_k^* = \bot$ (where k is computed based on the ciphertext and ctr^*). If this occurs, the challenger updates hsk^* by retrieving $i^* \leftarrow \mathsf{D}_1[\mathsf{pk}^*]$ and setting $\mathsf{hsk}_k^* \leftarrow \mathsf{D}_3[i^*, k]$ for $k \in [0, n]$. We now analyze how $\mathsf{D}_3[i^*, k]$ is assigned. Observe that registration assigns $\mathsf{D}_3[\mathsf{ctr} + 1 - 2^k + i, k]$ only for ctr such that $(\mathsf{ctr} \bmod 2^k) + 1 = 2^k$, or equivalently, $\mathsf{ctr} = c \cdot 2^k - 1$ for some unique $c \in \mathbb{N}$. Because $i \in [2^k]$ and ctr only increases, it follows that any given $\mathsf{D}_3[i', k]$ is assigned exactly once. Consequently $\mathsf{hsk}_k^* \leftarrow \mathsf{D}_3[i^*, k]$ is never overwritten, and updates to any entry of $\mathsf{hsk}^* = (\mathsf{ctr}^*, \mathsf{hsk}_0^*, \mathsf{hsk}_1^*, \ldots, \mathsf{hsk}_n^*)$ is permanent. Finally, there are only $n = \log N$ different hsk_k^* in hsk^* initially set to $\bot$ that can be updated, and once they are all non-$\bot$, decryption ceases to output GetUpdate and the challenger no longer invokes the update algorithm.

The efficiency of the update runtime follows from the compactness of Π_{sRABE}. In particular, updating n elements in $\mathsf{hsk}^* = (\mathsf{ctr}^*, \mathsf{hsk}_0^*, \mathsf{hsk}_1^*, \ldots, \mathsf{hsk}_n^*)$ takes (assuming the dictionaries can be searched in constant time) running time bounded by $\log N \cdot \mathsf{poly}'(\lambda, |A|, \log N) = \mathsf{poly}(\lambda, |A|, \log N)$, where $\mathsf{poly}'(\cdot, \cdot, \cdot)$ denotes the universal polynomial that bounds the size of hsk_k^* output by aggregation and A is the union of all attribute sets registered through the game. □

Theorem B.8 (Static Security). *Suppose Π_{sRABE} is index-set-selectively secure without corruptions. Then Construction B.1 is statically secure.*

Proof. Take any positive integer-valued polynomials $N = N(\lambda)$ and $R = R(\lambda)$ with $R(\lambda) \leq N(\lambda)$ for all λ, and any collection of index sequences $I = \{I_\lambda\}_{\lambda \in \mathbb{N}}$ where each $I_\lambda = \{\mathsf{id}_i\}_{i \in [R]} \subseteq \mathcal{I}_\lambda$. Consider the following $n+2$ hybrids implicitly parameterized by an efficient admissible adversary $\mathcal{A}$ and the security parameter λ, where $n = \log N$. We will assume, without loss of generality, that $\mathcal{A}$ does not make corrupted or honest registration queries that do not match the set C it declared, and that it does not submit invalid public keys. In particular, given an $\mathcal{A}$ that does not satisfy these, we can construct one that does by forwarding valid oracle queries and internally returning $\bot$ otherwise. We write the full sequence of hybrids below as a single experiment parameterized by a variable $k^* \in [0, n+1]$ and use green to denote changes.

- **Hybrid** Hyb_{k^*}: This is the static security game except that the challenger encrypts μ_1^* for the first k^* components of the challenge ciphertext and μ_0^* for the remaining components. We recall the main steps here:
  - **Setup phase:** The adversary $\mathcal{A}$ begins by submitting a set $C \subseteq [0, R-1]$ where each $i \in C$ indicates that the $(i+1)^{\text{th}}$ registration query will be a corrupted registration query. The challenger responds to $\mathcal{A}$ with $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^N)$. By construction, $\mathsf{crs} = (\mathsf{crs}_0, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$ where $\mathsf{crs}_k \leftarrow \mathsf{Setup}'(1^\lambda, 1^{2^k})$ for $k \in [0, n]$. The challenger also initializes a counter $\mathsf{ctr} = 0$, an empty curator state $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$ and an empty master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$ where each $\mathsf{mpk}_k = \bot$.
  - **Query phase:** During the query phase, the adversary $\mathcal{A}$ is able to make the following queries:
    - **Corrupted registration query:** The adversary submits a public key $\mathsf{pk} = (\mathsf{id}_{\mathsf{ctr}+1}, \mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and a set of attributes $S \subseteq \mathcal{U}_\lambda$. The challenger registers by checking that $\mathsf{IsValid}'(\mathsf{crs}_k, \mathsf{id}_{\mathsf{ctr}+1}, \mathsf{pk}_k) = 1$ for $k \in [0, n]$. If this succeeds, it assigns $\mathsf{D}_1[\mathsf{pk}] \leftarrow \mathsf{ctr}+1$ and for each $k \in [0, n]$, computes the index $i_k \leftarrow (\mathsf{ctr} \bmod 2^k) + 1$ and updates $\mathsf{D}_2[k, i_k] \leftarrow (\mathsf{pk}, S)$. If $i_k = 2^k$, it computes
    $$(\mathsf{mpk}'_k, \mathsf{hsk}_{k,1}, \ldots, \mathsf{hsk}_{k,2^k}) \leftarrow \mathsf{Aggregate}'(\mathsf{crs}_k, \mathsf{D}_2[k,1], \ldots, \mathsf{D}_2[k,2^k]).$$
    If $i_k \neq 2^k$, it sets $\mathsf{mpk}'_k \leftarrow \mathsf{mpk}_k$. Finally, the challenger updates the master public key by assigning $(\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n) \leftarrow (\mathsf{ctr}+1, \mathsf{mpk}'_0, \mathsf{mpk}'_1, \ldots, \mathsf{mpk}'_n)$ and additionally updates the curator state $\mathsf{aux} = (\mathsf{ctr}+1, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$.
    - **Honest registration query:** The adversary submits a set of attributes $S \subseteq \mathcal{U}_\lambda$. The challenger generates $\mathsf{pk} = (\mathsf{id}_{\mathsf{ctr}+1}, \mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ by sampling $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{KeyGen}'(\mathsf{crs}_k, \mathsf{id}_{\mathsf{ctr}+1})$ for each $k \in [0, n]$. It then registers the key using the same procedure described above, updates the master public key $\mathsf{mpk} \leftarrow \mathsf{mpk}'$ (which includes incrementing $\mathsf{ctr} \leftarrow \mathsf{ctr}+1$) and curator state $\mathsf{aux} \leftarrow \mathsf{aux}'$, and sends the public key pk to $\mathcal{A}$.
  - **Challenge phase:** The adversary submits a challenge policy $P^* \in \mathcal{P}_\lambda$ and two messages $\mu_0^*, \mu_1^* \in \mathcal{M}_\lambda$. The challenger responds with a challenge ciphertext $\mathsf{ct}^* = (\mathsf{ctr}, \mathsf{ct}_0^*, \mathsf{ct}_1^*, \ldots, \mathsf{ct}_n^*)$ by computing, for each $k \in [0, n]$, either $\mathsf{ct}_k^* \leftarrow \mathsf{Encrypt}'(\mathsf{mpk}_k, P^*, \mu_0^*)$ if $k \geq k^*$, or $\mathsf{ct}_k^* \leftarrow \mathsf{Encrypt}'(\mathsf{mpk}_k, P^*, \mu_1^*)$ if $k < k^*$, or by setting $\mathsf{ct}_k^* = \bot$ if $\mathsf{mpk}_k = \bot$.
  - **Output phase:** At the end, the adversary $\mathcal{A}$ outputs a bit b^*, which is the output of the hybrid.

Note that the counter maintained by the security game coincides with the registration counter maintained by the key curator and included in the master public key (except at the beginning of a registration query, where the former is one greater than the latter). We write them with a single variable ctr. Furthermore, note that the hybrid directly initializes an empty master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$ and an empty curator state $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$. This is equivalent to setting them to $\bot$ at the start of the game and initializing them at the start of the first (successful) registration query.

We now argue that each pair of hybrids is indistinguishable provided the adversary $\mathcal{A}$ is efficient and admissible. In particular, we bound the difference in the output distribution of each pair of consecutive hybrids in the lemma below.

Lemma B.9. *Suppose* Π_{sRABE} *is index-set-selectively secure without corruptions. Then, for all efficient admissible adversaries* $\mathcal{A}$*, there exists a negligible function* $\mathsf{negl}'(\cdot)$ *independent of* k^* *and a positive polynomial* $\mathsf{poly}'_{k^*}(\cdot)$ *such that* $|\Pr[\mathsf{Hyb}_{k^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{k^*+1}(\mathcal{A}) = 1]| \leq \mathsf{poly}'_{k^*}(\lambda) \cdot \mathsf{negl}'(\lambda)$.

Proof. Let $J = [2^{k^*}(\lfloor R/2^{k^*} \rfloor - 1) + 1, 2^{k^*} \lfloor R/2^{k^*} \rfloor]$. The lemma follows by constructing an efficient admissible reduction $\mathcal{B}$ against the static security game for Π_{sRABE} with parameters $N^* = 2^{k^*}$ (recall that $k^* \leq \log N + 1$ where $N = N(\lambda)$ is fixed above) and $I^* = \{I_\lambda^*\}_{\lambda \in \mathbb{N}}$ where $I_\lambda^* = \{\mathsf{id}_i\}_{i \in J}$ (recalling that $I_\lambda = \{\mathsf{id}_i\}_{i \in [R]}$ is fixed above). We describe the reduction below:

- The reduction $\mathcal{B}$ begins by running the adversary $\mathcal{A}$ on input $(1^\lambda, 1^N, I)$, which declares a set $C \subseteq [0, R-1]$ of corrupted registration query indices. The reduction internally computes the set $C^* = \{\mathsf{id}_i\}_{i \in J \cap \{j+1 : j \in C\}}$, then it receives $\mathsf{crs}_{k^*} \leftarrow \mathsf{Setup}'(1^\lambda, 1^{N^*})$.

- **Setup phase:** The reduction $\mathcal{B}$ samples $\mathsf{crs}_k \leftarrow \mathsf{Setup}'(1^\lambda, 1^{2^k})$ for each $k \in [0,n] \setminus \{k^*\}$. It additionally initializes a counter $\mathsf{ctr} = 0$, an empty curator state $\mathsf{aux} = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$ and an empty master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n)$ where each $\mathsf{mpk}_k = \perp$. Finally, it inputs $\mathsf{crs} = (\mathsf{crs}_0, \mathsf{crs}_1, \ldots, \mathsf{crs}_n)$ to $\mathcal{A}$.
- **Query phase:** The reduction $\mathcal{B}$ answers the queries $\mathcal{A}$ makes as follows:
  - **Corrupted registration query:** The adversary submits a public key $\mathsf{pk} = (\mathsf{id}_{\mathsf{ctr}+1}, \mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and a set of attributes $S \subseteq \mathcal{U}_\lambda$. The reduction registers the key by checking that $\mathsf{IsValid}'(\mathsf{crs}_k, \mathsf{id}_{\mathsf{ctr}+1}, \mathsf{pk}_k) = 1$ for each $k \in [0,n]$. If this succeeds, it assigns $\mathsf{D}_1[\mathsf{pk}] \leftarrow \mathsf{ctr}+1$ and, for each $k \in [0,n]$, the algorithm sets $i_k \leftarrow (\mathsf{ctr} \bmod 2^k) + 1$ and updates $\mathsf{D}_2[k, i_k] \leftarrow (\mathsf{pk}, S)$. Next, if $i_k = 2^k$, the reduction computes
    $$(\mathsf{mpk}'_k, \mathsf{hsk}_{k,1}, \ldots, \mathsf{hsk}_{k,2^k}) \leftarrow \mathsf{Aggregate}'(\mathsf{crs}_k, \mathsf{D}_2[k,1], \ldots, \mathsf{D}_2[k,2^k]).$$
    If $i_k \neq 2^k$, the reduction simply assigns $\mathsf{mpk}'_k \leftarrow \mathsf{mpk}_k$. Finally, it updates the master public key by setting $(\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n) \leftarrow (\mathsf{ctr}+1, \mathsf{mpk}'_0, \mathsf{mpk}'_1, \ldots, \mathsf{mpk}'_n)$ and additionally updates the curator state by setting $\mathsf{aux} = (\mathsf{ctr}+1, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$. Finally, it stores $\mathsf{pk}_{\mathsf{id}_{\mathsf{ctr}},k^*} \leftarrow \mathsf{pk}_{k^*}$ and $S_{\mathsf{id}_{\mathsf{ctr}}} \leftarrow S$.
  - **Honest registration query:** The adversary submits a set of attributes $S \subseteq \mathcal{U}_\lambda$. The reduction generates the public key $\mathsf{pk} = (\mathsf{id}_{\mathsf{ctr}+1}, \mathsf{pk}_0, \mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ by sampling $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{KeyGen}'(\mathsf{crs}_k, \mathsf{id}_{\mathsf{ctr}+1})$ for each $k \in [0,n] \setminus \{k^*\}$. Next, if $\mathsf{id}_{\mathsf{ctr}+1} \notin I^*_\lambda \setminus C^*$, the reduction $\mathcal{B}$ generates $(\mathsf{pk}_{k^*}, \mathsf{sk}_{k^*}) \leftarrow \mathsf{KeyGen}'(\mathsf{crs}_{k^*}, \mathsf{id}_{\mathsf{ctr}+1})$, otherwise, if $\mathsf{id}_{\mathsf{ctr}+1} \in I^*_\lambda \setminus C^*$, the reduction queries its own honest registration oracle to receive $\mathsf{pk}_{\mathsf{id}_{\mathsf{ctr}+1},k^*}$ and it assigns $\mathsf{pk}_{k^*} \leftarrow \mathsf{pk}_{\mathsf{id}_{\mathsf{ctr}+1},k^*}$. Finally, $\mathcal{B}$ registers the key following the same procedure as corrupted key registration. In particular, it checks that $\mathsf{IsValid}'(\mathsf{crs}_k, \mathsf{id}_{\mathsf{ctr}+1}, \mathsf{pk}_k) = 1$ for $k \in [0,n]$. If this succeeds, it assigns $\mathsf{D}_1[\mathsf{pk}] \leftarrow \mathsf{ctr}+1$ and, for each $k \in [0,n]$, the algorithm sets $i_k \leftarrow (\mathsf{ctr} \bmod 2^k) + 1$ and updates $\mathsf{D}_2[k, i_k] \leftarrow (\mathsf{pk}, S)$. If $i_k = 2^k$, the reduction computes
    $$(\mathsf{mpk}'_k, \mathsf{hsk}_{k,1}, \ldots, \mathsf{hsk}_{k,2^k}) \leftarrow \mathsf{Aggregate}'(\mathsf{crs}_k, \mathsf{D}_2[k,1], \ldots, \mathsf{D}_2[k,2^k]).$$
    If $i_k \neq 2^k$, the reduction directly assigns $\mathsf{mpk}'_k \leftarrow \mathsf{mpk}_k$. Finally, it updates the master public key by setting $(\mathsf{ctr}, \mathsf{mpk}_0, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_n) \leftarrow (\mathsf{ctr}+1, \mathsf{mpk}'_0, \mathsf{mpk}'_1, \ldots, \mathsf{mpk}'_n)$ and additionally updates the curator state by setting $\mathsf{aux} = (\mathsf{ctr}+1, \mathsf{D}_1, \mathsf{D}_2, \mathsf{D}_3, I, \mathsf{mpk})$. Finally, it stores $S_{\mathsf{id}_{\mathsf{ctr}}} \leftarrow S$.
- **Challenge phase:** Upon the adversary $\mathcal{A}$ submitting a challenge policy $P^* \in \mathcal{P}_\lambda$ and messages $\mu^*_0, \mu^*_1 \in \mathcal{M}_\lambda$, the reduction $\mathcal{B}$ does the following. For each $\mathsf{id}^* \in I^*_\lambda$, the reduction $\mathcal{B}$ submits the set of attributes S_{id^*} to the game it plays, and additionally provides $\mathsf{pk}_{\mathsf{id}^*,k^*}$ if $\mathsf{id}^* \in C^*$. It receives a challenge ciphertext $\mathsf{ct}^*_{k^*}$ as a response. Next, for $k \in [0,n] \setminus \{k^*\}$, the reduction computes either $\mathsf{ct}^*_k \leftarrow \mathsf{Encrypt}'(\mathsf{mpk}_k, P^*, \mu^*_0)$ if $k > k^*$, or $\mathsf{ct}^*_k \leftarrow \mathsf{Encrypt}'(\mathsf{mpk}_k, P^*, \mu^*_1)$ if $k < k^*$, or it sets $\mathsf{ct}^*_k = \perp$ if $\mathsf{mpk}_k = \perp$. Finally, it provides the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ctr}, \mathsf{ct}^*_0, \mathsf{ct}^*_1, \ldots, \mathsf{ct}^*_n)$ as input to $\mathcal{A}$.
- **Output phase:** The adversary $\mathcal{A}$ outputs a bit b^*, which the reduction $\mathcal{B}$ returns as its own output.

Observe that $\mathcal{B}$ is admissible, provided $\mathcal{A}$ is admissible. In particular, if $\mathcal{A}$ is admissible, then the policy P^* selected by $\mathcal{A}$ is not satisfied by any set of attributes submitted during a corrupted registration query (on, say, index id). Equivalently, S_{id} does not satisfy P^* for $\mathsf{id} \in \{\mathsf{id}_i\}_{i \in \{j+1 : j \in C\}}$. It follows that S_{id^*} does not satisfy P^* for $\mathsf{id}^* \in C^*$, where $C^* = \{\mathsf{id}_i\}_{i \in J \cap \{j+1 : j \in C\}}$ is the set of corrupted indices for which $\mathcal{B}$ submits public keys during the challenge phase. This is because $C^* \subseteq C$. Hence, $\mathcal{B}$ is admissible.

We now argue that the reduction $\mathcal{B}$ perfectly simulates Hyb_{k^*} to $\mathcal{A}$ when it receives an encryption of μ_0 (i.e., $b = 0$ in the static security game for Π_{sRABE}) and it perfectly simulates Hyb_{k^*+1} to $\mathcal{A}$ when it receives an encryption of μ_1 (i.e., $b = 1$ in the static security game for Π_{sRABE}). For $k \in [0,n]$ such that $k \neq k^*$, we note that the simulation for the k^{th} slotted scheme is perfect, because $\mathcal{B}$ directly executes the setup, key generation, aggregation, and encryption as is necessary. We now analyze the simulation of the $k^{*\text{th}}$ slotted scheme.

- **Setup phase:** Here, the reduction directly forwards the crs_{k^*} it receives to the adversary $\mathcal{A}$. Because the game computes $\mathsf{crs}_{k^*} \leftarrow \mathsf{Setup}'(1^\lambda, 1^{N^*})$, this phase is simulated perfectly.

- **Query phase:** We first note that the reduction may use crs_{k^*} to aggregate during the query phase; however, the outputs will eventually be overwritten. We move on to the public keys output by honest registration queries. Recall that if $\mathsf{id}_{\mathsf{ctr}+1} \notin I^*_\lambda \setminus C^*$, then the reduction directly generates $(\mathsf{pk}_{k^*}, \mathsf{sk}_{k^*}) \leftarrow \mathsf{KeyGen}'(\mathsf{crs}_{k^*}, \mathsf{id}_{\mathsf{ctr}+1})$, and if $\mathsf{id}_{\mathsf{ctr}+1} \in I^*_\lambda \setminus C^*$, then the reduction sends the $\mathsf{pk}_{\mathsf{id},k^*}$ it received from the index-set-selective security without corruptions game for the slotted scheme. The game itself computes $(\mathsf{pk}_{\mathsf{id},k^*}, \mathsf{sk}_{\mathsf{id},k^*}) \leftarrow \mathsf{KeyGen}'(\mathsf{crs}_{k^*}, \mathsf{id})$ for $\mathsf{id} \in I^*_\lambda \setminus C^*$, and thus the public keys are computed properly in either case.
- **Challenge phase:** Because the adversary re-aggregates the mpk_{k^*} component of the master public key every time $(\mathsf{ctr} \bmod 2^{k^*}) + 1 = 2^{k^*}$, we need only analyze the last time this occurs before the challenge phase. After the final R^{th} registration query, we have that $R = 2^{k^*}\lfloor R/2^{k^*} \rfloor + r$ where $r \in [0, 2^{k^*} - 1]$ is a remainder. It follows that index of the last query aggregated is $2^{k^*}\lfloor R/2^{k^*} \rfloor$, which implies that the 2^{k^*} registration queries before were also aggregated into the final master public key. In particular, the set $J = [2^{k^*}(\lfloor R/2^{k^*} \rfloor - 1) + 1, 2^{k^*}\lfloor R/2^{k^*} \rfloor]$ is exactly the indices of the registration queries that $\mathcal{A}$ expects to be aggregated into the mpk_{k^*}, and these are the same indices that the static security game for the slotted scheme aggregates. Furthermore, the reduction saves each public key and attribute set under the associated index, and hence outputs the correct keys and sets during the challenge phase. It follows that the mpk_{k^*} computed by the game the reduction $\mathcal{B}$ plays is correct and $\mathcal{B}$ receives $\mathsf{ct}^*_{k^*} \leftarrow \mathsf{Encrypt}'(\mathsf{mpk}_{k^*}, P^*, \mu^*_b)$. When $b = 0$, $\mathsf{ct}^*_{k^*}$ encrypts μ^*_0 so the reduction simulates hybrid Hyb_{k^*} to $\mathcal{A}$, and when $b = 1$, $\mathsf{ct}^*_{k^*}$ encrypts μ^*_1 so the reduction simulates hybrid Hyb_{k^*+1} to $\mathcal{A}$.

It follows that

$$\begin{aligned}|\Pr[\mathsf{Hyb}_{k^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{k^*+1}(\mathcal{A}) = 1]| &= |\Pr[b^* = 1 : b = 0] - \Pr[b^* = 1 : b = 1]| \\ &\le \mathsf{poly}'_{k^*}(\lambda) \cdot \mathsf{negl}'(\lambda)\end{aligned}$$

where b is the bit sampled in the static security game $\mathcal{B}$ plays and b^* is its output, and where $\mathsf{negl}'(\cdot)$ and $\mathsf{poly}'_{k^*}(\cdot)$ are the negligible function and positive polynomial respectively guaranteed by the static security of Π_{sRABE}. □

Theorem B.8 now follows by applying Lemma B.9 using a hybrid argument as follows:

$$\begin{aligned}\mathsf{StaticAdv}_{\mathcal{A},N,R,I}(\lambda) &= |\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{n+1}(\mathcal{A}) = 1]| \\ &\le \sum_{k^*=0}^{n} |\Pr[\mathsf{Hyb}_{k^*}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{k^*+1}(\mathcal{A}) = 1]| \\ &\le \sum_{k^*=0}^{n} \mathsf{poly}'_{k^*}(\lambda) \cdot \mathsf{negl}'(\lambda) \\ &= \mathsf{negl}'(\lambda) \cdot \sum_{k^*=0}^{n} \mathsf{poly}'_{k^*}(\lambda) \\ &= \mathsf{negl}(\lambda),\end{aligned}$$

where we have used the fact that Hyb_0 is the registered ABE static security game with $b = 0$ and Hyb_{n+1} is the registered ABE static security game with $b = 1$. □

C Generic Hardness of Assumption 4.4

In this section, we prove that Assumption 4.4 holds unconditionally in the generic bilinear group model [Sho97, BBG05]. We use the exact same model definition and conventions as [GWWW26, Appendix B]. Specifically, we model a generic asymmetric bilinear group of prime order p with associated label space $\mathcal{L}$ by three random injective functions $\varphi_1, \varphi_2, \varphi_{\mathsf{T}} : \mathbb{Z}_p \to \mathcal{L}$. An algorithm in the generic asymmetric bilinear group model then has access to the following two oracles:

- **Evaluation oracle:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$ and a group index $i \in \{1, 2, \mathsf{T}\}$, the evaluation oracle first checks that ℓ_1, ℓ_2 are in the image of φ_i. If not, it outputs $\perp$. Otherwise, it returns $\varphi_i(\varphi_i^{-1}(\ell_1) + \varphi_i^{-1}(\ell_2) \bmod p)$.

- **Pairing oracle:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, the pairing oracle first checks that ℓ_1 is in the image φ_1 and ℓ_2 is in the image of φ_2. If not, it outputs $\perp$. Otherwise, it returns $\varphi_T(\varphi_1^{-1}(\ell_1) \cdot \varphi_2^{-1}(\ell_2) \bmod p)$.

Boneh, Boyen, and Goh [BBG05] describe a set of sufficient conditions for a cryptographic assumption to hold unconditionally in the generic bilinear group model. Below, we describe a special case (specifically, the version from [GWWW26]) of the assumption (where the assumption only gives out group elements in $\mathbb{G}_1$ and $\mathbb{G}_2$) that suffices for our analysis. We start with the definition of linear independence from [BBG05] and then state the version of the hardness theorem formalized in [GWWW26, Theorem B.2].

Definition C.1 (Independence of Polynomials)**.** Let $\mathcal{P} = \{P_i\}_{i\in[k]}$ be a collection of n-variate polynomials $P_i \in \mathbb{Z}_p[X_1, \ldots, X_n]$. We say a polynomial $f \in \mathbb{Z}_p[X_1, \ldots, X_n]$ is dependent with respect to $\mathcal{P}$ if there exists coefficients $\alpha_0, \ldots, \alpha_k$ such that

$$f(X_1, \ldots, X_n) = \alpha_0 + \sum_{i\in[k]} \alpha_i P_i(X_1, \ldots, X_n).$$

Conversely, we say that f is independent with respect to $\mathcal{P}$ if f is not dependent with respect to $\mathcal{P}$.

Theorem C.2 (Generic Hardness in Prime-Order Groups [BBG05, Theorem A.2, as adapted by [GWWW26]])**.** *Let p be a prime and $\mathcal{P} = \{P_i\}_{i\in[k]}$ and $\mathcal{Q} = \{Q_j\}_{j\in[m]}$ be two collections of n-variate polynomials $P_i, Q_j \in \mathbb{Z}_p[X_1, \ldots, X_n]$ and where $P_1 = Q_1 = 1$. Let $T \in \mathbb{Z}_p[X_1, \ldots, X_n]$ be a polynomial. For an adversary $\mathcal{A}$ and a bit $b \in \{0, 1\}$, we define the following distinguishing experiment in the generic asymmetric bilinear group of order p:*

- *At the beginning of the game, the challenger samples $x_1, \ldots, x_n \xleftarrow{R} \mathbb{Z}_p$. For each $i \in [k]$, it computes $\ell_i = \varphi_1(P_i(x_1, \ldots, x_n))$ and for each $j \in [m]$, it computes $\ell'_j = \varphi_2(Q_j(x_1, \ldots, x_n))$.*
- *If $b = 0$, the challenger computes $\tau = \varphi_T(T(x_1, \ldots, x_n))$. If $b = 1$, the challenger samples $r \xleftarrow{R} \mathbb{Z}_p$ and sets $\tau = \varphi_T(r)$.*

The challenger gives $(\ell_1, \ldots, \ell_k, \ell'_1, \ldots, \ell'_m, \tau)$ to $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$ which is the output of the experiment. Let $\mathcal{PQ} = \{P_iQ_j : i \in [k], j \in [m]\}$. Let d be a bound on the total degree of the polynomials in $\mathcal{PQ} \cup \{T\}$. If T is independent of $\mathcal{PQ}$, then for all adversaries $\mathcal{A}$ making at most q queries to the generic asymmetric bilinear group oracle, it holds that

$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| \leq \frac{(q + k + m + 1)^2 d}{p}.$$

Generic hardness of Assumption 4.4. We now use Theorem C.2 to show Assumption 4.4 holds in the generic asymmetric bilinear group model.

Theorem C.3 (Generic Hardness of Assumption 4.4)**.** *Let $\lambda \in \mathbb{N}$ be a security parameter and take any sequence of sets $S = \{S_\lambda\}_{\lambda\in\mathbb{N}}$ where $S_\lambda \subseteq \{0, 1, \ldots, 2^\lambda - 1\}$ for all $\lambda \in \mathbb{N}$. Let $\mathcal{A}$ be any adversary for Assumption 4.4 with set S. If $\mathcal{A}$ makes at most $q = q(\lambda)$ generic group oracle queries, then the advantage of $\mathcal{A}$ is at most $O(q^2|S_\lambda|^3)/p$ in the generic asymmetric bilinear group model with order $p \geq 2^{2\lambda}$. In particular, whenever $p > 2^{2\lambda}$, the advantage of $\mathcal{A}$ is negligible for all sequence of sets S as long as $|S_\lambda| = \mathrm{poly}(\lambda)$.*

Proof. Let $\lambda \in \mathbb{N}$ be a security parameter, $p \geq 2^{2\lambda}$ be a group order, and $\mathcal{L}$ be the label space for the generic bilinear group model. Let $S_\lambda = \{s_1, \ldots, s_N\} \subseteq \{0, 1, \ldots, 2^\lambda - 1\}$. We now define a sequence of hybrid distributions:

- **Hybrid** Hyb_0: Sample random injective functions $\sigma_1, \sigma_2, \sigma_T : \mathbb{Z}_p \to \mathcal{L}$. Next, sample exponents $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}$, $\gamma \xleftarrow{R} \mathbb{Z}_p$, and $\tau \xleftarrow{R} \mathbb{Z}_p \setminus S_\lambda$. Let $\beta = \sum_{i\in[N]} \frac{1}{\tau - s_i}$. Define the sets

$$\mathcal{P} = \left\{r_1\tau^{i-1},\ \frac{r_1\gamma}{\tau - s_i}\right\}_{i\in[N]} \cup \{r_1\beta\},$$
$$\mathcal{Q} = \left\{r_2\tau^{i-1},\ r_2\gamma\tau^{i-1},\ r_2\gamma\beta\tau^{i-1}\right\}_{i\in[N-1]} \cup \{r_2\tau^{N-1}\}.$$

  Let $\mathsf{params} = \{\sigma_1(P)\}_{P\in\mathcal{P}} \cup \{\sigma_2(Q)\}_{Q\in\mathcal{Q}}$. Let $T = \sigma_T(r_1r_2\gamma\beta^2\tau^{N-1})$. Output (params, T). This coincides with the pseudorandom distribution in Assumption 4.4 with respect to the generators $\hat{g}_1 = g^{r_1}$ and $\hat{g}_2 = g^{r_2}$.

- **Hybrid** Hyb_1: Same as Hyb_0, except the challenger first samples $\tilde{r}_1, \tilde{r}_2 \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}$. Next, the challenger defines polynomials $Z(X) = \prod_{i\in[N]}(X - s_i)$ and $L_i(X) = Z(X)/(X - s_i) = \prod_{j\in[N]\setminus\{i\}}(X - s_j)$. It then sets

  $$r_1 = \tilde{r}_1 \cdot Z(\tau) \quad \text{and} \quad r_2 = \tilde{r}_2 \cdot Z(\tau).$$

  In particular, under this choice of variables, for all $i \in [N]$,

  $$\frac{r_1}{\tau - s_i} = \frac{\tilde{r}_1 \cdot Z(\tau)}{\tau - s_i} = \tilde{r}_1 \cdot L_i(\tau) \quad \text{and} \quad \frac{r_2}{\tau - s_i} = \frac{\tilde{r}_2 \cdot Z(\tau)}{\tau - s_i} = \tilde{r}_2 \cdot L_i(\tau).$$

  Under this choice of variables, the components of $\mathcal{P}$ and $\mathcal{Q}$ can be re-written as:

  $$\mathcal{P} = \left\{\tilde{r}_1\tau^{i-1} \cdot Z(\tau),\ \tilde{r}_1\gamma \cdot L_i(\tau)\right\}_{i\in[N]} \cup \{\tilde{r}_1 \textstyle\sum_{i\in[N]} L_i(\tau)\}$$
  $$\mathcal{Q} = \left\{\tilde{r}_2\tau^{i-1} \cdot Z(\tau),\ \tilde{r}_2\gamma\tau^{i-1} \cdot Z(\tau),\ \tilde{r}_2\gamma\tau^{i-1} \cdot \textstyle\sum_{j\in[N]} L_j(\tau)\right\}_{i\in[N-1]} \cup \{\tilde{r}_2\tau^{N-1} \cdot Z(\tau)\}.$$

  Finally, as before, let $\mathsf{params} = \{\sigma_1(P)\}_{P\in\mathcal{P}} \cup \{\sigma_2(Q)\}_{Q\in\mathcal{Q}}$. Let

  $$T = \sigma_T(r_1 r_2 \gamma \beta^2 \tau^{N-1}) = \sigma_T\left(\tilde{r}_1\tilde{r}_2\gamma\tau^{N-1}\left(\sum_{k\in[N]} L_k(\tau)\right)^2\right).$$

  Output (params, T).

- **Hybrid** Hyb_2: Same as Hyb_1, except the challenger now samples $\tilde{r}_1, \tilde{r}_2, \tau \xleftarrow{R} \mathbb{Z}_p$.
- **Hybrid** Hyb_3: Same as Hyb_2, except the challenger now samples $t \xleftarrow{R} \mathbb{Z}_p$ and sets $T = \sigma_T(t)$.
- **Hybrid** Hyb_4: Same as Hyb_3, except the challenger now reverts to sampling $\tilde{r}_1, \tilde{r}_2 \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}$ and $\tau \xleftarrow{R} \mathbb{Z}_p \setminus S_\lambda$.
- **Hybrid** Hyb_5: Same as Hyb_4, except the challenger samples $r_1, r_2 \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}$ (as in Hyb_0). This coincides with the random distribution in Assumption 4.4 with respect to the generators $\hat{g}_1 = g^{r_1}$ and $\hat{g}_2 = g^{r_2}$.

Take any adversary $\mathcal{A}$ that makes at most q queries to the generic group oracle. For an index i, let $\mathsf{Hyb}_i(\mathcal{A})$ denote the output distribution of $\mathcal{A}$ on the input (params, T) sampled using the procedure in Hyb_i. We now analyze each adjacent pair of distributions.

Lemma C.4. $\Pr[\mathsf{Hyb}_0(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1]$.

Proof. The only (syntactic) difference between these two experiments is the distribution of r_1 and r_2. In both Hyb_0 and Hyb_1, we claim that r_1, r_2 is uniform over $\mathbb{Z}_p \setminus \{0\}$. This is by construction in Hyb_0, whereas in Hyb_1, the challenger sets $r_1 = \tilde{r}_1 \cdot Z(\tau)$ and $r_2 = \tilde{r}_2 \cdot Z(\tau)$, where $\tilde{r}_1, \tilde{r}_2 \xleftarrow{R} \mathbb{Z}_p \setminus \{0\}$ (and in particular, independent of τ). As long as $Z(\tau) \neq 0$, these distributions are identical. By construction, the roots of Z are the elements in S_λ and $\tau \notin S_\lambda$. Thus, $Z(\tau) \neq 0$ and the claim holds. □

Lemma C.5. $|\Pr[\mathsf{Hyb}_1(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1]| \leq (|S_\lambda| + 2)/p$.

Proof. The only difference between Hyb_1 and Hyb_2 is the distribution of $\tilde{r}_1, \tilde{r}_2, \tau$. First, $\tilde{r}_1, \tilde{r}_2$ are uniform over $\mathbb{Z}_p \setminus \{0\}$ in Hyb_1 and uniform over $\mathbb{Z}_p$ in Hyb_2. The statistical distance between these two distributions is $1/p$. Similarly, τ is uniform over $\mathbb{Z}_p \setminus S_\lambda$ in Hyb_1 and uniform over $\mathbb{Z}_p$ in Hyb_2. The statistical distance between these two distributions is $|S_\lambda|/p$. Thus, the overall statistical distance between these two distributions is at most $(|S_\lambda| + 2)/p$. □

Lemma C.6. $|\Pr[\mathsf{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_3(\mathcal{A}) = 1]| \leq O(q^2|S_\lambda|^3/p)$.

Proof. We will use Theorem C.2. To do so, let $\mathcal{PQ} = \{P_iQ_j : P_i \in \mathcal{P}, Q_j \in \mathcal{Q}\}$ be a set of polynomials in the formal variables $\tilde{r}_1, \tilde{r}_2, \gamma, \tau$. To invoke Theorem C.2, it suffices to show that the polynomial

$$T(\tilde{r}_1, \tilde{r}_2, \gamma, \tau) = \tilde{r}_1\tilde{r}_2\gamma\tau^{N-1} \cdot \left(\sum_{k\in[N]} L_k(\tau)\right)^2 \tag{C.1}$$

is linearly independent with respect to the polynomials $\mathcal{PQ}$. Suppose otherwise. Then, there exists coefficients $\alpha_{i,j} \in \mathbb{Z}_p$ such that

$$T(\tilde{r}_1, \tilde{r}_2, \gamma, \tau) = \sum_{P_i\in\mathcal{P}} \sum_{Q_j\in\mathcal{Q}} \alpha_{i,j} \cdot P_i(\tilde{r}_1, \tilde{r}_2, \gamma, \tau) \cdot Q_j(\tilde{r}_1, \tilde{r}_2, \gamma, \tau). \tag{C.2}$$

Consider the value of the polynomials T and $L_1, \ldots, L_N$ when $\tau = s_i \in S_\lambda$ for some $i \in [N]$. By definition, we have

$$\begin{aligned} Z(\tau) = Z(s_i) &= \prod_{j\in[N]} (s_i - s_j) = 0 \\ L_i(\tau) = L_i(s_i) &= \prod_{j\neq i} (s_i - s_j) \neq 0 \\ \forall j \neq i : L_j(\tau) = L_j(s_i) &= \prod_{k\neq j} (s_i - s_k) = 0, \end{aligned} \tag{C.3}$$

where the second equality holds since $\mathbb{Z}_p$ is a field and $s_i - s_j \neq 0$ for all $j \neq i$. Suppose we consider Eq. (C.2) when we restrict the value of τ to $\tau \in S_\lambda$. In this case, if P_i or Q_j is a multiple of $Z(\tau)$, then the value of $P_i(\tilde{r}_1, \tilde{r}_2, \gamma, \tau)$ and $Q_j(\tilde{r}_1, \tilde{r}_2, \gamma, \tau)$ is guaranteed to be 0. Thus, Eq. (C.2) requires that for all $\tau \in S_\lambda$,

$$T(\tilde{r}_1, \tilde{r}_2, \gamma, \tau) = \sum_{i\in[N]} \sum_{j\in[0,N-2]} \left(\alpha^{(1)}_{i,j} \tilde{r}_1\tilde{r}_2\gamma^2\tau^j \cdot L_i(\tau) \cdot \sum_{k\in[N]} L_k(\tau)\right) + \sum_{j\in[0,N-2]} \alpha^{(2)}_j \tilde{r}_1\tilde{r}_2\gamma\tau^j \left(\sum_{k\in[N]} L_k(\tau)\right)^2, \tag{C.4}$$

where $\alpha^{(1)}_{i,j}$ and $\alpha^{(2)}_j$ are scalars. From Eq. (C.1), we have that the degree of γ in T is 1. For Eq. (C.4) to hold, this means $\alpha^{(1)}_{i,j} = 0$ for all $i \in [N], j \in [0, N-2]$. Thus, Eq. (C.4) implies that

$$\forall\tau \in S_\lambda : \tilde{r}_1\tilde{r}_2\gamma\tau^{N-1} \cdot \left(\sum_{k\in[N]} L_k(\tau)\right)^2 = \sum_{j\in[0,N-2]} \alpha^{(2)}_j \tilde{r}_1\tilde{r}_2\gamma\tau^j \cdot \left(\sum_{k\in[N]} L_k(\tau)\right)^2. \tag{C.5}$$

From Eq. (C.3), we have that for all $\tau = s_i \in S_\lambda$, $\sum_{k\in[N]} L_k(\tau) = L_i(s_i) \neq 0$. Thus Eq. (C.5) holds only if

$$\forall\tau \in S_\lambda : \tau^{N-1} = \sum_{j\in[0,N-2]} \alpha^{(2)}_j \tau^j,$$

or equivalently, if the polynomial $\tau^{N-1} - \sum_{j\in[0,N-2]} \alpha^{(2)}_j \tau^j = 0$ for all $\tau \in S_\lambda$. This is a non-zero polynomial of degree $N-1$, which has at most $N-1$ roots. However, $|S_\lambda| = N$, which is a contradiction. We conclude that Eq. (C.4) cannot hold for all $\tau \in S_\lambda$, which means that we cannot write the target polynomial T as a linear combination of the polynomials in $\mathcal{PQ}$. Thus, T is linearly independent with respect to $\mathcal{PQ}$.

To complete the proof, we observe that $|\mathcal{P}| = O(N)$ and $|\mathcal{Q}| = O(N)$. The total degree of polynomials in $\mathcal{PQ}$ is also $O(N)$ since the polynomials $Z(X), L_1(X), \ldots, L_N(X)$ all have degree at most N. Similarly, the degree of T is also $O(N)$. Thus, by Theorem C.2, if $\mathcal{A}$ makes at most q queries, then

$$|\Pr[\mathrm{Hyb}_2(\mathcal{A}) = 1] - \Pr[\mathrm{Hyb}_3(\mathcal{A}) = 1]| \leq \frac{O(q^2N^3)}{p} = \frac{O(q^2|S_\lambda|^3)}{p}.$$ □

Lemma C.7. $|\Pr[\mathrm{Hyb}_3(\mathcal{A}) = 1] - \Pr[\mathrm{Hyb}_4(\mathcal{A}) = 1]| \leq (N+2)/p$.

Proof. Follows by the same argument as in the proof of Lemma C.5. □

Lemma C.8. $\Pr[\mathsf{Hyb}_4(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_5(\mathcal{A}) = 1]$.

Proof. Follows by the same argument as in the proof of Lemma C.4. □

Theorem C.3 now follows by combining Lemmas C.4 to C.8 and a standard hybrid argument. □