Reducing the CRS Size in Registered ABE Systems

Rachit Garg[1], George Lu[1], Brent Waters[1,2], and David J. Wu[1]

[1]UT Austin
{rachg96, gclu, bwaters, dwu4}@cs.utexas.edu
[2]NTT Research

Abstract

Attribute-based encryption (ABE) is a generalization of public-key encryption that enables fine-grained access control to encrypted data. In (ciphertext-policy) ABE, a central *trusted* authority issues decryption keys for attributes x to users. In turn, ciphertexts are associated with a decryption policy $\mathcal{P}$. Decryption succeeds and recovers the encrypted message whenever $\mathcal{P}(x) = 1$. Recently, Hohenberger, Lu, Waters, and Wu (Eurocrypt 2023) introduced the notion of *registered ABE*, which is an ABE scheme *without* a trusted central authority. Instead, users generate their own public/secret keys (just like in public-key encryption) and then register their keys (and attributes) with a key curator. The key curator is a transparent and untrusted entity.

Currently, the best pairing-based registered ABE schemes support monotone Boolean formulas and an a priori bounded number of users L. A major limitation of existing schemes is that they require a (structured) common reference string (CRS) of size $L^2 \cdot |\mathcal{U}|$ where $|\mathcal{U}|$ is the size of the attribute universe. In other words, the size of the CRS scales *quadratically* with the number of users and multiplicatively with the size of the attribute universe. The large CRS makes these schemes expensive in practice and limited to a small number of users and a small universe of attributes.

In this work, we give two ways to reduce the CRS size in pairing-based registered ABE schemes. First, we introduce a combinatoric technique based on *progression-free sets* that enables registered ABE for the same class of policies but with a CRS whose size is sub-quadratic in the number of users. Asymptotically, we obtain a scheme where the CRS size is *nearly linear* in the number of users L (i.e., $L^{1+o(1)}$). If we take a more concrete-efficiency-oriented focus, we can instantiate our framework to obtain a construction with a CRS of size $L^{\log_2 3} \approx L^{1.6}$. For instance, in a scheme for 100,000 users, our approach reduces the CRS by a factor of over $115\times$ compared to previous approaches (and without incurring any overhead in encryption/decryption time). Our second approach for reducing the CRS size is to rely on a partitioning-based argument when arguing security of the registered ABE scheme. Previous approaches took a dual-system approach. Using a partitioning-based argument yields a registered ABE scheme where the size of the CRS is *independent* of the size of the attribute universe. The cost is the resulting scheme satisfies a weaker notion of static security. Our techniques for reducing the CRS size can be combined, and taken together, we obtain a pairing-based registered ABE scheme that supports monotone Boolean formulas with a CRS size of $L^{1+o(1)}$. Notably, this is the first pairing-based registered ABE scheme that does not require imposing a bound on the size of the attribute universe during setup time.

As an additional application, we also show how to apply our techniques based on progression-free sets to the batch argument (BARG) for NP scheme of Waters and Wu (Crypto 2022) to obtain a scheme with a nearly-linear CRS *without* needing to rely on non-black-box bootstrapping techniques.

1 Introduction

Attribute-based encryption (ABE) [SW05, GPSW06] is a cryptographic primitive that enables fine-grained access control to encrypted data. Specifically, in (ciphertext-policy) ABE, secret keys are associated with a set of attributes S and ciphertexts are associated with a decryption policy $\mathcal{P}$ and a message m. Decryption recovers the message m

if the attributes S satisfy the decryption policy $\mathcal{P}$ (i.e., if $\mathcal{P}(S) = 1$). Conversely, a user who does not have a key for a satisfying collection of attributes should not learn anything about the message.

While ABE is a versatile generalization of public-key encryption, it comes with a significant drawback of introducing a central authority who is responsible for generating keys for individual users. To issue keys to users, a central authority must hold on to a *long-term* master secret key for the lifetime of the system. If an attacker compromises the central authority and exfiltrates its secret key, then the attacker is able to decrypt every ciphertext in the system, both in the past and in the future. This is in stark contrast to traditional public-key encryption where users are individually responsible for generating and safeguarding their own secret keys. In exchange for expressivity, ABE has introduced a central point of failure, and this vulnerability to key-exfiltration attacks introduces significant hurdles for deploying ABE in practice.

The registration model of cryptography. A recent line of works has introduced a new registration model of cryptography whose goal is to replace the trusted key-issuer in identity-based encryption (IBE) [GHMR18, GHM+19, GV20, CES21, GKMR22, DKL+23, FKdP23], ABE [HLWW23, FWW23, FFM+23, ZZGQ23], broadcast encryption [BZ14, FWW23, KMW23, GLWW23], and functional encryption (FE) [FFM+23, DP23, DPY23, BLM+24] with a *transparent* and untrusted *key curator* whose sole job is to *aggregate* users' public keys. Specifically, in the setting of registered ABE, users independently generate their own public/secret key-pairs, exactly as in vanilla public-key encryption. They then *register* their public keys with a key curator along with the set of attributes they possess. Like the key-issuer in ABE or the certificate authority in a classic public-key infrastructure, the key curator is responsible for validating the attributes the user claims. The key curator then takes the public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ from the different users along with their respective attribute sets $S_1, \ldots, S_L$, and aggregates them together into a single *short* master public key mpk. Critically, the size of the aggregated mpk must be polylogarithmic in the number of users L. Like many traditional ABE schemes [GPSW06, LOS+10, Att14, Wee14, CGW15], we do allow the length of the master public key to scale with the size of the attribute universe. The aggregated master public key mpk now functions as the public key for a standard ABE scheme. Namely, a ciphertext encrypted with respect to a policy $\mathcal{P}$ can be decrypted by any registered user whose set of attributes S_i satisfy the policy.

As users register in a registered IBE or ABE scheme, the master public key is continually updated. Thus, registered users must periodically return to the key curator to request helper decryption keys that they will use during decryption. The number of times each user needs to update their key should be small, and we specifically require it to be polylogarithmic in the number of users L; in fact, under mild assumptions, $\Omega(\log L/\log\log L)$ updates are needed [MQR22]. Finally, the key curator in a registered IBE or ABE scheme is *transparent.* It is a *deterministic* algorithm and maintains no long-term secrets. Even if the key curator is compromised, the secret keys of existing registered users remain secret and are not compromised. An adversary that corrupts a key curator cannot decrypt any past ciphertext. Of course, such an adversary would be able to register a key that decrypts all future ciphertexts, but such behavior is publicly detectable by auditing the state of the key curator [GV20].

Constructions of registered ABE. Hohenberger et al. [HLWW23] introduced the notion of registered ABE and gave a construction from assumptions over composite-order pairing groups. Their scheme supports all monotone Boolean formula policies and an a priori bounded number of users L. Moreover, the scheme relies on a structured common reference string (CRS) whose size scales *quadratically* with the number of users L (and linearly with the size of the attribute universe). Subsequently, a number of constructions of registered ABE have been introduced [DP23, DPY23, FWW23, ZZGQ23, FFM+23] that expand on both the functionality and the security of the original scheme. These constructions generally fall into two categories: ones which require non-black-box use of cryptography (i.e., obfuscation-based or witness-encryption-based approaches) [FFM+23, DP23, DPY23, FWW23], or pairing-based constructions [FFM+23, ZZGQ23] that only require black box use of cryptography. The latter set of constructions enable more concretely efficient realizations, but thus far, all of them inherit the quadratic-size CRS from the original [HLWW23] construction. The size of the CRS in [FFM+23] does not depend on the size of the attribute universe, but the scheme supports a different and incomparable class of policies (inner products). The size of the CRS in the [HLWW23, ZZGQ23] constructions for Boolean formulas (and extensions thereof) also scale linearly with the size of the attribute universe. Our goal in this work is to develop new techniques to reduce the size of the CRS for registered ABE schemes.

1.1 Our Contributions

In this work we introduce two techniques to reduce the CRS size in registered ABE. The first is a combinatoric approach to reduce the CRS size from quadratic in the number of users L to nearly linear (specifically, $L^{1+o(1)}$). The second is a new partitioning-based proof strategy that enables a construction where the CRS size is *independent* of the number of attributes. The two techniques are complementary and can be used simultaneously to obtain a construction with a CRS whose size is nearly linear in the number of users and independent of the number of attributes. Previous pairing-based registered ABE schemes [HLWW23, FFM+23, ZZGQ23] had a CRS whose size was quadratic in the number of users, and for the schemes that supported general Boolean formulas, also linear in the number of attributes. We now summarize our main contributions and provide a comparison in Table 1.

Sub-quadratic size CRS using progression-free sets. First, we introduce a new combinatoric approach based on *progression-free sets* [ET36, Beh46, SS46, Elk10] to construct a registered ABE scheme whose CRS size is *nearly linear* in the number of users. As we discuss in Section 1.2, existing group-based constructions of registered ABE [HLWW23, FFM+23, ZZGQ23] aggregate user public keys by multiplying them together. Thus, when a user encrypts to the master public key, they are technically encrypting to the *product* of *every* users' key. To enable decryption, the CRS contains "cross terms" that can be used to remove the interactions across different users' public keys. The cross terms allow a user to take a ciphertext encrypted to the master public key and cancel out the components from the other users, leaving only a ciphertext encrypted to her own secret key, which she can then decrypt normally.

The CRS in recent pairing-based constructions of registered ABE include cross terms for *every* pair of users in the system. This leads to a quadratic-size CRS. The starting point in our work is to observe that we do *not* necessarily require a *different* cross term for each distinct pair of users. It can be the case that different pairs of users use a common cross term during decryption. In this work, we take a combinatoric approach and "embed" a progression-free set into the CRS of the registered ABE scheme. On the one hand, the progression-free set allows us to simultaneously reduce the number of cross terms in the CRS, and by extension, the size of the CRS. On the other hand, the "progression-free" property ensures that the components in the CRS do not allow unauthorized users to decrypt ciphertexts. Our security reduction will critically rely on the progression-free property; we provide more details in Section 1.2. The idea of using progression-free sets to improve the efficiency of cryptographic constructions was also previously used in a work of Lipmaa [Lip12] who focused on succinct non-interactive zero-knowledge arguments.

Using the state-of-the-art progression-free set constructions [Beh46, SS46, Elk10], we obtain a registered ABE scheme (for monotone Boolean formulas) where the size of the structured CRS for L users is *nearly linear*: $L^{1+o(1)}$. If we take a more concrete-efficiency-oriented view, we can use a lightweight progression-free set construction of Erdös and Turán [ET36] to obtain a registered ABE scheme with a CRS size of size $L^{\log_2 3} \approx L^{1.6}$. As we illustrate in Section 6, for a scheme that supports 100,000 users, the use of progression-free sets yields a 115× reduction in the size of the CRS compared to a construction with a quadratic-size CRS (from 447 GB to 3.8 GB).[1] We also note that the use of progression-free sets does *not* incur any overhead in the size of the master public key, the helper decryption keys, or the running time of the encryption/decryption algorithms. In Sections 4 and 5, we show how to combine the techniques from [HLWW23] with progression-free sets to obtain registered ABE schemes with a nearly-linear-size CRS in both prime-order groups (satisfying static security) and composite-order groups (satisfying adaptive security).

A partitioning proof strategy to support an arbitrary number of attributes. In this work, we also show how to use a partitioning-based strategy (similar to [BB04, Wat05, Wat11]) to argue the security of our constructions (in contrast to the dual-system approaches taken in previous works [HLWW23, ZZGQ23]). The advantage of using a partitioning proof strategy is it avoids the need to fix the universe of attributes at the time the CRS is generated. Instead, only the number of users needs to be declared in advance. Correspondingly, the size of the CRS no longer grows with the number of attributes. Previous pairing-based registered ABE schemes for formulas [HLWW23, ZZGQ23] relied on a dual-system methodology for the security analysis, which in turn required the universe of attributes to be fixed *a priori*. In these constructions, the size of the CRS also scaled linearly with the size of the attribute universe. The downside of a partitioning proof is that the scheme achieves a weaker notion of *static* security where the adversary is not allowed to make

[1]For fairness, we compare against a *prime-order* analog of the [HLWW23] construction (Appendix B), so the improvement in CRS size is purely from the use of progression-free sets. The actual level of improvement over the *composite-order* construction of [HLWW23] is even larger.

	$\lvert\mathsf{crs}\rvert$	$\lvert\mathsf{st}\rvert$	T_{reg}	$\lvert\mathsf{mpk}\rvert$	$\lvert\mathsf{hsk}\rvert$	$\lvert\mathsf{ct}\rvert$	**Policy**	**Assumption**	**Security**
[HLWW23]	$\lvert\mathcal{U}\rvert\, L^2$	$\lvert\mathcal{U}\rvert\, L^2$	$\lvert\mathcal{U}\rvert\, L$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	Formulas†	Composite-order	Adaptive
[FWW23]	$\lvert\mathcal{P}\rvert^{\delta}$	L	$\lvert\mathcal{P}\rvert^{\delta}$	$\lvert\mathcal{P}\rvert^{\delta}$	$\lvert\mathcal{P}\rvert^{\delta}$	$\lvert\mathcal{P}\rvert^{\delta}$	Circuits	Witness encryption	Static
[HLWW23]	1	L	1	1	1	$\lvert\mathcal{P}\rvert^{\delta}$	Circuits	$i\mathcal{O}$	Adaptive
[ZZGQ23]	$\lvert\mathcal{U}\rvert L^2$	$\lvert\mathcal{U}\rvert L^2$	$\lvert\mathcal{U}\rvert L$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	ABP	Prime-Order	Adaptive
Construction 4.3	$L^{1+o(1)}$	$\lvert\mathcal{U}\rvert\, L$	$\lvert\mathcal{U}\rvert\, L$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	Formulas	Prime-order	Static
Construction 5.5	$\lvert\mathcal{U}\rvert\, L^{1+o(1)}$	$\lvert\mathcal{U}\rvert\, L$	$\lvert\mathcal{U}\rvert\, L$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	Formulas†	Composite-order	Adaptive
Construction B.3	L^2	$\lvert\mathcal{U}\rvert\, L$	$\lvert\mathcal{U}\rvert\, L$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{U}\rvert$	$\lvert\mathcal{P}\rvert$	Formulas	Prime-order	Static

† Construction 5.5 and [HLWW23] require an a-priori bound on the number of times an attribute is used in the formula (due to only supporting single-use LSSS), while the other constructions allow unbounded reuse of attributes in the policy.

Table 1: Comparison with previous registered ABE schemes. Here, crs denotes the size of the common reference string, st denotes the auxiliary information maintained by the key curator (*excluding* the CRS), T_{reg} denotes the registration time, mpk denotes the size of the master public key, hsk denotes the size of the helper decryption key, and ct denotes the size of the ciphertext. We consider a system with L users, an attribute universe $\mathcal{U}$, and a policy $\mathcal{P}$. The schemes that support formulas are restricted to *monotone* formulas; we write "ABP" to denote arithmetic branching programs. We write "composite-order" to refer to schemes based on composite-order pairing groups and "prime-order" to refer to ones based on prime-order pairing groups. We write $\delta > 1$ to denote some constant (corresponding to the overhead in the underlying obfuscation or witness encryption scheme). In our asymptotic statements, we suppress polynomials in the security parameter λ and all polylogarithmic terms. We say a scheme is statically secure if the adversary is not allowed to make any corruption queries in the security game (Definition 3.7) and that it is *adaptively* secure otherwise (Definition 3.6).

any corruption queries.[2] Previous approaches (based on the dual-system methodology [Wat09, LW10]) achieved adaptive security. While static security is a weaker security notion, the work of [FWW23] showed how to transform a registered ABE scheme that does not allow corruption queries into one that does in the random oracle model. Our partitioning proof strategy is compatible with the use of progression-free sets and in Section 4, we describe a construction with both techniques over prime-order pairing groups. (For comparison purposes, we also describe a variant of [HLWW23] with a quadratic-size CRS that supports an arbitrary polynomial-size attribute universe in Appendix B). Our construction is the first pairing-based registered ABE scheme for Boolean formulas that does not require an *a priori* bound on the number of attributes at setup time. We summarize the main constructions we introduce in this work in Table 1.

Application to batch arguments. Our approach of using progression-free sets to reduce the CRS size can also be applied to the pairing-based batch arguments of Waters and Wu [WW22]. Like the registered ABE scheme of [HLWW23], the basic version of Waters-Wu batch argument has a CRS whose size is quadratic in the number of instances. The quadratic overhead there is also due to the presence of "cross terms." Using our combinatoric techniques, we obtain a version of [WW22] where the CRS has size $O(N^{1+o(1)})$ and N is the number of instances. Notably, this gives a BARG from pairings with a *sub-quadratic* CRS that does not rely on non-black-box use of cryptography. While there are approaches to generically reduce the size of the CRS in batch arguments [KPY19, WW22, KLVW23], all of these rely heavily on non-black-box use of cryptography. Our approach is purely algebraic and incurs minimal overhead over the original [WW22] construction.

Additional applications. A number of recent works have focused on batching cryptographic primitives using cross-term cancellation techniques. This includes (sub)-vector commitments [CF13, LM19], batch arguments [WW22], and registered ABE [HLWW23, FFM+23, ZZGQ23]. In each of these settings, the batching capability is enabled through a large CRS whose size scales quadratically with the number of users. Our work provides a direct path to reducing the size of the CRS in such constructions, and we expect that our techniques can be used to improve the asymptotic and concrete efficiency of existing and future constructions that rely on cross term cancellation.

[2]Note that the static adversary is allowed to register keys of its own. However, it is not allowed to request the secret key for an honest user (i.e., "corrupt" an honest user) in the static security game.

Recently, the work of [BLM+24] showed how to construct a registered quadratic functional encryption scheme with a *linear-size* CRS. However, this scheme comes with the caveat that the function key associated with each user need to be *determined* at setup time. This is sufficient for their application to registered traitor tracing, but is a departure from the standard notions of registered ABE and FE where the user (or key curator) can determine the attribute or function at *registration* time (i.e., when users join the system).

1.2 Technical Overview

We now provide an overview of our techniques for reducing the CRS size in registered ABE schemes. Throughout this work, we primarily focus on the simpler notion of *slotted* registered ABE introduced by Hohenberger et al. [HLWW23]. We consider ciphertext-policy ABE where each secret key is associated with a set of attributes S and each ciphertext is associated with a decryption policy; decryption is allowed whenever the attributes satisfy the policy. In a slotted registered ABE scheme, we additionally assume an *a priori* bound on the number of users or slots L and the size of the CRS can depend on L. Instead of users dynamically registering as in a standard registered ABE scheme, there is instead a single *aggregation* algorithm that takes as input a tuple of L public keys $\mathsf{pk}_1, \dots, \mathsf{pk}_L$ along with their associated attributes $S_1, \dots, S_L$, and outputs a succinct master public key mpk. The aggregation algorithm also outputs a set of L helper decryption keys, one for each user. Typically, the key curator would be responsible for running the aggregation algorithm. The master public key allows a user to encrypt to an arbitrary decryption policy, and all registered users whose set of attributes satisfy the decryption policy are able to decrypt. While the slotted primitive seems weaker than a registered ABE scheme, Hohenberger et al. showed that a slotted scheme can be generically compiled into a standard registered ABE scheme that supports dynamic registration (i.e., where users can register at any point in time) with only polylogarithmic overhead. Previous works [HLWW23, ZZGQ23] constructed slotted registered ABE schemes that support monotone Boolean formulas (and more) from pairings. However, these constructions required a CRS whose size scales quadratically with the number of users L and linearly with the size of the attribute universe. In this work, we develop techniques to achieve a CRS whose size scales nearly linearly with the number of users and independently of the size of the attribute universe.

Starting point: the [HLWW23] construction. The starting point of our work is the registered ABE scheme for monotone Boolean formulas from [HLWW23] based on composite-order pairing groups. We begin by describing a slimmed-down version of their scheme where the attribute universe contains a *single* attribute (denoted a) and the only supported policy is checking whether the user possesses the attribute or not. Moreover, we describe the scheme using a prime-order pairing group. The full [HLWW23] construction operates over a composite-order pairing group, but for correctness, we only need to consider the scheme in a single (prime-order) subgroup of the full group. The additional subgroups in their construction are used for re-randomization and implementing a dual-system proof strategy [Wat09, LW10]. In this work, we show that if we adopt a partitioning proof strategy, then a version of this slimmed-down prime-order construction is (statically)-secure.

Let L be the number of slots (or users) for the slotted registered ABE scheme. Let $(\mathbb{G}, \mathbb{G}_T, e, p, g)$ be a (symmetric) prime-order pairing group, where $\mathbb{G}, \mathbb{G}_T$ are groups of prime-order p, g is a generator of $\mathbb{G}$, and $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is an efficiently-computable non-degenerate bilinear map. The scheme now proceeds as follows:

- **CRS components:** The common reference string includes a description of the group along with the following set of components (grouped together by their semantic properties):
  - **Slot-specific components:** Each slot $i \in [L]$ is associated with three group elements:
  $$A_i = g^{t_i} \quad \text{and} \quad B_i = g^{\alpha} h^{t_i} \quad \text{and} \quad P_i = g^{\delta_i}.$$
  Here, α and h (randomly sampled) are common to all of the slots while $t_i, \delta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ are random slot-specific exponents. The slot-specific components (A_i, B_i, P_i) ensure that decryption is possible only in settings where the user possesses a secret key associated with some slot $i \in [L]$ (i.e., that the decrypter is a registered user).

- **Attribute-specific components:** Each slot also includes a group element $U_i = g^{u_i}$ associated with the (single) attribute in the scheme.[3] The attribute-specific components ensure that decryption is only possible if the user has a key for a slot i where the associated set of attributes satisfy the decryption policy.
- **Cross terms:** Each slot-attribute pair is also associated with a "cross term" $W_{j,i} = g^{t_j u_i}$ for all $i \neq j$. These will be used to construct helper decryption keys and facilitate decryption.
- **General components:** Finally, the CRS also contains a random group element $h \xleftarrow{\text{R}} \mathbb{G}$ and $Z = e(g,g)^{\alpha}$. These are used to encrypt the message and for linking together the slot-specific and attribute-specific components during decryption.

- **User key-generation:** To generate a key for a slot i, the user starts by sampling a secret exponent $r_i \xleftarrow{\text{R}} \mathbb{Z}_p$. The public key then consists of the group elements

$$T_i = g^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad \forall j \neq i : V_{j,i} = A_j^{r_i}.$$

Here T_i can be viewed as the user's main public key, $V_{j,i}$ are the "cross terms" (used to generate helper decryption keys at aggregation time), and Q_i is auxiliary information about the user's public key used for facilitating the security analysis.

- **Aggregation:** Given $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ and attributes $S_1, \ldots, S_L \subseteq \{\mathtt{a}\}$ for each user (recall that we are considering the simplified setting where the attribute universe consists of a single attribute $\mathtt{a}$), the aggregation algorithm computes the master public key as

$$\hat{T} = \prod_{i \in [L]} T_i \quad \text{and} \quad \hat{U} = \prod_{i \in [L] : \mathtt{a} \notin S_i} U_i$$

Here $\hat{T}$ functions as the attribute-independent public key and $\hat{U}$ the attribute-specific public key for the attribute $\mathtt{a}$. When there are multiple attributes, each attribute will have its own attribute-specific public key. Moreover, observe that the attribute-specific public key $\hat{U}$ for the attribute $\mathtt{a}$ is the product of the attribute-specific components U_i for the slots i that do *not* contain the attribute $\mathtt{a}$ (i.e., the indices $i \in [L]$ where $\mathtt{a} \notin S_i$). For each slot $i \in [L]$, the aggregation algorithm also computes the cross terms

$$\hat{V}_i = \prod_{j \neq i} V_{i,j} \quad \text{and} \quad \hat{W}_i = \prod_{j \neq i : \mathtt{a} \notin S_j} W_{i,j}. \tag{1.1}$$

The helper decryption for user i contains both $\hat{V}_i$ and $\hat{W}_i$.

- **Encryption:** To encrypt a message μ, the encrypter samples encryption randomness $s \xleftarrow{\text{R}} \mathbb{Z}_p$ and $h_1, h_2 \xleftarrow{\text{R}} \mathbb{G}$ such that $h_1 h_2 = h$. The ciphertext is then

$$\mathsf{ct} = (C_1, C_2, C_3, C_4) = \left(\mu \cdot Z^s \, , \, g^s \, , \, h_2^s \hat{U}^{-s} \, , \, h_1^s \hat{T}^{-s}\right).$$

We often refer to C_3 as the "attribute-specific" component and C_4 as the "slot-specific" component.

- **Decryption:** The decryption process affirms two main properties: (1) that the users' purported secret key is associated with some slot $i \in [L]$ and (2) that the attributes associated with slot i satisfy the challenge policy. At a high level, if the user possesses a secret key for a public key registered to slot i, then it is able to compute $e(g, h_1)^{st_i}$, where s is the encryption randomness and t_i is the slot-specific exponent for slot i. Moreover, if the attributes associated with slot i satisfy the challenge policy, then the user will be able to compute $e(g, h_2)^{st_i}$. Taken together, the user is able to compute

$$e(g, h_1)^{st_i} e(g, h_2)^{st_i} = e(g, h_1 h_2)^{st_i} = e(g, h)^{st_i}.$$

Now, pairing the ciphertext component $C_2 = g^s$ with the component $B_i = g^{\alpha} h^{t_i}$ from the CRS, the user obtains $e(C_2, B_i) = e(g,g)^{\alpha s} e(g,h)^{st_i}$, which can be used to recover the message. We provide more details below:

[3]When the attribute universe $\mathcal{U}$ contains more than one attribute, then there is a group element $U_{w,i} = g^{u_{i,w}}$ for each attribute $w \in \mathcal{U}$ and each slot index $i \in [L]$.

– **Slot check:** Suppose the user know the secret key r_i associated with slot i. Then, to compute $e(g,h_1)^{st_i}$, the user computes

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

Using r_i as well as its cross term $\hat{V}_i = \prod_{j \neq i} V_{i,j} = \prod_{j\neq i} g^{t_i r_j}$, the user can now compute

$$e(C_2, A_i)^{r_i} e(C_2, \hat{V}_i) = e(g^s, g^{t_i})^{r_i} \prod_{j \neq i} e(g^s, g^{t_i r_j}) = e(g,g)^{st_i r_i} \prod_{j\neq i} e(g,g)^{st_i r_j} = \prod_{j \in [L]} e(g,g)^{st_i r_j}$$

In particular, this means that

$$D_{\text{slot}} = e(A_i, C_4) e(C_2, A_i^{r_i} \hat{V}_i) = e(g,h_1)^{st_i}.$$

– **Policy check:** Next, if the attributes S_i associated with slot i contains attribute a (i.e., satisfies the decryption policy), then the user is able to compute $e(g,h_2)^{st_i}$. Here, the user relies on the cross terms $\hat{W}_i$:

$$e(A_i, C_3) = e(g^{t_i}, h_2^s \hat{U}^{-s}) = e(g,h_2)^{st_i} \prod_{j\in[L]:\mathsf{a}\notin S_j} e(g, U_j)^{-st_i} = e(g,h_2)^{st_i} \prod_{j \in [L]:\mathsf{a}\notin S_j} e(g,g)^{-st_i u_j}.$$

When $\mathsf{a} \in S_i$ and using the fact that the cross terms $\hat{W}_i = \prod_{j\neq i:\mathsf{a}\notin S_j} W_{i,j} = \prod_{j\neq i:\mathsf{a}\notin S_j} g^{t_i u_j}$, we have

$$e(A_i, C_3) = e(g,h_2)^{st_i} \prod_{j\neq i:\mathsf{a}\notin S_j} e(g,g)^{-st_i u_j} = e(g,h_2)^{st_i} \prod_{j \neq i:\mathsf{a}\notin S_j} e(g^s, g^{t_i u_j})^{-1} = e(g,h_2)^{st_i} e(C_2, \hat{W}_i)^{-1}. \quad (1.2)$$

This means that

$$D_{\text{attrib}} = e(A_i, C_3) e(C_2, \hat{W}_i) = e(g,h_2)^{st_i}.$$

Given both $D_{\text{slot}} = e(g,h_1)^{st_i}$ and $D_{\text{attrib}} = e(g,h_2)^{st_i}$ and using the fact that $h_1 h_2 = h$, the user can now recover the message by computing

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

Progression-free sets. The size of the CRS in the above variant of [HLWW23] is *quadratic* in the number of slots because it contains a "cross term" $W_{j,i} = g^{t_j u_i}$ for all $i \neq j$. In fact, when there are multiple attributes in the attribute universe, there needs to be a separate set of (quadratically-many) cross terms for each attribute. These cross terms are needed to generate the attribute-specific helper decryption key for each user (i.e., the $\hat{W}_i$ in Eq. (1.1)), which are in turn used in the "policy check" step during decryption (Eq. (1.2)). For correctness, it is essential that the CRS contains the term $W_{j,i}$ for all $j \neq i$ (to cancel out the interaction between an attribute registered to slot j and the decryption process with respect to slot i). For security, it is critical that the CRS does not contain the non-cross-term $W_{i,i} = g^{t_i u_i}$.[4]

Our first technique for reducing the CRS size is observing that the cross terms $W_{j,i} = g^{t_j u_i}$ do not have to be distinct for every pair $i \neq j$. For instance, suppose we choose the slot-specific exponents $t_1, \ldots, t_L$ and the attribute-specific exponents $u_1, \ldots, u_L$ from a distribution where $t_i u_j = t_{i'} u_{j'}$ for many pairs $(i,j) \neq (i',j')$. In this case, we only need to publish a *single* group element $W_{i,j}$ that can be shared across all pairs of indices (i',j') where $t_{i'} u_{j'} = t_i u_j$. Of course, we still require the invariant that we never give out a non-cross-term: namely, there does not exist an index $k \in [L]$ and a pair $i \neq j$ such that $t_k u_k = t_i u_j$.

Specifically, we choose the exponents t_i and u_i to be powers of a (random) value $a \xleftarrow{\text{R}} \mathbb{Z}_p$. We set $t_i := a^{d_i}$ for some integer $d_i \in \mathbb{N}$ and $u_i = b \cdot a^{d_i}$, where $b \xleftarrow{\text{R}} \mathbb{Z}_N$ is a randomizing term (shared across $u_1, \ldots, u_L$). Observe now that $u_i t_j = ba^{d_i} a^{d_j} = ba^{d_i + d_j}$. We need to choose a set of powers $\mathcal{D} = \{d_i : i \in [L]\} \subset \mathbb{N}$. so as to satisfy the following correctness and security properties:

[4]If the non-cross-term $W_{i,i}$ was given out, then a user who does not satisfy the ciphertext policy would also be able to decrypt.

- **Security:** For security, it should be the case that $d_k + d_k \neq d_i + d_j$ for any distinct $i, j, k \in [L]$. This ensures that even if the CRS contains $W_{j,i} = g^{ba^{d_i+d_j}}$ for all $i \neq j$, it never contains a non-cross-term of the form $g^{ba^{d_k+d_k}}$. If $\mathcal{D}$ is a set of non-negative integers with the property that for all distinct $d_i, d_j, d_k \in \mathcal{D}$, it holds that $d_i + d_j \neq 2d_k$, then the set $\mathcal{D}$ does not contain any arithmetic progressions of length 3. Such sets are often referred to as "progression-free sets."

- **Efficiency:** The second property we desire is that there should be many overlaps in the values of $d_i + d_j$ for distinct $i \neq j$. This is because the number of cross terms in the CRS scales with the size of the set $\{d_i + d_j : i, j \in [L], i \neq j\}$. Observe that we can always bound the size of this set (and thus, the number of cross terms) by $2 \cdot \max(\mathcal{D})$ where $\max(\mathcal{D})$ denotes the largest element in the set $\mathcal{D}$. Thus, it suffices to construct a progression-free set $\mathcal{D}$ with small values.

Progression-free sets are a well-studied combinatoric object [ET36, Beh46, Elk10] and state-of-the-art constructions show how to construct a progression-free set $\mathcal{D} \subset \mathbb{N}$ of size L where the maximum element has magnitude $L^{1+o(1)}$. Translating back to the setting of registered ABE, this means the number of cross terms we need to include in the CRS is also $L^{1+o(1)}$ (i.e., *nearly linear* in the number of users L). This is a substantial improvement over the quadratic-size CRS from [HLWW23].

On the flip side, choosing the slot-specific exponents t_i and the attribute-specific exponents u_i to be of the form a^{d_i} and ba^{d_i} for some fixed $a, b \xleftarrow{R} \mathbb{Z}_N$ will require us to make a more *complex* computational assumption when analyzing security. Specifically, we rely on "q-type" assumptions (c.f., [BBG05, BGW05]) where the size of the assumption grows with the number of users L, and moreover, where the terms given out in the assumption are parameterized by a progression-free set. While these are new and non-standard assumptions, it is straightforward to show that they hold in a generic (bilinear) group model [Sho97, BBG05, Boy08] and we refer to Appendix D for further discussion.

CRS size dependence on the number of attributes. Progression-free sets allow us to reduce the number of cross-terms in the CRS from quadratic to nearly linear in the number of users L. However, as described, we still need to give out a set of cross-terms for each attribute in the attribute universe. In the simplified scheme described above, there is only a single attribute, and as such, only one set of cross terms $W_{j,i} = g^{t_j u_i}$. However, when there are multiple attributes, the [HLWW23] scheme associates a different exponent $u_{w,i}$ for each attribute $w \in \mathcal{U}$ in the attribute universe $\mathcal{U}$ and each slot $i \in \mathbb{N}$, and there is a cross term $W_{w,j,i} = g^{t_j u_{w,i}}$ for each attribute w and each pair of distinct slots $i \neq j$. Having a different exponent $u_{w,i}$ for each attribute-slot index (w, i) is important for implementing the dual-system security proof (specifically, these exponents are used to switch the parameters from slot i from normal mode to semi-functional mode for the setting where the attributes associated with slot i do *not* satisfy the challenge policy). More broadly, the fact that we need a set of cross-terms for *each* attribute means that the size of the CRS scales with $|\mathcal{U}| \cdot L^2$ in the case of [HLWW23]. Using progression-free sets, we can reduce the size of each collection of cross terms from L^2 to $L^{1+o(1)}$, but if the CRS needs to contain $|\mathcal{U}|$ collections of cross terms, then the overall size of the CRS still scales with $|\mathcal{U}| \cdot L^{1+o(1)}$.

As noted above, the main reason [HLWW23] needed $|\mathcal{U}|$ sets of cross terms in the CRS is to facilitate a dual-system proof of adaptive security. Specifically, in the adaptive security game, the adversary is able to corrupt any slot i (i.e., request the secret key for an honest user registered to slot i). This is permitted as long as the adversary later specifies a set of attributes S which does not satisfy the challenge policy with slot i. Thus, when the reduction algorithm generates a key for a slot i, it needs to be prepared to give out the associated secret key for slot i. At the same time, the reduction algorithm should not be able to generate keys that would allow it to decrypt the challenge ciphertext itself (as otherwise, it would not need the adversary at all). In the dual-system argument, there are two types of slots: normal slots (whose parameters are generated according to the real scheme) and "semi-functional" slots whose parameters are generated in a special way. Similarly, there are normal ciphertexts and semi-functional ciphertexts. The proof maintains the invariant that keys registered to semi-functional slots can be used to decrypt normal ciphertexts and keys registered to normal slots can decrypt semi-functional ciphertexts. However, keys registered to semi-functional slots *cannot* decrypt semi-functional ciphertexts. The proof then consists of a sequence of hybrid experiments where the challenge ciphertext is first replaced by a semi-functional ciphertext; next, the proof carefully switches each slot from normal mode to semi-functional mode. At the very end of the proof, all of the slots as well as the challenge ciphertext are semi-functional and it is straightforward to argue that the adversary cannot break semantic security. In order to switch slot i from normal to semi-functional, the reduction algorithm critically relies on there being a different set of attribute exponents $u_{w,i}$ associated with each attribute $w \in \mathcal{U}$ and slot i. For this reason, the size of the CRS in the previous adaptively secure constructions scaled *multiplicatively* with the size of the attribute universe.

A partitioning-based proof strategy. To achieve a shorter CRS (whose size is *independent* of the size of the attribute universe), we take a different approach for arguing security. In particular, we consider a weaker "static" security model where the adversary must declare the set of corrupted slots $i \in [L]$ at the very beginning of the game. In this model, the reduction algorithm "knows" in advance which slots it needs to be able to generate the secret key for and which ones it does not. This enables us to use a "partitioning" strategy to argue security, where the indices of the corrupted slots are programmed into the CRS itself. The programming ensures that the adversary is able to generate secret keys for all of the corrupted slots (but not for the non-corrupted slots). While this is a weaker security notion that adaptive security, it still captures a meaningful security property, and moreover, the work of [FWW23] show how generically compile a registered ABE scheme that does not allow corruption queries into a scheme that supports adaptive corruptions in the random oracle model. The advantage of using a partitioning-based argument is we no longer require a different sets of attribute exponents for each slot, and in fact, all of the attribute can share the *same* set of attribute-slot components. This means the size of the CRS becomes *independent* of the size of the attribute universe. This has the added benefit that the size of the attribute universe no longer needs to be *fixed* at setup time. Note however that the size of the public key still grows with the number of attributes since we still need to associate a group element with each attribute which encodes which slots in the scheme are associated with the attribute.

Our partitioning-based approach can be applied with or without progression-free sets. For completeness, we describe both versions. In Section 4, we describe the scheme with both of our techniques for CRS size reduction. This yields a scheme with a CRS of size $L^{1+o(1)}$. Then, in Appendix B, we show an adaptation of [HLWW23] with a quadratic-size CRS and a partitioning-based security analysis. Since we use a partitioning-based proof strategy, we can rewrite the scheme over prime-order groups, and moreover, use a single set of attribute-slot exponents. The CRS size in both constructions is independent of the size of the attribute universe. We refer to Table 1 for a detailed comparison between our schemes as well as to those of previous works.

Proving adaptive security via a dual system approach. As noted above, using a partitioning-based proof strategy allows us to eliminate the dependence on the size of the attribute universe from the CRS. However, it comes at the cost of being able to prove adaptive security. In Section 5, we show how to integrate progression-free sets into the construction of [HLWW23] to obtain an *adaptively-secure* scheme where the CRS size scales nearly linearly with the number of slots. Adaptive security relies on a similar dual system argument as in [HLWW23], and consequently, we require an independent set of attribute exponents for each slot. As such, the size of the CRS in this construction scales multiplicatively with the size of the attribute universe.

Note that integrating progression-free sets into the construction of [HLWW23] requires making additional adjustments to the scheme. Notably, since we now sample exponents from a *correlated set* (as opposed to uniformly random), our modified scheme (Construction 5.5) requires an additional subgroup for re-randomization. Moreover, we also need to introduce additional re-randomization for the ciphertexts in order to facilitate the dual-system proof strategy; we refer to Section 5.3 for more technical details.

Incremental aggregation. Thus far, we have focused primarily on a slotted registered ABE scheme where the system is initialized with a fixed number of slots L. Instead of users joining dynamically, the slotted notion assumes that all L keys are provided together (to the aggregation algorithm). The work of [HLWW23] show how to generically transform any slotted registered ABE scheme into a normal registered ABE scheme that supports dynamic user registrations. In the normal setting, a key curator would be responsible for registering users, updating the scheme parameters, and issuing helper decryption keys. Thus, applied naïvely, the [HLWW23] transformation would require the key curator to store up to L public keys (before it is able to aggregate them together using the slotted registered ABE scheme). In our slotted registered ABE scheme (and as in previous constructions), the size of each user's public key is $O(L)$, which means the key curator would need to maintain an $O(L^2)$-size state to support dynamic registrations. Needing to maintain quadratic state would then remain a bottleneck in registered ABE schemes.

In this work, we show that if the underlying slotted registered ABE scheme supports *incremental* aggregation (where the aggregation algorithm operates in a streaming manner where the user public keys arrive sequentially and requires at most a linear-size state), then it is possible to adapt the [HLWW23] transformation to obtain a registered ABE scheme where the key curator only needs to maintain a linear-size state. All of the schemes we construct (as well as the original construction of [HLWW23]) support incremental aggregation, and thus, we are able to obtain a (standard) registered ABE scheme where the key curator only needs a *linear* amount of storage to support dynamic registrations. We describe this property in Definition 3.8 and describe our adaptation of the [HLWW23] transformation in Appendix C.

2 Preliminaries

Throughout this work, we write λ to denote the security parameter. For a positive integer $n \in \mathbb{N}$, we write $[n]$ to denote the set $\{1, \ldots, n\}$, and $[0, n]$ to denote the set $\{0, \ldots, n\}$. We use bold uppercase letters (e.g., $\mathbf{M}$) to denote matrices and bold lowercase letters (e.g., $\mathbf{v}$) to denote vectors. We use non-boldface letters to refer to their components (e.g., $\mathbf{v} = [v_1, \ldots, v_n]$). For a positive integer $N \in \mathbb{N}$, we write $\mathbb{Z}_N$ to denote the integers modulo N. For a set S, we write $\max(S)$ to denote the maximum element in the set, and $\min(S)$ to refer to the minimum element in the set. We write 2^S to denote the power set of S (i.e., the set containing all subsets $T \subseteq S$).

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

2.1 Access Structures and Linear Secret Sharing

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

Definition 2.1 (Access Structure [Bei96])**.** Let S be a set and let 2^S denote the power set of S (i.e., the set of all subsets of S). An access structure on S is a set $\mathbb{A} \subseteq 2^S \setminus \varnothing$ of non-empty subsets of S. We refer to the elements of $\mathbb{A}$ as the *authorized* sets and those not in $\mathbb{A}$ as the *unauthorized* sets. We say an access structure is *monotone* if for all sets $S_1, S_2 \in 2^S$, if $S_1 \in \mathbb{A}$ and $S_1 \subseteq S_2$, then $S_2 \in \mathbb{A}$.

Definition 2.2 (Linear Secret Sharing Scheme [Bei96])**.** Let $\mathcal{P}$ be a set of parties. A linear secret sharing scheme over a ring $\mathbb{Z}_N$ for $\mathcal{P}$ is a pair $(\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_N^{\ell \times n}$ is a "share-generating" matrix and $\rho \colon [\ell] \to \mathcal{P}$ is a "row-labeling" function. The pair $(\mathbf{M}, \rho)$ satisfy the following properties:

- **Share generation:** To share a value $s \in \mathbb{Z}_N$, sample $v_2, \ldots, v_n \xleftarrow{\text{R}} \mathbb{Z}_N$ and define the vector $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, $\mathbf{u} = \mathbf{M}\mathbf{v}$ is the vector of shares where $u_i \in \mathbb{Z}_N$ belongs to party $\rho(i)$ for each $i \in [\ell]$.
- **Share reconstruction:** Let $S \subseteq \mathcal{P}$ be a set of parties and let $I_S = \{i \in [\ell] : \rho(i) \in S\}$ be the row indices associated with S. Let $\mathbf{M}_S \in \mathbb{Z}_N^{|I_S| \times n}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ that are indexed by I_S. If S is an authorized set of parties, then there exists a vector $\boldsymbol{\omega}_S \in \mathbb{Z}_N^{|I_S|}$ such that $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$, where $\mathbf{e}_1^{\mathsf{T}} = [1, 0, \ldots, 0]$ denotes the first elementary basis vector. Conversely, if $S \subseteq$ is an *unauthorized* sets of parties, then $\mathbf{e}_1^{\mathsf{T}}$ is not in the row-span of $\mathbf{M}$ (i.e., there does not exist $\boldsymbol{\omega}_S \in \mathbb{Z}_N^{|S|}$ where $\boldsymbol{\omega}_S^{\mathsf{T}} \mathbf{M}_S = \mathbf{e}_1^{\mathsf{T}}$). Equivalently, when S is unauthorized, there exists a vector $\mathbf{v}^* \in \mathbb{Z}^n$ where the first component $v_1^* = 1$ such that $\mathbf{M}_S \mathbf{v}^* = \mathbf{0}$ (i.e., the vector $\mathbf{v}^*$ is orthogonal to the rows of $\mathbf{M}$ associated with the attributes in S).

We say that a policy $(\mathbf{M}, \rho)$ is *one-use* if the row-labeling function ρ is injective (i.e., each party appears at most once in the policy).

Remark 2.3 (Monotone Boolean Formulas)**.** Our pairing-based registered ABE constructions support monotone access policies that can be described by any linear secret sharing scheme. As a special case, this captures the class of monotone Boolean formulas. There are multiple ways to take a monotone Boolean formula and express it as a linear secret sharing scheme; we refer to [LW11, Appendix G] for one such approach.

2.2 Progression-Free Sets

The main combinatoric notion we use in this work is a progression-free set [ET36], which are sets that do not contain any arithmetic progressions of length 3). We provide the formal definition below:

Definition 2.4 (Progression-Free Set [ET36])**.** A set $\mathcal{D} \subset \mathbb{N}$ is progression-free if for all $i, j, k \in \mathcal{D}$ where $i \neq j$, it follows that $i + j \neq 2 \cdot k$.

Theorem 2.5 (Constructions of Progression-Free Sets [Beh46, Elk10])**.** *There exists an efficiently-computable family of progression-free sets* $\{\mathcal{D}_n\}_{n\in\mathbb{N}}$ *where* $|\mathcal{D}_n| = n$ *and* $\max(\mathcal{D}_n) = n^{1+o(1)}$.

Double-free sets. We will also consider "double-free" sets which are sets of positive integers where there does not exist i, j where $i = 2j$. We define this formally below and then show that any progression-free sets can be converted into a double-free and progression-free set.

Definition 2.6 (Double-Free Set)**.** We say a set $\mathcal{D} \subset \mathbb{N}$ is double-free if for all $i, j \in \mathcal{D}$, it follows that $i \neq 2 \cdot j$.

Corollary 2.7 (Progression-Free and Double-Free Sets)**.** *There exists an efficiently-computable family of progression and double-free sets* $\{\mathcal{D}_n\}$ *where* $\max(\mathcal{D}_n) \in n^{1+o(1)}$.

Proof. Let $\mathcal{D}'_{n+1}$ be a progression-free set of size $n+1$. Define $\mathcal{D}^* = \{d - \min(\mathcal{D}'_{n+1}) \mid d \in \mathcal{D}'_{n+1}\}$. Since we are simply subtracting a constant from the elements in $\mathcal{D}'_{n+1}$, any arithmetic sequence in $\mathcal{D}^*$ corresponds to an arithmetic sequence in $\mathcal{D}'_{n+1}$. This means that $\mathcal{D}^*$ is progression-free. Next, by construction, $0 \in \mathcal{D}^*$. Since $\mathcal{D}^*$ is progression-free, there does not exist any pair of indices $i, j \in \mathcal{D}^*$ such that $i + 0 = 2 \cdot j$. Now we can take $\mathcal{D}_n = \mathcal{D}^* \setminus \{0\}$ to be our progression and double-free set of size n. □

3 Registered Attribute-Based Encryption

We recall the preliminaries for a registered ABE scheme. The definition and discussion is copied verbatim from [HLWW23].

Definition 3.1 (Registered Attribute-Based Encryption [HLWW23])**.** Let λ be a security parameter. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be a universe of attributes and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies on $\mathcal{U}$ (i.e., every $P \in \mathcal{P}_\lambda$ is a function $P\colon 2^{\mathcal{U}_\lambda} \to \{0,1\}$). A registered attribute-based encryption scheme with attribute universe $\mathcal{U}$ and policy space $\mathcal{P}$ consists of a tuple of efficient algorithms $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}) \to \mathsf{crs}$: On input the security parameter λ and the size of the attribute universe $\mathcal{U}_\lambda$, the setup algorithm outputs a common reference string crs. We assume the crs (implicitly) contains the security parameter 1^λ and a description of the message space $\mathcal{M}_\lambda$ (where $|\mathcal{M}_\lambda| \geq 2$).
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{aux}) \to (\mathsf{pk}, \mathsf{sk})$: On input the common reference string crs, and a (possibly empty) state aux, the key-generation algorithm outputs a public key pk and a secret key sk.
- $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S_{\mathsf{pk}}) \to (\mathsf{mpk}, \mathsf{aux}')$: On input the common reference string crs, a (possibly empty) state aux, a public key pk, and a set of attributes $S_{\mathsf{pk}} \subseteq \mathcal{U}_\lambda$, the registration algorithm *deterministically* outputs the master public key mpk and an updated state aux′. We assume that mpk implicitly contains the security parameter 1^λ and a description of the message space $\mathcal{M}_\lambda$ (from crs).
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu) \to \mathsf{ct}$: 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 state aux, and a public key pk, the update algorithm *deterministically* outputs a helper decryption key hsk. We assume that hsk implicitly contains the security parameter 1^λ and a description of the message space $\mathcal{M}_\lambda$ (from crs).

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

Definition 3.2 (Correctness and Efficiency of Registered ABE)**.** Let $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ be a registered ABE scheme with attribute universe $\mathcal{U}$ and policy space $\mathcal{P}$. For a security parameter λ and an adversary $\mathcal{A}$, we define the following game between $\mathcal{A}$ and the challenger:

- **Setup phase:** The challenger starts by sampling the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|})$. It then initializes the auxiliary input $\mathsf{aux} = \perp$ and initial master public key $\mathsf{mpk}_0 = \perp$. It also initializes a counter $\mathsf{ctr}[\mathsf{reg}] = 0$ to keep track of the number of registration queries and another counter $\mathsf{ctr}[\mathsf{enc}] = 0$ to keep track of the number of encryption queries. Finally, it initializes $\mathsf{ctr}[\mathsf{reg}]^* = \infty$ as the index for the target key (which will also be updated during the course of the game). Finally, it gives crs to $\mathcal{A}$.

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

  - **Register non-target key query:** In a non-target-key registration query, the adversary $\mathcal{A}$ specifies a public key pk and a set of attributes $S \subseteq \mathcal{U}_\lambda$. The challenger first increments the counter $\mathsf{ctr}[\mathsf{reg}] = \mathsf{ctr}[\mathsf{reg}] + 1$ and then registers the key by computing $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S)$. The challenger updates its auxiliary data by setting $\mathsf{aux} = \mathsf{aux}'$ and replies to $\mathcal{A}$ with $(\mathsf{ctr}[\mathsf{reg}], \mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux})$.
  - **Register target key query:** In a target-key registration query, the adversary specifies a set of attributes $S^* \subseteq \mathcal{U}_\lambda$. If the adversary has previously made a target-key registration query, then the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr}[\mathsf{reg}] = \mathsf{ctr}[\mathsf{reg}] + 1$, samples $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and registers $(\mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*, S^*)$. It computes the helper decryption key $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$. The challenger updates its auxiliary data by setting $\mathsf{aux} = \mathsf{aux}'$, stores the index of the target identity $\mathsf{ctr}[\mathsf{reg}]^* = \mathsf{ctr}[\mathsf{reg}]$, and replies to $\mathcal{A}$ with $(\mathsf{ctr}[\mathsf{reg}], \mathsf{mpk}_{\mathsf{ctr}[\mathsf{reg}]}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{hsk}^*, \mathsf{sk}^*)$.
  - **Encryption query:** In an encryption query, the adversary submits the index $\mathsf{ctr}[\mathsf{reg}]^* \le i \le \mathsf{ctr}[\mathsf{reg}]$ of a public key,[5] a message $\mu_{\mathsf{ctr}[\mathsf{enc}]} \in \mathcal{M}_\lambda$ (where $\mathcal{M}_\lambda$ is the message space associated with crs), and a policy $P_{\mathsf{ctr}[\mathsf{enc}]} \in \mathcal{P}_\lambda$. If the adversary has not yet registered a target key, or if the target set of attributes S^* does not satisfy the policy $P_{\mathsf{ctr}[\mathsf{enc}]}$, the challenger replies with $\perp$. Otherwise, the challenger increments the counter $\mathsf{ctr}[\mathsf{enc}] = \mathsf{ctr}[\mathsf{enc}] + 1$ and computes $\mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}_i, P_{\mathsf{ctr}[\mathsf{enc}]}, \mu_{\mathsf{ctr}[\mathsf{enc}]})$. The challenger replies to $\mathcal{A}$ with $(\mathsf{ctr}[\mathsf{enc}], \mathsf{ct}_{\mathsf{ctr}[\mathsf{enc}]})$.
  - **Decryption query:** In a decryption query, the adversary submits a ciphertext index $1 \le j \le \mathsf{ctr}[\mathsf{enc}]$. The challenger computes $m'_j \leftarrow \mathsf{Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_j)$. If $m'_j = \mathsf{GetUpdate}$, then the challenger computes an updated helper decryption key $\mathsf{hsk}^* \leftarrow \mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$ and recomputes $m'_j \leftarrow \mathsf{Decrypt}(\mathsf{sk}^*, \mathsf{hsk}^*, \mathsf{ct}_j)$. If $m'_j \ne m_j$, the experiment halts with outputs $b = 1$.

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

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

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

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

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

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

Security. The security requirement for a registered ABE scheme asserts that a user with keys for attribute sets $S_1, \ldots, S_k$ should not be able to learn anything about the message associated with a ciphertext encrypted to a policy P where $P(S_i) = 0$ for all $i \in [k]$. We give the formal definition from [HLWW23]:

Definition 3.3 (Registered ABE Security [HLWW23])**.** Let $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ be a registered ABE scheme with attribute universe $\mathcal{U}$ and policy space $\mathcal{P}$. For a security parameter λ, an adversary $\mathcal{A}$, and a bit $b \in \{0, 1\}$, we define the following game between $\mathcal{A}$ and the challenger:

- **Setup phase:** The challenger samples the common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|})$. It then initializes the auxiliary input $\mathsf{aux} = \bot$, the initial master public key $\mathsf{mpk} = \bot$, a counter $\mathsf{ctr} = 0$ for the number of honest-key-registration queries the adversary has made, an empty set of keys $C = \varnothing$ (to keep track of corrupted public keys), and an empty dictionary mapping public keys to registered attribute sets $\mathsf{Dict} = \varnothing$. For notational convenience, if $\mathsf{pk} \notin \mathsf{Dict}$, then we define $\mathsf{Dict}[\mathsf{pk}] \coloneqq \varnothing$. to be the empty set. The challenger gives the crs to $\mathcal{A}$.
- **Query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
  - **Register corrupted key query:** In a corrupted-key-registration query, the adversary $\mathcal{A}$ specifies a public key pk and a set of attributes $S \subseteq \mathcal{U}_\lambda$. The challenger registers the key by computing $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S)$. The challenger updates its copy of the public key $\mathsf{mpk} = \mathsf{mpk}'$, its auxiliary data $\mathsf{aux} = \mathsf{aux}'$, adds pk to C, and updates $D[\mathsf{pk}] = D[\mathsf{pk}] \cup \{S\}$. It replies to $\mathcal{A}$ with $(\mathsf{mpk}', \mathsf{aux}')$.
  - **Register honest key query:** In an honest-key-registration query, the adversary specifies a set of attributes $S \subseteq \mathcal{U}_\lambda$. The challenger increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(1^\lambda)$, and registers $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}_{\mathsf{ctr}}, S)$. The challenger updates its public key $\mathsf{mpk} = \mathsf{mpk}'$, its auxiliary data $\mathsf{aux} = \mathsf{aux}'$, and $D[\mathsf{pk}_{\mathsf{ctr}}] = D[\mathsf{pk}_{\mathsf{ctr}}] \cup \{S\}$. It replies to $\mathcal{A}$ with $(\mathsf{ctr}, \mathsf{mpk}', \mathsf{aux}', \mathsf{pk}_{\mathsf{ctr}})$.
  - **Corrupt honest key query:** In a corrupt-honest-key query, the adversary specifies an index $1 \le i \le \mathsf{ctr}$. Let $(\mathsf{pk}_i, \mathsf{sk}_i)$ be the i^{th} public/secret key the challenger samples when responding to the i^{th} honest-key-registration query. The challenger adds pk_i to C and replies to $\mathcal{A}$ with sk_i.
- **Challenge phase:** The adversary $\mathcal{A}$ chooses two messages $\mu_0^*, \mu_1^* \in \mathcal{M}_\lambda$ (where $\mathcal{M}_\lambda$ is the message space associated with crs) and an access policy $P^* \in \mathcal{P}_\lambda$. The challenger replies with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P^*, \mu_b^*)$.
- **Output phase:** At the end of the game, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$.

Let $\mathcal{S} = \{S \in \mathsf{Dict}[\mathsf{pk}] : \mathsf{pk} \in C\}$ be the set of corrupted attributes. We say that an adversary $\mathcal{A}$ is admissible if the challenge policy P^* is not satisfied by any attribute set $S \in \mathcal{S}$. Note that it could be the case that P^* is satisfied by the attributes S from an honest key query (that was not subsequently corrupted). We say that a registered ABE scheme is secure if for all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, we have that $|\Pr[b' = 1 \mid b = 0] - \Pr[b' = 1 \mid b = 1]| = \mathsf{negl}(\lambda)$ in the above security game.

Definition 3.4 (Bounded Registered ABE)**.** As in [HLWW23], we say a registered ABE scheme Π_{RABE} is *bounded* if there is an *a priori* bound on the number of registered users in the system. In this setting, the Setup algorithm takes as input an additional bound parameter 1^L which specifies the maximum number of registered users. In the correctness and security definitions (Definitions 3.2 and 3.3), the adversary specifies the bound 1^L at the beginning of the experiment, and is then allowed to make a maximum of L registration queries.

Definition 3.5 (Static Security)**.** In this work, we also also consider the weaker notion of *static security* for a registered ABE scheme. In the static security game, the adversary must pre-commit to the number N of registration queries that it will make, and moreover, for each index $i \in [N]$, the adversary must also declare upfront whether its i^{th} registration query will be to register a "corrupted key" or an "honest key." Note that the adversary does *not* need to choose the corrupted keys themselves at the beginning of the game (which may not even be possible as the structure of a public key may depend on the common reference string). In addition, in this model, the adversary is not allowed to make "corrupt honest key" queries during the query phase. While static security is considerably weaker than the security definition in Definition 3.3, this relaxation will enable more *efficient* constructions.

3.1 Slotted Registered Attribute-Based Encryption

Similar to [HLWW23], we focus on constructing the simpler notion of a *slotted* registered ABE scheme. A slotted registered ABE scheme is simpler in the sense that it does *not* have to support dynamic registrations where users register their public keys (and attribute sets) one at a time. Instead, the scheme is initialized with a fixed number of slots L, and there is a single aggregation algorithm that takes *all* L public keys (together with their attribute sets) and outputs the aggregated public key. While the slotted version of the scheme may seem to provide a weaker functionality than a full registered ABE scheme (Definition 3.1), previous works have shown that the slotted version implies a scheme with dynamic registration via a powers-of-two compiler [GHM+19, GHM+19, HLWW23].

Definition 3.6 (Slotted Registration-Based Encryption [HLWW23])**.** Let λ be a security parameter. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be a universe of attributes and $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies on $\mathcal{U}$ (i.e., every $P \in \mathcal{P}_\lambda$ is a function $P\colon 2^{\mathcal{U}_\lambda} \to \{0,1\}$). A slotted registered ABE scheme with attribute universe $\mathcal{U}$ and policy space $\mathcal{P}$ is a tuple of efficient algorithms $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L) \to \mathsf{crs}$: On input the security parameter λ, the size of the attribute universe $\mathcal{U}_\lambda$, and the number of slots L, the setup algorithm outputs a common reference string crs. We assume that crs implicitly contains the security parameter 1^λ as well as a description of the message space $\mathcal{M}_\lambda$ associated with the scheme (where $|\mathcal{M}_\lambda| \geq 2$).
- $\mathsf{KeyGen}(\mathsf{crs}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$: On input the common reference string crs, a slot index $i \in [L]$, the key-generation algorithm outputs a public key pk_i and a secret key sk_i for slot i.
- $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) \to \{0,1\}$: On input the common reference string crs, a slot index $i \in [L]$, and a public key pk_i, the key-validation algorithm outputs a bit $b \in \{0,1\}$ indicating whether pk_i is valid or not. This algorithm is *deterministic.*
- $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \dots, (\mathsf{pk}_L, S_L)) \to (\mathsf{mpk}, \mathsf{hsk}_1, \dots, \mathsf{hsk}_L)$: On input the common reference string crs and a list of public keys and the associated attributes $(\mathsf{pk}_1, S_1), \dots, (\mathsf{pk}_L, S_L)$, the aggregate algorithm outputs the master public key mpk and a collection of helper decryption keys $\mathsf{hsk}_1, \dots, \mathsf{hsk}_L$. This algorithm is *deterministic.* We assume that the master public key mpk and the helper decryption keys hsk_i also contain (implicitly) the security parameter 1^λ as well as a description of the message space $\mathcal{M}_\lambda$ (from crs).
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu) \to \mathsf{ct}$: 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.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct}) \to m$: On input a decryption key sk, the helper decryption key hsk, and a ciphertext ct, the decryption algorithm outputs a message $\mu \in \mathcal{M}_\lambda \cup \{\bot\}$. This algorithm is *deterministic.*

Moreover, the above algorithms should satisfy the following properties:

- **Completeness:** For all parameters $\lambda \in \mathbb{N}$, $L \in \mathbb{N}$, and all indices $i \in [L]$,
$$\Pr\left[\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1 : \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L); (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)\right] = 1.$$
- **Correctness:** We say Π_{sRABE} is correct if for all security parameters $\lambda \in \mathbb{N}$, all slot lengths $L \in \mathbb{N}$, all indices $i \in [L]$, if we sample $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L)$, $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$, then for all collections of public keys $\{\mathsf{pk}_j\}_{j \neq i}$ (which may be correlated with pk_i) where $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$, all messages $\mu \in \mathcal{M}_\lambda$ (where

$\mathcal{M}_\lambda$ is the message space associated with crs), all sets of attributes $S_1, \ldots, S_L \subseteq \mathcal{U}_\lambda$, all policies $P \in \mathcal{P}_\lambda$ where S_i satisfies policy P, the following holds:

$$\Pr\left[\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{hsk}_i, \mathsf{ct}) = \mu : \begin{array}{c}(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L)) \\ \mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P, \mu)\end{array}\right] = 1,$$

where the probability is taken over the randomness in Setup, KeyGen, and Encrypt.

- **Compactness:** There exists a universal polynomial $\mathrm{poly}(\cdot,\cdot,\cdot)$ such that the length of the master public key and individual helper secret keys output by Aggregate are $\mathrm{poly}(\lambda, |\mathcal{U}_\lambda|, \log L)$.

- **Security:** Let $b \in \{0,1\}$ be a bit. For an adversary $\mathcal{A}$, define the following security game between $\mathcal{A}$ and a challenger:
  - **Setup phase:** The adversary $\mathcal{A}$ sends a slot count 1^L to the challenger. The challenger then samples $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L)$ and gives crs to $\mathcal{A}$. The challenger also initializes a counter $\mathsf{ctr} = 0$, a dictionary Dict, and a set of slot indices $C = \varnothing$.
  - **Pre-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
    - **Key-generation query:** In a key-generation query, the adversary specifies a slot index $i \in [L]$. The challenger responds by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$, sampling $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$ and replies with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$ to $\mathcal{A}$. The challenger adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary Dict.
    - **Corruption query:** In a corruption query, the adversary specifies an index $1 \le c \le \mathsf{ctr}$. In response, the challenger looks up the tuple $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{Dict}[c]$ and replies to $\mathcal{A}$ with sk'.
  - **Challenge phase:** For each slot $i \in [L]$, adversary $\mathcal{A}$ specifies a tuple $(c_i, S_i, \mathsf{pk}_i^*)$ where either $c_i \in \{1, \ldots, \mathsf{ctr}\}$ to reference a challenger-generated key or $c_i = \perp$ to reference a key outside this set. The adversary also specifies a challenge policy $P^* \in \mathcal{P}_\lambda$ and two messages $\mu_0^*, \mu_1^* \in \mathcal{M}_\lambda$ (where $\mathcal{M}_\lambda$ is the message space associated with crs). The challenger responds by first constructing pk_i as follows:
    - If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, then the challenger looks up the entry $\mathsf{Dict}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, then the challenger sets $\mathsf{pk}_i = \mathsf{pk}'$. Moreover, if the adversary previously issued a "corrupt identity" query on index c_i, then the challenger adds the slot index i to C. Otherwise, if $i \ne i'$, then the experiment halts.
    - If $c_i = \perp$, then the challenger checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, the experiment halts. If the key is valid, the challenger sets $\mathsf{pk}_i = \mathsf{pk}_i^*$ and adds the slot index i to C.

    The challenger computes $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$ and replies with the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P^*, \mu_b^*)$. Note that because Aggregate is *deterministic* and can be run by $\mathcal{A}$ itself, there is no need to additionally provide $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L)$ to $\mathcal{A}$. Similarly, there is no advantage to allowing the adversary to select the challenge policy and messages *after* seeing the aggregated key.
  - **Post-challenge query phase:** Adversary $\mathcal{A}$ can now issue the following queries:
    - **Corruption query:** In a corruption query, the adversary specifies an index $c \in \{1, \ldots, \mathsf{ctr}\}$. In response the challenger looks up the tuple $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{Dict}[c]$ and replies to $\mathcal{A}$ with sk'. Moreover, if the adversary registered a tuple of the form (c, S, pk^*) in the challenge phase for some choice of $S \subseteq \mathcal{U}_\lambda$ and pk^*, then the challenger adds the slot index $i' \in [L]$ to C.
  - **Output phase:** At the end of the experiment, algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say an adversary $\mathcal{A}$ is admissible if for all corrupted slot indices $i \in C$, the set S_i does not satisfy P^* (i.e., the attributes associated with a corrupted slot does not satisfy the challenge policy). Finally, we say that a slotted registration-based encryption scheme is secure if for all polynomials $L = L(\lambda)$ and all efficient and admissible adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

in the above security experiment.

Definition 3.7 (Static Security)**.** Similar to Definition 3.5, we say that a slotted registered ABE scheme satisfies *static* security if the adversary has to declare upfront (at the beginning of the security game before the challenger samples the CRS) the slot indices $i \in [L]$ where it will provide its own key during the challenge phase (i.e., the set of indices $i \in [L]$ where $c_i = \perp$). In addition, in the static security game, the adversary is not allowed to make any corruption queries in either query phase.

Incremental aggregation. Finally, we introduce the notion of incremental aggregation for a slotted registered ABE scheme. As mentioned above, the work of [HLWW23] describes a transformation from slotted registered ABE to registered ABE. However, a naïve implementation of this transformation would require the key curator to store a large amount of auxiliary data. Specifically, in the [HLWW23] transformation, as users join the system, their public keys and attribute sets are assigned to a collection of slotted registered ABE schemes (where the number of slots in each scheme are consecutive powers of two). Once a public key has been assigned to every slot of a particular scheme, the key curator runs the aggregation algorithm to derive an updated master public key for the slotted scheme. Since the key curator cannot run the aggregation algorithm for the slotted scheme until a key has been assigned to every slot, the key curator will need to cache a large number of public keys (up to L of them if there are L slots). In the [HLWW23] scheme (and our system), each user's public key in the slotted scheme also has size $\Omega(L)$. As a result, if the key curator has to store L public keys, this means the key curator needs to maintain a state of size $\Omega(L^2)$. However, in algebraic constructions of registered ABE such as [HLWW23] and our scheme, the underlying slotted registered ABE scheme supports "incremental aggregation." Namely, the aggregation algorithm for the slotted scheme essentially reads in a single public key and attribute set and uses them to "update" the master public key and helper decryption components. Once a public key and attribute set has been incorporated into the master public key and helper decryption components, the key curator no longer needs to keep the user public key around. In our setting, this will bring now the key curator set from $\Omega(L^2)$ to $O(L)$. We describe this transformation in Appendix C. We now define the incremental aggregation property we rely on formally:

Definition 3.8 (Incremental Aggregation)**.** Let $f(\cdot,\cdot)$ be a function. We say a slotted registered ABE scheme $\Pi_{\mathsf{sRABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ supports f-incremental aggregation if there exists an efficient algorithm AggregateUpdate with the following syntax:

- $\mathsf{AggregateUpdate}(\mathsf{crs}, \mathsf{st}, (\mathsf{pk}, S)) \to \mathsf{st}'$: On input the common reference string crs, state st, public key pk, and an associated set of attributes, the aggregate update algorithm outputs an updated state st$'$.

Then, we say that Π_{sRABE} supports f-incremental aggregation if $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$ can be implemented as follows:

1. Initialize $\mathsf{st}_0 = \perp$.
2. For each $i \in [L]$, compute $\mathsf{st}_i \leftarrow \mathsf{AggregateUpdate}(\mathsf{crs}, \mathsf{st}_{i-1}, (\mathsf{pk}_i, S_i))$.
3. Output $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{AggregateUpdate}(\mathsf{crs}, \mathsf{st}_L, \perp)$.

Moreover, we require that $\max_{i \in L} |\mathsf{st}_i| \le f(L, |\mathcal{U}_\lambda|) \cdot \mathsf{poly}(\lambda)$.

4 Statically-Secure Registered ABE via Progression-Free Sets

In this section, we show how to construct a statically-secure slotted registered ABE scheme over prime-order pairing groups. By instantiating the construction with state-of-the-art progression-free sets (Theorem 2.5), we obtain a scheme whose CRS size is $O(L^{1+o(1)})$. This improves upon the schemes with a *quadratic* CRS from prior work [HLWW23, FFM+23, ZZGQ23]. Similarly, by using a *partitioning* argument in the security proof, the CRS in our scheme does *not* grow with the size of the attribute universe. Previous registered ABE schemes for monotone Boolean formulas require a CRS whose size scales linearly with the size of the attribute universe.

4.1 Prime-Order Pairing Groups

Our construction of slotted registered ABE will rely on prime-order pairing groups. We recall the formal definition below:

Definition 4.1 (Prime-Order Bilinear Group)**.** A (symmetric) prime-order bilinear group generator is an efficient algorithm PrimeGroupGen that takes as input the security parameter λ and outputs a description $(\mathbb{G}, \mathbb{G}_T, p, g, e)$ of a bilinear group where $p = 2^{\Omega(\lambda)}$ is a prime, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order p, g is a generator of $\mathbb{G}$, and $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map (called the "pairing"). We require that the group operation in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the pairing operation be efficiently computable.

Set-consistent decisional bilinear Diffie-Hellman exponent assumption. The security of our construction relies on a new assumption on prime-order bilinear groups, which we call the set-consistent bilinear Diffie-Hellman exponent assumption. This is a variant of the bilinear Diffie-Hellman exponent (BDHE) assumption from [BBG05]. In the q-BDHE assumption from [BBG05], the adversary's goal is to distinguish $e(g,g)^{a^{q+1}s}$ from random given the group elements

$$(g, g^s, g^a, g^{a^2}, \ldots, g^{a^q}, g^{a^{q+2}}, \ldots, g^{a^{2q}}).$$

In the q-set-consistent bilinear Diffie-Hellman exponent assumption, the adversary's goal is to distinguish the element $e(g,g)^{a^q s}$ from random given g, g^s, and for each $i \in [q-1]$, either the element g^{a^i} *or* the element $g^{a^{q-i}s}$. Similar to the q-BDHE assumption, the adversary also gets additional group elements corresponding to the powers of a beyond q. Observe that if the adversary had both g^{a^i} *and* $g^{a^{q-i}s}$ for a particular index $i \in [q-1]$, the adversary can trivially distinguish by pairing these two elements together. However, given only one element from each pair, the adversary cannot trivially compute $e(g,g)^{a^q s}$. We give the formal statement of the assumption below, and in Appendix D (Theorem D.6), we show that this assumption holds unconditionally in the generic bilinear group model.

Assumption 4.2 (Set-Consistent Bilinear Diffie-Hellman Exponent)**.** Let PrimeGroupGen be a prime-order group generator. For a security parameter λ and a bit $b \in \{0,1\}$, we define the q-set-consistent bilinear Diffie-Hellman exponent game between an adversary $\mathcal{A}$ and a challenger:

- On input the security parameter 1^λ, the adversary $\mathcal{A}$ starts by outputting set $S \subseteq [q-1]$.
- The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \text{PrimeGroupGen}(1^\lambda)$ and exponents $a, s \xleftarrow{\text{R}} \mathbb{Z}_p$.
- The challenger computes $Y = g^s$ and for each $i \in [2q]$, let $X_i = g^{a^i}$ and $Z_i = g^{a^i s}$. Let $Q = e(g,g)^{a^q}$. The challenger also computes $T_0 = e(g,g)^{a^q s}$ and samples $T_1 \xleftarrow{\text{R}} \mathbb{G}_T$.
- The challenger gives the following challenge to $\mathcal{A}$:
$$\mathcal{G}\ ,\ g\ ,\ Y\ ,\ \{X_i\}_{i \in S \cup [q+1,2q]}\ ,\ \{Z_{q-i}\}_{i \in [q-1] \setminus S}\ ,\ \{Z_i\}_{[q+1,2q]}\ ,\ Q\ ,\ \boxed{T_b}.$$
- The adversary outputs a bit $b' \in \{0,1\}$ which is the output of the experiment.

We say the q-set-consistent bilinear Diffie-Hellman exponent assumption holds with respect to PrimeGroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

in the q-set-consistent bilinear Diffie-Hellman exponent game.

4.2 Slotted Registered ABE Construction

We now give the construction and analysis of our slotted registered ABE scheme from prime-order pairing groups.

Construction 4.3 (Slotted Attribute-Based Registration-Based Encryption)**.** Let $\mathsf{PrimeGroupGen}$ be a prime-order bilinear group generator. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ be a (polynomial-size) attribute space. 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 (Definition 2.2) over $\mathcal{U}$, where each policy $P \in \mathcal{P}_\lambda$ is defined over a maximum of $K = K(\lambda)$ attributes. We construct a slotted attribute-based registration-based encryption scheme $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with attribute space $\mathcal{U}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^L)$:[6] On input the security parameter λ, and the number of slots L, the setup algorithm starts by sampling $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$. The setup algorithm now constructs the following quantities:
  - Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set of size L (Corollary 2.7). In the following, we define $f(i,j) \coloneqq d_i + d_j$ and the set $\mathcal{E}$ of all distinct pairwise sums of elements in $\mathcal{D}$:
  $$\mathcal{E} = \{f(i,j) \mid i,j \in [L] : i \neq j\}.$$
  Let $d_{\max} = 3 \cdot \max(\mathcal{D})$.
  - Sample random exponents $a, b \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and set $\alpha = -a^{d_{\max}}$. Compute $h = \prod_{i \in [L]} g^{a^{d_{\max} - d_i}}$.
  - For each index $i \in [L]$, sample $\delta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, and let $t_i = a^{d_i}$. Then, define the following group elements:
  $$A_i = g^{t_i} \quad , \quad B_i = g^{\alpha} h^{t_i} \quad , \quad P_i = g^{\delta_i} \quad , \quad U_i = g^{b t_i}.$$
  Then, for each $z \in \mathcal{E}$, let $W_z = g^{b a^z}$.
  - Finally let $Z = e(g,g)^\alpha$. Output the common reference string
  $$\mathsf{crs} = \big(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}\,,\, \{W_z\}_{z \in \mathcal{E}}\big) \tag{4.1}$$

  The associated message space $\mathcal{M}_\lambda$ is defined to be $\mathcal{M}_\lambda \coloneqq \mathbb{G}_T$.

- $\mathsf{KeyGen}(\mathsf{crs}, i)$: On input the common reference string crs (with components given by Eq. (4.1)) and a slot index $i \in [L]$, the key-generation algorithm samples $r_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and computes
  $$T_i = g^{r_i} \quad , \quad Q_i = P_i^{r_i}.$$
  Then for each $j \neq i$, it computes the cross terms $V_{j,i} = A_j^{r_i}$. Finally, it outputs the public key pk_i and the secret key sk_i defined as follows:
  $$\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i}) \quad \text{and} \quad \mathsf{sk}_i = r_i.$$
  Note that this key-generation algorithm does *not* depend on the set of attributes.

- $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$: On input the common reference string crs (with components given by Eq. (4.1)), a slot index $i \in [L]$, and a purported public key $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$, the key-validation algorithm first affirms that each of the components in pk_i is a valid group element (i.e., an element in $\mathbb{G}$). If so, it then checks
  $$e(T_i, P_i) = e(g, Q_i)$$
  Next, for each $j \neq i$, the algorithm checks that
  $$e\big(g, V_{j,i}\big) = e(T_i, A_j)$$
  If all checks pass, it outputs 1; otherwise, it outputs 0.

[6]Since the setup algorithm does not depend on the size of the attribute universe, we omit the parameter $1^{\mathcal{U}_\lambda}$ to this algorithm.

- $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$: On input the common reference string crs (with components given by Eq. (4.1)), a collection of L public keys $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ together with their attribute sets $S_i \subseteq \mathcal{U}_\lambda$, the aggregation algorithm starts by computing the attribute-independent public key $\hat{T}$ and the attribute-independent slot key $\hat{V}_i$ for each $i \in [L]$:
$$\hat{T} = \prod_{j \in [L]} T_j \quad \text{and} \quad \hat{V}_i = \prod_{j \neq i} V_{i,j}.$$
Next, for each attribute $w \in \mathcal{U}_\lambda$, it computes the attribute-specific public key $\hat{U}_w$ and the attribute-specific slot key $\hat{W}_{i,w}$ for each $i \in [L]$:
$$\hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_j \quad \text{and} \quad \hat{W}_{i,w} = \prod_{j \neq i : w \notin S_j} W_{f(i,j)}.$$
Finally, it outputs the master public key mpk and the slot-specific helper decryption keys hsk_i where
$$\mathsf{mpk} = \left(\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}\right) \quad \text{and} \quad \mathsf{hsk}_i = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}_\lambda}\right).$$

- $\mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{M}, \rho), \mu)$: On input the master public key $\mathsf{mpk} = (\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda})$, a policy $(\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function, and a message $\mu \in \mathbb{G}_T$, the encryption algorithm starts by sampling a secret exponent $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $h_1, h_2 \xleftarrow{\mathrm{R}} \mathbb{G}$ such that $h = h_1 h_2$. Then, it constructs the ciphertext components as follows:
  - **Message-embedding components:** First, let $C_1 = \mu \cdot Z^s$ and $C_2 = g^s$.
  - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ for the linear secret sharing scheme and let $\mathbf{v} = [1, v_2, \ldots, v_n]^\mathsf{T}$. Then, for each $k \in [K]$, sample $s_k \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and set $C_{3,k} = h_2^{s \mathbf{m}_k^\mathsf{T} \mathbf{v}} \hat{U}_{\rho(k)}^{-s_k}$ and $C_{4,k} = g^{s_k}$, where $\mathbf{m}_k^\mathsf{T} \in \mathbb{Z}_p^n$ denotes the kth row of $\mathbf{M}$.
  - **Slot-specific component:** Set $C_5 = (h_1 \hat{T}^{-1})^s$.

  It then outputs the ciphertext
$$\mathsf{ct} = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\right).$$

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input the secret key $\mathsf{sk} = r$, the helper key $\mathsf{hsk} = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}_\lambda}\right)$, where $\mathsf{mpk} = (\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda})$, and the ciphertext $\mathsf{ct} = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\right)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function, the decryption algorithm proceeds as follows:
  - If the set of attributes S_i is not authorized by $(\mathbf{M}, \rho)$, then the decryption algorithm outputs $\perp$.
  - Otherwise, let $I = \{k \in [K] : \rho(k) \in S_i\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes $S_i \subseteq \mathcal{U}_\lambda$. Write the elements as $I = \{k_1, \ldots, k_{|I|}\}$.
  - Let $\mathbf{M}_{S_i}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I. Since S_i is authorized, let $\boldsymbol{\omega}_{S_i} \in \mathbb{Z}_p^{|I|}$ be a vector such that $\boldsymbol{\omega}_{S_i}^\mathsf{T} \mathbf{M}_{S_i} = \mathbf{e}_1^\mathsf{T}$.
  - Then, compute and output
$$\frac{C_1}{e(C_2, B_i)} \cdot \underbrace{e(C_5, A_i) \cdot e(C_2, A_i^r \hat{V}_i)}_{D_{\mathsf{slot}}} \cdot \underbrace{\prod_{1 \le j \le |I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)})\right)^{\omega_{S_i,j}}}_{D_{\mathsf{attrib}}}. \tag{4.2}$$

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

Theorem 4.4 (Completeness). *Construction 4.3 is complete.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$ and slot parameter $L \in \mathbb{N}$. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$. Then, we can write

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i\in[L]}\,,\, \{W_z\}_{z\in\mathcal{E}}\right).$$

Take any index $i \in [L]$ and let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. By construction of KeyGen, we can write $\mathsf{pk}_i = \left(T_i, Q_i, \{V_{j,i}\}_{j\neq i}\right)$, where

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

for some $r_i \in \mathbb{Z}_p$. We now consider each of the pairing checks in IsValid:

- $e(T_i, P_i) = e(g^{r_i}, P_i) = e(g, P_i^{r_i}) = e(g, Q_i)$.
- $e(g, V_{j,i}) = e(g, A_j^{r_i}) = e(g^{r_i}, A_j) = e(T_i, A_j)$.

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

Theorem 4.5 (Correctness)**.** *Construction 4.3 is correct.*

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

- Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$ where $\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i\in[L]}\,,\, \{W_z\}_{z\in\mathcal{E}}\right)$. Recall that the slot components can be written as $A_i = g^{t_i}$, $B_i = g^\alpha h^{t_i}$, and $P_i = g^{\delta_i}$. The attribute components can be written as $U_i = g_1^{bt_i}$ and $W_z = g_1^{ba^z}$ (where $t_i = a^{d_i}$).
- Let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. Then, we can write $\mathsf{sk}_i = r_i$ and $\mathsf{pk}_i = \left(T_i, Q_i, \{V_{j,i}\}_{j\neq i}\right)$ where
$$T_i = g^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad V_{j,i} = A_j^{r_i} = g^{t_j r_i}. \tag{4.3}$$
- Take any set of public keys $\{\mathsf{pk}_j\}_{j\neq i}$ where $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$. Since pk_j satisfies the IsValid predicate, we can write $\mathsf{pk}_j = \left(T_j, Q_j, \{V_{\ell,j}\}_{\ell\neq j}\right)$.
- For each $j \in [L]$, let $S_j \subseteq \mathcal{U}_\lambda$ be the attributes associated with pk_j.
- Let $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$. Then, the master public key mpk and the ith slot-specific helper decryption key hsk_i can be written as follows:
$$\mathsf{mpk} = \left(\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w\in\mathcal{U}_\lambda}\right) \quad \text{and} \quad \mathsf{hsk}_i = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w\in\mathcal{U}_\lambda}\right),$$
where $\hat{T} = \prod_{j\in[L]} T_j$, $\hat{V}_i = \prod_{j\neq i} V_{i,j}$, and
$$\hat{U}_w = \prod_{j\in[L]:w\notin S_j} U_j = \prod_{j\in[L]:w\notin S_j} g^{bt_j}$$
$$\hat{W}_{i,w} = \prod_{j\neq i:w\notin S_j} W_{f(i,j)} = \prod_{j\neq i:w\notin S_j} g^{a^{f(i,j)}b}$$
- Let $(\mathbf{M}, \rho)$ be the challenge policy where $\mathbf{M} \in \mathbb{Z}_p^{K\times n}$ and $\rho\colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function. Take any message $\mu \in \mathbb{G}_T$. The challenge ciphertext ct can be written as
$$\mathsf{ct} = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k\in[K]}, C_5\right)$$
where $C_1 = \mu \cdot Z^s$, $C_2 = g^s$, $C_{3,k} = h_2^{\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s_k}$, $C_{4,k} = g^{s_k}$, and $C_5 = h_1^s \hat{T}^{-s}$.

We now show that $\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{hsk}_i, \mathsf{ct})$ outputs μ. Let $I = \{k \in [K] : \rho(k) \in S_i\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes S_i. Write the elements of I as $I = \{k_1, \ldots, k_{|I|}\}$. Let $\mathbf{M}_{S_i}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I, and let $\boldsymbol{\omega}_{S_i} \in \mathbb{Z}_N^{|I|}$ be a vector such that $\boldsymbol{\omega}_{S_i}^{\mathsf{T}}\mathbf{M}_{S_i} = \mathbf{e}_1^{\mathsf{T}}$. We break up the decryption relation (Eq. (4.2)) into several pieces and analyze them individually:

- **Policy check:** First, consider $D_{\mathsf{attrib}} = \prod_{1 \le j \le |I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}) \right)^{\omega_{S_i,j}}$. By definition,

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

$$e\big(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}\big) = \prod_{\ell \ne i : \rho(k_j) \notin S_\ell} e\left(g^{s_{k_j}}, W_{f(i,\ell)} \right) = \prod_{\ell \ne i : \rho(k_j) \notin S_\ell} e(g,g)^{a^{f(i,\ell)} s_{k_j} b}$$

By construction, $\rho(k_j) \in S_i$ and by definition, $t_i t_\ell = a^{d_i + d_\ell} = a^{f(i,\ell)}$, so

$$\prod_{\ell \in [L] : \rho(k_j) \notin S_\ell} e(g,g)^{-s_{k_j} t_i t_\ell b} = \prod_{\ell \ne i : \rho(k_j) \notin S_\ell} e(g,g)^{-a^{f(i,\ell)} s_{k_j} b},$$

and so we can write

$$e(C_{3,k_j}, A_i) e\big(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}\big) = e(h_2, g)^{st_i \mathbf{m}_{k_j}^\mathsf{T} \mathbf{v}}.$$

Finally noting that $\mathbf{e}_1^\mathsf{T}\mathbf{v} = 1$, we have

$$\begin{aligned} D_{\mathsf{attrib}} &= \prod_{1 \le j \le |I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}) \right)^{\omega_{S_i,j}} \\ &= e(h_2, g)^{st_i \sum_{1 \le j \le |I|} \omega_{S_i,j} \mathbf{m}_{k_j}^\mathsf{T} \mathbf{v}} \\ &= e(h_2, g)^{st_i \boldsymbol{\omega}_{S_i}^\mathsf{T} \mathbf{M}_{S_i} \mathbf{v}} \\ &= e(h_2, g)^{st_i \mathbf{e}_1^\mathsf{T} \mathbf{v}} = e(h_2, g)^{st_i}. \end{aligned}$$

- **Slot check:** Next, consider the component $D_{\mathsf{slot}} = e(C_5, A_i) e(C_2, A_i^{r_i} \hat{V}_i)$. By definition

$$e(C_5, A_i) = e\big(h_1^s \hat{T}^{-s}, g^{t_i}\big) = e(h_1, g)^{st_i} \prod_{j \in [L]} e(T_j, g)^{-st_i} = e(h_1, g)^{st_i} \prod_{j \in [L]} e(T_j, A_i)^{-s}$$

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

Now, since we know for all $j \in [L]$, $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$, we have that for all $j \ne i$, $e(g, V_{i,j}) = e(T_j, A_i)$. Thus, using Eq. (4.3), we can now write

$$\begin{aligned} D_{\mathsf{slot}} = e(C_5, A_i) e(C_2, A_i^{r_i} \hat{V}_i) &= \left(e(h_1, g)^{st_i} e(T_i, A_i)^{-s} \prod_{j \ne i} e(T_j, A_i)^{-s} \right) \left(e(g,g)^{s r_i t_i} \prod_{j \ne i} e(g, V_{i,j})^s \right) \\ &= e(h_1, g)^{st_i} e(T_i, A_i)^{-s} e(g,g)^{s r_i t_i} \\ &= e(h_1, g)^{st_i} e(g^{r_i}, g^{t_i})^{-s} e(g,g)^{s r_i t_i} = e(h_1, g)^{st_i}. \quad \square \end{aligned}$$

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

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

Next, we can see that have

$$e(C_2, B_i) = e(g^s, g^\alpha h^{t_i}) = e(g,g)^{\alpha s} e(h, g)^{st_i}.$$

Thus, putting everything together, Eq. (4.2) becomes

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

Theorem 4.6 (Compactness)**.** *Construction 4.3 is compact.*

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

Theorem 4.7 (Incremental Aggregation)**.** *Construction 4.3 supports f-incremental aggregation for $f(L, |\mathcal{U}_\lambda|) = O(L \cdot |\mathcal{U}_\lambda|)$.*

Proof. We construct the AggregateUpdate algorithm as follows:

- $\mathsf{AggregateUpdate}(\mathsf{crs}, \mathsf{st}, (\mathsf{pk}, S))$: On input the common reference string

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}\,,\, \{W_z\}_{z \in \mathcal{E}}\right),$$

  a state st (which could be $\perp$), and a public key (pk, S) (or the special symbol $\perp$), the update algorithm proceeds as follows:

  1. If $\mathsf{st} = \perp$, then the update algorithm initializes $k = 0$ and $\hat{T}^{(k)} = 1$, $\hat{V}_i^{(k)} = 1$ for all $i \in [L]$, $\hat{U}_w^{(k)} = 1$ for all $w \in \mathcal{U}_\lambda$, and $\hat{W}_{i,w}^{(k)} = 1$ for all $i \in [L]$ and $w \in \mathcal{U}_\lambda$. Otherwise, the update algorithm parses

$$\mathsf{st} = \left(k\,,\, \hat{T}^{(k)}\,,\, \{\hat{V}_i^{(k)}\}_{i \in [L]}\,,\, \{\hat{U}_w^{(k)}\}_{w \in \mathcal{U}_\lambda}\,,\, \{\hat{W}_{i,w}^{(k)}\}_{i \in [L], w \in \mathcal{U}_\lambda}\right)$$

  2. If $(\mathsf{pk}, S) = \perp$, then the algorithm outputs

$$\mathsf{mpk} = \left(\mathcal{G}, g, h, Z, \hat{T}^{(k)}, \{\hat{U}_w^{(k)}\}_{w \in \mathcal{U}_\lambda}\right) \quad , \quad \forall i \in [L] : \mathsf{hsk}_i = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i^{(k)}, \{\hat{W}_{i,w}^{(k)}\}_{w \in \mathcal{U}_\lambda}\right).$$

  3. Otherwise, the update algorithm parses $\mathsf{pk} = \left(T_{k+1}, Q_{k+1}, \{V_{i,k+1}\}_{i \neq k+1}\right)$ and updates the state as follows:
     - $\hat{T}^{(k+1)} = \hat{T}^{(k)} \cdot T_{k+1}$.
     - For each $i \in [L]$, if $i \neq k+1$ then $\hat{V}_i^{(k+1)} = \hat{V}_i^{(k)} \cdot V_{i,k+1}$. Otherwise, if $i = k+1$, then set $\hat{V}_i^{(k+1)} = \hat{V}_i^{(k)}$.
     - For each $w \in \mathcal{U}_\lambda$, if $w \notin S_{k+1}$, then $\hat{U}_w^{(k+1)} = \hat{U}_w^{(k)} \cdot U_{k+1}$. Otherwise, if $w \in S_{k+1}$, then $\hat{U}_w^{(k+1)} = \hat{U}_w^{(k)}$.
     - For each $i \in [L]$ and $w \in \mathcal{U}_\lambda$, if $i \neq k+1$ and $w \notin S_{k+1}$, then $\hat{W}_{i,w}^{(k+1)} = \hat{W}_{i,w}^{(k)} \cdot W_{f(i,k+1)}$. Otherwise, set $\hat{W}_{i,w}^{(k+1)} = \hat{W}_{i,w}^{(k)}$.
  4. Output the updated state

$$\mathsf{st} = \left(k+1\,,\, \hat{T}^{(k+1)}\,,\, \{\hat{V}_i^{(k+1)}\}_{i \in [L]}\,,\, \{\hat{U}_w^{(k+1)}\}_{w \in \mathcal{U}_\lambda}\,,\, \{\hat{W}_{i,w}^{(k+1)}\}_{i \in [L], w \in \mathcal{U}_\lambda}\right).$$

To complete the proof, we show that this incremental aggregation procedure implements the same behavior as the standard aggregation procedure. Specifically, we show inductively that for all $k \leq L$, the following properties hold for the elements in the AggregateUpdate algorithm:

- $\hat{T}^{(k)} = \prod_{j \in [k]} T_j$.
- For all $i \in [L]$, $\hat{V}_i^{(k)} = \prod_{j \in [k] \setminus \{i\}} V_{i,j}$.
- For all $w \in \mathcal{U}_\lambda$, $\hat{U}_w^{(k)} = \prod_{j \in [k] : w \notin S_j} U_j$.
- For all $i \in [L]$ and $w \in \mathcal{U}_\lambda$, $\hat{W}_{i,w}^{(k)} = \prod_{j \in [k] \setminus \{i\} : w \notin S_j} W_{f(i,j)}$.

By construction, all of these properties hold for $k = 0$. Moreover, the inductive step follows by inspection: namely, each of the updates in Step 3 simply multiplies in the next component into the product (if present). When $k = L$, the components $\hat{T}^{(L)}$, $\hat{V}_i^{(L)}$, $\hat{U}_w^{(L)}$, and $\hat{W}_{i,w}^{(L)}$ precisely coincide with the quantities in the Aggregate algorithm. Finally, the intermediate state st always contains $O(L \cdot |\mathcal{U}_\lambda|)$ group elements, which proves the claim. □

Theorem 4.8 (Static Security)**.** *Let L be a bound on the number of slots and let $q = 4 \cdot d_{\max} \cdot L \cdot K$. If the q-set-consistent bilinear Diffie-Hellman exponent assumption (Assumption 4.2) holds with respect to* PrimeGroupGen*, then Construction 4.3 is statically secure (for up to L slots).*

Proof. Our security proof relies on a partitioning strategy where we program the indices of the corrupted slots into the common reference string. We begin by defining a sequence of hybrid experiments. Each of our experiments is parameterized by a bit $v \in \{0, 1\}$ (and implicitly, by the security parameter λ). We refer to Section 1.2 for a high-level overview of the reduction strategy.

- $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$: This is the real security game where the challenger encrypts message μ_v^*. We recall the main steps here:
  - **Setup phase:** At the beginning of the game, the adversary $\mathcal{A}$ specifies the number of slots 1^L and the indices of the corrupted slots $C \subseteq [L]$.[7] (In the description, we will also define the indices of the non-corrupted slots as $\mathcal{N} \coloneqq [L] \setminus C$.) The challenger then constructs the common reference string as follows according to the specification of Setup:
    - Specifically, the challenger initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary Dict.
    - The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$.
    - Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set. As in Construction 4.3, we define $f(i, j) \coloneqq d_i + d_j$, $\mathcal{E} \coloneqq \{f(i, j) \mid i, j \in [L] : i \neq j\}$, and $d_{\max} \coloneqq 3 \cdot \max(\mathcal{D})$.
    - The challenger samples random exponents $a, b \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and sets $\alpha = -a^{d_{\max}}$. It also computes $h = \prod_{i \in [L]} g^{a^{d_{\max} - d_i}}$. Then, for each index $i \in [L]$, the challenger also samples $\delta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, and lets $t_i = a^{d_i}$. Then, it defines the following group elements:
      $$A_i = g^{t_i} \quad , \quad B_i = g^{\alpha} h^{t_i} \quad , \quad P_i = g^{\delta_i} \quad , \quad U_i = g^{b t_i}.$$
      For each $z \in \mathcal{E}$, it also sets $W_z = g^{b a^z}$.
    - Finally compute $Z = e(g, g)^{\alpha}$. The challenger constructs the common reference string
      $$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}\,,\, \{W_z\}_{z \in \mathcal{E}}\right) \tag{4.4}$$
      and gives crs to $\mathcal{A}$.
  - **Query phase:** The challenger responds to the adversary's key-generation queries as follows:
    - **Key-generation query:** Whenever algorithm $\mathcal{A}$ makes a key-generation query on a non-corrupted slot index $i \in \mathcal{N}$, the challenger starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. It then computes $T_i = g^{r_i}$, $Q_i = P_i^{r_i}$, and $V_{j,i} = A_j^{r_i}$ for $j \neq i$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}})$ to the dictionary Dict.

    Recall that in the static security game, the adversary is not allowed to make any corruption queries.
  - **Challenge phase:** In the challenge phase, the adversary specifies a challenge policy $P^* = (\mathbf{M}, \rho) \in \mathcal{P}_\lambda$, where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function and two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$.[8] In addition, the adversary specifies a key for for each slot $i \in [L]$ as follows:
    - For each corrupted slot $i \in C$ the adversary specifies a public key $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$ and an attribute set S_i. The challenger checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$ and halts with output $\perp$ if not. Specifically, the challenger checks that $e(T_i, P_i) = e(g, Q_i)$ and for each $j \neq i$, that $e(g, V_{j,i}) = e(T_i, A_j)$.

[7] We assume that the adversary $\mathcal{A}$ chooses a fixed value $L = L(\lambda)$ for each security parameter λ. This is without loss of generality since any algorithm $\mathcal{A}$ that succeeds with non-negligible advantage ε implies a (non-uniform) adversary $\mathcal{B}$ that chooses a fixed value of $L = L(\lambda)$ for each security parameter λ and succeeds with advantage at least ε / L.

[8] Recall that the policy family $\mathcal{P}_\lambda$ consists of policies $(\mathbf{M}, \rho)$ that depend on at most $K(\lambda)$ attributes (i.e., where the share-generation matrix $\mathbf{M}$ has at most K rows). For ease of exposition, we will assume that $\mathbf{M}$ has exactly K rows (since we can always pad the share-generation matrix $\mathbf{M}$ with dummy rows of all-zeroes).

    * For each non-corrupted slot $i \in \mathcal{N}$, the adversary specifies an index $c_i \in [\mathrm{ctr}]$. The challenger looks up the entry $\mathrm{Dict}[c_i] = (i', \mathrm{pk}')$. If $i = i'$, the challenger sets $\mathrm{pk}_i = \mathrm{pk}'$. If $\mathrm{pk}_i \neq \mathrm{pk}'$, the challenger halts with output $\perp$.

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

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

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

$$\hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_j \quad \text{and} \quad \hat{W}_{i,w} = \prod_{j \neq i : w \notin S_j} W_{f(i,j)}.$$

  The challenger then constructs the challenge ciphertext by sampling a secret exponent $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $h_1, h_2 \xleftarrow{\mathrm{R}} \mathbb{G}$ such that $h = h_1 h_2$. It constructs the ciphertext components as follows:

    * **Message-embedding components:** First, let $C_1 = \mu_v^* \cdot Z^s$ and $C_2 = g^s$.
    * **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ for the linear secret sharing scheme and let $\mathbf{v} = [1, v_2, \ldots, v_n]^\intercal$. Then, for each $k \in [K]$, sample $s_k \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, let $C_{3,k} = h_2^{s \mathbf{m}_k^\intercal \mathbf{v}} \hat{U}_{\rho(k)}^{-s_k}$ and $C_{4,k} = g^{s_k}$, where $\mathbf{m}_k^\intercal$ denotes the kth row of $\mathbf{M}$.
    * **Slot-specific component:** Let $C_5 = (h_1 \hat{T}^{-1})^s$.

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

$$\mathrm{ct}^* = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5 \big).$$

  - **Output phase:** At the end of the game, the adversary outputs a bit $v' \in \{0,1\}$, which is also the output of the experiment.

- $\mathsf{Hyb}_1^{(v)}$: Same as $\mathsf{Hyb}_0^{(v)}$, except the challenger makes the following *syntactic* changes:
  - **Setup phase:** In the setup phase, the challenger additionally samples $\beta_{i,k} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ for all $i \in [L]$ and $k \in [K]$. Then, instead of sampling $b \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, the challenger sets

$$b = \sum_{i \in [L]} \sum_{k \in [K]} \frac{1}{\beta_{i,k}} a^{d_{\max} - 2d_i}.$$

  Finally, instead of sampling the encryption randomness $s \in \mathbb{Z}_p$ in the challenge phase, the challenger now samples $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ in the setup phase. For the corrupted slots $i \in \mathcal{C}$, the challenger now sets $P_i = g^{s\delta_i}$ (instead of $P_i = g^{\delta_i}$).
  - **Query phase:** When responding to a key-generation query for a slot $i \in \mathcal{N}$, instead of sampling $r_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, the challenger samples $r_i' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and sets $r_i = a^{d_{\max} - d_i} + r_i'$.
  - **Challenge phase:** After the adversary outputs its challenge policy $P^* = (\mathbf{M}, \rho)$, the challenger computes for each $i \in \mathcal{C}$ a vector $\mathbf{v}_i^* \in \mathbb{Z}_p^n$ with first entry 1 and which is orthogonal to every row $\mathbf{m}_k^\intercal$ of $\mathbf{M}$ where $\rho(k) \in S_i$. Note that such a vector exists (see also Definition 2.2) since the attributes in S_i (for a corrupted slot) do *not* satisfy the challenge policy $P^* = (\mathbf{M}, \rho)$. When generating the challenge ciphertext, the challenger generates the attribute-specific components $C_{3,k}$ and $C_{4,k}$ as well as the slot-specific component C_5 using the following modified procedure:

* **Attribute-specific components:** The challenger sets

$$s_k^* = s \cdot \sum_{i \in C : \rho(k) \notin S_i} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*$$

and constructs the attribute-specific components as

$$C'_{3,k} = g^{s\mathbf{m}_k^{\mathsf{T}} \sum_{i \in C} a^{d_{\max} - d_i} \mathbf{v}_i^*} \cdot \hat{U}_{\rho(k)}^{-s_k^*} \quad \text{and} \quad C'_{4,k} = g^{s_k^*}$$

* **Slot-specific component:** The challenger sets the slot-specific component as

$$C'_5 = g^{s \cdot \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}}$$

Finally, the challenger rerandomizes the attribute-specific and slot-specific ciphertext components using the following rerandomization procedure:

```
Rerand$\big(\{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k \in [K]}, C'_5\big)$:
  1. Sample $\gamma, v'_2, v'_3, \ldots v'_n \xleftarrow{R} \mathbb{Z}_p$ and set $\mathbf{v}' = [1, v'_2, v'_3, \ldots, v'_n]$ and $s'_k \xleftarrow{R} \mathbb{Z}_p$ for each $k \in [K]$.
  2. Compute the rerandomized ciphertext:
       $C_{3,k} = C'_{3,k} \cdot g^{\gamma \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} \cdot \hat{U}_{\rho(k)}^{-s'_k}$ and $C_{4,k} = C'_{4,k} \cdot g^{s'_k}$ and $C_5 = C'_5 g^{-\gamma}$.
  3. Output $\big(\{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big)$.
```

Figure 1: Ciphertext rerandomization algorithm.

The challenger then computes

$$\big(\{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big) = \mathsf{Rerand}\big(\{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k \in [K]}, C'_5\big)$$

and gives the rerandomized ciphertext to the adversary:

$$\mathsf{ct}^* = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big).$$

– **Output phase:** At the end of the game, the adversary outputs a bit $\nu' \in \{0, 1\}$, which is also the output of the experiment.

• $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$: Same as $\mathsf{Hyb}_1^{(\nu)}$ except when constructing the challenge ciphertext, the challenger samples $C_1 \xleftarrow{R} \mathbb{G}_T$. Importantly, this distribution is *independent* of the message.

For a hybrid experiment Hyb and an adversary $\mathcal{A}$, we write $\mathsf{Hyb}(\mathcal{A})$ to denote the output distribution of an execution of Hyb with adversary $\mathcal{A}$. In the following, we argue that each the output distribution of each adjacent pair of hybrid is indistinguishable.

Lemma 4.9. *For all adversaries $\mathcal{A}$ and all $\nu \in \{0, 1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. We show that $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$ and $\mathsf{Hyb}_1^{(\nu)}$ are statistically close by showing that the adversary's view (i.e., the crs from the setup phase, the public keys in the query phase, and the challenge ciphertext ct^* in the challenge phase) in the two distributions is statistically close. We consider each phase separately.

Setup phase. The only difference is how the challenger samples b and P_i. In $\mathsf{Hyb}_1^{(\nu)}$, the challenger sets

$$b = \sum_{i \in [L]} \sum_{k \in [K]} \frac{1}{\beta_{i,k}} a^{d_{\max} - 2d_i},$$

where $\beta_{i,k} \xleftarrow{\text{R}} \mathbb{Z}_p$ for all $i \in [L]$ and $k \in [K]$. We consider the distribution of b:

- Since the challenger samples $a \xleftarrow{\text{R}} \mathbb{Z}_p$ and p is prime, the probability a is a zero to the polynomial $x^{(d_{\max} - 2d_1}$ - i.e. $a^{(d_{\max} - 2d_1} = 0$ has probability at most $((d_{\max} - 2d_1)/p$. Since $\mathcal{D}$ is efficiently-computable, it follows that $d_{\max} \leq \mathsf{poly}(L) = \mathsf{poly}(\lambda)$. Since $p = 2^{-\Omega(\lambda)}$, we conclude that $(d_{\max} - 2d_1)/p = \mathsf{negl}(\lambda)$ and so, $a^{d_{\max} - 2d_i}$ is non-zero with overwhelming probability.
- Since each $\beta_{i,k}$ is uniform over $\mathbb{Z}_p$, they are non-zero with overwhelming probability. In this case, the distribution of each $\beta_{i,k}^{-1}$ is independent and uniform over $\mathbb{Z}_p$.

Thus, with overwhelming probability, $a^{d_{\max} - 2d_i} \neq 0$ and each $\beta_{i,k}^{-1}$ is an uniform (non-zero) value over $\mathbb{Z}_p$. We conclude that the distribution of b is statistically close to uniform over $\mathbb{Z}_p$, which is the distribution of b in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. Next, consider the distribution of P_i for $i \in \mathcal{C}$. In $\mathsf{Hyb}_0^{(\nu)}$, the challenger sets $P_i = g^{\delta_i}$ while in $\mathsf{Hyb}_1^{(\nu)}$, the challenger sets $P_i = g^{s\delta_i}$, where $s \xleftarrow{\text{R}} \mathbb{Z}_p$ and $\delta_i \xleftarrow{\text{R}} \mathbb{Z}_p$. As long as $s \neq 0$, these two distributions are identical. Since s is sampled uniformly, these two distributions are statistically close.

Query phase. The only change is how the challenger samples r_i for $i \in \mathcal{N}$. In $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$, the challenger samples $r_i \xleftarrow{\text{R}} \mathbb{Z}_p$. In $\mathsf{Hyb}_1^{(\nu)}$, the challenger samples $r_i' \xleftarrow{\text{R}} \mathbb{Z}_p$ and sets

$$r_i = a^{d_{\max} - d_i} + r_i'.$$

These two distributions are identical.

Challenge phase. In $\mathsf{Hyb}_1^{(\nu)}$, the distribution of the attribute-specific and slot-specific ciphertext components can be written as follows:

$$\begin{aligned}
C_{3,k} &= g^{s\mathbf{m}_k^{\mathsf{T}} \sum_{i \in C} a^{d_{\max} - d_i} \mathbf{v}_i^*} \cdot \hat{U}_{\rho(k)}^{-s_k^*} \cdot g^{\gamma \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} \cdot \hat{U}_{\rho(k)}^{-s_k'} = g^{s\mathbf{m}_k^{\mathsf{T}} \left((\gamma/s) \cdot \mathbf{v}' + \sum_{i \in C} a^{d_{\max} - d_i} \mathbf{v}_i^* \right)} \hat{U}_{\rho(k)}^{-(s_k^* + s_k')} \\
C_{4,k} &= g^{s_k^*} g^{s_k'} = g^{s_k^* + s_k'} \\
C_5 &= g^{s \cdot \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} g^{-\gamma} = g^{s \left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i} \right)} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}},
\end{aligned}$$

where $\gamma, v_2', \ldots, v_n' \xleftarrow{\text{R}} \mathbb{Z}_p$, $s_k' \xleftarrow{\text{R}} \mathbb{Z}_p$ for all $k \in [K]$, and $\mathbf{v}' = [1, v_2', \ldots, v_n']$. For each $i \in C$, let (pk_i, S_i) be the public key and set of attributes the adversary chooses for slot $i \in C$. Parse $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$, and let $r_i \in \mathbb{Z}_p$ be the discrete log of T_i (i.e., $T_i = g^{r_i}$). Without loss of generality, we can assume that for all $i \in C$, $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$. Otherwise, the output in both experiments is $\perp$. In $\mathsf{Hyb}_1^{(\nu)}$, the challenger sets $P_i = g^{s\delta_i}$, so by construction of $\mathsf{IsValid}$,

$$e(g, Q_i) = e(T_i, P_i) = e(g, g)^{s\delta_i r_i}.$$

In particular, $Q_i = g^{s\delta_i r_i}$. Thus, we can rewrite C_5 as

$$\begin{aligned}
C_5 = g^{s \left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i} \right)} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} &= g^{s \left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i} \right)} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} g^{-sr_i} \\
&= g^{s \left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i} \right)} \prod_{j \in [L]} T_j^{-s} \\
&= g^{s \left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i} \right)} \hat{T}^{-s}.
\end{aligned}$$

We claim now that the distribution in $\mathsf{Hyb}_1^{(\nu)}$ is equivalent to an execution of $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$ with the following variable assignments:

$$h_1 := g^{-\gamma/s + \sum_{i \in \mathcal{N}} a^{d_{\max} - d_i}} \quad \text{and} \quad h_2 := g^{\gamma/s + \sum_{i \in C} a^{d_{\max} - d_i}},$$

and for all $k \in [K]$, $s_k := s_k^* + s_k'$, and

$$\mathbf{v} := \frac{(\gamma/s)\mathbf{v}' + \sum_{i \in C} a^{d_{\max} - d_i} \mathbf{v}_i^*}{\gamma/s + \sum_{i \in C} a^{d_{\max} - d_i}}.$$

For this assignment of variables, observe that

$$\begin{aligned} h_2^{s\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s_k} &= g^{s\left(\gamma/s + \sum_{i \in C} a^{d_{\max} - d_i}\right)\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-(s_k^* + s_k')} \\ &= g^{s\mathbf{m}_k^{\mathsf{T}}\left((\gamma/s)\mathbf{v}' + \sum_{i \in C} a^{d_{\max} - d_i}\mathbf{v}_i^*\right)} \cdot \hat{U}_{\rho(k)}^{-(s_k^* + s_k')} = C_{3,k}, \end{aligned}$$

$g^{s_k} = g^{s_k^* + s_k'} = C_{4,k}$, and $(h_1 \hat{T}^{-1})^s = C_5$, which coincides with the definitions in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. To complete the proof, it suffices to argue that this choice of assignments are distributed according to the specification in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. We analyze each component as follows:

- In $\mathsf{Hyb}_1^{(\nu)}$, the challenger samples $\gamma, s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. As long as $s \neq 0$ (which happens with overwhelming probability), then over the random choice of γ, the distribution of γ/s is uniform. Thus, with overwhelming probability over the choice of s, the distribution of h_1 is uniform over $\mathbb{G}$. Moreover,

  $$h_1 h_2 = g^{\sum_{i \in \mathcal{N}} a^{d_{\max} - d_i} + \sum_{i \in C} a^{d_{\max} - d_i}} = g^{\sum_{i \in [L]} a^{d_{\max} - d_i}} = h,$$

  since C and $\mathcal{N}$ are a partition of $[L]$.

- Since the challenger samples $s_k' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, the distribution of s_k is also uniform over $\mathbb{Z}_p$, which matches the distribution in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$.

- Write $\mathbf{v} = [v_1, v_2, \ldots, v_n]$ and $\mathbf{v}' = [1, v_2', \ldots, v_n']$. By construction, the first component of $\mathbf{v}'$ and $\mathbf{v}_i^*$ for all $i \in C$ is 1. This means $v_1 = 1$, just as in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. For $i > 1$, the challenger in $\mathsf{Hyb}_2^{(\nu)}$ samples $v_i' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. Thus, as long as $\gamma, s \neq 0$, the distribution of $\gamma/s \cdot v_i'$ is uniformly random (and independent of all other components). Correspondingly, this means that the distributions of $v_2, \ldots, v_n$ are independent and uniform over $\mathbb{Z}_p$, exactly as required in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. Since the challenger samples $\gamma, s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, they are non-zero with overwhelming probability.

Thus, with overwhelming probability over the choice of a, γ, and s, the challenge ciphertext $\mathsf{Hyb}_1^{(\nu)}$ is distributed exactly according to the distribution in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. We conclude that the adversary's view in the two experiments are statistically indistinguishable, and the claim holds. □

Analyzing $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_2^{(\nu)}$. Next, we show that under the set-consistent decisional bilinear Diffie-Hellman exponent assumption (Assumption 4.2), the output distributions of $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_2^{(\nu)}$ are computationally indistinguishable. To simplify this analysis, we start by defining the following intermediate assumption which is implied by Assumption 4.2. The structure of the intermediate assumption enables a more direct reduction. This intermediate assumption can be viewed as a generalization of the parallel bilinear Diffie-Hellman exponent assumption introduced in [Wat11]. In Appendix E (Lemma E.1), we show that the intermediate assumption directly reduces to (a suitably parameterized version of) Assumption 4.2.

Assumption 4.10 (Intermediate Set-Consistent Bilinear Diffie Hellman Exponent)**.** Let PrimeGroupGen be a prime-order group generator. For a security parameter λ and a bit $b \in \{0,1\}$ we define the (q_1, q_2)-intermediate set-consistent bilinear Diffie-Hellman exponent game between an adversary $\mathcal{A}$ and a challenger as follows:

- On input the security parameter 1^λ, adversary $\mathcal{A}$ outputs a set $S \subseteq [q_1 - 1]$.

- The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$ and exponents $a, s, \beta_1, \dots \beta_{q_2} \xleftarrow{\mathrm{R}} \mathbb{Z}_p$.
- The challenger then constructs the following components:
  - For each $i \in [2q_1]$, it sets $X_i = g^{a^i}$ and $Z_i = g^{a^i s}$.
  - For each $i \in [2q_1]$ and $j, k \in [q_2]$, it sets $Y^{(j)} = g^{s\beta_j}$, $X_i^{(j)} = g^{a^i/\beta_j}$, and $Z_i^{(j,k)} = g^{a^i s \beta_k/\beta_j}$.
  - Finally, it computes $Q = e(g,g)^{a^{q_1}}$, $T_0 = e(g,g)^{a^{q_1} s}$, and samples $T_1 \xleftarrow{\mathrm{R}} \mathbb{G}_T$.

  The challenger gives the following components to the adversary:
  - $\mathcal{G}$, g, Y, $\{X_i\}_{i \in S \cup [q_1+1, 2q_1]}$, $\{Z_{q_1 - i}\}_{i \in [q_1 - 1] \setminus S}$, $\{Z_i\}_{i \in [q_1+1, 2q_1]}$; and
  - $\{Y^{(j)}\}_{j \in [q_2]}$, $\{X_i^{(j)}\}_{i \in [2q_1] \setminus \{q_1\}, j \in [q_2]}$, $\{Z_i^{(j,k)}\}_{i \in [2q_1] \setminus \{q_1\}, j \neq k}$, Q, $\boxed{T_b}$.
- The adversary outputs a bit $b' \in \{0,1\}$ which is the output of the experiment.

We say the (q_1, q_2)-intermediate set-consistent bilinear Diffie-Hellman exponent assumption holds with respect to $\mathsf{PrimeGroupGen}$ if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

in the (q_1, q_2)-intermediate set-consistent bilinear Diffie-Hellman exponent game.

Lemma 4.11. *Let $q = 4 \cdot d_{\max} \cdot L \cdot K$. Suppose the q-set-consistent bilinear Diffie-Hellman exponent assumption (Assumption 4.2) holds with respect to* $\mathsf{PrimeGroupGen}$. *Then, the (q_1, q_2)-intermediate set-consistent bilinear Diffie-Hellman exponent assumption holds with respect to* $\mathsf{PrimeGroupGen}$ *for $q_1 = d_{\max}$ and $q_2 = L \cdot K$.*

Proof. We give the proof in Appendix E (see Lemma E.1). □

Lemma 4.12. *Suppose Assumption 4.2 holds for $q = 4 \cdot d_{\max} \cdot L \cdot K$ with respect to* $\mathsf{PrimeGroupGen}$. *Then, for all efficient adversaries $\mathcal{A}$, and all $\nu \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. Take any $\nu \in \{0,1\}$ and suppose there exists an efficient adversary $\mathcal{A}$ where

$$\left|\Pr[\mathsf{Hyb}_1^{(\nu)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}(\mathcal{A}) = 1]\right| = \varepsilon$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the (q_1, q_2)-intermediate set-consistent BDHE assumption, where $q_1 = d_{\max}$ and $q_2 = L \cdot K$. In the following, we will refer to elements of the set $[q_2] = \{1, \dots, q_2\}$ by a *pair* of indices $(i, k) \in [L] \times [K]$. We now give the description of $\mathcal{B}$:

- **Setup phase:** Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ starts by specifying the number of slots 1^L and the indices of the corrupted slots $C \subseteq [L]$. Algorithm $\mathcal{B}$ then initializes the following quantities:
  - Algorithm $\mathcal{B}$ initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary Dict to keep track of the key-generation queries.
  - Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently computable progression and double-free set. As in Construction 4.3, we define $f(i,j) = d_i + d_j$ and the set $\mathcal{E} = \{f(i,j) \mid i, j \in [L] : i \neq j\}$. Let $d_{\max} = 3 \cdot \max(\mathcal{D})$.

  Algorithm $\mathcal{B}$ sends the set

  $$S = \mathcal{D} \cup \{d_{\max} - d_i\}_{i \in [L]} \cup \{d_{\max} - d_i + d_j\}_{i \neq j \in [L]}$$

  to the (q_1, q_2)-intermediate set-consistent BDHE challenger and receives the challenge:
  - $\mathcal{G}$, g, Y, $\{X_j\}_{j \in S \cup [d_{\max}+1, 2d_{\max}]}$, $\{Z_{d_{\max} - j}\}_{j \in [d_{\max} - 1] \setminus S}$, $\{Z_j\}_{j \in [d_{\max}+1, 2d_{\max}]}$; and

- $\{Y^{(i,k)}\}_{i \in [L], k \in [K]}$, $\{X_j^{(i,k)}\}_{j \in [2d_{\max}] \setminus \{d_{\max}\}, i \in [L], k \in [K]}$, $\{Z_j^{((i,k),(i',k'))}\}_{j \in [2d_{\max}] \setminus \{d_{\max}\}, (i,k) \neq (i',k') \in [L] \times [K]}$; and
- Q , T.

For emphasis, we color the components from the challenge in green. Next, algorithm $\mathcal{B}$ computes $h = \prod_{i \in [L]} X_{d_{\max} - d_i}$. Then, for each slot $i \in [L]$, it computes

$$A_i = X_{d_i} \quad \text{and} \quad B_i = \prod_{j \in [L], j \neq i} X_{d_{\max} - d_j + d_i} \quad \text{and} \quad U_i = \prod_{j \in [L], k \in [K]} X^{(j,k)}_{d_{\max} - 2d_j + d_i}$$

Next, for each $i \in [L]$ it samples $\delta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. If $i \in \mathcal{N}$, it sets $P_i = g^{\delta_i}$, and if $i \in C$, it sets $P_i = Y^{\delta_i}$. For each $z \in \mathcal{E}$, algorithm $\mathcal{B}$ computes $W_z = \prod_{j \in [L], k \in [K]} X^{(j,k)}_{d_{\max} - 2d_j + z}$. Finally, algorithm $\mathcal{B}$ sets $Z = Q^{-1}$ and defines the common reference string to be

$$\mathsf{crs} = \left(\mathcal{G}, Z, g, h, \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}, \{W_z\}_{z \in \mathcal{E}}\right). \tag{4.5}$$

Algorithm $\mathcal{B}$ gives crs to $\mathcal{A}$.

- **Query phase:** During the query phase, whenever algorithm $\mathcal{A}$ makes a key-generation query on a non-corrupted slot index $i \in \mathcal{N}$, algorithm $\mathcal{B}$ starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. It then sets

$$T_i = X_{d_{\max} - d_i} g^{r_i'} \quad \text{and} \quad Q_i = \left(X_{d_{\max} - d_i} g^{r_i'}\right)^{\delta_i} \quad \text{and} \quad V_{j,i} = X_{d_{\max} - d_i + d_j} X_{d_j}^{r_i'},$$

for all $j \neq i$. Then $\mathcal{B}$ sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}})$ to the dictionary Dict.

- **Challenge phase:** In the challenge phase, algorithm $\mathcal{A}$ specifies a challenge policy $P^* = (\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function, along with two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$. In addition, algorithm $\mathcal{A}$ specifies a key for for each slot $i \in [L]$ as follows:

  - For each corrupted slot $i \in C$, algorithm $\mathcal{A}$ specifies a public key pk_i and an attribute set S_i. Algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$ and halts with output $\perp$ if not.
  - For each non-corrupted slot $i \in \mathcal{N}$, the adversary specifies an index $c_i \in [\mathsf{ctr}]$. Algorithm $\mathcal{B}$ looks up the entry $\mathsf{Dict}[c_i] = (i', \mathsf{pk}')$. If $i = i'$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}'$. If $i \neq i'$, then algorithm $\mathcal{B}$ halts with output $\perp$.

  For each slot $i \in [L]$, algorithm $\mathcal{B}$ parses the associated public key pk_i as $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$. Algorithm $\mathcal{B}$ then computes the attribute-independent public key $\hat{T}$ and attribute-independent slot key $\hat{V}_i$ for each $i \in [L]$ as follows:

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

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

$$\hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_j \quad \text{and} \quad \hat{W}_{i,w} = \prod_{j \neq i : w \notin S_j} W_{f(i,j)}.$$

  Next, algorithm $\mathcal{B}$ constructs the challenge ciphertext. Since $\mathcal{A}$ is admissible, the attributes S_i for all corrupted indices $i \in C$ do *not* satisfy the challenge policy P^*. Thus, for each $i \in C$, there exists a vector $\mathbf{v}_i^*$ with first entry 1 and which is orthogonal to every row $\mathbf{m}_k^{\mathsf{T}}$ of $\mathbf{M}$ where $\rho(k) \in S_i$. Algorithm $\mathcal{B}$ now proceeds as follows:

  - **Message-embedding components:** First, algorithm $\mathcal{B}$ sets $C_1 = \mu_v^* / T$ and $C_2 = Y$.

- **Attribute-specific components:** For ease of notation, for each $k \in [K]$, we define the following sets of indices $\Upsilon_1^{(k)}$, $\Upsilon_2^{(k)}$, and Υ_3:

$$\Upsilon_1^{(k)} = \{i \in [L] : \rho(k) \notin S_i\} \quad \text{and} \quad \Upsilon_2^{(k)} = \{i \in C : \rho(k) \notin S_i\} \quad \text{and} \quad \Upsilon_3 = [L] \times [K] \tag{4.6}$$

Then, algorithm $\mathcal{B}$ computes $C'_{3,k}$ as

$$C'_{3,k} = \left(\prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)}} \prod_{\substack{(j',k') \in \Upsilon_3 \\ (j',k') \neq (j,k)}} \left(Z^{((j',k'),(j,k))}_{d_{\max}-2d_j+d_i} \right)^{-\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^*} \right) \left(\prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)} \setminus \{i\}} Z_{d_{\max}-2d_j+d_i}^{-\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^*} \right),$$

and $C'_{4,k}$ as

$$C'_{4,k} = \prod_{i \in C : \rho(k) \notin S_i} \left(Y^{(i,k)} \right)^{\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*}.$$

- **Slot-specific component:** Algorithm $\mathcal{B}$ computes C'_5 as

$$C'_5 = \prod_{i \in N} Y^{-r'_i} \prod_{i \in C} Q_i^{-\delta_i^{-1}}.$$

Finally, algorithm $\mathcal{B}$ computes

$$\left(\{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5 \right) = \mathsf{Rerand}\left(\{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k \in [K]}, C'_5 \right).$$

Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the challenge ciphertext

$$\mathsf{ct}^* = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5 \right).$$

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

We start by showing that algorithm $\mathcal{B}$ is able to simulate all of the parameters for $\mathcal{A}$ using the group elements from the (q_1, q_2)-intermediate set-consistent BDHE challenge. This will crucially rely on the progression-free and double-free properties of the set $\mathcal{D}$.

- **CRS components:** First, we consider the components of the CRS.
  - Computing h, A_i, and B_i for $i \in [L]$ requires knowledge of the elements $X_{d_{\max}-d_i}$, X_{d_i}, and $X_{d_{\max}-d_i+d_j}$ for all $j \in [L] \setminus \{i\}$. These are precisely the components of the set $\mathcal{S}$, so each of these components are included as part of the challenge.
  - Next, the component U_i depends on $X^{(j,k)}_{d_{\max}-2d_j+d_i}$ for all $i, j \in [L]$ and $k \in [K]$. By construction, the challenge contains $X_\ell^{(i,k)}$ for all $i \in [L]$, $k \in [K]$, and $\ell \in [2d_{\max}] \setminus \{d_{\max}\}$. Thus, it suffices to show that $d_{\max} - 2d_j + d_i \in [2d_{\max}] \setminus \{d_{\max}\}$. Since $d_{\max} = 3 \cdot \max(\mathcal{D})$ and $d_i, d_j \in \mathcal{D} \subset \mathbb{N}$, this means $d_{\max} - 2d_j > 0$ and $d_{\max} + d_i \le 2d_{\max}$. Correspondingly, this means that $d_{\max} - 2d_j + d_i \in [2d_{\max}]$. It remains to argue that $d_{\max} - 2d_j + d_i \neq d_{\max}$. Suppose otherwise. Then, it must be the case that $d_i = 2d_j$, which contradicts the fact that $\mathcal{D}$ is double-free (Definition 2.6). Hence, $d_{\max} - 2d_j + d_i \in [2d_{\max}] \setminus \{d_{\max}\}$, and algorithm $\mathcal{B}$ is able to construct U_i for all $i \in [L]$.
  - The component $P_i = Y^{\delta_i}$ can be simulated using Y.
  - For each $z \in \mathcal{E}$ where $\mathcal{E} = \{f(i,j) \mid i,j \in [L] : i \neq j\}$, the term W_z depends on $X^{(j,k)}_{d_{\max}-2d_j+z}$ for $j \in [L]$ and $k \in [K]$. As in the previous case, it suffices to show that $d_{\max} - 2d_j + z \in [2d_{\max}] \setminus \{d_{\max}\}$. Since $z \in \mathcal{E}$, it can be written as $d_{i'} + d_{j'}$ for some choice of $i' \neq j' \in [L]$. Thus, $z \le 2\max(\mathcal{D})$. Since $d_{\max} = 3 \cdot \max(\mathcal{D})$, this means $d_{\max} - 2d_j > 0$ and $d_{\max} + z < 2d_{\max}$. Hence, $d_{\max} - 2d_j + z \in [2d_{\max}]$. It remains to argue that $d_{\max} - 2d_j + z \neq d_{\max}$. Suppose otherwise. Then, it must be the case that $2d_j = z$. Since $z \in \mathcal{E}$, there exists $i' \neq j' \in [L]$ such that $d_{i'} + d_{j'} = z = 2d_j$, which contradicts the fact that $\mathcal{D}$ is progression-free (Definition 2.4). Hence, $d_{\max} - 2d_j + z \in [2d_{\max}] \setminus \{d_{\max}\}$, and algorithm $\mathcal{B}$ is able to construct W_z for all $z \in \mathcal{E}$.

  - Finally, algorithm $\mathcal{B}$ sets $Z = Q^{-1}$ which can be computed from the challenge.

- **Key-generation queries:** Next, we consider the elements algorithm $\mathcal{B}$ uses to simulate public keys when responding to the adversary's key-generation queries. For each $i \in \mathcal{N}$, the elements T_i, Q_i, and $V_{j,i}$ for $j \neq i$ require knowledge of $X_{d_{\max}-d_i}$, $X_{d_{\max}-d_i+d_j}$, and X_{d_j}. Once again, these are the components of $\mathcal{S}$, and thus part of the challenge.

- **Challenge ciphertext:** Finally, we consider the components of the challenge ciphertext:
  - To construct C_1 and C_2, algorithm $\mathcal{B}$ requires T and Y, which are part of the challenge.
  - To construct $C'_{3,k}$, algorithm $\mathcal{B}$ requires $Z^{((j',k'),(j,k))}_{d_{\max}-2d_j+d_i}$ and $Z_{d_{\max}-2d_j+d_i}$ for all $i \in \Upsilon^{(k)}_1$, $j \in \Upsilon^{(k)}_2$, and $(j',k') \in \Upsilon_3$ where $(j',k') \neq (j,k)$. We consider the two terms individually:
    * Consider $Z_{d_{\max}-2d_j+d_i}$ where $i \in \Upsilon^{(k)}_1 \subseteq [L]$ and $j \in \Upsilon^{(k)}_2 \subseteq [L]$. By construction, the challenge contains $Z_{d_{\max}-\ell}$ if $\ell \in [d_{\max}-1] \setminus \mathcal{S}$ and Z_ℓ for $\ell \in [d_{\max}+1, 2d_{\max}]$. As argued previously, since $d_{\max} = 3 \cdot \max(\mathcal{D})$ and $d_i, d_j \in \mathcal{D}$ and $\mathcal{D}$ is double-free, it follows that

$$d_{\max} - 2d_j + d_i \in [2d_{\max}] \setminus \{d_{\max}\}.$$

    If $d_{\max} - 2d_j + d_i \geq d_{\max} + 1$, then we are done. It suffices to consider the case where $d_{\max} - 2d_j + d_i \leq d_{\max} - 1$. In this case, the challenge contains $Z_{d_{\max}-2d_j+d_i}$ as long as $2d_j - d_i \in [d_{\max}-1] \setminus \mathcal{S}$. Since $d_{\max} = 3 \cdot \max(\mathcal{D})$, it follows that $2d_j - d_i < 3 \cdot \max(\mathcal{D}) = d_{\max}$. We only need to show that $2d_j - d_i \notin \mathcal{S}$. Here, we consider three possibilities:
      · Suppose $2d_j - d_i \in \mathcal{D}$. Then, there exists some $i' \in [L]$ such that $2d_j = d_i + d_{i'}$. When $i = i'$, this contradicts the assumption that $\mathcal{D}$ is double-free. When $i \neq i'$, this contradicts the assumption that $\mathcal{D}$ is progression-free. Thus, this cannot happen.
      · Suppose $2d_j - d_i = d_{\max} - d_{i'}$ for some $i' \in [L] \leftrightarrow 2d_j + d_{i'} \neq d_{\max} + d_i$. This means that $2d_j + d_{i'} - d_i = d_{\max}$. Since $\mathcal{D} \subset \mathbb{N}$, we have $d_i > 0$. Then $d_{\max} = 2d_j + d_{i'} - d_i < 3 \cdot \max(\mathcal{D}) = d_{\max}$, which is a contradiction.
      · Suppose $2d_j - d_i = d_{\max} - d_{i'} + d_{j'}$ for some $i' \neq j' \in [L]$. This means that $2d_j + d_{i'} = d_{\max} + d_i + d_{j'}$. Similar to the previous case, we appeal to the fact that $d_{\max} = 3 \cdot \max(\mathcal{D})$ to obtain a contradiction.

    We conclude that whenever $d_{\max} - 2d_j + d_i \leq d_{\max} - 1$, it holds that $2d_j - d_i \notin \mathcal{S}$. In this case, the challenge contains the term $Z_{d_{\max}-2d_j+d_i}$, as required.
    * Consider $Z^{((j',k'),(j,k))}_{d_{\max}-2d_j+d_i}$. By construction, the challenge contains $Z^{((j',k'),(j,k))}_\ell$ for all $\ell \in [2d_{\max}] \setminus \{d_{\max}\}$ and $(j',k') \neq (j,k)$. It suffices to argue that for all $i \in \Upsilon^{(k)}_1 \subseteq [L]$ and $j \in \Upsilon^{(k)}_2 \subseteq [L]$, it holds that $d_{\max} - 2d_j + d_i \in [2d_{\max}] \setminus \{d_{\max}\}$. As in the previous case, this follows when $d_{\max} = 3 \cdot \max(\mathcal{D})$ and $d_i, d_j \in \mathcal{D}$, and $\mathcal{D}$ is double-free.
  - To construct $C'_{4,k}$, algorithm $\mathcal{B}$ requires $Y^{(i,k)}$ for all $i \in C$ where $\rho(k) \notin S_i$. Since the challenge contains $Y^{(i,k)}$ for all $i \in [L]$ and $k \in [K]$, algorithm $\mathcal{B}$ can construct this term.
  - To construct C'_5, algorithm $\mathcal{B}$ needs Y, which is included in the challenge.

We conclude that the challenge contains all of the components algorithm $\mathcal{B}$ needs for simulating the CRS, the key-generation queries, and the challenge ciphertext. To complete the proof, we show that depending on the distribution of T, algorithm $\mathcal{B}$ either simulates an execution of $\mathsf{Hyb}^{(\nu)}_1$ or $\mathsf{Hyb}^{(\nu)}_{\mathsf{rand}}$ for $\mathcal{A}$. Let $a, s, \beta_{i,k} \in \mathbb{Z}_p$ for $i \in [L]$ and $k \in [K]$ be the exponents sampled by the (q_1,q_2)-intermediate set-consistent BDHE challenger. Then, the challenge components are defined as follows:

$$Y = g^s\,,\ X_i = g^{a^i}\,,\ Z_i = g^{a^i s}\,,\ Y^{(i,k)} = g^{s\beta_{i,k}}\,,\ X^{(i,k)}_j = g^{a^j/\beta_{i,k}}\,,\ Z^{(i,k),(i',k')}_j = g^{a^j s\beta_{i',k'}/\beta_{i,k}}\,,\ Q = e(g,g)^{a^{d_{\max}}}.$$

We claim that algorithm $\mathcal{B}$ simulates an execution of $\mathsf{Hyb}^{(\nu)}_1$ or $\mathsf{Hyb}^{(\nu)}_{\mathsf{rand}}$ where the exponents $a, s, \beta_{i,k}$ are the corresponding ones sampled by the (q_1,q_2)-intermediate set-consistent BDHE challenger.

CRS components. Consider first the components of the CRS. Then in an execution of $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ where the randomness is $a, s, \{\beta_{i,k}\}_{i\in[L],k\in[K]}$, the challenger constructs the components of the CRS as follows. First, the challenger sets $\alpha = -a^{d_{\max}}$ and $t_i = a^{d_i}$. It also computes

$$h = \prod_{i\in[L]} g^{a^{d_{\max}-d_i}} = \prod_{i\in[L]} X_{d_{\max}-d_i},$$

which matches the behavior of algorithm $\mathcal{B}$. Then, for each $i \in [L]$, the challenger would compute

$$A_i = g^{t_i} = g^{a^{d_i}} = X_{d_i}$$

$$B_i = g^{\alpha} h^{t_i} = g^{-a^{d_{\max}}} \left(\prod_{j\in[L]} g^{a^{d_{\max}-d_j}} \right)^{a^{d_i}} = \prod_{j\in[L],j\neq i} g^{a^{d_{\max}-d_j+d_i}} = \prod_{j\in[L],j\neq i} X_{d_{\max}-d_j+d_i},$$

which matches the behavior of algorithm $\mathcal{B}$. In both $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger sets

$$b = \sum_{j\in[L]} \sum_{k\in[K]} \frac{1}{\beta_{j,k}} a^{d_{\max}-2d_j} = \sum_{(j,k)\in\Upsilon_3} \frac{1}{\beta_{j,k}} a^{d_{\max}-2d_j}, \tag{4.7}$$

where $\Upsilon_3 = [L] \times [K]$ from Eq. (4.6). The challenger then computes

$$U_i = g^{bt_i} = g^{\left(\sum_{j\in[L],k\in[K]} \frac{1}{\beta_{j,k}} a^{d_{\max}-2d_j}\right) a^{d_i}} = \prod_{j\in[L],k\in[K]} g^{\frac{1}{\beta_{j,k}}\left(a^{d_{\max}-2d_j+d_i}\right)} = \prod_{j\in[L],k\in[K]} X^{(j,k)}_{d_{\max}-2d_j+d_i},$$

which again matches the behavior of $\mathcal{B}$. Next, for each $i \in [L]$, the challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ sets $P_i = g^{s\delta_i} = Y^{\delta_i}$ if $i \in \mathcal{C}$ and $P_i = g^{\delta_i}$ if $i \in \mathcal{N}$. This is the same procedure used by algorithm $\mathcal{B}$. Next, for each $z \in \mathcal{E}$, the challenger would set

$$W_z = g^{ba^z} = g^{\left(\sum_{j\in[L],k\in[K]} \frac{1}{\beta_{j,k}} a^{d_{\max}-2d_j}\right) a^z} = \prod_{j\in[L],k\in[K]} g^{\frac{1}{\beta_{j,k}}\left(a^{d_{\max}-2d_j+z}\right)} = \prod_{j\in[L],k\in[K]} X^{(j,k)}_{d_{\max}-2d_j+z}.$$

Finally, the challenger sets

$$Z = e(g,g)^{\alpha} = e(g,g)^{-a^{d_{\max}}} = Q^{-1}.$$

We conclude that algorithm $\mathcal{B}$ constructs the components in the CRS using the identical procedure as the challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$.

Key-generation queries. For the key-generation queries on indices $i \in \mathcal{N}$, the challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ generates $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j\neq i})$ by first sampling $r_i' \xleftarrow{\text{R}} \mathbb{Z}_p$, setting $r_i = a^{d_{\max}-d_i} + r_i'$ and then setting

- $T_i = g^{r_i} = g^{a^{d_{\max}-d_i}+r_i'} = X_{d_{\max}-d_i} g^{r_i'}$.
- $Q_i = P_i^{r_i} = g^{\delta_i(a^{d_{\max}-d_i}+r_i')} = (X_{d_{\max}-d_i} g^{r_i'})^{\delta_i}$. Recall that $i \in \mathcal{N}$ so $P_i = g^{\delta_i}$.
- $V_{j,i} = A_j^{r_i} = g^{a^{d_j}(a^{d_{\max}-d_i}+r_i')} = X_{d_{\max}-d_i+d_j} X_{d_j}^{r_i'}$.

Again, algorithm $\mathcal{B}$ perfectly simulates the responses to the key-generation queries.

Challenge ciphertext. We now analyze the challenge ciphertext components. First, we consider the distribution of C_1. We have two possibilities:

- Suppose $T = e(g,g)^{a^{d_{\max}} s}$. Then algorithm $\mathcal{B}$ sets $C_1 = \mu_\nu^* / T = \mu_\nu^* \cdot e(g,g)^{-a^{d_{\max}} s} = \mu_\nu^* \cdot Z^s$, which matches the distribution of C_1 in $\mathsf{Hyb}_1^{(\nu)}$.
- Suppose the challenger samples $T \xleftarrow{\text{R}} \mathbb{G}_T$. Then, $C_1 = \mu_\nu^*/T$ is also uniform over $\mathbb{G}_T$, and algorithm $\mathcal{B}$ simulated the distribution of C_1 in $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$.

To complete the proof, it suffices to argue that the remaining components in the challenge ciphertext are simulated exactly according to the specification of $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. First, in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger would set $C_2 = g^s = Y$. which coincides with the behavior of algorithm $\mathcal{B}$. Next, consider $C'_{3,k}$ for $k \in [K]$. In $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger would first set

$$s_k^* = s \cdot \sum_{i \in C : \rho(k) \notin S_i} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* = s \cdot \sum_{i \in \Upsilon_2^{(k)}} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*, \tag{4.8}$$

using the definition of $\Upsilon_2^{(k)} = \{i \in C : \rho(k) \notin S_i\}$ from Eq. (4.6). Then, the challenger computes

$$C'_{3,k} = g^{s \mathbf{m}_k^{\mathsf{T}} \sum_{i \in C} a^{d_{\max} - d_i} \mathbf{v}_i^*} \cdot \hat{U}_{\rho(k)}^{-s_k^*} = \hat{U}_{\rho(k)}^{-s_k^*} \prod_{i \in C} g^{a^{d_{\max} - d_i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} = \hat{U}_{\rho(k)}^{-s_k^*} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{d_{\max} - d_i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s}, \tag{4.9}$$

using the fact that in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger chooses $\mathbf{v}_i^*$ such that $\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* = 0$ for all $k \in [K]$ where $\rho(k) \in S_i$. Consider the term $\hat{U}_{\rho(k)}^{-s_k^*}$. By definition,

$$\hat{U}_{\rho(k)} = \prod_{i \in [L] : \rho(k) \notin S_i} U_i = \prod_{i \in \Upsilon_1^{(k)}} g^{b t_i} = \prod_{i \in \Upsilon_1^{(k)}} g^{b a^{d_i}}$$

using the definition of $\Upsilon_1^{(k)} = \{i \in [L] : \rho(k) \notin S_i\}$ from Eq. (4.6). For ease of notation, let $\hat{U}_{\rho(k)}^{-s_k^*} = g^{\xi}$ for some $\xi \in \mathbb{Z}_p$. Then, substituting in the definitions of b from Eq. (4.7) and s_k^* from Eq. (4.8), we have

$$\begin{aligned} \xi &= \sum_{i \in \Upsilon_1^{(k)}} -b a^{d_i} s_k^* \\ &= \sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} -b \beta_{j,k} a^{d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s && \text{by Eq. (4.8)} \\ &= \sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} \sum_{(j',k') \in \Upsilon_3} -\frac{\beta_{j,k}}{\beta_{j',k'}} a^{d_{\max} - 2d_{j'} + d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s && \text{by Eq. (4.7).} \end{aligned}$$

We decompose ξ into the terms ξ_1 where $(j',k') \neq (j,k)$ and the terms ξ_2 where $(j',k') \neq (j,k)$. Then, we have

$$\xi = \underbrace{\sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} \sum_{\substack{(j',k') \in \Upsilon_3 \\ (j',k') \neq (j,k)}} \left(-\frac{\beta_{j,k}}{\beta_{j',k'}} a^{d_{\max} - 2d_{j'} + d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s \right)}_{\xi_1} + \underbrace{\sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} -a^{d_{\max} - 2d_j + d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s}_{\xi_2}. \tag{4.10}$$

We further decompose ξ_2 into terms $\xi_{2,1}$ where $i \neq j$ and terms $\xi_{2,2}$ where $i = j$:

$$\xi_2 = \sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} -a^{d_{\max} - 2d_j + d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s = \underbrace{\sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)} \setminus \{i\}} -a^{d_{\max} - 2d_j + d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s}_{\xi_{2,1}} + \underbrace{\sum_{i \in \Upsilon_1^{(k)} \cap \Upsilon_2^{(k)}} -a^{d_{\max} - d_j} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s}_{\xi_{2,2}}. \tag{4.11}$$

From Eq. (4.6), we have that $\Upsilon_2^{(k)} \subseteq \Upsilon_1^{(k)}$, so we can write

$$\xi_{2,2} = \sum_{i \in \Upsilon_1^{(k)} \cap \Upsilon_2^{(k)}} -a^{d_{\max}-d_j} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s = \sum_{i \in \Upsilon_2^{(k)}} -a^{d_{\max}-d_j} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s. \tag{4.12}$$

Combining Eqs. (4.10) to (4.12), we can write

$$\begin{aligned}
g^{\xi_1} &= \prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)}} \prod_{\substack{(j',k') \in \Upsilon_3 \\ (j',k') \neq (j,k)}} g^{\left(-\frac{\beta_{j,k}}{\beta_{j',k'}} a^{d_{\max}-2d_{j'}+d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s\right)} = \prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)}} \prod_{\substack{(j',k') \in \Upsilon_3 \\ (j',k') \neq (j,k)}} \left(Z_{d_{\max}-2d'_j+d_i}^{((j',k'),(j,k))}\right)^{-\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^*} \\
g^{\xi_{2,1}} &= \prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)} \setminus \{i\}} g^{-a^{d_{\max}-2d_j+d_i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s} = \prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)} \setminus \{i\}} Z_{d_{\max}-2d_j+d_i}^{-\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^*} \\
g^{\xi_{2,2}} &= \prod_{i \in \Upsilon_2^{(k)}} g^{-a^{d_{\max}-d_j} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s}.
\end{aligned}$$

Substituting back into Eq. (4.9) and using the fact that $\hat{U}_{\rho(k)}^{-s_k^*} = g^{\xi} = g^{\xi_1+\xi_{2,1}+\xi_{2,2}}$, we conclude that

$$\begin{aligned}
C'_{3,k} = \hat{U}_{\rho(k)}^{-s_k^*} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{d_{\max}-d_i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} &= g^{\xi_1} g^{\xi_{2,1}} g^{\xi_{2,2}} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{d_{\max}-d_i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} \\
&= g^{\xi_1} g^{\xi_{2,1}} \left(\prod_{i \in \Upsilon_2^{(k)}} g^{a^{d_{\max}-d_i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} \right)^{-1} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{d_{\max}-d_i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} \\
&= \left(\prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)}} \prod_{\substack{(j',k') \in \Upsilon_3 \\ (j',k') \neq (j,k)}} \left(Z_{d_{\max}-2d'_j+d_i}^{((j',k'),(j,k))}\right)^{-\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^*} \right) \left(\prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)} \setminus \{i\}} Z_{d_{\max}-2d_j+d_i}^{-\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^*} \right).
\end{aligned}$$

This is precisely how algorithm $\mathcal{B}$ constructs $C'_{3,k}$. Next, consider $C'_{4,k}$. The challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ sets

$$C'_{4,k} = g^{s_k^*} = g^{s \cdot \sum_{i \in C : \rho(k) \notin S_i} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*} = \prod_{i \in C : \rho(k) \notin S_i} \left(Y^{(i,k)}\right)^{\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*},$$

which is how algorithm $\mathcal{B}$ constructs $C'_{4,k}$. Finally, consider C'_5. The challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ sets

$$C'_5 = g^{s \cdot \sum_{i \in \mathcal{N}} a^{d_{\max}-d_i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}}. \tag{4.13}$$

By construction, for all $i \in \mathcal{N}$, the challenger sets $T_i = g^{r_i} = g^{a^{d_{\max}-d_i}+r'_i}$. Thus, we can write

$$\prod_{i \in \mathcal{N}} T_i^{-s} = \prod_{i \in \mathcal{N}} g^{-sa^{d_{\max}-d_i}-sr'_i} = g^{-s \sum_{i \in \mathcal{N}} a^{d_{\max}-d_i}} \prod_{i \in \mathcal{N}} g^{-r'_i s}.$$

Substituting back into Eq. (4.13), this means

$$\begin{aligned}
C'_5 &= g^{s \cdot \sum_{i \in \mathcal{N}} a^{d_{\max}-d_i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} \\
&= g^{s \cdot \sum_{i \in \mathcal{N}} a^{d_{\max}-d_i}} \left(g^{s \cdot \sum_{i \in \mathcal{N}} a^{d_{\max}-d_i}}\right)^{-1} \prod_{i \in \mathcal{N}} g^{-r'_i s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} \\
&= \prod_{i \in \mathcal{N}} Y^{-r'_i} \prod_{i \in C} Q_i^{-\delta_i^{-1}},
\end{aligned}$$

which is how algorithm $\mathcal{B}$ constructs C'_5. Finally, algorithm $\mathcal{B}$ computes

$$\left(\{C_{3,k}, C_{4,k}\}_{k\in[K]}, C_5\right) = \mathsf{Rerand}\left(\{\hat{U}_w\}_{w\in\mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k\in[K]}, C'_5\right),$$

which exactly coincides with the challenger's behavior in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. We conclude that if $T = e(g,g)^{a^{d_{\max}}s}$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(\nu)}$ whereas if $T \xleftarrow{\mathrm{R}} \mathbb{G}_T$, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. Thus, algorithm $\mathcal{B}$ succeeds with the same advantage of $\mathcal{A}$, and the claim follows. □

By construction the adversary's view in $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ is independent of ν. As such, for all adversaries $\mathcal{A}$, the output distributions $\mathsf{Hyb}_{\mathsf{rand}}^{(0)}(\mathcal{A})$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(1)}(\mathcal{A})$ are identically distributed. The claim now follows from Lemmas 4.9 and 4.12. □

Summary. Putting all the pieces together (and invoking the generic compiler from a slotted registered ABE scheme to a standard registered ABE scheme in Appendix C), we obtain the following corollary:

Corollary 4.13 (Bounded Registered ABE from Prime-Order Pairing Groups)**.** *Let λ be a security parameter. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be any (polynomial-size) attribute space, and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies that can be described by a linear secret sharing scheme over $\mathcal{U}$ of size at most K (i.e., each policy is over at most K attributes). Then, under Assumption 4.2, for every polynomial $L = L(\lambda)$, there exists a statically-secure bounded registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and supporting up to L users with the following properties:*

- *The size of the CRS is $L^{1+o(1)} \cdot \mathsf{poly}(\lambda)$.*
- *The size of the auxiliary data maintained by the key curator is $L \cdot |\mathcal{U}_\lambda| \cdot \mathsf{poly}(\lambda, \log L)$*
- *The running time of key-generation is $L \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The running time of registration is $L \cdot \mathsf{poly}(\lambda, |\mathcal{U}_\lambda|, \log L)$.*
- *The size of the master public key and the helper decryption keys are both $|\mathcal{U}_\lambda| \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The size of a ciphertext is $K \cdot \mathsf{poly}(\lambda, \log L)$.*

5 Adaptively-Secure Registered ABE via a Dual System Proof

Construction 4.3 from Section 4 gives an efficient construction of registered ABE where the size of the CRS scales nearly linearly with the bound on the number of users (when instantiated with state-of-the-art progression-free sets) and independently of the size of the attribute universe. In contrast, previous schemes that support monotone Boolean formulas [HLWW23, ZZGQ23] required a CRS whose size scaled quadratically with the number of users and linearly with the size of the attribute universe. The downside is the construction achieves a weaker notion of *static* security (Definition 3.5). In this section, we show that using a dual system approach [Wat09, LW11] similar to the one taken in [HLWW23, ZZGQ23], we can obtain adaptive security with a nearly-linear-sized CRS through the use of progression-free sets. However, similar to previous adaptively-secure constructions, the CRS in this scheme scales linearly with the size of the attribute universe.

5.1 Composite-Order Preliminaries

Similar to the registered ABE construction from [HLWW23], our construction relies on a composite-order pairing group. To incorporate progression-free sets into our construction, we will work over a composite-order pairing group where the modulus N is a product of *four* primes. We recall the definition below and then state the concrete computational assumptions we use in our construction.

Definition 5.1 (Four-Prime Composite-Order Bilinear Group [BGN05])**.** A (symmetric) four-prime composite-order bilinear group generator is an efficient algorithm CompGroupGen that takes as input the security parameter λ and outputs a description $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e)$ of a bilinear group where p_1, p_2, p_3, p_4 are distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order $N = p_1p_2p_3p_4$, g is a generator of $\mathbb{G}$, and $e\colon \mathbb{G}\times\mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map. We require that the group operation in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the pairing operation be efficiently computable.

Notation. Let $\mathbb{G}$ be a cyclic group with order $N = p_1p_2p_3p_4$ and generator g. In the following, we will write $\mathbb{G}_1 = \langle g^{p_2p_3p_4} \rangle$ to denote the subgroup of $\mathbb{G}$ of order p_1. We define $\mathbb{G}_2, \mathbb{G}_3, \mathbb{G}_4$ analogously. By the Chinese Remainder Theorem, if g_1, g_2, g_3, g_4 are generators of $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, \mathbb{G}_4$, respectively, then $g_1g_2g_3g_4 \in \mathbb{G}$ is a generator of $\mathbb{G}$, and moreover, every element $h \in \mathbb{G}$ can be *uniquely* written as $g_1^{x_1}g_2^{x_2}g_3^{x_3}g_4^{x_4}$ where $x_i \in \mathbb{Z}_{p_i}$ for all $i \in \{1,2,3,4\}$. In the following description, we will say $h \in \mathbb{G}$ has a non-trivial component in the $\mathbb{G}_i$ subgroup if $x_i \neq 0$.

Generalized subgroup assumptions. Security of our construction relies on several variants of the subgroup decision assumptions introduced by Lewko and Waters [LW10] for constructing adaptively-secure (hierarchical) identity-based encryption, and subsequently by Lewko et al. [LOS+10] for constructing adaptively-secure attribute-based encryption. The first four assumptions are special cases of the generalized subgroup decision assumption from [BWY11]. The final assumption is a variant of the corresponding assumption from [LW10, HLWW23]. Finally, we state a simple implication (Lemma 5.3) which is similar to one shown in [LW10] that will be useful in our security analysis.

Assumption 5.2 (Subgroup Decision Assumptions)**.** Let CompGroupGen be a four-prime composite-order bilinear group generator. Let $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$, $N = p_1p_2p_3p_4$, $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, and $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_2 \xleftarrow{R} \mathbb{G}_2$, $g_3 \xleftarrow{R} \mathbb{G}_3$, and $g_4 \xleftarrow{R} \mathbb{G}_4$. We now define several pairs of assumptions $\mathcal{D}_0, \mathcal{D}_1$ where each distribution $\mathcal{D}_b = (D, T_b)$ consists of a set of common components D together with a challenge element T_b. We say that such an assumption holds with respect to CompGroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

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

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

$$D = (\mathcal{G}, g_1, g_3, g_4), \qquad T_0 = g_1^r, \qquad T_1 = (g_1g_2)^r.$$

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

$$D = (\mathcal{G}, g_1, g_3, g_4, (g_1g_2)^{s_{12}}, (g_2g_3)^{s_{23}}), \qquad T_0 = (g_1g_3)^r, \qquad T_1 = (g_1g_2g_3)^r.$$

Assumption 5.2c: Sample $s_{12}, s_{24}, r \xleftarrow{R} \mathbb{Z}_N$, and let

$$D = (\mathcal{G}, g_1, g_3, g_4, (g_1g_2)^{s_{12}}, (g_2g_4)^{s_{24}}), \qquad T_0 = (g_1g_4)^r, \qquad T_1 = (g_1g_2g_4)^r.$$

Assumption 5.2d: Sample $s_{12}, s_{23}, s_{24}, r \xleftarrow{R} \mathbb{Z}_N$, and let

$$D = (\mathcal{G}, g_1, g_3, g_4, (g_1g_2)^{s_{12}}, (g_2g_3)^{s_{23}}, (g_2g_4)^{s_{24}}), \qquad T_0 = g^r, \qquad T_1 = (g_1g_3g_4)^r.$$

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

$$D = \left(\mathcal{G}, g_1, g_2, g_3, g_4, g_1^\alpha g_2^{t_1}, g_1^s g_2^{t_2}\right), \qquad T_0 = e(g_1, g_1)^{\alpha s}, \qquad T_1 = e(g, g)^r.$$

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

$$\Pr\left[1 < \gcd(x, N) < N : \begin{array}{c} (\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda), \\ N = p_1p_2p_3p_4\ ,\ \mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e), \\ g_1 \xleftarrow{R} \mathbb{G}_1, g_3 \xleftarrow{R} \mathbb{G}_3, g_4 \xleftarrow{R} \mathbb{G}_4, s_{12}, s_{23} \xleftarrow{R} \mathbb{Z}_N \\ x \leftarrow \mathcal{A}\left(\mathcal{G}, g_1, g_3, g_4, (g_1g_2)^{s_{12}}, (g_2g_3)^{s_{23}}\right), \end{array}\right] = \mathsf{negl}(\lambda).$$

In words, given $\left(\mathcal{G}, g_1, g_3, g_4, (g_1g_2)^{s_{12}}, (g_2g_3)^{s_{23}}\right)$, no efficient adversary can output a non-trivial factor of N.

Proof. This is a 4-prime analogue of [LW10, Lemma 5]. To show this, consider an adversary $\mathcal{A}$ where the probability in Lemma 5.3 is ε for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2d:

1. On input the challenge $(\mathcal{G}, g_1, g_3, g_4, X_1, X_3, X_4, T)$, algorithm $\mathcal{B}$ forwards $(\mathcal{G}, g_1, g_3, g_4, X_1, Y_3)$ to $\mathcal{A}$. By construction, we can write $X_1 = (g_1g_2)^{s_{12}}$, $X_3 = (g_2g_3)^{s_{23}}$, and $X_4 = (g_2g_4)^{s_{24}}$ for some choice of exponents $s_{12}, s_{13}, s_{14} \in \mathbb{Z}_N$.
2. Algorithm $\mathcal{A}$ outputs a value α. If $\gcd(x, z) \in \{1, N\}$, then algorithm $\mathcal{B}$ halts with output $\perp$ Otherwise, algorithm $\mathcal{B}$ constructs β as follows:
   - If $X_i^\alpha = 1$ for some $i \in \{1, 3, 4\}$, then set $\beta = N/\alpha$.
   - If $X_i^{N/\alpha} = 1$ for some $i \in \{1, 3, 4\}$, then set $\beta = \alpha$.
   - Otherwise, output 1 if $T^\alpha = 1$ or $T^{N/\alpha} = 1$ and 0 otherwise.

   If algorithm $\mathcal{B}$ has not halted, then it sets $i \in \{1, 3, 4\}$ to be an index where $g_i^\beta = 1$ and outputs 1 if $e(T, X_i^\beta) = 1$ and 0 otherwise.

First, algorithm $\mathcal{B}$ perfectly simulates an execution of the game in Lemma 5.3 for algorithm $\mathcal{A}$. Thus, with probability at least ε, algorithm $\mathcal{A}$ outputs a non-trivial factor α of N. We consider several possibilities:

- Suppose $\alpha = p_2$ or $\alpha = p_1p_3p_4$. In this case, $X^\alpha, Y^\alpha, Z^\alpha \neq 1$ and similarly $X^{N/\alpha}, Y^{N/\alpha}, Z^{N/\alpha} \neq 1$. Suppose $T = g^r$. In this case, $T^\alpha \neq 1$ and $T^{N/\alpha} \neq 1$, so algorithm $\mathcal{B}$ always outputs 0. Suppose $T = (g_1g_3g_4)^r$. In this case, either $T^\alpha = 1$ or $T^{N/\alpha} = 1$, so algorithm $\mathcal{B}$ always outputs 1.
- Suppose $p_ip_2 \mid \alpha$ for some $i \in \{1, 3, 4\}$. Then, algorithm $\mathcal{B}$ sets $\beta = N/\alpha$. In particular, this means that $p_2 \nmid \beta$. Suppose there exists an index $j \in \{1, 3, 4\}$ where $g_j^\beta = 1$. Note such an index exists since other $\beta = 1$ (and $\alpha = N$ which is a trivial factorization). Since $p_2 \nmid \beta$, this means that $X_j^\beta \in \mathbb{G}_2$. Correspondingly, when $T = g^r$, this means $e(T, X_j^\beta) \neq 1$ and algorithm $\mathcal{B}$ outputs 0. If $T = (g_1g_3g_4)^r$, then $e(T, X_j^\beta) = 1$ and algorithm $\mathcal{B}$ outputs 1.
- Suppose $p_ip_2 \mid N/\alpha$ for some $i \in \{1, 3, 4\}$. Then algorithm $\mathcal{B}$ sets $\beta = \alpha$, which means $p_2 \nmid \beta$. The claim now follows as in the previous case.

We conclude that if $T = g^r$, algorithm $\mathcal{B}$ always outputs 0 and if $T = (g_1g_3g_4)^r$, then algorithm $\mathcal{B}$ always outputs 1. Correspondingly, algorithm $\mathcal{B}$ breaks Assumption 5.2d with advantage ε and the claim follows. □

Progression-free sets in composite-order groups. Security of our composite-order construction will also rely on a new hardness assumption related to progression-free sets. We state our assumption below and show that it holds in the generic bilinear group model in Appendix D (Lemma D.8).

Assumption 5.4 (Progression-Free Indistinguishability)**.** Let CompGroupGen be a four-prime composite-order bilinear group generator. We define the following game between an adversary $\mathcal{A}$ and a challenger. The game is parameterized by a security parameter λ and a bit $\beta \in \{0, 1\}$.

1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ chooses an input length 1^L, a progression-free and double-free set $\mathcal{D} = \{d_i\}_{i \in [L]}$ together with a challenge index $i^* \in [L]$. Define the function $f(i, j) := d_i + d_j$.
2. The challenger samples a group $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It then sets $N = p_1p_2p_3p_4$ and samples $g_i \xleftarrow{\text{R}} \mathbb{G}_i$ for $i \in \{1, 2, 3, 4\}$. The challenger first samples $r \xleftarrow{\text{R}} \mathbb{Z}_N$ and sets $g_{23} = (g_2g_3)^r$. The challenger also samples exponents $a, b, s, \tau \xleftarrow{\text{R}} \mathbb{Z}_N$. For $i \in [L]$, the challenger sets $t_i = a^{d_i}$ and defines the following elements for indices $i, j \in [L]$:

$$A'_i = g_1^{t_i} \quad , \quad U'_i = g_1^{bt_i} \quad , \quad W'_{f(i,j)} = g_1^{bt_it_j} \quad , \quad X = (g_1g_2)^s \quad , \quad \forall i \neq i^* : Y_i = g_1^{sbt_i} \quad , \quad Y_{i^*} = (g_1g_2)^{sbt_{i^*}} .$$

Then the challenger computes the challenge elements

$$T_0 = g_1^{t_{i^*}} g_3^{\tau} \quad \text{and} \quad T_1 = g_1^{t_{i^*}} (g_2 g_3)^{\tau}$$

The challenger then gives the following to $\mathcal{A}$:

$$\left(\mathcal{G}, g_1, g_3, g_4, g_{23}, \{A_i'\}_{i \in [L] \setminus \{i^*\}}\ ,\ \{U_i'\}_{i \in [L]}\ ,\ \{W'_{f(i,j)}\}_{i \neq j \in [L]}\ ,\ X\ ,\ \{Y_i\}_{i \in [L] \setminus \{i^*\}}\ ,\ Y_{i^*}\ ,\ T_\beta\right).$$

3. Adversary $\mathcal{A}$ outputs a bit $\beta' \in \{0, 1\}$, which is the output of the experiment.

We say that the progression-free indistinguishability assumption holds with respect to CompGroupGen if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

5.2 Adaptively-Secure Registered ABE with Progression-Free Sets

In this section, we give our construction of an adaptively-secure (slotted) registered ABE scheme for monotone Boolean formulas with a sub-quadratic CRS by relying on progression-free sets.

Construction 5.5 (Slotted Attribute-Based Registration-Based Encryption)**.** Let CompGroupGen be a four-prime composite-order bilinear group generator. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ be a (polynomial-size) attribute space. Let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of policies that can be described by a *one-use* linear secret sharing scheme over $\mathcal{U}$ (Definition 2.2). We construct a slotted attribute-based registration-based encryption scheme $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with attribute space $\mathcal{U}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L)$: On input the security parameter λ, the size of the attribute space $|\mathcal{U}_\lambda|$, and the number of slots L, the setup algorithm starts by sampling $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, \mathbb{G}_4$ be the subgroups of $\mathbb{G}$ of orders p_1, p_2, p_3, p_4, respectively. The setup algorithm now constructs the following quantities:
  - Let $N = p_1 p_2 p_3 p_4$ and let $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ be the (public) group description.
  - Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set of size L (Theorem 2.5). In the following, we define $f(i, j) \coloneqq d_i + d_j$ and the set $\mathcal{E}$ of all distinct pairwise sums of elements in $\mathcal{D}$:
  $$\mathcal{E} = \{f(i, j) \mid i, j \in [L] : i \neq j\}.$$
  - Sample generators $g_1 \xleftarrow{\text{R}} \mathbb{G}_1$, $g_3 \xleftarrow{\text{R}} \mathbb{G}_3$, $g_4 \xleftarrow{\text{R}} \mathbb{G}_4$ and exponents $\alpha, \beta, a \xleftarrow{\text{R}} \mathbb{Z}_N$. Let $h = g_1^\beta$.
  - For each slot index $i \in [L]$, let $t_i = a^{d_i}$. Then, sample $\delta_i \xleftarrow{\text{R}} \mathbb{Z}_N, \tau_i, \tau_i' \xleftarrow{\text{R}} \mathbb{Z}_N$. Define the slot components as follows:
  $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad P_i = (g_1 g_3)^{\delta_i}.$$
  For each attribute $w \in \mathcal{U}_\lambda$, sample $b_w \xleftarrow{\text{R}} \mathbb{Z}_N$. For each $w \in \mathcal{U}_\lambda$, slot index $i \in [L]$, and cross term index $z \in \mathcal{E}$, define the attribute-specific slot components $U_{i,w}$ and $W_{z,w}$ as follows:
  $$U_{i,w} = g_1^{b_w t_i} \quad , \quad W_{z,w} = g_1^{b_w a^z}.$$
  - Finally, the setup algorithm sets $Z = e(g_1, g_1)^\alpha$ and outputs the common reference string
  $$\mathsf{crs} = \left(\mathcal{G}\ ,\ Z\ ,\ g_1\ ,\ h\ ,\ g_3\ ,\ g_4\ ,\ \{(A_i, B_i, P_i)\}_{i \in [L]}\ ,\ \{U_{i,w}, W_{z,w}\}_{w \in \mathcal{U}_\lambda, i \in [L], z \in \mathcal{E}}\right) \tag{5.1}$$

- $\mathsf{KeyGen}(\mathsf{crs}, i)$: On input the common reference string crs (with components given by Eq. (5.1)) and a slot index $i \in [L]$, the key-generation algorithm samples $r_i \xleftarrow{R} \mathbb{Z}_N$ and computes
$$T_i = g_1^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad R_i = g_3^{r_i}.$$
Then for each $j \neq i$, it computes the cross terms $V_{j,i} = A_j^{r_i}$. Finally, it outputs the public key pk_i and the secret key sk_i defined as follows:
$$\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i}) \quad \text{and} \quad \mathsf{sk}_i = r_i.$$
Note that this particular key-generation algorithm does *not* depend on the set of attributes.

- $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$: On input the common reference string crs (with components given by Eq. (5.1)), a slot index $i \in [L]$, and a purported public key $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$, the key-validation algorithm first affirms that each of the components in pk_i is a valid group element (i.e., an element in $\mathbb{G}$). If so, it then checks
$$e(g_3, T_i) = 1 = e(g_1, R_i) \quad \text{and} \quad e(T_i, P_i) = e(g_1, Q_i) \quad \text{and} \quad e(R_i, P_i) = e(g_3, Q_i).$$
Next, for each $j \neq i$, the algorithm checks that
$$e\big(g_1, V_{j,i}\big) = e(T_i, A_j) \quad \text{and} \quad e\big(g_3, V_{j,i}\big) = e(R_i, A_j).$$
Finally, the algorithm checks that none of the public key components have a component in the $\mathbb{G}_4$ subgroup. Namely for all $j \neq i$:
$$e(g_4, T_i) = e(g_4, Q_i) = e(g_4, R_i) = e(g_4, V_{j,i}) = 1$$
If all checks pass, the key-validation algorithm outputs 1; otherwise, it outputs 0.

- $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$: On input the common reference string crs (with components given by Eq. (5.1)), a collection of L public keys $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ together with their attribute sets $S_i \subseteq \mathcal{U}_\lambda$, the aggregation algorithm starts by computing the attribute-independent public key $\hat{T}$ and the attribute-independent slot key $\hat{V}_i$ for each $i \in [L]$:
$$\hat{T} = \prod_{j \in [L]} T_j \quad , \quad \hat{V}_i = \prod_{j \neq i} V_{i,j}.$$
Next, for each attribute $w \in \mathcal{U}_\lambda$, it computes the attribute-specific public key $\hat{U}_w$ and the attribute-specific slot key $\hat{W}_{i,w}$ for each $i \in [L]$:
$$\hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_{j,w} \quad , \quad \hat{W}_{i,w} = \prod_{j \neq i : w \notin S_j} W_{f(i,j),w}$$
Finally, it outputs the master public key mpk and the slot-specific helper decryption keys hsk_i where
$$\mathsf{mpk} = \big(\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}\big) \quad \text{and} \quad \mathsf{hsk}_i = \big(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}_\lambda}\big).$$

- $\mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{M}, \rho), \mu)$: On input the master public key $\mathsf{mpk} = (\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda})$, a policy $(\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is an injective row-labeling function, and a message $\mu \in \mathbb{G}_T$, the encryption algorithm starts by sampling a secret exponent $s \xleftarrow{R} \mathbb{Z}_N$ and $h_1, h_2 \xleftarrow{R} \mathbb{G}_1$ such that $h = h_1 h_2$. Then, it constructs the ciphertext components as follows:
  - **Message-embedding components:** First, let $C_1 = \mu \cdot Z^s$ and $C_2 = g_1^s$.
  - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ for the linear secret sharing scheme and let $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, for each $k \in [K]$, sample $s_k, \eta_k \xleftarrow{R} \mathbb{Z}_N$, and let $C_{3,k} = h_2^{\mathbf{m}_k^{\mathsf{T}} \mathbf{v}} \hat{U}_{\rho(k)}^{-s_k} g_4^{\eta_k}$ and $C_{4,k} = (g_1 g_4)^{s_k}$, where $\mathbf{m}_k^{\mathsf{T}} \in \mathbb{Z}_N^n$ denotes the k^{th} row of $\mathbf{M}$.
  - **Slot-specific component:** Let $C_5 = (h_1 \hat{T}^{-1})^s$

It then outputs the ciphertext

$$\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big).$$

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input the secret key $\mathsf{sk} = r$, the helper key $\mathsf{hsk} = \big(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}_\lambda}\big)$, where $\mathsf{mpk} = (\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda})$, and the ciphertext $\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big)$ where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is an injective row-labeling function, the decryption algorithm proceeds as follows:
  - If the set of attributes S_i is not authorized by $(\mathbf{M}, \rho)$, then the decryption algorithm outputs $\perp$.
  - Otherwise, let $I = \{k \in [K] : \rho(k) \in S_i\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes $S_i \subseteq \mathcal{U}_\lambda$. Write the elements as $I = \{k_1, \ldots, k_{|I|}\}$.
  - Let $\mathbf{M}_{S_i}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I. Since S_i is authorized, let $\boldsymbol{\omega}_{S_i} \in \mathbb{Z}_N^{|I|}$ be a vector such that $\boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} = \mathbf{e}_1^{\mathsf{T}}$.
  - Then, compute and output

$$\frac{C_1}{e(C_2, B_i)} \cdot \underbrace{e(C_5, A_i) \cdot e(C_2, A_i^r \hat{V}_i)}_{D_{\mathsf{slot}}} \cdot \underbrace{\prod_{1 \le j \le |I|} \Big(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)})\Big)^{\omega_{S_i,j}}}_{D_{\mathsf{attrib}}}. \tag{5.2}$$

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

Correctness. We now show that Construction 5.5 satisfies completeness, correctness, compactness, and incremental aggregation.

Theorem 5.6 (Completeness). *Construction 5.5 is complete.*

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

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

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

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

Finally, since $T_i, Q_i, R_i, T_{i,j}$ do not have non-zero components in the $\mathbb{G}_4$ subgroup, it follows that

$$e(g_4, T_i) = e(g_4, Q_i) = e(g_4, R_i) = e(g_4, V_{j,i}) = 1.$$

We conclude that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$ outputs 1 and completeness holds. □

Theorem 5.7 (Correctness). *Construction 5.5 is correct.*

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

- Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L)$ where
$$\mathsf{crs} = \left(\mathcal{G}, Z, g_1, h, g_3, g_4, \{(A_i, B_i, P_i)\}_{i\in[L]}, \{U_{i,w}, W_{z,w}\}_{w\in\mathcal{U}_\lambda, i\in[L], z\in\mathcal{E}}\right).$$
By construction, the slot components can be written as $A_i = g_1^{t_i} g_3^{\tau_i'}$, $B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i}$, and $P_i = (g_1 g_3)^{\delta_i}$. The attribute components can be written as $U_{i,w} = g_1^{b_w t_i}$ and $W_{z,w} = g_1^{b_w a^z}$.

- Let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. Then, we can write $\mathsf{sk}_i = r_i$ and $\mathsf{pk}_i = \left(T_i, Q_i, R_i, \{V_{j,i}\}_{j\neq i}\right)$ where
$$T_i = g_1^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad R_i = g_3^{r_i} \quad , \quad V_{j,i} = A_j^{r_i} = (g_1 g_3)^{t_j r_i}. \tag{5.3}$$

- Take any set of public keys $\{\mathsf{pk}_j\}_{j\neq i}$ where $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j)$ holds. Since pk_j satisfies the IsValid predicate, we can write $\mathsf{pk}_j = \left(T_j, Q_j, R_j, \{V_{\ell,j}\}_{\ell\in[L]\setminus\{j\}}\right)$. For each $j \in [L]$, let $S_j \subseteq \mathcal{U}_\lambda$ be the attributes associated with pk_j. Let $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$. Then, the master public key mpk and the i^{th} slot-specific helper decryption key hsk_i can be written as follows:
$$\mathsf{mpk} = \left(\mathcal{G}, g_1, h, Z, \hat{T}, \{\hat{U}_w\}_{w\in\mathcal{U}_\lambda}\right) \quad \text{and} \quad \mathsf{hsk}_i = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w\in\mathcal{U}_\lambda}\right),$$
where $\hat{T} = \prod_{j\in[L]} T_j$, $\hat{V}_i = \prod_{j\neq i} V_{i,j}$, and
$$\hat{U}_w = \prod_{j\in[L]:w\notin S_j} U_{j,w} = \prod_{j\in[L]:w\notin S_j} g_1^{b_w t_j} \quad \text{and} \quad \hat{W}_{i,w} = \prod_{j\neq i:w\notin S_j} W_{f(i,j),w} = \prod_{j\neq i:w\notin S_j} g_1^{b_w a^{f(i,j)}}.$$

- Take any message $\mu \in \mathbb{G}_T$ and any policy $(\mathbf{M}, \rho) \in \mathcal{P}_\lambda$ where $\mathbf{M} \in \mathbb{Z}_N^{K\times n}$ and $\rho\colon [K] \to \mathcal{U}_\lambda$ is an injective row-labeling function. Let $\mathsf{ct} \leftarrow \mathsf{Encrypt}(\mathsf{mpk}, P, \mu)$. Then,
$$\mathsf{ct} = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k\in[K]}, C_5\right),$$
where
$$C_1 = \mu \cdot Z^s, \quad C_2 = g_1^s, \quad C_{3,k} = h_2^{\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s_k} g_4^{\eta_k}, \quad C_{4,k} = (g_1 g_4)^{s_k}, \quad C_5 = h_1^s \hat{T}^{-s}.$$

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

- **Policy check:** First, consider $D_{\mathsf{attrib}} = \prod_{1\le j\le|I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)})\right)^{\omega_{S_i,j}}$. First, since $h_2, \hat{U}_{\rho(k_j)} \in \mathbb{G}_1$, we can write
$$e(C_{3,k_j}, A_i) = e\left(h_2^{\mathbf{m}_{k_j}^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k_j)}^{-s_{k_j}} g_4^{\eta_k}, (g_1 g_3)^{t_i}\right) = e(h_2, g_1)^{t_i \mathbf{m}_{k_j}^{\mathsf{T}}\mathbf{v}} \prod_{\ell\in[L]:\rho(k_j)\notin S_\ell} e(g_1, g_1)^{-s_{k_j} t_i t_\ell b_{\rho(k_j)}}$$
$$e\left(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}\right) = \prod_{\ell\neq i:\rho(k_j)\notin S_\ell} e\left(g_1^{s_{k_j}}, W_{f(i,\ell),\rho(k_j)}\right) = \prod_{\ell\neq i:\rho(k_j)\notin S_\ell} e(g_1, g_1)^{a^{f(i,\ell)} s_{k_j} b_{\rho(k_j)}}.$$
By construction, $\rho(k_j) \in S_i$ and by definition, $t_i t_\ell = a^{d_i + d_j} = a^{f(i,\ell)}$, so
$$\prod_{\ell\in[L]:\rho(k_j)\notin S_\ell} e(g_1, g_1)^{-s_{k_j} t_i t_\ell b_{\rho(k_j)}} = \prod_{\ell\neq i:\rho(k_j)\notin S_\ell} e(g_1, g_1)^{-a^{f(i,\ell)} s_{k_j} b_{\rho(k_j)}},$$

and so we can write

$$e(C_{3,k_j}, A_i) e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}) = e(h_2, g_1)^{t_i \mathbf{m}_{k_j}^{\mathsf{T}} \mathbf{v}}.$$

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

$$\begin{aligned} D_{\mathsf{attrib}} = \prod_{1 \le j \le |I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}) \right)^{\omega_{S_i,j}} &= e(h_2, g_1)^{t_i \sum_{1 \le j \le |I|} \omega_{S_i,j} \mathbf{m}_{k_j}^{\mathsf{T}} \mathbf{v}} \\ &= e(h_2, g_1)^{t_i \boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} \mathbf{v}} \\ &= e(h_2, g_1)^{t_i \mathbf{e}_1^{\mathsf{T}} \mathbf{v}} = e(h_2, g_1)^{s t_i}. \end{aligned} \tag{5.4}$$

- **Slot check:** Next, consider the component $D_{\mathsf{slot}} = e(C_5, A_i) e(C_2, A_i^{r_i} \hat{V}_i)$. Since $h_1 \in \mathbb{G}_1$,

$$e(C_5, A_i) = e(h_1^s \hat{T}^{-s}, A_i) = e(h_1^s, A_i) e(\hat{T}^{-s}, A_i) = e(h_1^s, g_1^{t_i} g_3^{\tau_i'}) \prod_{j \in [L]} e(T_j, A_i)^{-s} = e(h_1, g_1)^{s t_i} \prod_{j \in [L]} e(T_j, A_i)^{-s}$$

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

Now, for all $j \in [L]$, $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$. Thus, for all $j \ne i$, we have $e(g_1, V_{i,j}) = e(T_j, A_i)$. Thus, we can now write

$$\begin{aligned} D_{\mathsf{slot}} = e(C_5, A_i) e(C_2, A_i^{r_i} \hat{V}_i) &= \left(e(h_1, g_1)^{s t_i} e(T_i, A_i)^{-s} \prod_{j \ne i} e(T_j, A_i)^{-s} \right) \left(e(g_1, g_1)^{s r_i t_i} \prod_{j \ne i} e(g_1, V_{i,j})^s \right) \\ &= e(h_1, g_1)^{s t_i} e(T_i, A_i)^{-s} e(g_1, g_1)^{s r_i t_i} \\ &= e(h_1, g_1)^{s t_i} e(g_1^{r_i}, (g_1^{t_i} g_3^{\tau_i'})^{-s}) e(g_1, g_1)^{s r_i t_i} \\ &= e(h_1, g_1)^{s t_i}. \end{aligned} \tag{5.5}$$

- **Message reconstruction:** Using the fact that $h = h_1 h_2$, and combining Eqs. (5.4) and (5.5), we have that

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

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

$$e(C_2, B_i) = e(g_1^s, g_1^{\alpha} A_i^{\beta} (g_3 g_4)^{\tau_i}) = e(g_1, g_1)^{\alpha s} e(g_1^s, (g_1^{t_i} g_3^{\tau_i'})^{\beta}) = e(g_1, g_1)^{\alpha s} e(h, g_1)^{s t_i}.$$

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

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

Theorem 5.8 (Compactness). *Construction 5.5 is compact.*

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

Theorem 5.9 (Incremental Aggregation). *Construction 5.5 suppose f-incremental aggregation for $f(L, |\mathcal{U}_\lambda|) = O(L \cdot |\mathcal{U}_\lambda|)$.*

Proof. We construct the AggregateUpdate function as follows:

- AggregateUpdate(crs, st, (pk, S)): On input the common reference string

$$\text{crs} = \left(\mathcal{G}, Z, g_1, h, g_3, g_4, \{(A_i, B_i, P_i)\}_{i\in[L]}, \{U_{i,w}, W_{z,w}\}_{w\in\mathcal{U}_\lambda, i\in[L], z\in\mathcal{E}}\right),$$

a state st (which could be $\perp$), and a public key (pk, S) (or the special symbol $\perp$), the update algorithm proceeds as follows:

  1. If st $= \perp$, then the update algorithm initializes $k = 0$ and $\hat{T}^{(k)} = 1$. Then, for all $i \in [L]$ and $w \in \mathcal{U}_\lambda$, $\hat{V}_i^{(k)} = 1$, $\hat{U}_w^{(k)} = 1$, and $\hat{W}_{i,w}^{(k)} = 1$. Otherwise, the update algorithm parses

$$\text{st} = \left(k,\ \hat{T}^{(k)},\ \{\hat{V}_i^{(k)}\}_{i\in[L]},\ \{\hat{U}_w^{(k)}\}_{w\in\mathcal{U}_\lambda},\ \{\hat{W}_{i,w}^{(k)}\}_{i\in[L], w\in\mathcal{U}_\lambda}\right).$$

  2. If $(\text{pk}, S) = \perp$, then the algorithm outputs

$$\text{mpk} = \left(\mathcal{G}, g, h, Z, \hat{T}^L, \{\hat{U}_w^{(k)}\}_{w\in\mathcal{U}_\lambda}\right) \quad \text{and} \quad \forall i \in [L] : \text{hsk}_i = \left(\text{mpk}, i, S_i, A_i, B_i, \hat{V}_i^{(k)}, \{\hat{W}_{i,w}^{(k)}\}_{w\in\mathcal{U}_\lambda}\right).$$

  3. Otherwise, the update algorithm parses pk $= \left(T_{k+1}, Q_{k+1}, R_{k+1}, \{V_{i,k+1}\}_{i\neq k+1}\right)$ and updates the state as follows:
     – $\hat{T}^{(k+1)} = \hat{T}^{(k)} \cdot T_{k+1}$.
     – For each $i \in [L]$, if $i \neq k+1$ then $\hat{V}_i^{(k+1)} = \hat{V}_i^{(k)} \cdot V_{i,k+1}$. Otherwise, if $i = k+1$, then set $\hat{V}_i^{(k+1)} = \hat{V}_i^{(k)}$.
     – For each $w \in \mathcal{U}_\lambda$, if $w \notin S_{k+1}$, then $\hat{U}_w^{(k+1)} = \hat{U}_w^{(k)} \cdot U_{k+1,w}$. Otherwise, if $w \in S_{k+1}$, then $\hat{U}_w^{(k+1)} = \hat{U}_w^{(k)}$.
     – For each $i \in [L]$ and $w \in \mathcal{U}_\lambda$, if $i \neq k+1$ and $w \notin S_{k+1}$, then $\hat{W}_{i,w}^{(k+1)} = \hat{W}_{i,w}^{(k)} \cdot W_{f(i,k+1),w}$. Otherwise, set $\hat{W}_{i,w}^{(k+1)} = \hat{W}_{i,w}^{(k)}$.
  4. Output the updated state

$$\text{st} = \left(k+1,\ \hat{T}^{(k+1)},\ \{\hat{V}_i^{(k+1)}\}_{i\in[L]},\ \{\hat{U}_w^{(k+1)}\}_{w\in\mathcal{U}_\lambda},\ \{\hat{W}_{i,w}^{(k+1)}\}_{i\in[L], w\in\mathcal{U}_\lambda}\right).$$

To complete the proof, we show that this incremental aggregation procedure implements the same behavior as the standard aggregation procedure. Specifically, we show inductively that for all $k \le L$, the following properties hold for the elements in the AggregateUpdate algorithm:

- $\hat{T}^{(k)} = \prod_{j\in[k]} T_j$.
- For all $i \in [L]$, $\hat{V}_i^{(k)} = \prod_{j\in[k]\setminus\{i\}} V_{i,j}$.
- For all $w \in \mathcal{U}_\lambda$, $\hat{U}_w^{(k)} = \prod_{j\in[k]: w\notin S_j} U_{j,w}$.
- For all $i \in [L]$ and $w \in \mathcal{U}_\lambda$, $\hat{W}_{i,w}^{(k)} = \prod_{j\in[k]\setminus\{i\}: w\notin S_j} W_{f(i,j),w}$.

By construction, all of these properties hold for $k = 0$. Moreover, the inductive step follows by inspection: namely, each of the updates in Step 3 simply multiplies in the next component into the product (if present). When $k = L$, the components $\hat{T}^{(L)}$, $\hat{V}_i^{(L)}$, $\hat{U}_w^{(L)}$, and $\hat{W}_{i,w}^{(L)}$ precisely coincide with the quantities in the Aggregate algorithm. Finally, the intermediate state st always contains $O(L \cdot |\mathcal{U}_\lambda|)$ group elements, which proves the claim. □

Theorem 5.10 (Security). *Suppose Assumption 5.2 and Assumption 5.4 holds with respect to* CompGroupGen. *Then, Construction 5.5 is secure.*

Proof. Similar to the proof of [HLWW23, Theorem 5.9], our proof follows the dual-system methodology [Wat09, LW10]. Some of our description and structure is directed adapted from that of [HLWW23]. Specifically, in the proof, we define two types of ciphertexts: normal ciphertexts (as output by the honest Encrypt algorithm) and "semi-functional ciphertexts." Similarly, there are two types of slots: normal slots (where the slot parameters in the CRS are generated according to the honest Setup algorithm) and "semi-functional slots." The keys that are registered to a semi-functional slot can be used to decrypt normal ciphertexts and the keys registered to a normal slot can be used to decrypt semi-functional ciphertexts. However, a key registered to a semi-functional slot is *not* able to decrypt a semi-functional ciphertext. The proof leverages a hybrid argument where we iteratively replace the challenge ciphertext as well as the components associated with each slot with semi-functional analogs. In the final hybrid experiment, the slots parameters and the challenge ciphertext are semi-functional. In this setting, we show it is computationally infeasible for the adversary to win the semantic security game. Before describing the hybrid argument, we give a high-level description of the semi-functional ciphertexts and the semi-functional slot components in our construction. We follow with a description of the main set of hybrid experiments.

- **Semi-functional ciphertext:** Semi-functional ciphertexts contain an additional component in the $\mathbb{G}_2$ subgroup. Specifically, suppose $\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big) \leftarrow \mathsf{Encrypt}(\mathsf{crs}, \mathsf{mpk}, \mathsf{id}, \mu)$. Then, a semi-functional ciphertext has the following structure:
$$\mathsf{ct}' = \Big((\mathbf{M}, \rho)\,,\, C_1\,,\, C_2 g_2^{\zeta_2}\,,\, \big\{C_{3,k} g_2^{\zeta_{3,k}}\,,\, C_{4,k}\big\}_{k \in [K]}\,,\, C_5 g_2^{\zeta_5}\Big)$$
for some choice of exponents $\zeta_2, \{\zeta_{3,k}\}, \zeta_5 \in \mathbb{Z}_N$. The exact definition of ζ_2, $\zeta_{3,k}$, ζ_5 in terms of other scheme parameters is complex, so we defer their exact specification to the description of the hybrid experiments. Here, our goal is to illustrate the *general* structure of the components of the semi-functional ciphertext.
- **Semi-functional slot:** The slot components (A_i, B_i, P_i) for a semi-functional slot at index $i \in [L]$ are generated like the normal slot components, except we also introduce $\mathbb{G}_2$ components into B_i and P_i. Specifically, we construct the semi-functional slot components as follows:
$$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} (g_2 g_3 g_4)^{\tau_i} \quad , \quad P_i = (g_1 g_2 g_3)^{\delta_i}.$$

Outer hybrids. We start by defining our primary sequence of hybrid experiments. The outer sequence is intended to convey the general structure of the argument, and arguing indistinguishability between specific pairs of adjacent distributions will require an additional sequence of hybrid experiments (which we defer to the subsequent sections). Each of our hybrids is defined with respect to an (implicit) security parameter λ, a bit $\nu \in \{0, 1\}$, and an adversary $\mathcal{A}$.

- $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$: This is the real security game where the challenger encrypts message μ_ν^*. We recall the main steps here:
  - **Setup phase:** In the setup phase, the adversary $\mathcal{A}$ sends the number of slots 1^L to the challenger. The challenger then samples the common reference string crs according to the specification of the real setup algorithm:
    - The challenger initializes a counter $\mathsf{ctr} \leftarrow 0$ and an (empty) dictionary Dict to keep track of the key-generation queries.
    - The challenger samples $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $N = p_1 p_2 p_3 p_4$ and $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ be the group description.
    - Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set. We define $f(i, j) := d_i + d_j$ and the set $\mathcal{E} := \{f(i, j) \mid i, j \in [L] : i \neq j\}$.
    - The challenger samples generators $g_1 \xleftarrow{\mathrm{R}} \mathbb{G}_1$, $g_3 \xleftarrow{\mathrm{R}} \mathbb{G}_3$, $g_4 \xleftarrow{\mathrm{R}} \mathbb{G}_4$ as well as exponents $\alpha, \beta, a \xleftarrow{\mathrm{R}} \mathbb{Z}_N$, and sets $h = g_1^{\beta}$.
    - For each slot $i \in [L]$, the challenger samples $\delta_i, \tau_i, \tau_i' \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and sets $t_i = a^{d_i}$. Then, the challenger constructs the slot components as follows:
    $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} (g_3 g_4)^{\tau_i} \quad , \quad P_i = (g_1 g_3)^{\delta_i}.$$
    For each attribute $w \in \mathcal{U}_\lambda$, the challenger samples $b_w \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. Then, for each $w \in \mathcal{U}_\lambda$, slot index $i \in [L]$, and cross term index $z \in \mathcal{E}$, the challenger constructs the attribute-specific slot components $U_{i,w}$ and $W_{z,w}$ as follows:
    $$U_{i,w} = g_1^{b_w t_i} \quad , \quad W_{z,w} = g_1^{b_w a^z}.$$

  - Finally, the challenger computes $Z = e(g_1, g_1)^\alpha$ and gives algorithm $\mathcal{A}$ the common reference string

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, g_4\,,\, \{(A_i, B_i, P_i)\}_{i\in[L]}\,,\, \{U_{i,w}, W_{z,w}\}_{w\in\mathcal{U}_\lambda, i\in[L], z\in\mathcal{E}}\right). \tag{5.6}$$

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

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

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

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

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

  The challenger then constructs the challenge ciphertext by sampling a secret exponent $s \xleftarrow{R} \mathbb{Z}_N$ and $h_1, h_2 \xleftarrow{R} \mathbb{G}_1$ such that $h = h_1 h_2$. It then constructs the challenge ciphertext components as follows:
  - **Message-embedding components:** First, it sets let $C_1 = \mu_v^* \cdot Z^s$ and $C_2 = g_1^s$.
  - **Attribute-specific components:** The challenger samples $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ for the linear secret sharing scheme and let $\mathbf{v} = [s, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, for each $k \in [K]$, it samples $s_k, \eta_k \xleftarrow{R} \mathbb{Z}_N$ and sets $C_{3,k} = h_2^{\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s_k} g_4^{\eta_k}$ and $C_{4,k} = (g_1 g_4)^{s_k}$, where $\mathbf{m}_k^{\mathsf{T}}$ denotes the k^{th} row of $\mathbf{M}$.
  - **Slot-specific component:** Let $C_5 = (h_1 \hat{T}^{-1})^s$.

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

$$\mathsf{ct}^* = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k\in[K]}, C_5\right).$$

- **Output phase:** At the end of the game, the adversary outputs a bit $v' \in \{0, 1\}$, which is also the output of the experiment.

• $\mathsf{Hyb}_1^{(v)}$: Same as $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$ except for the following (primarily syntactic) changes:
  - **Setup phase:** The challenger samples $\beta_1, \beta_2 \xleftarrow{R} \mathbb{Z}_N$ and sets $\beta = \beta_1 + \beta_2$. It sets $h = g_1^\beta$ as in $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$. In addition, instead of sampling the secret exponent s during the challenge phase, the challenger samples $s \xleftarrow{R} \mathbb{Z}_N$ during the setup phase and sets $P_i = (g_1^s g_3)^{\delta_i}$ for all $i \in [L]$.

- **Challenge phase:** When simulating the challenge ciphertext, the challenger sets $h_1 = g_1^{\beta_1}$ and $h_2 = g_1^{\beta_2}$, where $\beta_1, \beta_2 \in \mathbb{Z}_N$ are the exponents sampled during the setup phase. Then it constructs the challenge ciphertext components as follows:
  - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^\intercal$. Then, for each $k \in [K]$, sample $s_k, \eta_k \xleftarrow{R} \mathbb{Z}_N$ and set
$$C_{4,k} = (g_1 g_4)^{s_k} \quad , \quad C_{3,k} = (g_1^s)^{\beta_2 \mathbf{m}_k^\intercal \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k}.$$
  - **Slot-specific component:** Set
$$C_5 = (g_1^s)^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right).$$

- $\mathsf{Hyb}_2^{(\nu)}$: Same as $\mathsf{Hyb}_1^{(\nu)}$, except the challenge ciphertext is replaced by a semi-functional ciphertext. Simultaneously, the challenger add a $\mathbb{G}_2$ component into the P_i component. Namely, during the setup phase, the challenger constructs P_i as follows for each $i \in [L]$ as $P_i = ((g_1 g_2)^s g_3)^{\delta_i}$. Then, in the challenge phase, after the adversary has chosen its attribute sets S_i and corresponding public keys $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ for each slot $i \in [L]$, the challenger constructs the challenge ciphertext components as follows:
  - **Message-embedding components:** Let $C_1 = \mu_\nu^* \cdot Z^s$ and $C_2 = (g_1 g_2)^s$.
  - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^\intercal$. Then, for each $k \in [K]$, sample $s_k, \eta_k \xleftarrow{R} \mathbb{Z}_N$ and set
$$C_{4,k} = (g_1 g_4)^{s_k} \quad , \quad C_{3,k} = ((g_1 g_2)^s)^{\beta_2 \mathbf{m}_k^\intercal \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k}.$$
  - **Slot-specific component:** Set
$$C_5 = ((g_1 g_2)^s)^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right).$$

- $\mathsf{Hyb}_{3,\ell}^{(\nu)}$ for each $\ell \in [L]$: Same as $\mathsf{Hyb}_2^{(\nu)}$, except we change the first ℓ slots to be semi-functional. Specifically, during the setup phase, for $i \le \ell$, the challenger samples the slot components A_i, B_i, and P_i as follows:
$$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} (g_2 g_3 g_4)^{\tau_i} \quad , \quad P_\ell = ((g_1 g_2)^s g_3)^{\delta_\ell}.$$
The remaining slot components A_i, B_i, and P_i for $i > \ell$ are generated as in $\mathsf{Hyb}_2^{(\nu)}$.

- $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$: Same as $\mathsf{Hyb}_{3,L}^{(\nu)}$ except when constructing the challenge ciphertext, the challenger samples $C_1 \xleftarrow{R} \mathbb{G}_T$. Importantly, this distribution is *independent* of the message.

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

Proof structure. The analysis from the initial experiment $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$ to experiment $\mathsf{Hyb}_{3,0}^{(\nu)}$ as well as the final transition from $\mathsf{Hyb}_{3,L}^{(\nu)}$ to $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ follow very similarly to the hybrid experiments in the proof of [HLWW23, Theorem 5.9]. As such, we defer their formal analysis (adapted to the use of progression-free sets) to Appendix A. The transition between $\mathsf{Hyb}_{3,0}^{(\nu)}$ to $\mathsf{Hyb}_{3,L}^{(\nu)}$ is where we will critically rely on the progression-free indistinguishability assumption (Assumption 5.4), and we include its proof in the following section (Section 5.3). We start by stating the lemmas asserting indistinguishability of each pair of adjacent hybrid experiments.

Lemma 5.11. *Suppose Assumption 5.2a holds with respect to* CompGroupGen. *Then, for all efficient adversaries $\mathcal{A}$ and $\nu \in \{0, 1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*
$$\left| \Pr[\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(\nu)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

We give the proof of Lemma 5.11 in Appendix A.1.

Lemma 5.12. *Suppose Assumption 5.2a holds with respect to* CompGroupGen*. Then, for all efficient adversaries $\mathcal{A}$ and $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

We give the proof of Lemma 5.12 in Appendix A.2.

Lemma 5.13. *For all efficient adversaries $\mathcal{A}$, $v \in \{0,1\}$, and $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_2^{(v)}(\mathcal{A}) = 1]\right| = \left|\Pr[\mathsf{Hyb}_{3,0}^{(v)}(\mathcal{A}) = 1]\right|.$$

Proof. The slot components B_i in $\mathsf{Hyb}_{3,0}^{(v)}$ are distributed identically to those in $\mathsf{Hyb}_2^{(v)}$, so the experiments are identical. □

Lemma 5.14. *Suppose Assumption 5.4 and Assumption 5.2 hold with respect to* CompGroupGen*. Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

We give the proof of Lemma 5.14 in Section 5.3.

Lemma 5.15. *Suppose Assumption 5.2e holds with respect to* CompGroupGen*. Then, for efficient adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_{3,L}^{(v)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(v)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

We give the proof of Lemma 5.15 in Appendix A.3.

Completing the proof. To complete the proof of Theorem 5.10, we observe that the distribution $\mathsf{Hyb}_{\mathsf{rand}}^{(v)}$ is *independent* of the bit $v \in \{0,1\}$. Thus, for all adversaries $\mathcal{A}$, it holds that $\mathsf{Hyb}_{\mathsf{rand}}^{(0)}(\mathcal{A}) \equiv \mathsf{Hyb}_{\mathsf{rand}}^{(1)}(\mathcal{A})$. Combining Lemmas 5.11 to 5.15, security follows via a hybrid argument. □

Summary. Putting all the pieces together (and invoking the generic compiler from a slotted registered ABE scheme to a standard registered ABE scheme in Appendix C), we obtain the following corollary:

Corollary 5.16 (Bounded Registered ABE from Composite-Order Pairing Groups)**.** *Let λ be a security parameter. Let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ be any (polynomial-size) attribute space, and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$ be a set of policies that can be described by a one-use linear secret sharing scheme over $\mathcal{U}$ of size at most K (i.e., each policy is over at most K attributes). Then, under Assumptions 5.2 and 5.4, for every polynomial $L = L(\lambda)$, there exists a bounded registered ABE scheme with attribute universe $\mathcal{U}$, policy space $\mathcal{P}$, and supporting up to L users with the following properties:*

- *The size of the CRS is $L^{1+o(1)} \cdot |\mathcal{U}_\lambda| \cdot \mathsf{poly}(\lambda)$.*
- *The size of the auxiliary data maintained by the key curator is $L \cdot |\mathcal{U}_\lambda| \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The running time of key-generation and registration is $L \cdot |\mathcal{U}_\lambda| \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The size of the master public key and the helper decryption keys are both $|\mathcal{U}_\lambda| \cdot \mathsf{poly}(\lambda, \log L)$.*
- *The size of a ciphertext is $K \cdot \mathsf{poly}(\lambda, \log L)$.*

5.3 Proof of Lemma 5.14

In this section, we give the formal proof of Lemma 5.14. The difference between experiments $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$ and $\mathsf{Hyb}_{3,\ell}^{(\nu)}$ is the distribution of the parameters for slot ℓ. In experiment $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$, the parameters for slot ℓ are normal while in experiment $\mathsf{Hyb}_{3,\ell}^{(\nu)}$, they are semi-functional.

Per-row randomization. Before giving the proof, we first remark on a key difference between Construction 5.5 and the previous dual-system registered ABE scheme from [HLWW23, Construction 5.4]. Construction 5.5 introduces *per-row randomization* in the ciphertexts. In more detail, the attribute-specific components $C_{3,k}$ and $C_{4,k}$ of the ciphertext in Construction 5.5 are each associated with an independent randomization factor $s_k \xleftarrow{\text{R}} \mathbb{Z}_N$. In contrast, in the previous construction of [HLWW23], all of the attribute-specific components shared a single randomizing exponent (in that case, there was also no need for $C_{4,k}$). As we discuss below, having independent randomizing components for each row of the policy matrix is essential for proving security from Assumption 5.4:

- In the progression-free indistinguishability assumption (Assumption 5.4), the Y_i terms for $i \neq j$ all live in the $\mathbb{G}_1$ subgroup. Only the Y_{i^*} terms for the challenge index i^* contains a component in the $\mathbb{G}_2$ subgroup. Note that this is inherent because if any Y_i for $i \neq i^*$ contained a non-zero component in the $\mathbb{G}_2$ subgroup, then the assumption is trivially broken: namely, the adversary can decide whether the challenge element T_β contains a $\mathbb{G}_2$ component or not by by checking whether $e(Y_i, T) = e\big(W'_{f(i,i^*)}, X\big)$.
- When giving a reduction to the progression-free indistinguishability assumption, the reduction algorithm (see Lemma 5.19) will use the Y_i terms from the challenge to simulate the attribute-specific components $C_{3,k}$ in the challenge ciphertext. However, to leverage a similar statistical argument as that used in [HLWW23] (see Lemma A.14), each component $C_{3,k}$ where $\rho(k) \notin S_{i^*}$ must be independently uniform in the $\mathbb{G}_2$ subgroup (here $\rho \colon [K] \to \mathcal{U}_\lambda$ is the row-labeling function associated with the challenge policy and S_{i^*} is the set of attributes associated with slot i^*). If we take the [HLWW23] approach where the same exponent $s \in \mathbb{Z}_N$ is shared across all of the attribute-specific ciphertext components $C_{3,k}$, then the ciphertext components will be correlated in the $\mathbb{G}_2$ subgroup. This in turn breaks the final information-theoretic analysis (Lemma A.14). In contrast, with per-row randomization, we are able to use $Y_{i^*}^{s_k}$ to introduce a *random* $\mathbb{G}_2$ component into each $C_{3,k}$ where $\rho(k) \notin S_{i^*}$. This in turn allows us to (eventually) leverage a similar statistical argument as in the previous proof from [HLWW23].

We now proceed with the formal argument. We start by defining an additional sequence of (simpler) hybrid experiments:

- $\mathsf{iHyb}_{\ell,0}^{(\nu)}$: Same as $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$ except the challenger introduces a component in the $\mathbb{G}_2$ subgroup in the challenge ciphertext components $C_{4,k}$ whenever $\rho(k) \notin S_\ell$. Specifically, the challenger constructs $C_{4,k}$ for $k \in [K]$ as follows:
  - If $\rho(k) \notin S_\ell$, it sets $C_{4,k} = ((g_1 g_2)^s g_4)^{s_k}$.
  - If $\rho(k) \in S_\ell$, it sets $C_{4,k} = (g_1 g_4)^{s_k}$ as in $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$.
- $\mathsf{iHyb}_{\ell,1}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,0}^{(\nu)}$, except the challenger changes how it constructs $C_{3,k}$ in the challenge ciphertext:
$$C_{3,k} = \begin{cases} (g_1 g_2)^{s(\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in [L] \setminus \{\ell\} : \rho(k) \notin S_i} t_i} g_4^{\eta_k} & \text{if } \rho(k) \notin S_\ell \\ (g_1 g_2)^{s \beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} g_1^{-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k} & \text{if } \rho(k) \in S_\ell. \end{cases}$$
- $\mathsf{iHyb}_{\ell,2}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ except the challenger sets $A_\ell = g_1^{t_\ell} (g_2 g_3)^{\tau'_\ell}$ in the setup phase.
- $\mathsf{iHyb}_{\ell,3}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,2}^{(\nu)}$ except the challenger sets
$$C_{3,k} = ((g_1 g_2)^s)^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k}.$$
- $\mathsf{iHyb}_{\ell,4}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,3}^{(\nu)}$ except the challenger sets $B_\ell = g_1^{\alpha} A_\ell^{\beta} (g_2 g_3 g_4)^{\tau_\ell}$ in the setup phase.

Hybrid	A_ℓ	B_ℓ	$C_{3,k}$ for $\rho(k) \notin S_\ell$	Justification	Analysis
$\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$	$g_1^{t_\ell} g_3^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} (g_1g_4)^{-s_k b_{\rho(k)} \sum_{i \in C_\ell \cup \{\ell\}} t_i} g_4^{\eta_k}$		
$\mathsf{iHyb}_{\ell,0}^{(\nu)}$	$g_1^{t_\ell} g_3^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i \in C_\ell \cup \{\ell\}} t_i} g_4^{\eta_k}$	Assumption 5.2c	Lemma 5.17
$\mathsf{iHyb}_{\ell,1}^{(\nu)}$	$g_1^{t_\ell} g_3^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s(\beta_2 \mathbf{m}_k^\top \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in C_\ell} t_i} g_4^{\eta_k}$	Statistical	Lemma 5.18
$\mathsf{iHyb}_{\ell,2}^{(\nu)}$	$g_1^{t_\ell} (g_2 g_3)^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s(\beta_2 \mathbf{m}_k^\top \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in C_\ell} t_i} g_4^{\eta_k}$	Assumption 5.4	Lemma 5.19
$\mathsf{iHyb}_{\ell,3}^{(\nu)}$	$g_1^{t_\ell} (g_2 g_3)^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i \in C_\ell \cup \{\ell\}} t_i} g_4^{\eta_k}$	Statistical	Lemma 5.20
$\mathsf{iHyb}_{\ell,4}^{(\nu)}$	$g_1^{t_\ell} (g_2 g_3)^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i \in C_\ell \cup \{\ell\}} t_i} g_4^{\eta_k}$	Assumption 5.2	Appendix A.4
$\mathsf{iHyb}_{\ell,5}^{(\nu)}$	$g_1^{t_\ell} (g_2 g_3)^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s(\beta_2 \mathbf{m}_k^\top \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in C_\ell} t_i} g_4^{\eta_k}$	Statistical	Lemma 5.22
$\mathsf{iHyb}_{\ell,6}^{(\nu)}$	$g_1^{t_\ell} g_3^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s(\beta_2 \mathbf{m}_k^\top \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in C_\ell} t_i} g_4^{\eta_k}$	Assumption 5.4	Lemma 5.23
$\mathsf{iHyb}_{\ell,7}^{(\nu)}$	$g_1^{t_\ell} g_3^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i \in C_\ell \cup \{\ell\}} t_i} g_4^{\eta_k}$	Statistical	Lemma 5.24
$\mathsf{Hyb}_{3,\ell}^{(\nu)}$	$g_1^{t_\ell} g_3^{\tau'_\ell}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} (g_1g_4)^{-s_k b_{\rho(k)} \sum_{i \in C_\ell \cup \{\ell\}} t_i} g_4^{\eta_k}$	Assumption 5.2c	Lemma 5.25

Table 2: Structure of the slot parameters A_ℓ, B_ℓ and the challenge ciphertext components $C_{3,k}$ for $\rho(k) \notin S_\ell$ in the intermediate hybrid experiments in the proof of Lemma 5.14. We write C_ℓ to denote the set $C_\ell := \{i \in [L] \backslash \{\ell\} : \rho(k) \notin S_\ell\}$. For each pair of adjacent hybrids, we indicate whether they are statistically indistinguishable or computationally indistinguishable.

- $\mathsf{iHyb}_{\ell,5}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,4}^{(\nu)}$ except the challenger samples $C_{3,k}$ as in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$

$$C_{3,k} = \begin{cases} (g_1g_2)^{s(\beta_2 \mathbf{m}_k^\top \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in [L] \backslash \{\ell\} : \rho(k) \notin S_i} t_i} g_4^{\eta_k} & \text{if } \rho(k) \notin S_\ell \\ (g_1g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} g_1^{-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k} & \text{if } \rho(k) \in S_\ell. \end{cases}$$

- $\mathsf{iHyb}_{\ell,6}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,5}^{(\nu)}$ except the challenger sets $A_\ell = g_1^{t_\ell} g_3^{\tau'_\ell}$ in the setup phase.
- $\mathsf{iHyb}_{\ell,7}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,6}^{(\nu)}$ except the challenger sets

$$C_{3,k} = ((g_1g_2)^s)^{\beta_2 \mathbf{m}_k^\top \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k}.$$

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

Lemma 5.17. *Suppose Assumption 5.2c holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $\nu \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{Hyb}_{3,\ell-1}^{(\nu)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{\ell,0}^{(\nu)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

Proof. The only difference between these two hybrids is how some of the challenge ciphertext components $C_{4,k}$ are sampled. Suppose that there exists an efficient adversary $\mathcal{A}$ that can distinguish these two experiments with non-negligible probability ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2c with the same advantage:

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_3, g_4, X, Y, T)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_3 \in \mathbb{G}_3$, $g_4 \in \mathbb{G}_4$, $X = (g_1g_2)^{s_{12}}$, $Y = (g_2g_4)^{s_{24}}$ for some $s_{12}, s_{24} \xleftarrow{R} \mathbb{Z}_N$, and either $T = (g_1g_4)^t$ or $T = (g_1g_2g_4)^t$ for some $t \xleftarrow{R} \mathbb{Z}_N$. The components that depend on the challenge elements X, Y, T are colored for clarity.
2. Algorithm $\mathcal{B}$ starts by running algorithm $\mathcal{A}$ to obtain the number of slots 1^L. Algorithm $\mathcal{B}$ then samples $\alpha, \beta_1, \beta_2, a \xleftarrow{R} \mathbb{Z}_N$ and sets $Z = e(g_1, g_1)^\alpha$, $\beta = \beta_1 + \beta_2$, and $h = g_1^\beta$.

3. Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set of size L. As in Construction 5.5, we write $f(i,j) \coloneqq d_i + d_j$ and $\mathcal{E} = \{f(i,j) \mid i,j \in [L] : i \neq j\}$.

4. For each slot $i \in [L]$, algorithm $\mathcal{B}$ samples $\delta_i, \tau_i, \tau_i' \xleftarrow{\text{R}} \mathbb{Z}_N$, and sets $t_i = a^{d_i}$. It then constructs the slot parameters as follows:

   - For $i < \ell$, algorithm $\mathcal{B}$ sets
   $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} Y^{\tau_i} g_3^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i}.$$
   - For $i \geq \ell$, algorithm $\mathcal{B}$ sets
   $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} (g_3 g_4)^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i}.$$

   Then, for each attribute $w \in \mathcal{U}_\lambda$, algorithm $\mathcal{B}$ samples $b_w \xleftarrow{\text{R}} \mathbb{Z}_N$. For each $w \in \mathcal{U}_\lambda$, slot index $i \in [L]$, and cross term index $z \in \mathcal{E}$, algorithm $\mathcal{B}$ constructs the attribute-specific slot components $U_{i,w}$ and $W_{z,w}$ as in $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$:
   $$U_{i,w} = g_1^{b_w t_i} \quad , \quad W_{z,w} = g_1^{b_w a^z}.$$
   Algorithm $\mathcal{B}$ gives the common reference string
   $$\mathsf{crs} = \big(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, g_4\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{z,w}\}_{w \in \mathcal{U}_\lambda, i \in [L], z \in \mathcal{E}}\big)$$
   to the adversary $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary Dict to keep track of the key-generation queries.

5. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,0}^{(\nu)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{\text{R}} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, $Q_i = P_i^{r_i}$, $R_i = g_3^{r_i}$, and $V_{j,i} = A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary Dict. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \leq i \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{Dict}[i]$ and replies to $\mathcal{A}$ with sk'.

6. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy $P^* = (\mathbf{M}, \rho)$, messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{Hyb}_{2,\ell-1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell}^{(\nu)}$:

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

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

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

   - **Message-embedding components:** Set $C_1 = \mu_\nu^* \cdot e(g_1, X)^{\alpha}$ and $C_2 = X$.
   - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$. For each $k \in [K]$, sample $s_k, \eta_k \xleftarrow{\text{R}} \mathbb{Z}_N$. If $\rho(k) \in S_\ell$, set $C_{4,k} = (g_1 g_4)^{s_k}$. Otherwise, set $C_{4,k} = T^{s_k}$. Finally, algorithm $\mathcal{B}$ sets
   $$C_{3,k} = X^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k}$$
   - **Slot-specific component:** Set
   $$C_5 = X^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

Algorithm $\mathcal{B}$ gives the challenge ciphertext to $\mathcal{A}$:

$$\mathsf{ct}^* = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big).$$

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

In the reduction, the exponent $s_{12} \xleftarrow{\text{R}} \mathbb{Z}_N$ plays the role of $s \xleftarrow{\text{R}} \mathbb{Z}_N$ in $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,0}^{(\nu)}$. Note that in the reduction,

$$C_1 = \mu_b^* \cdot e(g_1, (g_1 g_2)^{s_{12}})^\alpha = \mu_b^* \cdot e(g_1, g_1)^{\alpha s_{12}} = Z^{s_{12}},$$

which matches the distribution in $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,0}^{(\nu)}$. Similarly, the components $C_{3,k}$ and C_5 are constructed exactly as they would be in $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,0}^{(\nu)}$. Next, consider the distribution of B_i for $i < \ell$. As long as s_{24} is coprime to $p_2 p_4$ (which holds with overwhelming probability over the choice of $s_{24} \xleftarrow{\text{R}} \mathbb{Z}_N$), then the distributions

$$\big\{Y^{\tau_i} = (g_2 g_4)^{s_{24}\tau_i} : \tau_i \xleftarrow{\text{R}} \mathbb{Z}_N\big\} \quad \text{and} \quad \big\{(g_2 g_4)^{\tau_i} : \tau_i \xleftarrow{\text{R}} \mathbb{Z}_N\big\}$$

are identical. Finally, consider the distribution of $C_{4,k}$ when $\rho(k) \notin S_\ell$:

- If $T = (g_1 g_4)^r$ and r is coprime to $p_1 p_4$ (which holds with overwhelming probability over the choice of $r \xleftarrow{\text{R}} \mathbb{Z}_N$), then $T^{s'_k}$ is identically distributed to $(g_1 g_4)^{s_k}$ for uniform s_k. This matches the distribution of $C_{4,k}$ in $\mathsf{Hyb}_{3,\ell-1}^{(\nu)}$
- If $T = (g_1 g_2 g_4)^r$ and r, s are coprime to $p_1 p_2 p_4$ (which holds with overwhelming probability over the choice of $r, s \xleftarrow{\text{R}} \mathbb{Z}_N$), then T^{s_k} is identically distributed to $((g_1 g_2)^s g_4)^{s_k}$ for uniform $s_k \xleftarrow{\text{R}} \mathbb{Z}_N$. This matches the distribution of $C_{4,k}$ in $\mathsf{iHyb}_{\ell,0}^{(\nu)}$.

Thus, we conclude that the distinguishing advantage of algorithm $\mathcal{B}$ is negligibly smaller than the advantage of $\mathcal{A}$. The claim holds. ☐

Lemma 5.18. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $\nu \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. We argue that these two experiments are statistically indistinguishable. The only difference in these two experiments is the distribution of $C_{3,k}$. In the following, we write $C_{3,k}^{(0)}$ to denote the component $C_{3,k}$ computed according to the specification in $\mathsf{iHyb}_{\ell,0}^{(\nu)}$ and $C_{3,k}^{(1)}$ to denote the component computed according to the specification of $\mathsf{iHyb}_{\ell,1}^{(\nu)}$. For each $k \in [K]$, we consider two possible cases:

- Suppose $\rho(k) \in S_\ell$. Then, by definition,

$$\begin{aligned} C_{3,k}^{(0)} &= (g_1 g_2)^{s\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k} = (g_1 g_2)^{s\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} (g_1 g_4)^{-s_k b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k} \\ C_{3,k}^{(1)} &= (g_1 g_2)^{s\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} g_1^{-s_k b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k}. \end{aligned}$$

  The only difference between $C_{3,k}^{(0)}$ and $C_{3,k}^{(1)}$ is the extra $g_4^{-s_k b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i}$ term in $C_{3,k}^{(0)}$. However, since the challenger in both experiments sample $\eta_k \xleftarrow{\text{R}} \mathbb{Z}_N$, and moreover the only terms that depend on η_k is $C_{3,k}$ in both experiments, we conclude that the distribution of $C_{3,k}^{(0)}$ and $C_{3,k}^{(1)}$ in the $\mathbb{G}_4$ subgroup is uniform and independent. We conclude that $C_{3,k}^{(0)}$ and $C_{3,k}^{(1)}$ are identically distributed in this case.

- Suppose $\rho(k) \notin S_\ell$. Then, by definition

$$\begin{aligned} C_{3,k}^{(0)} &= (g_1 g_2)^{s\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k} = (g_1 g_2)^{s\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k} \\ C_{3,k}^{(1)} &= (g_1 g_2)^{s(\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in [L] \setminus \{\ell\}: \rho(k) \notin S_i} t_i} g_4^{\eta_k}. \end{aligned} \tag{5.7}$$

  To show that these two distributions are statistically indistinguishable, consider the following "alternative" sampling procedure for $b_{\rho(k)}$ and η'_k in $\mathsf{iHyb}_{\ell,0}^{(\nu)}$:

- Let $\xi^{(b)} \in \mathbb{Z}_N$ be the unique value where $\xi^{(b)} = 1 \bmod p_1 p_3 p_4$ and $\xi^{(b)} = t_\ell \left(\sum_{i \in [L] : \rho(k) \notin S_i} t_i\right)^{-1} \bmod p_2$. The challenger samples $b'_{\rho(k)} \xleftarrow{\text{R}} \mathbb{Z}_N$ and sets $b_{\rho(k)} = b'_{\rho(k)} \cdot \xi^{(b)}$. In the following analysis, we will argue that $\xi^{(b)}$ is well-defined with overwhelming probability (i.e., $\sum_{i \in [L] : \rho(k) \notin S_i} t_i$ is invertible modulo p_2 with overwhelming probability).
- Let $\xi^{(\eta)} \in \mathbb{Z}_N$ be the unique value where $\xi^{(\eta)} = 0 \bmod p_1 p_2 p_3$ and $\xi^{(\eta)} = s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i \bmod p_2$. The challenger samples $\eta'_k \xleftarrow{\text{R}} \mathbb{Z}_N$ and sets $\eta_k = \eta'_k + \xi^{(\eta)}$.

We now argue that if $b'_{\rho(k)}$ and η'_k are uniform over $\mathbb{Z}_N$, then the same holds for $b_{\rho(k)}$ and η_k. In other words, the alternative sampling procedure exactly coincides with the actual sampling procedure in $\mathsf{iHyb}_{\ell,0}^{(\nu)}$.

- Since $b'_{\rho(k)} \xleftarrow{\text{R}} \mathbb{Z}_N$, the distribution of $b_{\rho(k)} = b'_{\rho(k)} \cdot \xi^{(b)}$ is uniform as long as $\xi^{(b)} \neq 0 \bmod p_2$. First, $t_\ell = a^{d_\ell}$, where $a \xleftarrow{\text{R}} \mathbb{Z}_N$. Over the randomness of $a \xleftarrow{\text{R}} \mathbb{Z}_N$, the probability that $a^{d_\ell} = 0 \bmod p_2$ is at most $d_\ell / p_2 = \mathsf{negl}(\lambda)$, since $d_\ell \le \max(\mathcal{D}) = \mathsf{poly}(L) = \mathsf{poly}(\lambda)$. Next, consider the term
$$\sum_{i \in [L] : \rho(k) \notin S_i} t_i = \sum_{i \in [L] : \rho(k) \notin S_i} a^{d_i}.$$
By construction, the view of adversary $\mathcal{A}$ prior to the challenge phase is *independent* of the value of $a \bmod p_2$. In particular, the challenger in $\mathsf{iHyb}_{\ell,0}^{(\nu)}$ can defer the sampling of $a \bmod p_2$ until after the adversary has chosen the challenge policy $(\mathbf{M}, \rho)$. In this case, over the choice of $a \bmod p_2$, the probability that $\sum_{i \in [L] : \rho(k) \notin S_i} a^{d_i} = 0 \bmod p_2$ is at most $\max(\mathcal{D}) / p_2 = \mathsf{negl}(\lambda)$. We conclude that with overwhelming probability (over the choice of a), the distribution of $b_{\rho(k)}$ is uniform over $\mathbb{Z}_N$.
- Since $\eta'_k \xleftarrow{\text{R}} \mathbb{Z}_N$ and is independent of $\xi^{(\eta)}$, it follows that η_k is uniform over $\mathbb{Z}_N$.

Now consider the view of the adversary in $\mathsf{iHyb}_{\ell,0}^{(\nu)}$ under this particular variable substitution:

- First, consider the slot components $U_{\ell,\rho(k)}$ and $W_{z,\rho(k)}$ for all $i \in [L], z \in \mathcal{E}$. By definition,
$$U_{i,\rho(k)} = g_1^{b_{\rho(k)} t_i} = g_1^{b'_{\rho(k)} t_i}$$
$$W_{z,\rho(k)} = g_1^{b_{\rho(k)} a^z} = g_1^{b'_{\rho(k)} a^z},$$
since $\xi^{(b)} = 1 \bmod p_1$. The remaining components in the CRS do not depend on η_k or $b_{\rho(k)}$ and are unaffected.
- Next, the components the challenger constructs when responding to key-generation queries do not depend on the exponents η_k or $b_{\rho(k)}$, so their distributions (given the components in the CRS) are unchanged with this substitution.
- Finally, consider the components in the challenge ciphertext. The components $C_1, C_2, C_{4,k}, C_5$ are all unchanged (i.e., they are independent of η_k, $b_{\rho(k)}$). It suffices to consider the ciphertext component $C_{3,k}$. By definition of $b_{\rho(k)}$, we can write
$$-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i = -s_k b'_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i \bmod p_1$$
$$-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i = -s_k b'_{\rho(k)} t_\ell \left(\sum_{i \in [L] : \rho(k) \notin S_i} t_i \right)^{-1} \left(\sum_{i \in [L] : \rho(k) \notin S_i} t_i \right) = -s_k b'_{\rho(k)} t_\ell \bmod p_2.$$
Substituting into Eq. (5.7), we now have
$$\begin{aligned} C_{3,k}^{(0)} &= (g_1 g_2)^{s \beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k} \\ &= (g_1 g_2)^{s \beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} g_1^{-s s_k b'_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_2^{-s s_k b'_{\rho(k)} t_\ell} g_4^{-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta'_k + s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} \\ &= (g_1 g_2)^{s (\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}' - s_k b'_{\rho(k)} t_\ell)} g_1^{-s s_k b'_{\rho(k)} \sum_{i \in [L] \setminus \{\ell\} : \rho(k) \notin S_i} t_i} g_4^{\eta'_k}. \end{aligned}$$

Under this variable substitution, we have recovered the distribution in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ (with $b'_{\rho(k)}, \eta'_k \xleftarrow{\mathrm{R}} \mathbb{Z}_N$). Thus, the distributions of $C_{3,k}^{(0)}$ and $C_{3,k}^{(1)}$ are statistically indistinguishable and the claim holds. □

Lemma 5.19. *Suppose the progression-free indistinguishability assumption (Assumption 5.4) holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $\nu \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. Suppose that there exists an efficient adversary $\mathcal{A}$ that can distinguish these two experiments with non-negligible probability ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.4 with the same advantage:

1. Algorithm $\mathcal{B}$ starts by running algorithm $\mathcal{A}$ to obtain the number of slots 1^L. Let $\mathcal{D} = \{d_i\}_{i\in[L]}$ be an efficiently computable progression-free and double-free set. We define $f(i,j) := d_i + d_j$ and $\mathcal{E} = \{f(i,j) \mid i,j \in [L] : i \neq j\}$.

2. Algorithm $\mathcal{B}$ sends the input length 1^L, the set $\mathcal{D}$, and the index ℓ to its challenger. It receives a challenge of the form

$$\left(\mathcal{G}, g_1, g_3, g_4, g_{23}, \{A'_i\}_{i\in[L]\setminus\{i^*\}}\ ,\ \{U'_i\}_{i\in[L]}\ ,\ \{W'_{f(i,j)}\}_{i\neq j\in[L]}\ ,\ X\ ,\ \{Y_i\}_{i\in[L]\setminus\{i^*\}}\ ,\ Y_{i^*}\ ,\ T_\beta\right),$$

   where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$. In the following, we highlight the variables from the challenge in green for clarity.

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

4. For each slot $i \in [L]$, algorithm $\mathcal{B}$ samples $\delta_i, \tau_i, \tau'_i \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. It then constructs the slot components as follows:

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

$$A_i = A'_i g_3^{\tau'_i} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_{23} g_4)^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i}.$$

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

$$A_\ell = T \quad , \quad B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell} \quad , \quad P_\ell = (X g_3)^{\delta_\ell}.$$

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

$$A_i = A'_i g_3^{\tau'_i} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i}.$$

   Then, for each attribute $w \in \mathcal{U}_\lambda$, algorithm $\mathcal{B}$ samples $b'_w \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. For each $i \in [L]$ and $z \in \mathcal{E}$, algorithm $\mathcal{B}$ constructs the attribute-specific slot components as

$$U_{i,w} = (U'_i)^{b'_w} \quad , \quad W_{z,w} = (W'_z)^{b'_w}.$$

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

$$\mathsf{crs} = \left(\mathcal{G}\ ,\ Z\ ,\ g_1\ ,\ h\ ,\ g_3\ ,\ g_4\ ,\ \{(A_i, B_i, P_i)\}_{i\in[L]}\ ,\ \{U_{i,w}, W_{z,w}\}_{w\in\mathcal{U}_\lambda, i\in[L], z\in\mathcal{E}}\right)$$

   to the adversary $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary Dict to keep track of the key-generation queries.

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

6. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy $P^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is an injective row-labeling function, two messages μ_0^*, μ_1^*, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.
   - If $c_i \in \{1, \ldots, \mathsf{ctr}\}$, the challenger looks up the entry $\mathsf{Dict}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i = i'$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}'$. Otherwise, algorithm $\mathcal{B}$ aborts with output 0.
   - If $c_i = \perp$, then algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*)$ outputs 1. If not, it aborts with output 0. Otherwise, it sets $\mathsf{pk}_i = \mathsf{pk}_i^*$.

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

7. Algorithm $\mathcal{B}$ constructs the challenge ciphertext as follows:
   - **Message-embedding components:** First, algorithm $\mathcal{B}$ sets $C_1 = \mu_\nu^* \cdot e(g_1, X)^\alpha$ and $C_2 = X$.
   - **Attribute-specific components:** Algorithm $\mathcal{B}$ samples $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and sets $\mathbf{v}' = [1, v_2, \ldots, v_n]^\mathsf{T}$. For each $k \in [K]$, it samples $s_k, \eta_k \xleftarrow{R} \mathbb{Z}_N$ and depending on whether $\rho(k) \in S_\ell$, proceeds as follows:
     - If $\rho(k) \notin S_\ell$, set
     $$C_{3,k} = X^{\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} \prod_{i \in [L] : \rho(k) \notin S_i} Y_i^{-s_k b'_{\rho(k)}} g_4^{\eta_k} \quad , \quad C_{4,k} = (X g_4)^{s_k}.$$
     - If $\rho(k) \in S_\ell$, set
     $$C_{3,k} = X^{\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} \prod_{i \in [L] : \rho(k) \notin S_i} U_{i,\rho(k)}^{-s_k} g_4^{\eta_k} \quad , \quad C_{4,k} = (g_1 g_4)^{s_k}.$$
   - **Slot-specific component:** Algorithm $\mathcal{B}$ sets
   $$C_5 = X^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

   Algorithm $\mathcal{B}$ gives the challenge ciphertext to $\mathcal{A}$:
   $$\mathsf{ct}^* = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5 \big).$$

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

We now analyze the advantage of algorithm $\mathcal{B}$. First, the progression-free indistinguishability challenger samples $r \xleftarrow{R} \mathbb{Z}_N$ and sets $g_{23} = (g_2 g_3)^r$. It also samples exponents $a, b, s, \tau \xleftarrow{R} \mathbb{Z}_N$ and sets $t_i = a^{d_i}$ for $i \in [L]$. It also sets

$$A'_i = g_1^{t_i} \quad , \quad U'_i = g_1^{bt_i} \quad , \quad W'_{f(i,j)} = g_1^{bt_i t_j} \quad , \quad X = (g_1 g_2)^s \quad , \quad \forall i \neq \ell : Y_i = g_1^{sbt_i} \quad , \quad Y_\ell = (g_1 g_2)^{sbt_\ell}.$$

We now consider the different components in the reduction and argue that algorithm $\mathcal{B}$ correctly simulates an execution of either $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ or $\mathsf{iHyb}_{\ell,2}^{(\nu)}$ depending on the structure of the challenge component T:

- **CRS components:** In the reduction, the exponents s, a, t_i chosen by the challenger map to the analogous exponents in the execution of $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$. We consider each of the components in the CRS individually:
  - First, for $i \neq \ell$, algorithm $\mathcal{B}$ sets $A_i = A'_i g_3^{\tau'_i} = g_1^{t_i} g_3^{\tau'_i}$ where $\tau'_i \xleftarrow{R} \mathbb{Z}_N$. This matches the distribution of A_i in both experiments. When $i = \ell$, algorithm $\mathcal{B}$ sets $A_\ell = T$. If $T = g_1^{t_\ell} g_3^\tau$, then this corresponds to the distribution of A_ℓ in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$. If $T = g_1^{t_\ell} (g_2 g_3)^\tau$, then this matches the distribution of A_ℓ in $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.

  - Next, for $i < \ell$, algorithm $\mathcal{B}$ sets $B_i = g_1^\alpha A_i^\beta (g_{23} g_4)^{\tau_i}$. By construction, $g_{23} = (g_2 g_3)^r$. As long as $r \neq 0 \bmod p_2$ and $r \neq 0 \bmod p_3$, then the distribution of $((g_2 g_3)^r g_4)^{\tau_i}$ is distributed identically to $(g_2 g_3 g_3)^{\tau_i}$ over the choice of $\tau_i \xleftarrow{\text{R}} \mathbb{Z}_N$. Importantly, the randomizing exponent τ_i is only used to randomize B_i and no other components. Thus, with overwhelming probability over the choice of r, the distribution of B_i in the reduction is distributed correctly. For $i \geq \ell$, the challenger constructs B_i using the same procedure as in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.
  - Next, the exponent s sampled by the challenger plays the same role in the simulation of $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$. As such, algorithm $\mathcal{B}$ in reduction sets $P = (X g_3)^{\delta_i} = ((g_1 g_2)^s g_3)^{\delta_i}$, which matches the specification in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.
  - For the attribute-specific slot components, the reduction algorithm implicitly defines $b_w := b'_w b$, where $b \in \mathbb{Z}_N$ is the exponent chosen by the challenger. Since the challenger samples $b \xleftarrow{\text{R}} \mathbb{Z}_N$, with overwhelming probability, b is co-prime to N. As such, the distribution of $b_w = b'_w b$ is uniform over $\mathbb{Z}_N$ when $b'_w \xleftarrow{\text{R}} \mathbb{Z}_N$. Now, algorithm $\mathcal{B}$ defines

$$U_{i,w} = (U'_i)^{b'_w} = g_1^{bb'_w t_i} = g_1^{b_w t_i}$$
$$W_{f(i,j),w} = (W_{f(i,j)'})^{b'_w}) = g_1^{bb'_w t_i t_j} = g_1^{b_w t_i t_j} = g_1^{b_w a^{f(i,j)}},$$

    which coincides with the distribution in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.
  - Finally, the remaining components h and Z are constructed exactly as in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.

- **Key-generation queries:** By construction, algorithm $\mathcal{B}$ responds to key-generation queries using the identical procedure as in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.
- **Challenge query:** We consider each term in the challenge ciphertext. Recall from above that the exponent $s \in \mathbb{Z}_N$ chosen by the challenger plays its corresponding role in the simulated execution of $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$. First,

$$C_1 = \mu_\nu^* \cdot e(g_1, X)^\alpha = \mu_\nu^* \cdot e(g_1, (g_1 g_2)^s)^\alpha = \mu_\nu^* \cdot e(g_1, g_1)^{\alpha s} = \mu_\nu^* \cdot Z^s,$$

  which matches the distribution in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$. Next, algorithm $\mathcal{B}$ constructs the components C_2, $C_{4,k}$, and C_5 exactly as required in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$. It suffices to consider the distribution of $C_{3,k}$. We consider the two possibilities:

  - If $\rho(k) \notin S_\ell$, algorithm $\mathcal{B}$ computes

$$\begin{aligned} C_{3,k} &= X^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} \prod_{i \in [L] : \rho(k) \notin S_i} Y_i^{-s_k b'_{\rho(k)}} g_4^{\eta_k} \\ &= (g_1 g_2)^{s \beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} Y_\ell^{-s_k b'_{\rho(k)}} \prod_{i \in [L] \setminus \{\ell\} : \rho(k) \notin S_i} Y_i^{-s_k b'_{\rho(k)}} g_4^{\eta_k} \\ &= (g_1 g_2)^{s \beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} (g_1 g_2)^{-s s_k b b'_{\rho(k)} t_\ell} \prod_{i \in [L] \setminus \{\ell\} : \rho(k) \notin S_i} g_1^{-s s_k b b'_{\rho(k)} t_i} g_4^{\eta_k} \\ &= (g_1 g_2)^{s(\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}' - s_k b_{\rho(k)} t_\ell)} g_1^{-s s_k b_{\rho(k)} \sum_{i \in [L] \setminus \{\ell\} : \rho(k) \notin S_i} t_i} g_4^{\eta_k}, \end{aligned}$$

    which precisely coincides with the distribution of $C_{3,k}$ in $\mathsf{iHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{iHyb}_{\ell,2}^{(\nu)}$.

– If $\rho(k) \in S_\ell$, algorithm $\mathcal{B}$ computes

$$\begin{aligned} C_{3,k} &= X^{\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} \prod_{i \in [L] : \rho(k) \notin S_i} U_{i,\rho(k)}^{-s_k} g_4^{\eta_k} \\ &= (g_1 g_2)^{s \beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} \prod_{i \in [L] : \rho(k) \notin S_i} (U_i')^{-s_k b'_{\rho(k)}} g_4^{\eta_k} \\ &= (g_1 g_2)^{s \beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} \prod_{i \in [L] : \rho(k) \notin S_i} g_1^{-s_k b b'_{\rho(k)} t_i} g_4^{\eta_k} \\ &= (g_1 g_2)^{s \beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} g_1^{-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k}, \end{aligned}$$

which precisely coincides with the distribution of $C_{3,k}$ in $\mathsf{iHyb}_{\ell,1}^{(v)}$ and $\mathsf{iHyb}_{\ell,2}^{(v)}$.

We conclude that with overwhelming probability, algorithm $\mathcal{B}$ either perfectly simulates an execution of $\mathsf{iHyb}_{\ell,1}^{(v)}$ (when $T = g_1^{t_\ell} g_3^{\tau}$) and an execution of $\mathsf{iHyb}_{\ell,2}^{(v)}$ (when $T = g_1^{t_\ell} (g_2 g_3)^{\tau}$). Thus algorithm $\mathcal{B}$ breaks the progression-free indistinguishability assumption with advantage $\varepsilon - \mathsf{negl}(\lambda)$, which completes the proof. □

Lemma 5.20. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

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

Lemma 5.21. *Suppose Assumption 5.2b holds with respect to* CompGroupGen*. Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. The analysis here closely parallels the proof strategy from [HLWW23, Lemma 5.16]. Namely, the analysis depends on whether the adversary knows the secret key associated with slot ℓ or not. Due to the similarities with the proof from [HLWW23], we defer the formal argument to Appendix A.4. □

Lemma 5.22. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

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

Lemma 5.23. *Suppose the progression-free indistinguishability assumption (Assumption 5.4) holds with respect to* CompGroupGen*. Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

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

Lemma 5.24. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left| \Pr[\mathsf{iHyb}_{\ell,7}^{(v)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{\ell,6}^{(v)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

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

Lemma 5.25. *Suppose Assumption 5.2c holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $v \in \{0, 1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

$$\left|\Pr[\mathsf{Hyb}_{3,\ell}^{(v)}(\mathcal{A}) = 1] - \Pr[\mathsf{iHyb}_{\ell,7}^{(v)}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. This follows by a similar argument as the proof of Lemma 5.17. □

Combining Lemmas 5.17 to 5.25, Lemma 5.14 now follows by a hybrid argument. □

6 Concrete Efficiency Evaluation

In Sections 4 and 5, we showed how to leverage progression-free sets to construct new registered ABE schemes with a CRS whose size is *nearly linear* in the number of parties (when instantiated with the best-known progression-free sets [Beh46, SS46, Elk10]). In this section, we present a simple comparison to show that the improvements also translate into substantial *concrete* reductions in the size of the CRS in registered ABE schemes (when instantiated with a asymptotically-worse, but *concretely-efficient* progression-free set [ET36]). To allow for an apples-to-apples comparison where we specifically focus on the use of progression-free sets, we compare the following two schemes:

- The first construction is Construction 4.3, which gives a (statically-secure) registered ABE scheme that relies on progression-free sets.
- The second construction is a (statically-secure) variant of Construction 4.3 with a quadratic-size CRS that we describe in Construction B.3 (Construction B.3). Here, instead of sampling the exponents associated with each slot from a progression-free set (as in Construction 4.3), the slot components are sampled randomly (similar to earlier constructions [HLWW23, ZZGQ23]). As a result, the size of the CRS scales quadratically with the number of users. Note that we do not directly compare against [HLWW23, ZZGQ23] because these schemes have a dual-system proof and achieve adaptive security; the extra structure (in the forms of subgroups [HLWW23] or subspaces [ZZGQ23]) needed to implement the dual-system proof results in additional overhead. Our goal in this comparison is to highlight the efficiency gains from using progression-free sets, and for this reason, we elect to compare two schemes that are essentially identical except for how the individual slot components are chosen (from a progression-free set as in Construction 4.3 or randomly as in Construction B.3).

Evaluation methodology. In the following, we consider an instantiation of Constructions 4.3 and B.3 with the *asymmetric* BLS-381 pairing curve [BGM17, SKSW20]. We write $\mathbb{G}_1, \mathbb{G}_2$ to denote the two base groups and $\mathbb{G}_T$ to denote the target group. The group is defined over a 381-bit field (48 bytes), and the representation size of a $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ element is 48 bytes, 96 bytes, and 576 bytes, respectively.

Instantiating the progression-free set. For Construction 4.3, we instantiate the progression-free set with the classic ternary construction by Erdös and Turán [ET36]. To construct a progression-free set of size L, the set $\mathcal{E}$ contains the first L integers whose ternary representation only uses the digits 0 and 1. Thus $\max(\mathcal{E}) \le L^{\log_2 3}$. To obtain a progression-free and double-free set of size L, we can use the approach from Corollary 2.7 and instantiate with a progression-free set of size $(L+1)^{\log_2 3}$. Note that when comparing concrete efficiency, we do *not* use the asymptotically-better constructions from [Beh46, SS46, Elk10]. While these constructions satisfy $\max(\mathcal{E}) = L^{1+o(1)}$, if we consider the concrete size of the resulting progression-free sets for parameters of (practical) interest, they are significantly worse than using the progression-free sets based on the ternary encoding. As an example if we set $L = 10^5$ users, then the size of the largest element in the progression-free set from [Beh46] is 5.1×10^{10}, which is over 600× larger than the progression-free set obtained using the Erdös-Turán scheme.

Working over asymmetric groups. As written, Constructions 4.3 and B.3 operate over *symmetric* pairing groups, whereas the most efficient instantiations of pairing groups are asymmetric pairing groups. However, it is straightforward to translate both constructions to work using asymmetric pairing groups (and security will reduce to a corresponding asymmetric analog of the current assumptions). First, we describe how to assign the different components of the CRS, the public keys, and the ciphertexts to $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$. Since elements in $\mathbb{G}_1$ have shorter representations, we prefer assigning base-group elements to $\mathbb{G}_1$ rather than $\mathbb{G}_2$.

- **Common reference string:** In Construction 4.3, the common reference string consists of

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i\in[L]}\,,\, \{W_z\}_{z\in\mathcal{E}}\right).$$

  The same holds for Construction B.3, except the "cross-terms" $\{W_z\}_{z\in\mathcal{E}}$ are now $\{W_{i,j}\}_{i\neq j\in[L]}$. We assign the components to $\mathbb{G}_1$ and $\mathbb{G}_2$ as follows:

  - The following terms are computed in $\mathbb{G}_1$: W_z (or $W_{i,j}$), A_i, B_i, and P_i.
  - The following terms are computed in $\mathbb{G}_2$: h and U_i.
  - The following terms are computed in $\mathbb{G}_T$: Z.

  As described, Constructions 4.3 and B.3 set $B_i = g^\alpha h^{t_i}$. As described above, we assign h to $\mathbb{G}_2$ but B_i to $\mathbb{G}_1$. However, since these components are sampled as part of Setup, the algorithm "knows" the discrete log β of h and thus it can compute $B_i := g_1^{\alpha+\beta t_i}$ and $h := g_2^\beta$ where g_1 and g_2 are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively.

- **User public key:** For each slot $i \in [L]$, the user's public key in both schemes can be written as $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j\neq i})$.

  - The following terms are computed in $\mathbb{G}_1$: Q_i and $V_{j,i}$
  - The following terms are computed in $\mathbb{G}_2$: T_i.

- **Master public key:** In both schemes, the master public key can be written as $\mathsf{mpk} = \left(\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w\in\mathcal{U}_\lambda}\right)$.

  - The following terms are computed in $\mathbb{G}_2$: h, $\hat{T}$, and $\hat{U}_w$.
  - The following terms are computed in $\mathbb{G}_T$: Z.

- **Helper decryption key:** For each slot $i \in [L]$, the helper decryption key in both schemes can be written as $\mathsf{hsk}_i = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w\in\mathcal{U}_\lambda}\right)$.

  - The following terms are computed in $\mathbb{G}_1$: A_i, B_i, $\hat{V}_i$, $\hat{W}_{i,w}$.

- **Ciphertext:** In both schemes, the ciphertext can be written as $\mathsf{ct} = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k\in[K]}, C_5\right)$.

  - The following terms are computed in $\mathbb{G}_2$: C_2, $C_{3,k}$, $C_{4,k}$, C_5.
  - The following terms are computed in $\mathbb{G}_T$: C_1.

In Table 3, we report the number of group elements in each of these components as a function of the number of users the scheme supports, the size of the policy, and the size of the attribute universe. Then, in Table 4, we give estimated sizes for a concrete setting where we fix an attribute universe with 100 and policies over at most 25 attributes. Notably for a system with 10^5 users, the use of progression-free sets reduces the CRS size by over 115× (from over 447 GB to just 3.8 GB).

The Brown-Gallant-Cheon attacks. The curve BLS-381 provides roughly 128 bits of security. However, when using bilinear Diffie-Hellman exponent assumptions (e.g., Assumption 4.2) where the challenge contains terms of the form $g^a, g^{a^2}, \ldots, g^{a^q}$, the hardness of the assumption degrades with q. Specifically, Brown, Gallant, and Cheon [BG04, Che06]. showed how to recover the secret exponent a from $\left(g^a, g^{a^2}, \ldots, g^{a^q}\right)$ with an algorithm that runs in time $\tilde{O}\left(\sqrt{p/d} + \sqrt{d}\right)$, where $d \leq q$ can be any factor of either $p-1$ or $p+1$ less than q and p is the group order.

	$\mathbb{G}$	$\lvert\mathsf{crs}\rvert$	$\lvert\mathsf{mpk}\rvert$	$\lvert\mathsf{hsk}\rvert$	$\lvert\mathsf{ct}\rvert$
Construction 4.3	$\mathbb{G}_1$	$(L+1)^{\log_2 3} + 3L$	0	$3 + \lvert\mathcal{U}\rvert$	0
	$\mathbb{G}_2$	$L+1$	$2 + \lvert\mathcal{U}\rvert$	0	$2K+2$
	$\mathbb{G}_T$	1	1	0	1
Construction B.3	$\mathbb{G}_1$	$L^2 + 3L$	0	$3 + \lvert\mathcal{U}\rvert$	0
	$\mathbb{G}_2$	$L+1$	$2 + \lvert\mathcal{U}\rvert$	0	$2K+2$
	$\mathbb{G}_T$	1	1	0	1

Table 3: Number of group elements needed to instantiate the slotted registered ABE schemes from Constructions 4.3 and B.3 as a function of the number of users L, the size of the attribute universe $\lvert\mathcal{U}\rvert$, and the number of attributes K that each policy can depend on. We instantiate Construction 4.3 with a progression-free set based on the ternary encoding.

	L	$\lvert\mathsf{crs}\rvert$	$\lvert\mathsf{mpk}\rvert$	$\lvert\mathsf{hsk}\rvert$	$\lvert\mathsf{ct}\rvert$
Construction 4.3	10^3	3 MB	10 KB	5 KB	5 KB
Construction B.3	10^3	46 MB	10 KB	5 KB	5 KB
Construction 4.3	10^4	102 MB	10 KB	5 KB	5 KB
Construction B.3	10^4	4580 MB	10 KB	5 KB	5 KB
Construction 4.3	10^5	3.8 GB	10 KB	5 KB	5 KB
Construction B.3	10^5	447 GB	10 KB	5 KB	5 KB
Construction 4.3	10^6	145 GB	10 KB	5 KB	5 KB
Construction B.3	10^6	44 TB	10 KB	5 KB	5 KB

Table 4: Estimated size of different components of the slotted registered ABE schemes from Constructions 4.3 and B.3 as a function of the number of users L. For the comparisons, we fix the size of the attribute universe to be $\lvert\mathcal{U}\rvert = 100$ and consider a policy over 25 attributes. For simplicity, we only report the size of the group elements in the respective components (based on an instantiation with the BLS-381 pairing group). We do *not* include "auxiliary information" such as the size of the group description $\mathcal{G}$ or the description length of the policy P.

Since $d \le q \ll p$, the Brown-Gallant-Cheon attack effectively reduces the security by a factor $\sqrt{q}$.[9] Concretely, for a scheme with $L = 10^6$ users, and policies with up to 25 attributes (i.e., corresponding to the last row of Table 4), the largest power we give out in the CRS is $q = d_{\max} = (L+1)^{\log_2(3)} \approx 2^{32}$. Conservatively, in the target group, the largest power the adversary could compute is 2^{33} which results in a security loss of 17 bits (degrading security from 128 bits to around 111 bits).[10] We can compensate for the Brown-Gallant-Cheon attack by working over a larger pairing group (e.g., BLS-477 or BLS-581). Using a larger curve to achieve 128 bits of security would only affect the parameters by a small constant factor. Given the margins from Table 4, using progression-free sets still yields significant reductions in the CRS size relative to constructions that require a quadratic-size CRS.

7 Batch Arguments for NP from Composite-Order Bilinear Groups

In this section, we describe another setting where progression-free sets can be used to reduce the CRS size. Specifically, we focus on the Waters-Wu non-interactive batch arguments (BARG) for NP [WW22]. Very briefly, a batch argument for NP allows a prover to demonstrate that a set of L NP statements $x_1, \ldots, x_L$ are true with a proof whose size scales

[9]Technically, if $p-1$ and $p+1$ do not contain small prime factors (i.e., neither $p-1$ nor $p+1$ is smooth), then the Brown-Gallant-Cheon attacks would not apply. However, this may not be the case for elliptic curves used in practice, and in particular, is not true for the BLS-381 curve. In these settings, these attacks will indeed reduce the security level of the scheme.

[10]Alternatively, if we applied the Brown-Gallant-Cheon attack to our underlying assumption (Assumption 4.2), then the adversary is given g^{α^q} where $q = 4 \cdot d_{\max} \cdot \ell \cdot K \approx 2^{58}$ (Lemma 4.11). In this case, the security level of the assumption over the BLS-381 curve would be at most 99 bits. Note however that an attack on the assumption does not necessarily translate to an attack on the construction.

sublinearly with L. The base version of the Waters-Wu construction requires a CRS with size L^2. The quadratic blowup was due to the need to include "cross terms" in the CRS. Here, we show that using progression-free sets, we can reduce the number of cross terms, and correspondingly, the size of the CRS, from L^2 to $L^{1+o(1)}$. Note here that in this setting, alternative bootstrapping techniques [WW22, KLVW23] can also be used to reduce the CRS size, but these techniques all rely on recursive composition and as such, need to make *non-black-box* use of the group. In contrast, using progression-free sets, we can obtain a sub-quadratic CRS with minimal modifications to the original scheme. We believe this illustrates the general applicability of our techniques for reducing the parameter sizes in different cryptographic schemes.

7.1 Batch Arguments for NP

In this section, we recall the notion of a batch argument for NP. Our exposition is taken mostly verbatim from [WW22, §2.1]. We consider the NP-complete language of Boolean circuit satisfiability. We assume that the Boolean circuits are built from NAND gates. For a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ with t wires, we associate wires $1,\dots,n$ with the bits of the statement $x_1,\dots,x_n$, and wires $n+1,\dots,n+h$ with the bits of the witness $w_1,\dots,w_h$, respectively. We associate wire t with the output wire. We measure the size s of C by the number of NAND gates it has. By construction, $t \le n+h+s$. We now define the (batch) circuit satisfiability language we consider in this work:

Definition 7.1 (Circuit Satisfiability)**.** We define $\mathcal{L}_{\mathsf{CSAT}} = \{(C,\mathbf{x}) \mid \exists \mathbf{w} \in \{0,1\}^h : C(\mathbf{x},\mathbf{w}) = 1\}$ to be the language of Boolean circuit satisfiability, where $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is a Boolean circuit and $\mathbf{x} \in \{0,1\}^n$ is a statement. For a positive integer $L \in \mathbb{N}$, we define the *batch circuit satisfiability* language $\mathcal{L}_{\mathsf{BatchCSAT},L}$ as follows:

$$\mathcal{L}_{\mathsf{BatchCSAT},L} = \{(C,\mathbf{x}_1,\dots,\mathbf{x}_L) \mid \forall i \in [L] : \exists \mathbf{w}_i \in \{0,1\}^h : C(\mathbf{x}_i,\mathbf{w}_i) = 1\},$$

where $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ is a Boolean circuit and $\mathbf{x}_1,\dots,\mathbf{x}_L \in \{0,1\}^n$ are the instances.

Definition 7.2 (Batch Argument for Circuit Satisfiability)**.** A non-interactive batch argument (BARG) for circuit satisfiability is a tuple of three efficient algorithms $\Pi_{\mathsf{BARG}} = (\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ with the following properties:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^s) \to \mathsf{crs}$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $L \in \mathbb{N}$, and a bound on the circuit size $s \in \mathbb{N}$, the setup algorithm outputs a common reference string crs.
- $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1,\dots,\mathbf{x}_L), (\mathbf{w}_1,\dots,\mathbf{w}_L)) \to \pi$: On input the common reference string crs, a Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1,\dots,\mathbf{x}_L \in \{0,1\}^n$, and witnesses $\mathbf{w}_1,\dots,\mathbf{w}_L \in \{0,1\}^h$, the prove algorithm outputs a proof π.
- $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1,\dots,\mathbf{x}_L), \pi) \to b$: On input the common reference string crs, the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1,\dots,\mathbf{x}_L \in \{0,1\}^n$ and a proof π, the verification algorithm outputs a bit $b \in \{0,1\}$.

Definition 7.3 (Completeness)**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ is complete if for all $\lambda, L, s \in \mathbb{N}$, all Boolean circuits $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, all statements $\mathbf{x}_1,\dots,\mathbf{x}_L \in \{0,1\}^n$, and all witnesses $\mathbf{w}_1,\dots,\mathbf{w}_L \in \{0,1\}^h$ where $C(\mathbf{x}_i,\mathbf{w}_i) = 1$ for all $i \in [L]$,

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1,\dots,\mathbf{x}_L), \pi) = 1 : \begin{array}{c} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^s); \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1,\dots,\mathbf{x}_L), (\mathbf{w}_1,\dots,\mathbf{w}_L)) \end{array}\right] = 1.$$

Definition 7.4 (Somewhere Argument of Knowledge [CJJ21])**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup},\mathsf{Prove},\mathsf{Verify})$ is a somewhere argument of knowledge if there exists a pair of efficient algorithms $(\mathsf{TrapSetup},\mathsf{Extract})$ with the following properties:

- $\mathsf{TrapSetup}(1^\lambda, 1^L, 1^s, i^*) \to (\mathsf{crs}^*, \mathsf{td})$: On input the security parameter $\lambda \in \mathbb{N}$, the number of instances $L \in \mathbb{N}$, the size of the circuit $s \in \mathbb{N}$, and an index $i^* \in [L]$, the trapdoor setup algorithm outputs a common reference string crs^* and an extraction trapdoor td.
- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1,\dots,\mathbf{x}_L), \pi) \to \mathbf{w}^*$ On input the trapdoor td, statements $\mathbf{x}_1,\dots,\mathbf{x}_L$, and a proof π, the extraction algorithm outputs a witness $\mathbf{w}^* \in \{0,1\}^h$. The extraction algorithm is deterministic.

We require $(\mathsf{TrapSetup}, \mathsf{Extract})$ to satisfy the following two properties:

- **CRS indistinguishability:** For a bit $b \in \{0,1\}$, and an adversary $\mathcal{A}$, we define the CRS indistinguishability game as follows:
  1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs the number of statements 1^L, the size of the circuit 1^s, and an index $i^* \in [L]$.
  2. If $b = 0$, the challenger gives $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^s)$ to $\mathcal{A}$. If $b = 1$, the challenger gives $\mathsf{crs}^* \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^L, 1^s, i^*)$ to $\mathcal{A}$.
  3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

  Then, Π_{BARG} satisfies CRS indistinguishability if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

  in the above CRS indistinguishability game.

- **Somewhere extractable in trapdoor mode:** For an adversary $\mathcal{A}$, we define the somewhere extractable security game as follows:
  - On input the security parameter 1^λ, algorithm $\mathcal{A}$ starts by outputting the number of statements 1^L, the size of the circuit 1^s, and an index $i^* \in [L]$.
  - The challenger samples $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^L, 1^s, i^*)$ and gives crs^* to $\mathcal{A}$.
  - Algorithm $\mathcal{A}$ outputs a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, statements $\mathbf{x}_1, \dots, \mathbf{x}_L \in \{0,1\}^n$, and a proof π. Let $\mathbf{w}^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \dots, \mathbf{w}_L), \pi)$.
  - The output of the game is $b = 1$ if $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \dots, \mathbf{x}_L), \pi) = 1$ and $C(\mathbf{x}_{i^*}, \mathbf{w}^*) \neq 1$. Otherwise, the output is $b = 0$.

  Then Π_{BARG} is somewhere extractable in trapdoor mode if for every efficient adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, $\Pr[b = 1] = \mathsf{negl}(\lambda)$ in the somewhere extractable game.

Definition 7.5 (Succinctness)**.** A BARG $\Pi_{\mathsf{BARG}} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is succinct if there exists a fixed polynomial $\mathsf{poly}(\cdot,\cdot,\cdot)$ such that for all $\lambda, L, s \in \mathbb{N}$, all crs in the support of $\mathsf{Setup}(1^\lambda, 1^L, 1^s)$, and all Boolean circuits $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s, the following properties hold:

- **Succinct proofs:** The proof π output by $\mathsf{Prove}(\mathsf{crs}, C, \cdot, \cdot)$ satisfies $|\pi| \le \mathsf{poly}(\lambda, \log L, s)$.
- **Succinct CRS:** $|\mathsf{crs}| \le \mathsf{poly}(\lambda, L, n) + \mathsf{poly}(\lambda, \log L, s)$.
- **Succinct verification:** The verification algorithm runs in time $\mathsf{poly}(\lambda, L, n) + \mathsf{poly}(\lambda, \log L, s)$.

7.2 BARG for NP from Composite-Order Bilinear Maps

We now show how to use progression-free sets to reduce the size of the CRS in the BARG from [WW22]. For ease of exposition, we just consider the construction from composite-order groups. Like [WW22], we work with a two-prime composite-order group which we define formally below:

Definition 7.6 (Two-Prime Composite-Order Bilinear Group [BGN05])**.** A (symmetric) two-prime composite-order bilinear group generator is an efficient algorithm $\mathsf{CompGroupGen}$ that takes as input the security parameter λ and outputs a description $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, g, e)$ of a bilinear group where p_1, p_2 are distinct primes, $\mathbb{G}$ and $\mathbb{G}_T$ are cyclic groups of order $N = p_1 p_2$, g is a generator of $\mathbb{G}$, and $e \colon \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a non-degenerate bilinear map (called the "pairing"). We require that the group operation in $\mathbb{G}$ and $\mathbb{G}_T$ as well as the pairing operation be efficiently computable.

Progression-free indistinguishability assumption. The security of our variant of the BARG from [WW22] will rely on a new progression-free indistinguishability assumption. This is the analog of the progression-free assumptions we introduced when reasoning about our registered ABE schemes (Assumptions 4.2 and 5.4). We state the assumption here and in Appendix D (Lemma D.9), show that it holds in the generic group model.

Assumption 7.7 (Progression-Free Indistinguishability)**.** Let $\mathsf{CompGroupGen}$ be a two-prime composite-order bilinear group generator. For a security parameter λ and a bit $b \in \{0,1\}$, we define the following game between an adversary $\mathcal{A}$ and a challenger:

1. On input the security parameter 1^λ, algorithm $\mathcal{A}$ chooses an input length 1^L, a progression-free and double-free set $\mathcal{D} = \{d_i\}_{i \in [L]}$ together with a challenge index $i^* \in [L]$. Define the function $f(i,j) \coloneqq d_i + d_j$.
2. Challenger samples a group $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, g, e)$. Let $N = p_1 p_2$ and g_1, g_2 be generators of the corresponding subgroups. The challenger sets the public group description $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, and samples exponents $a, s \xleftarrow{\text{R}} \mathbb{Z}_N$. It also sets $t_i = a^{d_i}$ for all $i \in [L]$. Then, for each $i, j \in [L]$, it defines
$$A'_i = g_1^{st_i} \quad \text{and} \quad B'_{f(i,j)} = g_1^{s^2 t_i t_j}.$$
It also defines $T_0 = g_1^{st_{i^*}}$ and $T_1 = g^{st_{i^*}}$. The challenger gives the challenge
$$\left(\mathcal{G}\ ,\ g_1\ ,\ \{A'_i\}_{i \in [L] \setminus \{i^*\}}\ ,\ \{B'_{f(i,j)}\}_{i \neq j \in [L]}\ ,\ T_b\right)$$
to $\mathcal{A}$.
3. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.

We say that the progression-free indistinguishability assumption holds with respect to $\mathsf{CompGroupGen}$ if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,
$$|\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]| = \mathsf{negl}(\lambda).$$

BARG for NP using progression-free sets. We now show how to adapt [WW22, Construction 3.3] to use progression-free sets.

Construction 7.8 (BARG for NP with Progression-Free Sets)**.** We construct a BARG for the language of circuit satisfiability as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^s)$: On input the security parameter λ, the number of instances L, and the bound on the circuit size s, the setup algorithm proceeds as follows:
  - Sample $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $\mathbb{G}_1, \mathbb{G}_2$ be the subgroups of $\mathbb{G}$ of orders p_1, p_2, respectively. Let $N = p_1 p_2$ and $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$.
  - Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set. We define $f(i,j) \coloneqq d_i + d_j$ and the set of cross terms $\mathcal{E} = \{f(i,j) \mid i, j \in [L] : i \neq j\}$.
  - Sample $g_1 \xleftarrow{\text{R}} \mathbb{G}_1$ and exponents $a, s \xleftarrow{\text{R}} \mathbb{Z}_N$. For each $i \in [L]$, compute $t_i = a^{d_i}$ and let $A_i = g_1^{st_i}$. Let $A = \prod_{i \in [L]} A_i$. Then, for each $z \in \mathcal{E}$, compute $B_z = g_1^{s^2 a^z}$.

  Finally, output the common reference string
$$\mathsf{crs} = \left(\mathcal{G}, g_1, A, \{A_i\}_{i \in [L]}, \{B_z\}_{z \in \mathcal{E}}\right).$$

- $\mathsf{Prove}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_L), (\mathbf{w}_1, \ldots, \mathbf{w}_L))$: On input $\mathsf{crs} = \left(\mathcal{G}, g_1, A, \{A_i\}_{i \in [L]}, \{B_z\}_{z \in \mathcal{E}}\right)$, a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_L \in \{0,1\}^n$, and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_L \in \{0,1\}^h$, define t to be the number of wires in C and s to be the number of gates in C. Then, for $i \in [L]$ and $j \in [t]$, let $y_{i,j} \in \{0,1\}$ be the value of wire j in $C(\mathbf{x}_i, \mathbf{w}_i)$. The prover proceeds as follows:

  - **Encoding wire values:** For each $k \in [t]$, let $U_k = \prod_{i \in [L]} A_i^{y_{i,k}}$
  - **Validity of wire assignments:** For each $k \in [t]$, let $V_k = \prod_{i \neq j} B_{f(i,j)}^{(1-y_{i,k})y_{j,k}}$.
  - **Validity of gate computation:** For each NAND gate $G_\ell = (k_1, k_2, k_3) \in [t]^3$ (where $\ell \in [s]$), compute $W_\ell = \prod_{i \neq j} B_{f(i,j)}^{1-y_{i,k_1} y_{j,k_2} - y_{j,k_3}}$

  Output the proof $\pi = \big(\{U_k, V_k\}_{k \in [t]}, \{W_\ell\}_{\ell \in [s]}\big)$.

- $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_L), \pi)$: On input $\mathsf{crs} = \big(\mathcal{G}, g_1, A, \{A_i\}_{i \in [L]}, \{B_z\}_{z \in \mathcal{E}}\big)$, a circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_L \in \{0,1\}^n$, and the proof $\pi = \big(\{U_k, V_k\}_{k \in [t]}, \{W_\ell\}_{\ell \in [s]}\big)$, the verification algorithm checks the following:
  - **Validity of statement:** For each input wire $k \in [n]$, $U_k = \prod_{i \in [L]} A_i^{x_{i,k}}$.
  - **Validity of wire assignments:** For each $k \in [t]$,
  $$e(A, U_k) = e(g_1, V_k) e(U_k, U_k). \tag{7.1}$$
  - **Validity of gate computation:** For each gate $G_\ell = (k_1, k_2, k_3) \in [t]^3$,
  $$e(A, A) = e(U_{k_1}, U_{k_2}) e(A, U_{k_3}) e(g_1, W_\ell). \tag{7.2}$$
  - **Output satisfiability:** The output encoding U_t satisfies $U_t = A$.

  The algorithm outputs 1 if all checks pass, and outputs 0 otherwise.

Theorem 7.9 (Completeness)**.** *Construction 7.8 is complete.*

Proof. Take any circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, instances $\mathbf{x}_1, \ldots, \mathbf{x}_L \in \{0,1\}^n$ and witnesses $\mathbf{w}_1, \ldots, \mathbf{w}_L \in \{0,1\}^h$ such that $C(\mathbf{x}_i, \mathbf{w}_i) = 1$ for all $i \in [L]$. Let t be the number of wires in C and s be the number of gates in C. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^s)$ and $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, (\mathbf{x}_1, \ldots, \mathbf{x}_L), (\mathbf{w}_1, \ldots, \mathbf{w}_L))$. We show that $\mathsf{Verify}(\mathsf{crs}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_L), \pi)$ outputs 1. Let $\pi = \big(\{U_k, V_k\}_{k \in [t]}, \{W_\ell\}_{\ell \in [s]}\big)$. Consider each of the verification relations:

- **Validity of statement:** By construction of Prove, $U_k = \prod_{i \in [L]} A_i^{y_{i,k}}$. By definition, the first n wires in C coincide with the wires to the statement, so $y_{i,k} = x_{i,k}$ for $k \in [n]$. Thus, the first verification check passes.

- **Validity of wire assignments:** To show Eq. (7.1), take any $k \in [t]$. Recall that $U_k = \prod_{i \in [L]} A_i^{y_{i,k}} = g_1^{s \sum_{i \in [L]} t_i y_{i,k}}$ Now,
$$\left(s \sum_{i \in [L]} t_i\right)\left(s \sum_{j \in [L]} t_j y_{j,k}\right) = s^2 \sum_{i \in [L]} t_i^2 y_{i,k} + s^2 \sum_{i \neq j} t_i t_j y_{j,k},$$
and
$$\left(s \sum_{i \in [L]} t_i y_{i,k}^2\right)\left(s \sum_{j \in [L]} t_j y_{j,k}\right) = s^2 \sum_{i \in [L]} t_i^2 y_{i,k}^2 + s^2 \sum_{i \neq j} t_i t_j y_{i,k} y_{j,k} = s^2 \sum_{i \in [L]} t_i^2 y_{i,k} + s^2 \sum_{i \neq j} t_i t_j y_{i,k} y_{j,k},$$
using the fact that $y_{i,k} \in \{0,1\}$ so $y_{i,k}^2 = y_{i,k}$. Finally $V_k = \prod_{i \neq j} B_{f(i,j)}^{(1-y_{i,k})y_{j,k}} = g_1^{s^2 \sum_{i \neq j} t_i t_j (1-y_{i,k}) y_{j,k}}$. Thus, we can write
$$\begin{aligned} e(g_1, V_k) e(U_k, U_k) &= e(g_1, g_1)^{s^2 \sum_{i \neq j} t_i t_j (1-y_{i,k}) y_{j,k} + s^2 \sum_{i \in [L]} t_i^2 y_{i,k} + s^2 \sum_{i \neq j} t_i t_j y_{i,k} y_{j,k}} \\ &= e(g_1, g_1)^{s^2 (\sum_{i \in [L]} t_i^2 y_{i,k} + \sum_{i \neq j} t_i t_j y_{j,k})} \\ &= e(A, U_k). \end{aligned}$$

- **Validity of gate computation:** Take any gate $G_\ell = (k_1, k_2, k_3) \in [t]^3$. Consider first the exponents for the terms $e(U_{k_1}, U_{k_2})$, $e(A, U_{k_3})$, and $e(A, A)$:

$$\left(s \sum_{i\in[L]} t_i y_{i,k_1}\right)\left(s \sum_{j\in[L]} t_j y_{j,k_2}\right) = s^2\left(\sum_{i\in[L]} t_i^2 y_{i,k_1} y_{i,k_2} + \sum_{i\neq j} t_i t_j y_{i,k_1} y_{j,k_2}\right)$$
$$\left(s \sum_{i\in[L]} t_i\right)\left(s \sum_{j\in[L]} t_j y_{j,k_3}\right) = s^2\left(\sum_{i\in[L]} t_i^2 y_{i,k_3} + \sum_{i\neq j} t_i t_j y_{j,k_3}\right)$$
$$\left(s \sum_{i\in[L]} t_i\right)\left(s \sum_{j\in[L]} t_j\right) = s^2\left(\sum_{i\in[L]} t_i^2 + \sum_{i\neq j} t_i t_j\right).$$

By definition $y_{i,k_3} = \mathsf{NAND}(y_{i,k_1}, y_{i,k_2})$. This means that for each $i \in [L]$, either $(y_{i,k_1} y_{i,k_2} = 1$ and $y_{i,k_3} = 0)$ or $(y_{i,k_1} y_{i,k_2} = 0$ and $y_{i,k_3} = 1)$. This means that

$$\sum_{i\in[L]} t_i^2 (y_{i,k_1} y_{i,k_2} + y_{i,k_3}) = \sum_{i\in[L]} t_i^2.$$

Combining the above relations in the exponent, we have that

$$\begin{aligned}\frac{e(A,A)}{e(U_{k_1},U_{k_2})e(A,U_{k_3})} &= \frac{e(g_1,g_1)^{s^2(\sum_{i\in[L]} t_i^2 + \sum_{i\neq j} t_i t_j)}}{e(g_1,g_1)^{s^2(\sum_{i\in[L]} t_i^2 + \sum_{i\neq j} t_i t_j (y_{i,k_1} y_{j,k_2} + y_{j,k_3}))}} \\ &= \prod_{i\neq j} e(g_1, B_{f(i,j)})^{1 - y_{i,k_1} y_{j,k_2} - y_{j,k_3}} \\ &= e(g_1, W_\ell).\end{aligned}$$

- **Output satisfiability:** Since $C(\mathbf{x}_i, \mathbf{w}_i) = 1$, it follows that $y_{i,t} = 1$ for all $i \in [L]$. By definition, $U_t = \prod_{i\in[L]} A_i^{y_{i,t}} = \prod_{i\in[L]} A_i = A$. □

Theorem 7.10 (Somewhere Argument of Knowledge)**.** *Suppose the progression-free indistinguishability assumption (Assumption 7.7) holds with respect to* CompGroupGen. *Then, Construction 7.8 is a somewhere argument of knowledge.*

Proof. We start by defining the trapdoor setup and extraction algorithms:

- $\mathsf{TrapSetup}(1^\lambda, 1^L, 1^s, i^*)$: On input the security parameter λ, the number of instances L, the size of the circuit s, and the index i^*, the trapdoor setup algorithm proceeds as follows (with differences relative to Setup highlighted in green):

  1. Sample $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. Let $\mathbb{G}_1, \mathbb{G}_2$ be the subgroups of $\mathbb{G}$ of orders p_1, p_2, respectively. Let $N = p_1 p_2$ and $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$.
  2. Let $\mathcal{D} = \{d_i\}_{i\in[L]}$ be an efficiently-computable progression-free and double-free set. We define $f(i,j) := d_i + d_j$ and the set of cross terms $\mathcal{E} = \{f(i,j) \mid i,j \in [L] : i \neq j\}$.
  3. Sample $g_1 \xleftarrow{R} \mathbb{G}_1$ and exponents $a, s \xleftarrow{R} \mathbb{Z}_N$. For each $i \in [L] \setminus \{i^*\}$, compute $t_i = a^{d_i}$ and let $A_i = g_1^{s t_i}$. Let $A_{i^*} = g^{s t_{i^*}}$. Then, compute $A = \prod_{i\in[L]} A_i$. Then, for each $z \in \mathcal{E}$, compute $B_z = g_1^{s^2 a^z}$.

  Finally, output the common reference string $\mathsf{crs} = \left(\mathcal{G}, g_1, A, \{A_i\}_{i\in[L]}, \{B_z\}_{z\in\mathcal{E}}\right)$ and the trapdoor $\mathsf{td} = g_2$.

- $\mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_L), \pi)$: On input the trapdoor $\mathsf{td} = g_2$, the Boolean circuit $C\colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$, statements $\mathbf{x}_1, \ldots, \mathbf{x}_L \in \{0,1\}^n$, and the proof $\pi = \left(\{U_k, V_k\}_{k\in[t]}, \{W_\ell\}_{\ell\in[s]}\right)$, the extraction algorithm sets $w_k^* = 0$ if $e(g_2, U_k) = 1$ and $w_k^* = 1$ otherwise for all $k = n+1, \ldots, n+h$. It outputs $\mathbf{w}^* = (w_{n+1}^*, \ldots, w_{n+h}^*)$.

We now show the CRS indistinguishability and somewhere extractable in trapdoor mode properties.

Lemma 7.11 (CRS Indistinguishability)**.** *If the progression-free indistinguishability assumption (Assumption 7.7) holds with respect to* CompGroupGen*, then Construction 7.8 satisfies CRS indistinguishability.*

Proof. Suppose there exists an efficient adversary $\mathcal{A}$ that has non-negligible advantage ε in the CRS indistinguishability game. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the progression-free indistinguishability assumption (Assumption 7.7):

1. Algorithm $\mathcal{B}$ starts by running $\mathcal{A}$ to receive the number of instances 1^L, the circuit size 1^s, and a challenge index $i^* \in [L]$. Algorithm $\mathcal{B}$ constructs an efficiently-computable progression-free and double-free set $\mathcal{D} = \{d_i\}_{i\in[L]}$ of size L. As usual, we write $f(i,j) := d_i + d_j$ and the set of cross terms $\mathcal{E} = \{f(i,j) \mid i,j \in [L] : i \neq j\}$.

2. Algorithm $\mathcal{B}$ sends 1^L, $\mathcal{D}$, and i^* to the challenger. The challenger replies with the challenge
$$\big(\mathcal{G}\ ,\ g_1\ ,\ \{A'_i\}_{i\in[L]\setminus\{i^*\}}\ ,\ \{B'_{f(i,j)}\}_{i\neq j\in[L]}\ ,\ T\big).$$

3. Algorithm $\mathcal{B}$ sets $A_{i^*} = T$. For $i \neq i^*$, it sets $A_i = A'_i$ and $A = \prod_{i\in[L]} A_i$. For $z \in \mathcal{E}$, algorithm $\mathcal{B}$ sets $B_z = B'_z$. Algorithm $\mathcal{B}$ gives $\mathsf{crs} = \big(\mathcal{G}, g_1, A, \{A_i\}_{i\in[L]}, \{B_z\}_{z\in\mathcal{E}}\big)$ to $\mathcal{A}$.

4. After algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, algorithm $\mathcal{B}$ outputs the same bit.

We now consider the advantage of $\mathcal{B}$. The progression-free indistinguishability challenger samples $a, s \xleftarrow{\text{R}} \mathbb{Z}_N$, sets $t_i = a^{d_i}$, and sets $A'_i = g_1^{st_i}$ and $B'_{f(i,j)} = g_1^{s^2 t_i t_j}$. We consider two possibilities:

- Suppose $T = g_1^{st_{i^*}}$. Then, the common reference string crs is sampled according to $\mathsf{Setup}(1^\lambda, 1^L, 1^s)$.
- Suppose $T = g^{st_{i^*}}$. Then, the common reference string crs is sampled according to $\mathsf{TrapSetup}(1^\lambda, 1^L, 1^s, i^*)$.

We conclude that algorithm $\mathcal{B}$ wins the progression-free indistinguishability game with the same advantage as $\mathcal{A}$. □

Lemma 7.12 (Somewhere Extractable in Trapdoor Mode)**.** *Construction 7.8 is somewhere extractable in trapdoor mode.*

Proof. Take any adversary $\mathcal{A}$ and let $(1^L, 1^s, i^*) \leftarrow \mathcal{A}(1^\lambda)$. Let $(\mathsf{crs}^*, \mathsf{td}) \leftarrow \mathsf{TrapSetup}(1^\lambda, 1^L, 1^s, i^*)$. By construction,
$$\mathsf{crs}^* = \big(\mathcal{G}, g_1, A, \{A_i\}_{i\in[L]}, \{B_z\}_{z\in\mathcal{E}}\big) \quad \text{and} \quad \mathsf{td} = g_2,$$
where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $N = p_1 p_2$, and g_1, g_2 are generators of the subgroups $\mathbb{G}_1, \mathbb{G}_2$, respectively. Let $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be the Boolean circuit, $\mathbf{x}_1, \ldots, \mathbf{x}_L \in \{0,1\}^n$ be the statements, and $\pi = \big(\{U_k, V_k\}_{k\in[t]}, \{W_\ell\}_{\ell\in[s]}\big)$ be the proof the adversary outputs. Suppose $\mathsf{Verify}(\mathsf{crs}^*, (\mathbf{x}_1, \ldots, \mathbf{x}_L), \pi) = 1$. By construction of TrapSetup, we can write $A_{i^*} = g^{\alpha_{i^*}} = g_1^{\alpha_{i^*,1}} g_2^{\alpha_{i^*,2}}$ for some $\alpha_{i^*,1} \in \mathbb{Z}_{p_1}$ and $\alpha_{i^*,2} \in \mathbb{Z}_{p_2}$. By construction of TrapSetup, $\alpha_{i^*} = s \cdot a^{d_{i^*}}$, where $s, a \xleftarrow{\text{R}} \mathbb{Z}_N$. Since $d_{i^*} \le \max(\mathcal{D}) = \mathrm{poly}(\lambda)$, it follows that $\alpha_{i^*,2} \neq 0 \bmod p_2$ with overwhelming probability over the choice of $s, a \xleftarrow{\text{R}} \mathbb{Z}_N$. Then the following properties hold:

- For all $k \in [t]$, either $U_k \in \mathbb{G}_1$ or $U_k / g_2^{\alpha_{i^*,2}} \in \mathbb{G}_1$. This follows from the wire validity checks. Specifically, suppose $U_k = g_1^{\beta_1} g_2^{\beta_2} \in \mathbb{G}$. We can write $A = g_1^{\sum_{i\in[L]} \alpha_i} g_2^{\alpha_{i^*,2}}$. Since verification succeeds, it must be the case that
$$e(A, U_k) = e(g_1, V_k) e(U_k, U_k).$$
Consider the projection into the order-p_2 subgroup of $\mathbb{G}_T$. This relation requires that $\alpha_{i^*,2} \cdot \beta_2 = \beta_2^2 \bmod p_2$. This means that either $\beta_2 = 0$ (in which case $U_k \in \mathbb{G}_1$) or $\beta_2 = \alpha_{i^*,2}$ (in which case $U_k / g_2^{\alpha_{i^*,2}} \in \mathbb{G}_1$).

- For each $k \in [t]$, if $U_k \in \mathbb{G}_1$, then set $y_k = 0$. If $U_k / g_2^{\alpha_{i^*,2}} \in \mathbb{G}_1$, then set $y_k = 1$. Then, for all gates $G_\ell = (k_1, k_2, k_3) \in [t]^3$ in the circuit, $y_{k_3} = \mathsf{NAND}(y_{k_1}, y_{k_2})$. This follows from the gate validity checks. In particular, if verification succeeds, then Eq. (7.2) holds. From the above analysis, we can write $U_k = g_1^{\beta_{k,1}} g_2^{y_k \alpha_{i^*,2}}$ for all $k \in [t]$ and some $\beta_{k,1} \in \mathbb{Z}_N$. Consider the projection of Eq. (7.2) into the order-q subgroup of $\mathbb{G}_T$. This yields the relation
$$\alpha_{i^*,2}^2 = (y_{k_1} \alpha_{i^*,2})(y_{k_2} \alpha_{i^*,2}) + \alpha_{i^*,2}(y_{k_3} \alpha_{i^*,2}) = \alpha_{i^*,2}^2 (y_{k_1} y_{k_2} + y_{k_3}).$$
Since $\alpha_{i^*,2} \neq 0 \bmod p_2$, this means that $1 = y_{k_1} y_{k_2} + y_{k_3} \bmod p_2$. Equivalently, $y_{k_3} = 1 - y_{k_1} y_{k_2} = \mathsf{NAND}(y_{k_1}, y_{k_2})$.

- Let $\mathbf{x}_{i^*} = (x_{i^*,1}, \ldots, x_{i^*,n})$. For $k \in [n]$, $y_k = x_{i^*,k}$. This follows from the statement validity check. Namely, for all $k \in [n]$, the verifier checks that $U_k = A_{i^*}^{x_{i^*,k}} \prod_{i \neq i^*} A_i^{x_{i,k}}$. Since $A_i \in \mathbb{G}_1$ for $i \neq i^*$, it follows that if $x_{i^*,k} = 0$, then $U_k \in \mathbb{G}_1$ (and $y_k = 0 = x_{i^*,k}$). Otherwise, if $x_{i^*,k} = 1$, then the component of U_k in $\mathbb{G}_2$ is exactly $g_2^{\alpha_{i^*,2}}$, in which case $y_k = 1 = x_{i^*,k}$.
- Finally $y_t = 1$. This follows from the output satisfiability check. Namely, the verifier checks that $U_t = A = g_1^{\sum_{i \in [L]} \alpha_i} g_2^{\alpha_{i^*,2}}$. If the verifier accepts, then this relation holds and $y_t = 1$.

The above properties show that $y_1, \ldots, y_t \in \{0, 1\}$ is a valid assignment to the wires of C on input $\mathbf{x}_{i^*}$ and witness $\mathbf{w} = (y_{n+1}, \ldots, y_{n+h})$. Moreover, $C(\mathbf{x}_{i^*}, \mathbf{w}) = y_t = 1$. To complete the proof, let $\mathbf{w}^* \leftarrow \mathsf{Extract}(\mathsf{td}, C, (\mathbf{x}_1, \ldots, \mathbf{x}_L), \pi)$. We claim that $\mathbf{w}^* = \mathbf{w}$. In particular, for $k \in [h]$, if $U_{n+k} \in \mathbb{G}_1$, then $e(g_2, U_k) = 1$ and $w_k^* = 0 = y_{n+k}$. Alternatively, if $U_{n+k}/g_2^{\alpha_{i^*,2}} \in \mathbb{G}_1$, then $e(g_2, U_k) = e(g_2, g_2)^{\alpha_{i^*,2}} \neq 1$, so $w_k^* = 1 = y_{n+k}$. Thus, with probability $1 - \mathsf{negl}(\lambda)$ over the randomness of $\mathsf{TrapSetup}$, either $\mathsf{Verify}(\mathsf{crs}^*, C, (\mathbf{x}_1, \ldots, \mathbf{x}_L), \pi) = 0$ or $C(\mathbf{x}, \mathbf{w}^*) = 1$ and the claim holds. □

By Lemmas 7.11 and 7.12, Construction 7.8 is a somewhere argument of knowledge. □

Theorem 7.13 (Succinctness)**.** *Construction 7.8 is succinct.*

Proof. Take any $\lambda, L, s \in \mathbb{N}$ and consider a Boolean circuit $C \colon \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ of size at most s. Let $t = \mathsf{poly}(s)$ be the number of wires in C. We check each property:

- **Proof size:** A proof π consists of $2t + s$ elements in $\mathbb{G}$, each of which can be represented in $\mathsf{poly}(\lambda)$ bits. Thus, the proof size satisfies $|\pi| = (2t + s) \cdot \mathsf{poly}(\lambda) = \mathsf{poly}(\lambda, s)$
- **CRS size:** The common reference string crs consists of the group description $\mathcal{G}$, and $L + 1 + |\mathcal{D}|$ elements in $\mathbb{G}$. Using state-of-the-art progression-free sets (Corollary 2.7), we have that $\max(\mathcal{D}) \leq L^{1+o(1)}$. Thus, $|\mathsf{crs}| = L^{1+o(1)} \cdot \mathsf{poly}(\lambda)$.
- **Verification time:** Checking the statements requires time Ln group operations. The remaining checks require $O(t + s)$ additional group operations. Thus, the total verification cost is $\mathsf{poly}(\lambda, s) + \mathsf{poly}(\lambda, L, n)$ operations. □

Acknowledgments

We thank Jonathan Bootle and Hoeteck Wee for helpful conversations about progression-free sets. Brent Waters is supported by NSF CNS-1908611, CNS-2318701, and a Simons Investigator award. David J. Wu is supported by NSF CNS-2140975, CNS-2318701, a Microsoft Research Faculty Fellowship, and a Google Research Scholar award.

References

[Att14] Nuttapong Attrapadung. Dual system encryption via doubly selective security: Framework, fully secure functional encryption for regular languages, and more. In *EUROCRYPT*, 2014.

[BB04] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *EUROCRYPT*, pages 223–238, 2004.

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

[Beh46] F. Behrend. On sets of integers which contain no three terms in arithmetical progression. *Proceedings of the National Academy of Sciences*, 32(12), 1946.

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

[BG04] Daniel R. L. Brown and Robert P. Gallant. The static diffie-hellman problem. *IACR Cryptol. ePrint Arch.*, page 306, 2004.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable multi-party computation for zk-snark parameters in the random beacon model. *IACR Cryptol. ePrint Arch.*, page 1050, 2017.

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

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In *CRYPTO*, 2005.

[BLM$^+$24] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. Traitor tracing without trusted authority from registered functional encryption. *IACR Cryptol. ePrint Arch.*, page 179, 2024.

[Boy08] Xavier Boyen. The uber-assumption family: A unified complexity framework for bilinear groups. In *International Conference on Pairing-Based Cryptography*, pages 39–56, 2008.

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

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

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

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

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In *EUROCRYPT*, 2015.

[Che06] Jung Hee Cheon. Security analysis of the strong diffie-hellman problem. In *EUROCRYPT*, pages 1–11, 2006.

[CJJ21] Arka Rai Choudhuri, Abhishek Jain, and Zhengzhong Jin. SNARGs for $\mathcal{P}$ from LWE. In *FOCS*, pages 68–79, 2021.

[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*, pages 417–446, 2023.

[DP23] Pratish Datta and Tapas Pal. Registration-based functional encryption. *IACR Cryptol. ePrint Arch.*, page 457, 2023.

[DPY23] Pratish Datta, Tapas Pal, and Shota Yamada. Registered fe beyond predicates:(attribute-based) linear functions and more. *Cryptology ePrint Archive*, 2023.

[Elk10] Michael Elkin. An improved construction of progression-free sets. In *SODA*, 2010.

[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*, pages 98–133, 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*, pages 498–531, 2023.

[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.

[GKMR22] Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. *IACR Cryptol. ePrint Arch.*, 2022.

[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*, pages 1093–1107, 2023.

[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.

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

[KLVW23] Yael Kalai, Alex Lombardi, Vinod Vaikuntanathan, and Daniel Wichs. Boosting batch arguments and RAM delegation. In *STOC*, pages 1545–1552, 2023.

[KMW23] Dimitris Kolonelos, Giulio Malavolta, and Hoeteck Wee. Distributed broadcast encryption from bilinear groups. In *ASIACRYPT*, pages 407–441, 2023.

[KPY19] Yael Tauman Kalai, Omer Paneth, and Lisa Yang. How to delegate computations publicly. In *STOC*, pages 1115–1124, 2019.

[KSW08] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In *EUROCRYPT*, 2008.

[KSW13] Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. *J. Cryptol.*, 26(2):191–224, 2013.

[Lip12] Helger Lipmaa. Progression-free sets and sublinear pairing-based non-interactive zero-knowledge arguments. In *TCC*, 2012.

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

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

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

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

[MQR22] Mohammad Mahmoody, Wei Qi, and Ahmadreza Rahimi. Lower bounds for the number of decryption updates in registration-based encryption. In *TCC*, 2022.

[Sho97] Victor Shoup. Lower bounds for discrete logarithms and related problems. In *EUROCRYPT*, pages 256–266, 1997.

[SKSW20] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad Wahby. Pairing-friendly curves. I-d, IETF, 9 2020.

[SS46] R. Salem and D. C. Spencer. On sets of integers which contain no three in arithmetic progression. *Proceedings of the National Academy of Sciences*, 32(12), 1946.

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

[Wat05] Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

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

[Wat11] Brent Waters. Ciphertext-policy attribute-based encryption: An expressive, efficient, and provably secure realization. In *PKC*, 2011.

[Wee14] Hoeteck Wee. Dual system encryption via predicate encodings. In *TCC*, 2014.

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

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

A Analysis of Construction 5.5

In this section, we provide the formal proofs of Lemmas 5.11, 5.12, 5.15 and 5.21 underlying the security analysis of Construction 5.5. The structure of these proofs arguments directly parallels the corresponding proof from [HLWW23, Theorem 5.9]. As such, we reuse the prose and notation as the corresponding proofs from [HLWW23].

A.1 Proof of Lemma 5.11

This follows from an adaptation of the proof from [HLWW23, Lemma 5.10]. As noted previously, we reuse the same or similar prose and exposition from [HLWW23]. These two experiments are *statistically indistinguishable* if all of the public keys pk_i^* the adversary specifies in the challenge phase either satisfy $\mathsf{pk}_i^* = \perp$ or pk_i^* is in the support of the honest key-generation algorithm (i.e., for every $i \in [L]$, there exists r_i such that pk_i^* is the public key output by $\mathsf{KeyGen}(\mathsf{crs}, i)$). We start by showing that under Assumption 5.2a, the only public keys pk_i^* that an *efficient* adversary can construct and which satisfy the validity check $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$ are those that are in the support of the honest key-generation algorithm. To do so, we start by characterizing the set of possible strategies available to an efficient adversary. Here, we extend the proof strategy of [HLWW23] by allowing our reduction to handle *structured* distributions.

Claim A.1. *For a security parameter λ, we define the following game between an adversary $\mathcal{A}$ and a challenger:*

1. *On input the security parameter 1^λ, algorithm $\mathcal{A}$ outputs 1^Q and 1^{q^*}. We require that $q^* > 0$.*
2. *Then, the challenger samples $(\mathbb{G}, \mathbb{G}_T, p_1, p_2, p_3, p_4, g, e) \leftarrow \mathsf{CompGroupGen}(1^\lambda)$. It sets $N = p_1p_2p_3p_4$, $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ and samples $g_1 \xleftarrow{R} \mathbb{G}_1$, $g_3 \xleftarrow{R} \mathbb{G}_3$, $g_4 \xleftarrow{R} \mathbb{G}_3$, and $a, \tau \xleftarrow{R} \mathbb{Z}_N$. Then, for all $i \in [Q]$, it sets $Z_i = g_1^{a^i}$ and $Z^* = g_1^{a^{q^*}} g_3^\tau$. The challenger gives the tuple $\big(\mathcal{G}, g_1, g_3, g_4, \{Z_i\}_{i \in [Q]}, Z^*\big)$ to $\mathcal{A}$.*
3. *Algorithm $\mathcal{A}$ outputs a tuple $(A, B, C) \in \mathbb{G}^3$.*
4. *The challenger outputs $b = 1$ if the following relations are satisfied:*

$$e(g_3, A) = 1 = e(g_1, B)\ ,\ e(A, Z^*) = e(g_1, C)\ ,\ e(B, Z^*) = e(g_3, C)\ ,\ e(g_4, A) = e(g_4, B) = e(g_4, C) = 1, \quad \text{(A.1)}$$

and moreover, there does not exist $r \in \mathbb{Z}_N$ such that $A = g_1^r$, $B = g_3^r$, and $C = (Z^*)^r$.

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

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

1. At the beginning of the game, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_3, g_4, T)$, where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_3 \in \mathbb{G}_3$, $g_4 \in \mathbb{G}_4$, and either $T = g_1^r$ or $T = (g_1 g_2)^r$.
2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ to obtain 1^Q and 1^{q^*}.
3. Algorithm $\mathcal{B}$ samples exponents $\gamma_A, \gamma_B \xleftarrow{R} \mathbb{Z}_N$ and for each $i \in [Q]$, computes $Z_i = g_1^{\gamma_A^i}$ and $Z^* = g_1^{\gamma_A^{q^*}} g_3^{\gamma_B}$.
4. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ on input $(\mathcal{G}, g_1, g_3, g_4, \{Z_i\}_{i \in [Q]}, Z^*)$ to obtain a triple (A, B, C).
5. Algorithm $\mathcal{B}$ computes $Z' = C / (A^{\gamma_A^{q^*}} B^{\gamma_B})$ and outputs 1 if $e(Z', T) = 1$ and 0 otherwise.

First, we argue that algorithm $\mathcal{B}$ perfectly simulates an execution of the security game from Claim A.1 for $\mathcal{A}$. This follows by construction: namely, in the reduction, the exponent $\gamma_A \xleftarrow{R} \mathbb{Z}_N$ plays the role of a and the exponent $\gamma_B \xleftarrow{R} \mathbb{Z}_N$ plays the role of τ in Claim A.1. Thus, with probability at least ε, algorithm $\mathcal{A}$ outputs a tuple (A, B, C) such that Eq. (A.1) holds:

$$e(g_3, A) = 1 = e(g_1, B)\,,\ e(A, Z^*) = e(g_1, C)\,,\ e(B, Z^*) = e(g_3, C)\,,\ e(g_4, A) = e(g_4, B) = e(g_4, C) = 1.$$

Moreover, there does not exist $r \in \mathbb{Z}_N$ such that $A = g_1^r$, $B = g_3^r$, and $C = (Z^*)^r$. We now argue that in this case, over the choice of $\gamma_A, \gamma_B \xleftarrow{R} \mathbb{Z}_N$, it will be the case that $Z' \in \mathbb{G}_2 \setminus \{1\}$ with overwhelming probability.

- First, we show that Z' does not have any non-trivial component in the $\mathbb{G}_1$, $\mathbb{G}_3$ and $\mathbb{G}_4$ subgroups (i.e., $Z' \in \mathbb{G}_2$). First, Z' is a product of A, B, C. Since $e(g_4, A) = e(g_4, B) = e(g_4, C) = 1$, it holds that $e(g_4, Z') = 1$. We now show that $e(g_1 g_3, Z') = 1$. First, using the fact that $e(g_1, C) = e(A, Z^*)$, $e(g_3, C) = e(B, Z^*)$, and $Z^* = g_1^{\gamma_A^{q^*}} g_3^{\gamma_B}$, we can write

$$e(g_1 g_3, Z') = \frac{e(g_1 g_3, C)}{e\big(g_1 g_3, A^{\gamma_A^{q^*}}\big) e\big(g_1 g_3, B^{\gamma_B}\big)} = \frac{e(A, Z^*) e(B, Z^*)}{e\big(g_1 g_3, A^{\gamma_A^{q^*}}\big) e\big(g_1 g_3, B^{\gamma_B}\big)} = \frac{e\big(A, g_1^{\gamma_A^{q^*}} g_3^{\gamma_B}\big) e\big(B, g_1^{\gamma_A^{q^*}} g_3^{\gamma_B}\big)}{e\big(g_1 g_3, A^{\gamma_A^{q^*}}\big) e\big(g_1 g_3, B^{\gamma_B}\big)}.$$

  Next, since $e(g_1, B) = 1 = e(g_3, A)$, we have

$$e(g_1 g_3, Z') = \frac{e\big(A, g_1^{\gamma_A^{q^*}} g_3^{\gamma_B}\big) e\big(B, g_1^{\gamma_A^{q^*}} g_3^{\gamma_B}\big)}{e\big(g_1 g_3, A^{\gamma_A^{q^*}}\big) e\big(g_1 g_3, B^{\gamma_B}\big)} = \frac{e(A, g_1)^{\gamma_A^{q^*}} e(B, g_3)^{\gamma_B}}{e(g_1, A)^{\gamma_A^{q^*}} e(g_3, B)^{\gamma_B}} = 1.$$

  Hence, we conclude that $e(g_1 g_3 g_4, Z') = 1$, so $Z' \in \mathbb{G}_2$. It remains to show that $Z' \neq 1$.

- Next, at least one of the group elements A, B, C must contain a non-trivial component in the $\mathbb{G}_2$ subgroup. Suppose otherwise: namely that $A = (g_1 g_3)^{r_A}$, $B = (g_1 g_3)^{r_B}$, and $C = (g_1 g_3)^{r_C}$ for some $r_A, r_B, r_C \in \mathbb{Z}_N$. Note that $e(g_4, A) = e(g_4, B) = e(g_4, C) = 1$ so A, B, C cannot contain non-trivial components in the $\mathbb{G}_4$ subgroup. Then, Eq. (A.1) imply the following:

  - Since $e(g_3, A) = e(g_3, g_3)^{r_A \bmod p_3} = 1$, it must be the case that $r_A \bmod p_3 = 0$. Thus, $A = g_1^{r_A \bmod p_1}$.
  - Since $e(g_1, B) = e(g_1, g_1)^{r_B \bmod p_1} = 1$, it must be the case that $r_B \bmod p_1 = 0$. Thus, $B = g_3^{r_B \bmod p_3}$.
  - Finally, $e(g_1, C) = e(A, Z^*)$ means that $e(g_1, g_1)^{r_C \bmod p_1} = e(A, Z^*) = e(g_1, g_1)^{\gamma_A^{q^*} r_A \bmod p_1}$. Analogously, $e(g_3, C) = e(B, Z^*)$ means that $e(g_3, g_3)^{r_C \bmod p_3} = e(B, Z^*) = e(g_3, g_3)^{\gamma_B r_B \bmod p_3}$. Putting these together, this means that $r_C = \gamma_A^{q^*} r_A \bmod p_1$ and $r_C = \gamma_B r_B \bmod p_3$. Take any $r \in \mathbb{Z}_N$ such that $r = r_A \bmod p_1$ and $r = r_B \bmod p_3$. Then, we can write

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

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

- Thus, at least one of A, B, C must contain a non-trivial component in the $\mathbb{G}_2$ subgroup. Denote these by $g_2^{a'}, g_2^{b'}$ and $g_2^{c'}$, respectively. We have that at least one of $a', b', c' \neq 0 \bmod p_2$. Next, by the Chinese Remainder Theorem, the exponents γ_A and γ_B are uniform over $\mathbb{Z}_N$, so $\gamma_A \bmod p_2$ and $\gamma_B \bmod p_2$ are uniform over $\mathbb{Z}_{p_2}$ and more importantly, *independent* of the view of the adversary, as they are not revealed by $\{Z_i\}_{i \in [Q]}, Z^*$. Thus, we can write the $\mathbb{G}_2$ components of Z' as $g_2^{c - a\gamma_A^{q^*} - b\gamma_B}$. Consider the exponent $c - a\gamma_A^{q^*} - b\gamma_B \bmod p_2$. Since a, b, c are not all identically 0, this is a non-zero polynomial in γ_A, γ_B with degree at most q^*. By the Schwartz-Zippel lemma,

$$\Pr\left[c - a\gamma_A^{q^*} - b\gamma_B = 0 \bmod p_2 : \gamma_A, \gamma_B \xleftarrow{R} \mathbb{Z}_{p_2}\right] \leq \frac{q^*}{p_2} = \mathsf{negl}(\lambda),$$

since $q^* = \mathsf{poly}(\lambda)$. Correspondingly, this means that with probability $1 - \mathsf{negl}(\lambda)$, Z' has a non-trivial $\mathbb{G}_2$ component.

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

Using Claim A.1, we now show that the only public keys pk_i^* the efficient adversary can construct that pass the validity check are those in the support of the honest key-generation algorithm.

Lemma A.2. *For each index $i \in [L]$, let pk_i^* be the public key algorithm $\mathcal{A}$ outputs for slot i in the challenge phase in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. Suppose Assumption 5.2a holds with respect to* CompGroupGen. *Then, for all indices $i \in [L]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, if* $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$, *then with probability* $1 - \mathsf{negl}(\lambda)$, *there exists $r_i \in \mathbb{Z}_N$ such that pk_i^* is the public key output of* $\mathsf{KeyGen}(\mathsf{crs}, i; r_i)$.

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

- We first show that there exists $r_i \in \mathbb{Z}_N$ such that $T_i = g_1^{r_i}$, $R_i = g_3^{r_i}$, and $Q_i = P_i^{r_i}$ where $P_i = (g_1 g_3)^{\delta_i}$ is the component from the CRS. Suppose otherwise. Then, we use $\mathcal{A}$ to construct an efficient algorithm $\mathcal{B}$ that wins the game in Claim A.1:

  1. At the beginning of the game, algorithm $\mathcal{B}$ sets $Q = 0$ and $q^* = 1$ and outputs 1^Q and 1^{q^*}. It receives a tuple $(\mathcal{G}, g_1, g_3, g_4, Z^*)$ from the challenger, where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$ and $Z^* = (g_1 g_3)^\delta$ for some $\delta \xleftarrow{R} \mathbb{Z}_N$. (Technically, the challenger sets $Z^* = g_1^a g_3^\tau$ where $a, \tau \xleftarrow{R} \mathbb{Z}_N$, but it is easy to see that these two distributions are identical via the Chinese Remainder Theorem).
  2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the number of slots 1^L. Algorithm $\mathcal{B}$ guesses an index $i^* \xleftarrow{R} [L]$ and uses $(\mathcal{G}, g_1, g_3, g_4)$ to construct the components of crs according to $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L)$. It uses Z^* in place of P_{i^*} in crs. All of the other components are sampled according to the procedure in Setup. Algorithm $\mathcal{B}$ gives crs to $\mathcal{A}$.
  3. Algorithm $\mathcal{B}$ responds to the key-generation queries exactly as described in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. All of the logic only requires knowledge of the crs (and none of the specific exponents).
  4. During the challenge phase, algorithm $\mathcal{B}$ constructs the public keys pk_i for each $i \in [L]$ using the same procedure as in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. Again, the procedure here only depends on the components of the crs and does *not* require any knowledge of exponents. It parses $\mathsf{pk}_{i^*} = (T_{i^*}, Q_{i^*}, R_{i^*}, \{V_{j,i^*}\}_{j \neq i^*})$ and outputs $(T_{i^*}, R_{i^*}, Q_{i^*})$.

  By construction, algorithm $\mathcal{B}$ perfectly simulates the distribution of the common reference string. Thus, with probability ε, there exists an index $i \in [L]$ where $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$ and there does not exist $r_i \in \mathbb{Z}_N$ where $\mathsf{pk}_i = \mathsf{KeyGen}(\mathsf{crs}, i; r_i)$. Since i^* is perfectly hidden from $\mathcal{A}$, with probability $1/L$ over the choice of i^*, it holds that $i = i^*$. If $\mathsf{IsValid}(\mathsf{crs}, i^*, \mathsf{pk}_{i^*}^*) = 1$, then

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

and

$$e(g_4, T_{i^*}) = e(g_4, Q_{i^*}) = e(g_4, R_{i^*}) = 1.$$

Suppose now that there does not exist $r_{i^*} \in \mathbb{Z}_N$ where $T_{i^*} = g_1^{r_{i^*}}$, $R_{i^*} = g_3^{r_{i^*}}$, and $Q_{i^*} = P_{i^*}^{r_{i^*}}$. Then $\mathcal{B}$ wins the game in Claim A.1. Correspondingly, if algorithm $\mathcal{A}$ outputs a malformed key with probability ε, then algorithm $\mathcal{B}$ succeeds with probability ε/L, which proves the claim.

- Next, we show that for all $j \neq i$, there exists $r_{j,i} \in \mathbb{Z}_N$ such that $T_i = g_1^{r_{j,i}}$, $R_i = g_3^{r_{j,i}}$, $V_{j,i} = A_j^{r_{j,i}}$ where A_j is the component from the CRS. Against, suppose this was not the case. Once again, we use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that wins the game in Claim A.1:

  1. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the number of slots 1^L. Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set. As usual, we define $f(i, j) := d_i + d_j$ and the set $\mathcal{E} := \{f(i, j) \mid i, j \in [L] : i \neq j\}$. Let $d_{\max} = \max(\mathcal{D})$. Algorithm $\mathcal{B}$ sets $Q = 2d_{\max}$ and $q^* = d_j$ and outputs 1^Q and 1^{q^*}.
  2. Algorithm $\mathcal{B}$ receives a tuple $(\mathcal{G}, g_1, g_3, g_4, \{Z_i\}_{i \in [Q]}, Z^*)$ from the challenger, where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $Z_i = g_1^{a^i}$ and $Z^* = g_1^{a^{q^*}} g_3^{\tau'_j}$, and where the challenger sampled $a, \tau'_j \xleftarrow{R} \mathbb{Z}_N$.
  3. Algorithm $\mathcal{B}$ now constructs the crs according to $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L)$. Specifically, algorithm $\mathcal{B}$ samples $\alpha, \beta \xleftarrow{R} \mathbb{Z}_N$. Then, for each $i \in [L]$, it samples $\delta_i, \tau_i, \tau'_i \xleftarrow{R} \mathbb{Z}_N$ and for each $w \in \mathcal{U}_\lambda$, it samples $b_w \xleftarrow{R} \mathbb{Z}_N$. Algorithm $\mathcal{B}$ sets $A_j = Z^*$ and the remaining elements for $i \in [L]$, $w \in \mathcal{U}_\lambda$, and $z \in \mathcal{E}$ as

  $$A_i = Z_{d_i} g_3^{\tau'_i} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad P_i = (g_1 g_3)^{\delta_i} \quad , \quad U_{i,w} = Z_{d_i}^{b_w} \quad , \quad W_{z,w} = Z_z^{b_w}.$$

  Finally, algorithm $\mathcal{B}$ sets $h = g_1^\beta$ and $Z = e(g_1, g_1)^\alpha$ and gives crs to $\mathcal{A}$ where

  $$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, g_4\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{z,w}\}_{w \in \mathcal{U}_\lambda, i \in [L], z \in \mathcal{E}}\right).$$

  4. Algorithm $\mathcal{B}$ responds to the key-generation queries exactly as described in $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$. All of the logic only requires knowledge of the crs (and none of the specific exponents).
  5. During the challenge phase, algorithm $\mathcal{B}$ constructs the public keys pk_i for each $i \in [L]$ using the same procedure as in $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$. Again, the procedure here only depends on the components of the crs and does *not* require any knowledge of exponents. Finally, algorithm $\mathcal{B}$ samples a random $i^* \xleftarrow{R} [L]$, parses $\mathsf{pk}_{i^*} = \left(T_{i^*}, Q_{i^*}, R_{i^*}, \{V_{j,i^*}\}_{j \neq i^*}\right)$ and outputs $(T_{i^*}, R_{i^*}, V_{j,i^*})$.

We claim that algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$ for $\mathcal{A}$, where the exponents $a, \tau_j \xleftarrow{R} \mathbb{Z}_N$ sampled by the challenger for Claim A.1 plays the role of the corresponding exponents in $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$. It suffices to consider the distribution of A_j. In the reduction, algorithm $\mathcal{B}$ sets $A_j = Z^* = g_1^{a^{d_j}} g_3^{\tau'_j}$, which is precisely the distribution of A_j in $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$. Thus, with probability ε, there exists an index $i \in [L]$ where $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$ and there does not exist $r_i \in \mathbb{Z}_N$ where $\mathsf{pk}_i = \mathsf{KeyGen}(\mathsf{crs}, i; r_i)$. Since i^* is randomly sampled at the very end, with probability $1/L$ over the choice of i^*, it holds that $i = i^*$. If $\mathsf{IsValid}(\mathsf{crs}, i^*, \mathsf{pk}_{i^*}^*) = 1$, then

$$e(g_3, T_{i^*}) = 1 = e(g_1, R_{i^*})\,,\ e(T_{i^*}, A_j) = e(g_1, V_{j,i^*})\,,\ e(R_{i^*}, A_j) = e(g_3, V_{j,i^*}),$$

and

$$e(g_4, T_{i^*}) = e(g_4, R_{i^*}) = e(g_4, V_{j,i^*}) = 1.$$

Suppose now that there does not exist $r_{j,i^*} \in \mathbb{Z}_N$ where $T_{i^*} = g_1^{r_{j,i^*}}$, $R_{i^*} = g_3^{r_{j,i^*}}$, and $V_{j,i^*} = A_j^{r_{j,i^*}}$. Since $A_j = Z^*$, we conclude that algorithm $\mathcal{B}$ wins the game in Claim A.1. Correspondingly, if algorithm $\mathcal{A}$ outputs a malformed key with probability ε, then algorithm $\mathcal{B}$ succeeds with probability ε/L, which proves the claim.

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

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

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

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

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

Returning now to the proof of Lemma 5.11, we can first appeal to Lemma A.2 to conclude that for all efficient adversaries $\mathcal{A}$, in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$, the public keys $\mathsf{pk}_1^*, \ldots, \mathsf{pk}_L^*$ chosen by $\mathcal{A}$ in the challenge phase are either $\perp$, do not satisfy the $\mathsf{IsValid}$ predicate, or are in the support of the honest key-generation algorithm. Thus, if the challenger does not abort, then it must be the case that for all $i \in [L]$, there exists $r_i \in \mathbb{Z}_N$ such that pk_i is the public key output of $\mathsf{KeyGen}(\mathsf{crs}, i; r_i)$. In particular, all of the keys pk_i sampled by the challenger in an (honest) key-generation query already satisfy this property. Thus, for each $i \in [L]$, we can write

$$T_i = g_1^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad R_i = g_3^{r_i} \quad , \quad V_{j,i} = A_j^{r_i}. \tag{A.2}$$

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

$$\hat{T} = \prod_{i \in [L]} T_i = \prod_{i \in [L]} g_1^{r_i} \quad \text{and} \quad \hat{U}_w = \prod_{i \in [L] : w \notin S_i} U_{i,w} = g_1^{b_w \sum_{i \in [L] : w \notin S_i} t_i}. \tag{A.3}$$

We now consider the components in the two experiments:

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

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

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

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

  Since δ_i is still uniform over $\mathbb{Z}_N$ (and independent of all other quantities), the distribution of $s\delta_i \bmod p_1$ is uniform over $\mathbb{Z}_{p_1}$ as long as $s \neq 0 \bmod p_1$ (which holds with overwhelming probability since $s \xleftarrow{\text{R}} \mathbb{Z}_N$). As such, the distribution of P_i in these two experiments is statistically indistinguishable.
- Consider the attribute-specific components $C_{3,k}$ in the challenge ciphertext. In $\mathsf{Hyb}_1^{(\nu)}$, for each $k \in [K]$,

$$\begin{aligned} C_{3,k} &= (g_1^s)^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} (C_{4,k})^{-b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k} \\ &= (g_1^s)^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}} (g_1 g_4)^{-s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k} \end{aligned}$$

  Since $h_2 = g_1^{\beta_2}$ and appealing to Eq. (A.3), this can be rewritten as

$$C_{3,k} = h_2^{\mathbf{m}_k^{\mathsf{T}} \mathbf{v}''} \hat{U}_{\rho(k)}^{-s_k} g_4^{\eta_k'},$$

  where $\mathbf{v}'' = s\mathbf{v}' = [s, sv_2, \ldots, sv_n]^{\mathsf{T}}$ and $\eta_k' = \eta_k - s_k b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i$. In $\mathsf{Hyb}_1^{(\nu)}$, the challenger samples $v_2, \ldots, v_n, \eta_k \xleftarrow{\text{R}} \mathbb{Z}_N$. Moreover, since η_k is only used to randomize $C_{3,k}$, the distribution of η_k' in the above expression is uniform over and independent over $\mathbb{Z}_N$. We conclude that the distribution of $C_{3,k}$ in $\mathsf{Hyb}_1^{(\nu)}$ matches the distribution in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$ with the substitution $\mathbf{v} \mapsto \mathbf{v}''$ and $\eta_k \mapsto \eta_k'$.

- Finally, consider the slot-specific component C_5 in the challenge ciphertext in $\mathsf{Hyb}_1^{(\nu)}$. By Eq. (A.2),

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

By Eq. (A.3), in $\mathsf{Hyb}_1^{(\nu)}$,

$$C_5 = (g_1^s)^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right) = h_1^s \prod_{i \in [L]} g_1^{-sr_i} = h_1^s \hat{T}^{-s}.$$

Thus, C_5 is distributed identically to $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. □

A.2 Proof of Lemma 5.12

This follows from an adaptation of the proof from [HLWW23, Lemma 5.13]. As noted previously, we follow and reuse much of the same prose and exposition from [HLWW23]. Suppose there exists an efficient adversary $\mathcal{A}$ that distinguishes between $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_2^{(\nu)}$ with non-negligible advantage ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2a with the same advantage:

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

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the number of slots 1^L. Then, algorithm $\mathcal{B}$ samples $\alpha, \beta_1, \beta_2, a \xleftarrow{\text{R}} \mathbb{Z}_N$. It sets $Z = e(g_1, g_1)^\alpha$, $\beta = \beta_1 + \beta_2$, and $h = g_1^\beta$.

3. Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set. As usual, we define $f(i,j) := d_i + d_j$ and the set of cross terms $\mathcal{E} = \{f(i,j) \mid i, j \in [L] : i \neq j\}$.

4. For each slot $i \in [L]$, algorithm $\mathcal{B}$ samples $\delta_i, \tau_i, \tau_i' \xleftarrow{\text{R}} \mathbb{Z}_N$ and sets $t_i = a^{d_i}$. It then constructs the slot components as follows:

$$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad P_i = (T g_3)^{\delta_i}.$$

   Then, for each attribute $w \in \mathcal{U}_\lambda$, algorithm $\mathcal{B}$ samples $b_w \xleftarrow{\text{R}} \mathbb{Z}_N$. For each $w \in \mathcal{U}_\lambda$, slot index $i \in [L]$, and cross term index $z \in \mathcal{E}$, algorithm $\mathcal{B}$ constructs the attribute-specific slot components $U_{i,w}$ and $W_{z,w}$ as in $\mathsf{Hyb}_1^{(\nu)}$:

$$U_{i,w} = g_1^{b_w t_i} \quad , \quad W_{z,w} = g_1^{b_w a^z}.$$

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

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, g_4\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{z,w}\}_{w \in \mathcal{U}_\lambda, i \in [L], z \in \mathcal{E}}\right)$$

   to the adversary $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary Dict to keep track of the key-generation queries.

5. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_2^{(\nu)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{\text{R}} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, $Q_i = P_i^{r_i}$, $R_i = g_3^{r_i}$, and $V_{j,i} = A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary Dict. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \leq i \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{Dict}[i]$ and replies to $\mathcal{A}$ with sk'.

6. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy $P^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$, two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs the public key pk_i as in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_2^{(\nu)}$:

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

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

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

   - **Message-embedding components:** Set $C_1 = \mu_\nu^* \cdot e(g_1, T)^\alpha$ and $C_2 = T$.
   - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^\intercal$. For each $k \in [K]$, sample $s_k, \eta_k \xleftarrow{R} \mathbb{Z}_N$ and set

   $$C_{3,k} = T^{\beta_2 \mathbf{m}_k^\intercal \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k} \quad , \quad C_{4,k} = (g_1 g_4)^{s_k}.$$

   - **Slot-specific component:** Set

   $$C_5 = T^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

   Algorithm $\mathcal{B}$ gives the challenge ciphertext to $\mathcal{A}$:

   $$\mathsf{ct}^* = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5 \big).$$

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

Observe that $e(g_1, T)^\alpha = e(g_1, g_1)^{\alpha s}$ regardless of whether $T = g_1^s$ or $T = (g_1 g_2)^s$. If $T = g_1^s$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(\nu)}$. Alternatively, when $T = (g_1 g_2)^s$, algorithm $\mathcal{B}$ perfectly simulates $\mathsf{Hyb}_{2,0}^{(\nu)}$. Thus, algorithm $\mathcal{B}$ breaks Assumption 5.2a with the same advantage ε. □

A.3 Proof of Lemma 5.15

This follows from an adaptation of the proof from [HLWW23, Lemma 5.33]. As noted previously, we reuse the same prose and exposition from [HLWW23]. Suppose there exists an efficient adversary $\mathcal{A}$ where

$$\left| \Pr[\mathsf{Hyb}_{3,L}^{(\nu)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}(\mathcal{A}) = 1] \right| = \varepsilon$$

for some non-negligible ε. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that breaks Assumption 5.2e with the same advantage:

1. First, algorithm $\mathcal{B}$ receives a challenge $(\mathcal{G}, g_1, g_2, g_3, g_4, X, Y, T)$ where $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, N, g, e)$, $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$, $g_3 \in \mathbb{G}_3$, $g_4 \in \mathbb{G}_4$, $X = g_1^\alpha g_2^{\gamma_1}$, $Y = g_1^s g_2^{\gamma_2}$ for some $\alpha, \gamma_1, \gamma_2 \xleftarrow{R} \mathbb{Z}_N$, and either $T = e(g_1, g_1)^{\alpha s}$ or $T = e(g, g)^r$, where $r \xleftarrow{R} \mathbb{Z}_N$. The components that depend on the challenge elements X, Y, T are colored for clarity.

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the number of slots 1^L. Then, algorithm $\mathcal{B}$ samples $\beta_1, \beta_2, a \xleftarrow{R} \mathbb{Z}_N$ and sets $Z = e(g_1, X)$, $\beta = \beta_1 + \beta_2$, and $h = g_1^\beta$.

3. Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set. As usual, we define $f(i, j) \coloneqq d_i + d_j$ and $\mathcal{E} \coloneqq \{f(i, j) \mid i, j \in [L] : i \neq j\}$.

4. For each slot $i \in [L]$, sample $\delta_i, \tau_i, \tau'_i \xleftarrow{R} \mathbb{Z}_N$ and compute $t_i = a^{d_i}$. Algorithm $\mathcal{B}$ constructs the (semi-functional) slot components as follows:

$$A_i = g_1^{t_i} g_3^{\tau'_i} \quad , \quad B_i = X A_i^{\beta} (g_2 g_3 g_4)^{\tau_i} \quad , \quad P_i = (Y g_3)^{\delta_i} .$$

Then, for each attribute $w \in \mathcal{U}_\lambda$, algorithm $\mathcal{B}$ samples $b_w \xleftarrow{R} \mathbb{Z}_N$. For each $w \in \mathcal{U}_\lambda$, slot index $i \in [L]$ and cross term index $z \in \mathcal{E}$, algorithm $\mathcal{B}$ constructs the attribute-specific slot components $U_{i,w}$ and $W_{z,w}$ as follows:

$$U_{i,w} = g_1^{b_w t_i} \quad , \quad W_{z,w} = g_1^{b_w a^z} .$$

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

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, g_4\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{z,w}\}_{w \in \mathcal{U}_\lambda, i \in [L], z \in \mathcal{E}}\right)$$

to the adversary $\mathcal{A}$. It also initializes a counter ctr = 0 and an (empty) dictionary Dict to keep track of the key-generation queries.

5. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter ctr = ctr + 1 and samples $r_i \xleftarrow{R} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, $Q_i = P_i^{r_i}$, $R_i = g_3^{r_i}$, and $V_{j,i} = A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary Dict. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \le i \le \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{Dict}[i]$ and replies to $\mathcal{A}$ with sk'.

6. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy $P^* = (\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho\colon [K] \to \mathcal{U}_\lambda$ is an injective row-labeling function, two messages μ_0^*, μ_1^*, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. For each $i \in [L]$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$:

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

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

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

   - **Message-embedding components:** First, algorithm $\mathcal{B}$ sets $C_1 = \mu_\nu^* \cdot T$ and $C_2 = Y$.
   - **Attribute-specific components:** Algorithm $\mathcal{B}$ samples $v_2, \ldots, v_n \xleftarrow{R} \mathbb{Z}_N$ and sets $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$. For each $k \in [K]$, it samples $s_k, \eta_k \xleftarrow{R} \mathbb{Z}_N$ and sets

   $$C_{4,k} = (g_1 g_4)^{s_k} \quad , \quad C_{3,k} = Y^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k} .$$

   - **Slot-specific component:** Algorithm $\mathcal{B}$ sets

   $$C_5 = Y^{\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

   Algorithm $\mathcal{B}$ gives the challenge ciphertext to $\mathcal{A}$:

   $$\mathsf{ct}^* = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\right).$$

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

In the reduction, the exponents α mod p_1 and s mod p_1 play their corresponding roles in $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. The exponent γ_2 mod p_2 (in Y) plays the role of s mod p_2. We now argue that algorithm $\mathcal{B}$ perfectly simulates an execution of either $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$:

- **CRS components:** First, $Z = e(g_1, X) = e(g_1, g_1^{\alpha} g_2^{\gamma_1}) = e(g_1, g_1)^{\alpha}$. Consider the remaining components of the CRS. Algorithm $\mathcal{B}$ sets $A_i = g_1^{t_i} g_3^{\tau_i'}$ where $t_i = a^{d_i}$ and $\tau_i' \xleftarrow{R} \mathbb{Z}_N$, which matches the distribution in $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. Next, algorithm $\mathcal{B}$ sets

  $$B_i = X A_i^{\beta} (g_2 g_3 g_4)^{\tau_i} = g_1^{\alpha} g_2^{\gamma_1} A_i^{\beta} (g_2 g_3)^{\tau_i} = g_1^{\alpha} A_i^{\beta} g_2^{\gamma_1 + \tau_i} (g_3 g_4)^{\tau_i}.$$

  Since $\tau_i \xleftarrow{R} \mathbb{Z}_N$, the distribution of $\gamma_1 + \tau_i$ is also uniform over $\mathbb{Z}_N$. Moreover, no other term depends on the value of τ_i, so the distribution of B_i is correctly simulated. Finally, algorithm $\mathcal{B}$ sets

  $$P_i = (Y g_3)^{\delta_i} = ((g_1^s g_2^{\gamma_2}) g_3)^{\delta_i},$$

  which matches the distribution in $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ when γ_2 mod p_2 plays the role of s mod p_2. Since the challenger samples $\gamma_2 \xleftarrow{R} \mathbb{Z}_N$, this matches the distribution of s in $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. Finally, the attribute-specific components in the CRS are sampled exactly as in $\mathsf{Hyb}_{3,L}^{(\nu)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$.
- **Key-generation queries:** By construction, algorithm $\mathcal{B}$ responds to key-generation queries using the identical procedure as in $\mathsf{Hyb}_{3,L}^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$.
- **Challenge query:** By construction, the challenge ciphertext components $C_2, C_{3,k}, C_{4,k}, C_5$ are distributed exactly as in $\mathsf{Hyb}_{3,L}^{(\nu)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ where γ_2 mod p_2 plays the role of s mod p_2. Consider now the distribution of C_1:
  - If $T = e(g_1, g_2)^{\alpha, s}$, then $C_1 = \mu_b^* \cdot T = \mu_b^* \cdot Z^s$. This corresponds to the distribution in $\mathsf{Hyb}_{3,L}^{(\nu)}$.
  - If $T = e(g_1, g_2)^r$, where $r \xleftarrow{R} \mathbb{Z}_N$, the distribution of C_1 is uniform in $\mathbb{G}_T$, which corresponds to the distribution in $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$.

We conclude that algorithm $\mathcal{B}$ either simulates an execution of $\mathsf{Hyb}_{3,L}^{(\nu)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. Thus, algorithm $\mathcal{B}$ breaks Assumption 5.2e with the same distinguishing advantage as $\mathcal{A}$ and the claim follows. □

A.4 Proof of Lemma 5.21

Our analysis follows a very similar style as the proof of [HLWW23, Lemma 5.16]. Like the proof from [HLWW23], our analysis depend on whether the adversary knows the secret key associated with slot ℓ or not. We begin with a general sketch of our argument (with some of the prose taken directly from [HLWW23]).

Proof overview. Let $(c_i, S_i, \mathsf{pk}_i^*)$ be the tuples adversary $\mathcal{A}$ outputs for each slot $i \in [L]$ in the challenge phase. Let ctr be the number of key-generation queries the adversary has made at the beginning of the challenge phase. We say that event NonCorrupt occurs if

$$c_\ell \in \{1, \ldots, \mathsf{ctr}\} \text{ and } \mathcal{A} \text{ did not make a corruption query on index } c_\ell,$$

Let $\mathsf{pk}_1, \ldots, \mathsf{pk}_L$ be the public keys the challenger constructs during the challenge phase. If event NonCorrupt occurs, then the public key pk_ℓ was *honestly* sampled by the challenger in a key-registration query, and moreover, the adversary did *not* corrupt the key to learn its associated secret key. We write $\overline{\mathsf{NonCorrupt}}$ to denote the complement of event NonCorrupt. Now, we can write

$$\Pr\left[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] + \Pr\left[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right]$$
$$\Pr\left[\mathsf{iHyb}_{\ell,4}^{(\nu)}(\mathcal{A}) = 1\right] = \Pr\left[\mathsf{iHyb}_{\ell,4}^{(\nu)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}\right] + \Pr\left[\mathsf{iHyb}_{\ell,4}^{(\nu)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right].$$

It suffices then to show that

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

$$\left|\Pr\left[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right] - \Pr\left[\mathsf{iHyb}_{\ell,4}^{(\nu)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}\right]\right| = \mathsf{negl}(\lambda). \tag{A.5}$$

Lemma 5.21 then follows by the triangle inequality. Our proof strategy for showing Eqs. (A.4) and (A.5) will construct a sequence of hybrid experiment culminating in an *information-theoretic* step that ensures the adversary cannot tell that ℓ^{th} slot has switched from normal mode to semi-functional mode. These two information-theoretic components critically relies on *different* admissibility properties on the adversary:

- If event NonCorrupt occurs, then the adversary does not know the secret key $\mathsf{sk}_\ell = r_\ell$ associated with slot ℓ (i.e., $r_\ell \xleftarrow{\text{R}} \mathbb{Z}_N$ is the secret exponent the challenger sampled when responding to the c_ℓ^{th} key-generation query). The final information-theoretic argument (Lemma A.7) in the proof of Eq. (A.4) critically relies on the distribution of $r_\ell \bmod p_2$ being uniform and hidden from the view of the adversary. The full sequence of hybrids is described in the proof of Claim A.3.
- If event $\overline{\mathsf{NonCorrupt}}$ occurs, then the adversary may know the secret key $\mathsf{sk}_\ell = r_\ell$ associated with slot ℓ, and as such, we cannot rely on the same information-theoretic argument as above. In this case, the admissibility requirement ensures that the set of attributes S_ℓ associated with slot ℓ do *not* satisfy the challenge policy. The final information-theoretic argument (Lemma A.14) in the proof of Eq. (A.5) relies on information-theoretic security of the underlying linear secret sharing scheme. The full sequence of hybrids is described in the proof of Claim A.11.

Analysis for the case where slot ℓ is not corrupted. We now show that Eq. (A.4) holds. As noted previously, when the public key pk_ℓ associated with slot ℓ is not corrupted, our analysis will (eventually) rely on the secret key $\mathsf{sk}_\ell = r_\ell$ associated with slot ℓ being hidden to argue that the semi-functional slot components look computationally indistinguishable from normal slot components. We state the precise claim below:

Claim A.3. *Suppose Assumption 5.2b holds with respect to* CompGroupGen*. Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

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

- $\mathsf{ncHyb}_{\ell,0}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,3}^{(\nu)}$ except during the challenge phase, the challenger constructs the challenge ciphertext as follows:
  - If event NonCorrupt did not occur, then the experiment halts with output 0.
  - Otherwise, if event NonCorrupt occurs, let pk_ℓ be the public key associated with slot ℓ. Since NonCorrupt occurs, the public key pk_ℓ was constructed by the challenger in response to the c_ℓ^{th} key-generation query the adversary made in the query phase. Let $r_\ell \in \mathbb{Z}_N$ be the randomness the challenger used to construct pk_ℓ (i.e., this is the secret key stored in $\mathsf{Dict}[c_\ell]$). Then, $\mathsf{pk}_\ell = \mathsf{KeyGen}(\mathsf{crs}, \ell; r_\ell)$. The challenger constructs the challenge ciphertext exactly as in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$, except it computes C_5 as follows:

$$C_5 = (g_1 g_2)^{s\beta_1} (g_1 g_2)^{-s r_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right).$$

    The other components of the challenge ciphertext are constructed as in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$. The output of the experiment is the output of $\mathcal{A}$, exactly as in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$.

  Importantly, in this experiment, the only component that depends on the exponent $\delta_\ell \in \mathbb{Z}_N$ is P_ℓ. The challenge ciphertext no longer depends on δ_ℓ.

Hybrid	B_ℓ	P_ℓ	Justification	
$\mathsf{iHyb}_{\ell,3}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$((g_1 g_2)^s g_3)^{\delta_\ell}$		
$\mathsf{ncHyb}_{\ell,0}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$((g_1 g_2)^s g_3)^{\delta_\ell}$	Identical	Lemma A.4
$\mathsf{ncHyb}_{\ell,1}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1 g_2 g_3)^{\delta_\ell}$	Statistical	Lemma A.5
$\mathsf{ncHyb}_{\ell,2}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1 g_3)^{\delta_\ell}$	Assumption 5.2b	Lemma A.6
$\mathsf{ncHyb}_{\ell,3}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1 g_3)^{\delta_\ell}$	Statistical	Lemma A.7
$\mathsf{ncHyb}_{\ell,4}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1 g_2 g_3)^{\delta_\ell}$	Assumption 5.2b	Lemma A.9
$\mathsf{ncHyb}_{\ell,5}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$((g_1 g_2)^s g_3)^{\delta_\ell}$	Statistical	Lemma A.10
$\mathsf{iHyb}_{\ell,4}^{(\nu)}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$((g_1 g_2)^s g_3)^{\delta_\ell}$	Identical	Lemma A.10

Table 5: Structure of slot parameters B_ℓ, P_ℓ in the hybrid experiments for analyzing the NonCorrupt branch (Claim A.3). For each pair of adjacent hybrids, we indicate whether they are identically distributed, statistically indistinguishable, or computationally indistinguishable. The highlighted row is the information-theoretic step that relies on event NonCorrupt occurring (i.e., that the adversary does not know the secret key for slot ℓ).

- $\mathsf{ncHyb}_{\ell,1}^{(\nu)}$: Same as $\mathsf{ncHyb}_{\ell,0}^{(\nu)}$, except the challenger sets $P_\ell = (g_1 g_2 g_3)^{\delta_\ell}$ in the setup phase.
- $\mathsf{ncHyb}_{\ell,2}^{(\nu)}$: Same as $\mathsf{ncHyb}_{\ell,1}^{(\nu)}$ except the challenger sets $P_\ell = (g_1 g_3)^{\delta_\ell}$ in the setup phase.
- $\mathsf{ncHyb}_{\ell,3}^{(\nu)}$: Same as $\mathsf{ncHyb}_{\ell,2}^{(\nu)}$ except the challenger sets $B_\ell = g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$ in the setup phase.
- $\mathsf{ncHyb}_{\ell,4}^{(\nu)}$: Same as $\mathsf{ncHyb}_{\ell,3}^{(\nu)}$ except the challenger sets $P_\ell = (g_1 g_2 g_3)^{\delta_\ell}$ in the setup phase
- $\mathsf{ncHyb}_{\ell,5}^{(\nu)}$: Same as $\mathsf{ncHyb}_{\ell,4}^{(\nu)}$ except the challenger sets $P_\ell = ((g_1 g_2)^s g_3)^{\delta_\ell}$ in the setup phase

We provide a summary of the hybrid experiments in Table 5. We now show that each pair of adjacent hybrids are computationally indistinguishable.

Lemma A.4. *For all adversaries $\mathcal{A}$ and $\nu \in \{0,1\}$,* $\Pr[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \land \mathsf{NonCorrupt}] = \Pr[\mathsf{ncHyb}_{\ell,0}^{(\nu)}(\mathcal{A}) = 1]$.

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

$$Q_\ell = P_\ell^{r_\ell} = ((g_1 g_2)^s) g_3)^{\delta_\ell r_\ell}$$

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

$$\frac{R_\ell}{Q_\ell^{\delta_\ell^{-1}}} = \frac{g_3^{r_\ell}}{(g_1 g_2)^{r_\ell s} g_3^{r_\ell}} = (g_1 g_2)^{-r_\ell s}.$$

Thus, if event NonCorrupt occurs, then C_5 in $\mathsf{ncHyb}_{\ell,0}^{(\nu)}$ satisfies

$$C_5 = (g_1 g_2)^{s\beta_1} (g_1 g_2)^{-s r_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right) = (g_1 g_2)^{s\beta_1} \left(\frac{R_\ell}{Q_\ell^{\delta_\ell^{-1}}} \right) \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right) = (g_1 g_2)^{s\beta_1} \left(\prod_{i \in [L]} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right).$$

This is exactly the distribution of C_5 in $\mathsf{iHyb}_{\ell}^{(\nu)}(\mathcal{A})$. Therefore, conditioned on event NonCorrupt, the output distribution of $\mathsf{ncHyb}_{\ell,0}^{(\nu)}(\mathcal{A})$ is identical to the output distribution of $\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A})$. Correspondingly,

$$\begin{aligned}\Pr[\mathsf{ncHyb}_{\ell,0}^{(\nu)}(\mathcal{A}) = 1] &= \Pr[\mathsf{NonCorrupt}] \cdot \Pr[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \mid \mathsf{NonCorrupt}] \\ &= \Pr[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}],\end{aligned}$$

and the claim follows. □

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

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

Lemma A.6. *Suppose Assumption 5.2b holds with respect to* CompGroupGen*. Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $\nu \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

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

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

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

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

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the number of slots 1^L. Let $\mathcal{D} = \{d_i\}_{i \in [L]}$ be an efficiently-computable progression-free and double-free set. We define $f(i,j) := d_i + d_j$ and $\mathcal{E} := \{f(i,j) \mid i,j \in [L] : i \neq j\}$.

3. Algorithm $\mathcal{B}$ samples $\alpha, \beta_1, \beta_2, a \xleftarrow{\mathrm{R}} \mathbb{Z}_N$. It sets $Z = e(g_1, g_1)^{\alpha}$, $\beta = \beta_1 + \beta_2$, and $h = g_1^{\beta}$.

4. For each $i \in [L]$, algorithm $\mathcal{B}$ samples $\delta_i, \tau_i, \tau_i' \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and computes $t_i = a^{d_i}$.

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

   $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} (Y g_4)^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i}.$$

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

   $$A_\ell = g_1^{t_i} (g_2 g_3)^{\tau_i'} \quad , \quad B_\ell = g_1^{\alpha} A_\ell^{\beta} (g_3 g_4)^{\tau_\ell} \quad , \quad P_\ell = T.$$

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

   $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^{\alpha} A_i^{\beta} (g_3 g_4)^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i}.$$

Then, for each attribute $w \in \mathcal{U}_\lambda$, algorithm $\mathcal{B}$ samples $b_w \xleftarrow{\text{R}} \mathbb{Z}_N$. For each $w \in \mathcal{U}_\lambda$, slot index $i \in [L]$, and cross term index $z \in \mathcal{E}$, algorithm $\mathcal{B}$ then constructs the attribute-specific slot components $U_{i,w}$ and $W_{z,w}$ as in $\mathsf{ncHyb}_{\ell,1}^{(\nu)}$ $\mathsf{ncHyb}_{\ell,2}^{(\nu)}$:

$$U_{i,w} = g_1^{b_w \cdot t_i} \quad , \quad W_{z,w} = g_1^{b_w a^z}.$$

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

$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, g_4\,,\, \{(A_i, B_i, P_i)\}_{i \in [L]}\,,\, \{U_{i,w}, W_{z,w}\}_{w \in \mathcal{U}_\lambda, i \in [L], z \in \mathcal{E}}\right)$$

to the adversary $\mathcal{A}$. It also initializes a counter ctr = 0 and an (empty) dictionary Dict to keep track of the key-generation queries.

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

6. In the challenge phase, after the adversary specifies a challenge policy $P^* = (\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is an injective row-labeling function, two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$, algorithm $\mathcal{B}$ constructs pk_i as in $\mathsf{ncHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{ncHyb}_{\ell,2}^{(\nu)}$:

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

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

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

   - **Message-embedding components:** Set $C_1 = \mu_\nu^* \cdot e(g_1, X)^\alpha$ and $C_2 = X$.
   - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{\text{R}} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^\mathsf{T}$. For each $k \in [K]$, algorithm $\mathcal{B}$ samples $s_k, \eta_k \xleftarrow{\text{R}} \mathbb{Z}_N$. If $\rho(k) \in S_\ell$, algorithm $\mathcal{B}$ sets $C_{4,k} = (g_1 g_4)^{s_k}$. Otherwise, if $\rho(k) \notin S_\ell$, it sets $C_{4,k} = (X g_4)^{s_k}$. Next, algorithm $\mathcal{B}$ sets

   $$C_{3,k} = X^{\beta_2 \mathbf{m}_k^\mathsf{T} \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L]: \rho(k) \notin S_i} t_i} g_4^{\eta_k}$$

   - **Slot-specific component:** Algorithm $\mathcal{B}$ sets

   $$C_5 = X^{\beta_1 - r_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right)$$

   Algorithm $\mathcal{B}$ gives the challenge ciphertext to $\mathcal{A}$:

   $$\mathsf{ct}^* = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\right).$$

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

In the reduction algorithm, the exponent s_{12} plays the role of s in $\mathsf{ncHyb}^{(v)}_{\ell,1}$ and $\mathsf{ncHyb}^{(v)}_{\ell,2}$. Next, consider the distribution of B_i for $i < \ell$. As long as $s_{23} \neq 0 \bmod p_2$ and $s_{23} \neq 0 \bmod p_3$ (which holds with overwhelming probability over the choice of $s_{23} \xleftarrow{R} \mathbb{Z}_N$), then the distributions

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

are identical. Note this is the only place where τ_i is revealed in the $\mathbb{G}_2$ or $\mathbb{G}_3$ subgroups. Thus, with overwhelming probability over the choice of s_{23}, algorithm $\mathcal{B}$ constructs B_i according to the same distribution as $\mathsf{ncHyb}^{(v)}_{\ell,1}$ and $\mathsf{ncHyb}^{(v)}_{\ell,2}$. Finally, consider the distribution of P_ℓ:

- If $T = (g_1 g_2 g_3)^\delta$ for some $\delta \xleftarrow{R} \mathbb{Z}_N$, then algorithm $\mathcal{B}$ simulates the distribution in $\mathsf{ncHyb}^{(v)}_{\ell,1}$.
- If $T = (g_1 g_3)^\delta$ for some $\delta \xleftarrow{R} \mathbb{Z}_N$, then algorithm $\mathcal{B}$ simulates the distribution in $\mathsf{ncHyb}^{(v)}_{\ell,2}$.

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

Lemma A.7. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\left|\Pr[\mathsf{ncHyb}^{(v)}_{\ell,3}(\mathcal{A}) = 1] - \Pr[\mathsf{ncHyb}^{(v)}_{\ell,2}(\mathcal{A}) = 1]\right| = \mathsf{negl}(\lambda)$.

Proof. We show that the distributions $\mathsf{ncHyb}^{(v)}_{\ell,2}(\mathcal{A})$ and $\mathsf{ncHyb}^{(v)}_{\ell,3}(\mathcal{A})$ are statistically close. Let $(c_\ell, S_\ell, \mathsf{pk}^*_\ell)$ be the tuple the adversary chooses for slot ℓ during the challenge phase. Let $r_\ell \xleftarrow{R} \mathbb{Z}_N$ be the randomness the challenger used to answer the c_ℓ^{th} key-generation query. For either experiment to output 1, event NonCorrupt must occur, which means the adversary does *not* issue a corruption query on index c_ℓ. Correspondingly, the challenger *never* gives r_ℓ to the adversary. This property will be critical for arguing that the two distributions are statistically indistinguishable. Consider the distributions $\mathsf{ncHyb}^{(v)}_{\ell,2}(\mathcal{A})$ and $\mathsf{ncHyb}^{(v)}_{\ell,3}(\mathcal{A})$. By construction, the only difference between them is the distribution of component B_ℓ in the $\mathbb{G}_2$ subgroup:

$$\begin{aligned}\mathsf{ncHyb}^{(v)}_{\ell,2} &: B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell} \\ \mathsf{ncHyb}^{(v)}_{\ell,3} &: B_\ell = g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}\end{aligned}$$

In both experiments, $A_\ell = g_1^{t_\ell}(g_2 g_3)^{\tau'_\ell}$. Suppose that $\tau'_\ell \neq 0 \bmod p_2$. Since $\tau'_\ell \xleftarrow{R} \mathbb{Z}_N$, this holds with probability $1 - 1/p_2 = 1 - \mathsf{negl}(\lambda)$. Consider the following relabeling of the variables β_1 and r_ℓ in $\mathsf{ncHyb}^{(v)}_{\ell,2}$:

- Let $\sigma \in \mathbb{Z}_N$ be the unique value where $\sigma = 0 \bmod p_1 p_3 p_4$ and $\sigma = (\tau'_\ell)^{-1}\tau_\ell \bmod p_2$.
- Suppose we now set $\beta_1 = \beta'_1 + \sigma$ and $r_\ell = r'_\ell + \sigma$ where $\beta'_1, r' \xleftarrow{R} \mathbb{Z}_N$. Observe that the distribution of β_1 and r_ℓ is still uniform over $\mathbb{Z}_N$ under this relabeling.

Consider now the other components in the adversary's view in $\mathsf{ncHyb}^{(v)}_{\ell,2}$ with the above relabeling. It suffices to only consider the components that depend on β_1 or r_ℓ since the other components are unchanged. Note also that by design, $\beta_1 = \beta'_1 \bmod p_1 p_3 p_4$ and $r_\ell = r'_\ell \bmod p_1 p_3 p_4$.

- Consider the components in the common reference string. First, $h = g_1^{\beta_1+\beta_2} = g_1^{\beta'_1+\beta_2}$. Next $A_i = g_1^{t_i} g_3^{\tau'_i}$ for all $i \neq \ell$ and $A_\ell = g_1^{t_\ell}(g_2 g_3)^{\tau'_\ell}$. Consider the distribution of each B_i:
  - If $i < \ell$, then $B_i = g_1^\alpha A_i^{\beta_1+\beta_2}(g_2 g_3 g_4)^{\tau_i} = g_1^\alpha A_i^{\beta'_1+\beta_2}(g_2 g_3 g_4)^{\tau_i}$.
  - If $i = \ell$, then
$$B_\ell = g_1^\alpha A_\ell^{\beta_1+\beta_2}(g_3 g_4)^{\tau_\ell} = g_1^\alpha g_1^{t_\ell(\beta_1+\beta_2)} g_2^{\tau'_\ell(\beta_1+\beta_2)} g_3^{\tau'_\ell(\beta_1+\beta_2)} (g_3 g_4)^{\tau_\ell} = g_1^\alpha A_\ell^{\beta'_1+\beta_2}(g_2 g_3 g_4)^{\tau_\ell},$$
    since $\beta_1 = \beta'_1 \bmod p_1 p_3 p_4$ and $\beta_1 = \beta'_1 + (\tau'_\ell)^{-1}\tau_\ell \bmod p_2$.
  - If $i > \ell$, then $B_i = g_1^\alpha A_i^{\beta_1+\beta_2}(g_3 g_4)^{\tau_i} = g_1^\alpha A_i^{\beta'_1+\beta_2}(g_3 g_4)^{\tau_i}$.

The remaining components in the CRS do not depend on either β_1 or r_ℓ and are thus unchanged.

- Consider the components in the key-generation queries. The only key-generation query that is affected by this change of variables is the c_ℓ^{th} query. When the adversary makes the c_ℓ^{th} key-generation query, the challenger constructs the public key $\mathsf{pk}_\ell = (T_\ell, Q_\ell, R_\ell, \{V_{j,\ell}\}_{j \neq \ell})$ using randomness r_ℓ. Under the above substitution this means $T_\ell = g_1^{r_\ell} = g_1^{r'_\ell}$, $Q_\ell = P_\ell^{r_\ell} = P_\ell^{r'_\ell}$, $R_\ell = g_3^{r_\ell} = g_3^{r'_\ell}$, and $V_{j,\ell} = A_j^{r_\ell} = A_j^{r'_\ell}$ for all $j \neq \ell$ since $r_\ell = r'_\ell \bmod p_1 p_3$, and the components P_ℓ and A_j for $j \neq \ell$ do not contain any non-trivial components in the $\mathbb{G}_2$ subgroup. Here, it is critical that $P_\ell = (g_1 g_3)^{\delta_\ell}$ in $\mathsf{ncHyb}_{\ell,2}^{(v)}$ does *not* contain any components in $\mathbb{G}_2$.
- Finally, consider the components in the challenge ciphertext. The components $C_1, C_2, C_{3,k}, C_{4,k}$ for $k \in [K]$ are all unchanged (i.e., they are independent of β_1 and r_ℓ). Consider now ciphertext component C_5. In $\mathsf{ncHyb}_{\ell,2}^{(v)}$,

$$C_5 = (g_1 g_2)^{s\beta_1} (g_1 g_2)^{-s r_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right) = (g_1 g_2)^{s\beta'_1} (g_1 g_2)^{-s r'_\ell} \left(\prod_{i \in [L] \setminus \{\ell\}} \frac{R_i}{Q_i^{\delta_i^{-1}}} \right),$$

since $\beta_1 = \beta'_1 \bmod p_1$ and $r_\ell = r'_\ell \bmod p_1$, and

$$s\beta_1 - s r_\ell = s(\beta'_1 + \sigma) - s(r'_\ell + \sigma)) = s\beta'_1 - s r'_\ell \bmod p_2.$$

Observe now that this is precisely the distribution in $\mathsf{ncHyb}_{\ell,3}^{(v)}$ (with the relabeling $\beta_1 \mapsto \beta'_1$ and $r_\ell \mapsto r'_\ell$). Thus, whenever $\tau'_\ell \neq 0 \bmod p_2$, hybrids $\mathsf{ncHyb}_{\ell,2}^{(v)}$ and $\mathsf{ncHyb}_{\ell,3}^{(v)}$ are identically distributed. Since this holds with probability $1 - \mathsf{negl}(\lambda)$ over the choice of τ'_ℓ, the claim holds. Note that this argument critically relies on the fact that r_ℓ is not given to the adversary in the game, as this allows us to reinterpret r_ℓ as $r'_\ell = r_\ell + \sigma$. □

Lemma A.8. *Suppose Assumption 5.2b holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. This follows by a similar argument as in the proof of Lemma A.6. □

Lemma A.9. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$ and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,* $\left| \Pr[\mathsf{ncHyb}_{\ell,5}^{(v)}(\mathcal{A}) = 1] - \Pr[\mathsf{ncHyb}_{\ell,4}^{(v)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda)$.

Proof. This follows by a similar argument as in the proof of Lemma A.5. □

Lemma A.10. *For all adversaries $\mathcal{A}$ and all $v \in \{0,1\}$,* $\Pr[\mathsf{iHyb}_{\ell,4}^{(v)}(\mathcal{A}) = 1 \wedge \mathsf{NonCorrupt}] = \Pr[\mathsf{ncHyb}_{\ell,5}^{(v)}(\mathcal{A}) = 1]$.

Proof. This follows by a similar argument as in the proof of Lemma A.4. □

Combining Lemmas A.4 to A.10, Claim A.3 now follows by a hybrid argument. □

Analysis for the case where slot ℓ is corrupted. Next, we show that Eq. (A.5) holds. As noted previously, when slot ℓ is corrupted (and the adversary knows the associated secret key), we are guaranteed that the set of attributes S_ℓ associated with slot ℓ does *not* satisfy the challenge policy. Our analysis here will (eventually) rely on the security of the linear secret sharing scheme to argue that that the semi-functional slot components look computationally indistinguishable from normal slot components. We state the precise claim below:

Claim A.11. *Suppose Assumption 5.2b and Assumption 5.2c holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $b \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. Similar to the proof of Claim A.3, we introduce an additional sequence of hybrid experiments:

- $\mathsf{cHyb}_{\ell,0}^{(\nu)}$: Same as $\mathsf{iHyb}_{\ell,3}^{(\nu)}$ except during the challenge phase, when constructing the challenge ciphertext, the challenger performs several additional checks:
  - If event $\mathsf{NonCorrupt}$ occurs, then the experiment halts with output 0.
  - Let pk_ℓ be the public key associated with slot ℓ and $S_\ell \subseteq \mathcal{U}_\lambda$ be the set of associated attributes. Let $P^* = (\mathbf{M}, \rho)$ be the challenge policy where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is an injective row-labeling function. Let $I = \{k \in [K] : \rho(k) \in S_\ell\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes in S_ℓ, and let $\mathbf{M}_I$ be the corresponding submatrix of $\mathbf{M}$. Since event $\mathsf{NonCorrupt}$ does not occur, this means that S_ℓ does not satisfy the policy $(\mathbf{M}, \rho)$, so $\mathbf{e}_1^{\mathsf{T}}$ is *not* in the row-span of $\mathbf{M}_I$. This means that there exists a vector $\mathbf{v}^* \in \mathbb{Z}_N^n$ such that $\mathbf{M}_I \mathbf{v}^* = \mathbf{0} \bmod N$ and $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* \neq 0 \bmod N$. In this experiment, the challenger computes $\mathbf{v}^* \in \mathbb{Z}_N^n$ using Gaussian elimination.
  - If $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$, the experiment halts with output 0.

  The rest of the experiment proceeds as in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$.

- $\mathsf{cHyb}_{\ell,1}^{(\nu)}$: Same as $\mathsf{cHyb}_{\ell,0}^{(\nu)}$ except the challenger changes how it constructs the $C_{3,k}$ components in the challenger ciphertext:
  - Sample $\xi \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and $\hat{v}_2', \ldots, \hat{v}_n' \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $\hat{\mathbf{v}}' = [\beta_2 - \xi v_1^*, \hat{v}_2', \ldots, \hat{v}_n']^{\mathsf{T}}$.
  - For each $k \in [K]$, sample $\eta_k \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and set $C_{3,k} = ((g_1 g_2)^s)^{\mathbf{m}_k^{\mathsf{T}}(\hat{\mathbf{v}}' + \xi \mathbf{v}^*)} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k}$

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

- $\mathsf{cHyb}_{\ell,2}^{(\nu)}$: Same as $\mathsf{cHyb}_{\ell,1}^{(\nu)}$ except the challenger sets $B_\ell = g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$ in the setup phase.

- $\mathsf{cHyb}_{\ell,3}^{(\nu)}$: Same as $\mathsf{cHyb}_{\ell,2}^{(\nu)}$, except the challenger changes how it constructs the $C_{3,k}$ components in the challenge ciphertext:
  - Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and let $\mathbf{v}' = [1, v_2, \ldots, v_n]^{\mathsf{T}}$.
  - For each $k \in [K]$, sample $\eta_k \xleftarrow{\mathrm{R}} \mathbb{Z}_N$ and set $C_{3,k} = ((g_1 g_2)^s)^{\beta_2 \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} C_{4,k}^{-b_{\rho(k)} \sum_{i \in [L] : \rho(k) \notin S_i} t_i} g_4^{\eta_k}$

We provide a summary of the hybrid experiments in Table 6. We now show that each pair of adjacent hybrids are computationally indistinguishable.

Lemma A.12. *Suppose Assumption 5.2d holds with respect to* $\mathsf{CompGroupGen}$*. Then, for all* $\ell \in [L]$*, all efficient adversaries* $\mathcal{A}$*, and all* $\nu \in \{0, 1\}$*, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

$$\left| \Pr[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}] - \Pr[\mathsf{cHyb}_{\ell,0}^{(\nu)}(\mathcal{A}) = 1] \right| = \mathsf{negl}(\lambda).$$

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

$$\left| \Pr[\mathsf{iHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1 \wedge \overline{\mathsf{NonCorrupt}}] - \Pr[\mathsf{cHyb}_{\ell,0}^{(\nu)}(\mathcal{A}) = 1] \right| = \varepsilon$$

for some non-negligible ε. Since these two experiments are identical except the additional check of whether $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$, this means that with probability at least ε, algorithm $\mathcal{A}$ outputs a challenge $(\mathbf{M}, \rho)$ such that $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* \neq 0 \bmod N$ but $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$, where $\mathbf{v}^*$ is the vector derived from $(\mathbf{M}, \rho)$ according to the specification of $\mathsf{cHyb}_{\ell,0}^{(\nu)}$. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ that outputs a nontrivial factor of N given the inputs of Lemma 5.3 with probability $\varepsilon - \mathsf{negl}(\lambda)$. Security can in turn be based on the hardness of Assumption 5.2d (Lemma 5.3).

Hybrid	B_ℓ	$C_{3,k}$	Justification	
$\mathsf{iHyb}^{(\nu)}_{\ell,3}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1 g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k}$		
$\mathsf{cHyb}^{(\nu)}_{\ell,0}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1 g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k}$	Assumption 5.2d	Lemma A.12
$\mathsf{cHyb}^{(\nu)}_{\ell,1}$	$g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$	$(g_1 g_2)^{s\mathbf{m}_k^\top (\hat{\mathbf{v}}' + \xi \mathbf{v}^*)} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k}$	Identical	Lemma A.13
$\mathsf{cHyb}^{(\nu)}_{\ell,2}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1 g_2)^{s\mathbf{m}_k^\top (\hat{\mathbf{v}}' + \xi \mathbf{v}^*)} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k}$	Statistical	Lemma A.14
$\mathsf{cHyb}^{(\nu)}_{\ell,3}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1 g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k}$	Identical	Lemma A.15
$\mathsf{iHyb}^{(\nu)}_{\ell,4}$	$g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$	$(g_1 g_2)^{s\beta_2 \mathbf{m}_k^\top \mathbf{v}'} ((g_1 g_2)^s g_4)^{-s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k}$	Assumption 5.2d	Lemma A.16

Table 6: Structure of the slot parameter B_ℓ and the challenge ciphertext component $C_{3,k}$ (for $\rho(k) \notin S_\ell$) in the hybrid experiments for analyzing the $\overline{\mathsf{NonCorrupt}}$ branch (Claim A.11). For each pair of adjacent hybrids, we indicate whether they are identically distributed, statistically indistinguishable, or computationally indistinguishable. The highlighted row is the information-theoretic step that relies on event $\overline{\mathsf{NonCorrupt}}$ occurring (i.e., that the set of attributes S_ℓ associated with slot ℓ does *not* satisfy the challenge policy P^*). Note that two of the hybrid experiments either introduce or remove an abort condition ($\mathsf{cHyb}^{(\nu)}_{\ell,0}$ and $\mathsf{cHyb}^{(\nu)}_{\ell,4}$) *without* changing the distribution of B_ℓ, and $C_{3,k}$.

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

2. Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$ and receives the number of slots 1^L. Let $\mathcal{D} = \{d_i\}_{i\in[L]}$ be an efficiently-computable progression-free and double-free set. We define $f(i,j) := d_i + d_j$ and $\mathcal{E} := \{f(i,j) \mid i,j \in [L] : i \neq j\}$.

3. Algorithm $\mathcal{B}$ samples $\alpha, \beta_1, \beta_2, a \xleftarrow{R} \mathbb{Z}_N$. It sets $Z = e(g_1, g_1)^\alpha$, $\beta = \beta_1 + \beta_2$, and $h = g_1^\beta$.

4. For each $i \in [L]$, algorithm $\mathcal{B}$ samples $\delta_i, \tau_i, \tau_i' \xleftarrow{R} \mathbb{Z}_N$ and set $t_i = a^{d_i}$

   - For $i < \ell$, algorithm $\mathcal{B}$ sets
   $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^\alpha A_i^\beta (Y g_4)^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i} .$$
   - For $i = \ell$, algorithm $\mathcal{B}$ sets
   $$A_\ell = g_1^{t_\ell} Y^{\tau_\ell'} \quad , \quad B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell} \quad , \quad P_\ell = (X g_3)^{\delta_\ell} .$$
   - For $i \geq \ell$, algorithm $\mathcal{B}$ sets
   $$A_i = g_1^{t_i} g_3^{\tau_i'} \quad , \quad B_i = g_1^\alpha A_i^\beta (g_3 g_4)^{\tau_i} \quad , \quad P_i = (X g_3)^{\delta_i} .$$

   For each attribute $w \in \mathcal{U}_\lambda$, it samples $b_w \xleftarrow{R} \mathbb{Z}_N$. In addition, for each $w \in \mathcal{U}_\lambda$, slot index $i \in [L]$, and cross term index $z \in \mathcal{E}$, algorithm $\mathcal{B}$ constructs the attribute-specific slot components $U_{i,w}$ and $W_{z,w}$ as in $\mathsf{iHyb}^{(\nu)}_{\ell,3}$:
   $$U_{i,w} = g_1^{b_w t_i} \quad , \quad W_{z,w} = g_1^{b_w a^z} .$$
   Algorithm $\mathcal{B}$ gives the common reference string
   $$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g_1\,,\, h\,,\, g_3\,,\, g_4\,,\, \{(A_i, B_i, P_i)\}_{i\in[L]}\,,\, \{U_{i,w}, W_{z,w}\}_{w\in\mathcal{U}_\lambda, i\in[L], z\in\mathcal{E}}\right)$$
   to the adversary $\mathcal{A}$. It also initializes a counter $\mathsf{ctr} = 0$ and an (empty) dictionary Dict to keep track of the key-generation queries.

5. In the query phase, algorithm $\mathcal{B}$ responds to the adversary's queries as in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$ and $\mathsf{cHyb}_{\ell,0}^{(\nu)}$. Namely, when algorithm $\mathcal{A}$ makes a key-generation query on a slot i, algorithm $\mathcal{B}$ increments the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{R} \mathbb{Z}_N$. It then computes $T_i = g_1^{r_i}$, $Q_i = P_i^{r_i}$, $R_i = g_3^{r_i}$, and $V_{j,i} = A_j^{r_i}$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It defines $\mathsf{sk}_{\mathsf{ctr}} = r_i$ and adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$ to the dictionary Dict. If the adversary $\mathcal{A}$ makes a corruption query on an index $1 \leq i \leq \mathsf{ctr}$, the challenger looks up the entry $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{Dict}[i]$ and replies to $\mathcal{A}$ with sk'.

6. In the challenge phase, after $\mathcal{A}$ specifies the challenge policy $P^* = (\mathbf{M}, \rho)$, the messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$, and for each slot $i \in [L]$, a tuple $(c_i, S_i, \mathsf{pk}_i^*)$. Algorithm $\mathcal{B}$ computes $\mathbf{v}^*$ from $\mathbf{M}$ as described in $\mathsf{cHyb}_{\ell,0}^{(\nu)}$, and outputs $\gcd(N, \mathbf{e}_1^{\mathsf{T}} \mathbf{v}^*)$.

The exponent $s_{12} \in \mathbb{Z}_N$ plays the role of $s \in \mathbb{Z}_N$ in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$ and $\mathsf{cHyb}_{\ell,0}^{(\nu)}$. Next, consider the distribution of B_i for $i < \ell$ as well as the distribution of A_ℓ. As long as s_{23} is coprime to $p_2 p_3$, (which holds with overwhelming probability over the choice of $s_{23} \xleftarrow{R} \mathbb{Z}_N$), then the distributions of B_i and A_ℓ are distributed exactly as required in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$ and $\mathsf{cHyb}_{\ell,0}^{(\nu)}$. All remaining components are simulated exactly as in $\mathsf{iHyb}_{\ell,3}^{(\nu)}$ and $\mathsf{cHyb}_{\ell,0}^{(\nu)}$, so with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, algorithm $\mathcal{A}$ outputs $(\mathbf{M}, \rho)$ such that $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* \neq 0 \bmod N$ but $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* = 0 \bmod p_2$. In this case, $\gcd(N, \mathbf{e}_1^{\mathsf{T}} \mathbf{v}^*)$ yields a non-trivial factor of N, and algorithm $\mathcal{B}$ wins the game in Lemma 5.3 with the same advantage. □

Lemma A.13. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $\nu \in \{0, 1\}$, $\Pr[\mathsf{cHyb}_{\ell,1}^{(\nu)}(\mathcal{A}) = 1] = \Pr[\mathsf{cHyb}_{\ell,0}^{(\nu)}(\mathcal{A}) = 1]$.*

Proof. Without loss of generality, we can assume that NonCorrupt does not occur and moreover, $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* \neq 0 \bmod p_2$. Otherwise, the output in both experiments is 0. The only difference between the two distributions is the distribution of the challenge ciphertext components $C_{3,k}$. In $\mathsf{cHyb}_{\ell,1}^{(\nu)}$, the challenger replaces $\beta_2 \mathbf{v}'$ with $\hat{\mathbf{v}}' + \xi v^*$. By definition, in $\mathsf{cHyb}_{\ell,1}^{(\nu)}$,

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

where $\hat{\mathbf{v}}'' = [1, \hat{v}_2'', \ldots, \hat{v}_n'']$, and the distribution of $\hat{v}_2'', \ldots, \hat{v}_n''$ are independent and uniform over $\mathbb{Z}_N$ (since $\hat{v}_2', \ldots, \hat{v}_n' \xleftarrow{R} \mathbb{Z}_N$). This is precisely the distribution of $C_{3,k}$ in $\mathsf{cHyb}_{\ell,0}^{(\nu)}$. □

Lemma A.14. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $\nu \in \{0, 1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. We show that the distributions $\mathsf{cHyb}_{\ell,1}^{(\nu)}(\mathcal{A})$ and $\mathsf{cHyb}_{\ell,2}^{(\nu)}(\mathcal{A})$ are statistically indistinguishable. This argument will rely on the fact that the attributes S_ℓ associated with slot ℓ do *not* satisfy the challenge policy. By construction, the only difference between the two experiments is the distribution of component B_ℓ in the $\mathbb{G}_2$ subgroup. In $\mathsf{cHyb}_{\ell,1}^{(\nu)}$, $B_\ell = g_1^\alpha A_\ell^\beta (g_3 g_4)^{\tau_\ell}$ while in $\mathsf{cHyb}_{\ell,2}^{(\nu)}$, $B_\ell = g_1^\alpha A_\ell^\beta (g_2 g_3 g_4)^{\tau_\ell}$. In both experiments, $A_\ell = g_1^{t_\ell} (g_2 g_3)^{\tau_\ell'} = g^{a^{d_\ell}} (g_2 g_3)^{\tau_\ell}$. We start by defining a few quantities that will be useful in our analysis:

- Let $P^* = (\mathbf{M}, \rho)$ be the challenge policy where $\mathbf{M} \in \mathbb{Z}_N^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is the injective row-labeling function.
- Let $S_\ell \subseteq \mathcal{U}_\lambda$ be the set of attributes associated with slot ℓ, and let $I = \{k \in [K] : \rho(k) \in S_\ell\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes in S_ℓ. Let $\mathbf{M}_I$ be the corresponding submatrix of $\mathbf{M}$.
- Let $\mathbf{v}^* \in \mathbb{Z}_N^n$ be the vector where $\mathbf{M}_I \mathbf{v}^* = \mathbf{0} \bmod p_2$ and $\mathbf{e}_1^{\mathsf{T}} \mathbf{v}^* \neq 0 \bmod p_2$.

Moreover, we have the following:

- Since the challenger in the two experiments sample $s_k, \tau_\ell' \xleftarrow{R} \mathbb{Z}_N$, it follows that $\tau_\ell' \neq 0 \bmod p_2$ and $s_k \neq 0 \bmod p_2$ with overwhelming probability.

- Next, we show that $\sum_{i \in [L]:\rho(k) \notin S_i} t_i = \sum_{i \in [L]:\rho(k) \notin S_i} a^{d_i} \neq 0 \bmod p_2$ for all $k \in I$. By construction, in experiments $\mathsf{cHyb}_{\ell,1}^{(\nu)}$ and $\mathsf{cHyb}_{\ell,2}^{(\nu)}$, the value of $a \bmod p_2$ is information-theoretically hidden to the view of $\mathcal{A}$ (specifically, the quantities $A_i, U_{i,w}, W_{z,q}$ that depend on a are only given out in the $\mathbb{G}_1$ subgroup). In particular, this means that the labeling function ρ as well as the attribute sets S_i for all $i \in [L]$ chosen by the adversary are *independent* of the value of $a \bmod p_2$. In fact, the challenger in these two experiments can defer the sampling of $a \bmod p_2$ until after the adversary chooses ρ and S_i for all $i \in [L]$. Since the challenger samples $a \xleftarrow{\text{R}} \mathbb{Z}_N$, the value of $a \bmod p_2$ is uniform over $\mathbb{Z}_{p_2}$. By the Schwartz-Zippel lemma, the probability that $\sum_{i \in [L]:\rho(k) \notin S_i} a^{d_i} = 0$ is then at most $\max(\mathcal{D})/p_2 = \mathsf{poly}(L)/p_2 = \mathsf{negl}(\lambda)$. The claim now follows by a union bound over all indices $k \in I$.

Consider the following relabeling of the variables in $\mathsf{cHyb}_{\ell,1}^{(\nu)}$:

- Let $\sigma^{(\beta)} \in \mathbb{Z}_N$ be the unique value where $\sigma^{(\beta)} = 0 \bmod p_1 p_3 p_4$ and $\sigma^{(\beta)} = (\tau'_\ell)^{-1} \tau_\ell \bmod p_2$. Suppose we write $\beta_2 = \beta'_2 + \sigma^{(\beta)}$ for some $\beta'_2 \xleftarrow{\text{R}} \mathbb{Z}_N$.
- Let $\sigma^{(\xi)} \in \mathbb{Z}_N$ be the unique value where $\sigma^{(\xi)} = 0 \bmod p_1 p_3 p_4$ and $\sigma^{(\xi)} = (v_1^*)^{-1} \sigma^{(\beta)} \bmod p_2$. Suppose we write $\xi = \xi' + \sigma^{(\xi)}$ for some $\xi' \xleftarrow{\text{R}} \mathbb{Z}_N$.
- For each $k \in [K]$, let $\sigma_k^{(b)}$ be the unique value where
$$\sigma_k^{(b)} = 0 \bmod p_1 p_3 p_4 \quad \text{and} \quad \sigma_k^{(b)} = \left(s_k \sum_{i \in [L]:\rho(k) \notin S_i} t_i \right)^{-1} \sigma^{(\xi)} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}^* \bmod p_2.$$
Suppose we write $b_{\rho(k)} = b'_{\rho(k)} + \sigma_k^{(b)}$ for some $b'_{\rho(k)} \xleftarrow{\text{R}} \mathbb{Z}_N$.

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

- Consider the components in the common reference string. First, $h = g_1^{\beta_1 + \beta_2} = g_1^{\beta_1 + \beta'_2}$. Next $A_i = g_1^{t_i} g_3^{\tau'_\ell}$ for all $i \neq \ell$ and $A_\ell = g_1^{t_\ell} (g_2 g_3)^{\tau'_\ell}$. Consider the distribution of each B_i:
  - If $i < \ell$, then $B_i = g_1^\alpha A_i^{\beta_1 + \beta_2} (g_2 g_3 g_4)^{\tau_i} = g_1^\alpha A_i^{\beta_1 + \beta'_2} (g_2 g_3 g_4)^{\tau_i}$.
  - If $i = \ell$, then
  $$\begin{aligned} B_\ell = g_1^\alpha A_\ell^{\beta_1 + \beta_2} (g_3 g_4)^{\tau_\ell} &= g_1^\alpha g_1^{t_\ell(\beta_1 + \beta_2)} g_2^{\tau'_\ell(\beta_1 + \beta_2)} g_3^{\tau'_\ell(\beta_1 + \beta_2)} (g_3 g_4)^{\tau_\ell} \\ &= g_1^\alpha A_\ell^{\beta_1 + \beta'_2} (g_2 g_3 g_4)^{\tau_\ell}, \end{aligned}$$
  since $\beta_2 = \beta'_2 \bmod p_1 p_3 p_4$ and $\beta_2 = \beta'_2 + (\tau'_\ell)^{-1} \tau_\ell \bmod p_2$.
  - If $i > \ell$, then $B_i = g_1^\alpha A_i^{\beta_1 + \beta_2} g_3^{\tau_i} = g_1^\alpha A_i^{\beta_1 + \beta'_2} (g_3 g_4)^{\tau_i}$.

  Consider the slot components $U_{i,\rho(k)}$ and $W_{z,\rho(k)}$ for each $i \in [L]$, $z \in \mathcal{E}$, and $k \in [K]$. By definition,
  $$\begin{aligned} U_{i,\rho(k)} &= g_1^{b_{\rho(k)} t_i} = g_1^{b'_{\rho(k)} t_i} \\ W_{z,\rho(k)} &= g_1^{b_{\rho(k)} a^z} = g_1^{b'_{\rho(k)} a^z}. \end{aligned}$$

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

- Finally, consider the components in the challenge ciphertext. The components $C_1, C_2, C_{4,k}, C_5$ for $k \in [K]$ are all unchanged (i.e., they are independent of $\beta_2, \xi, b_{\rho(k)}$). It suffices to consider the ciphertext components $C_{3,k}$. First, since $\beta_2 = \beta_2' + \sigma^{(\beta)}$ and $\xi = \xi' + (v_1^*)^{-1}\sigma^{(\beta)}$, we have

$$\hat{\mathbf{v}}' = [\beta_2 - \xi v_1^*, \hat{v}_2', \dots, \hat{v}_n'] = [\beta_2' - \xi' v_1^*, \hat{v}_2', \dots, \hat{v}_n'] \bmod N.$$

  We now consider two possibilities:

  - Suppose $\rho(k) \in S_\ell$. This means that $k \in I$ so $\mathbf{m}_k^\mathsf{T}\mathbf{v}^* = 0 \bmod N$. Moreover, $C_{4,k} = (g_1 g_4)^{s_k}$, so we can write $C_{3,k}$ as

$$\begin{aligned} C_{3,k} &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi\mathbf{v}^*)} C_{4,k}^{-b_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k} \\ &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}\hat{\mathbf{v}}'} C_{4,k}^{-b_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k} \\ &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi'\mathbf{v}^*)} C_{4,k}^{-b'_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k}. \end{aligned}$$

  - Suppose $\rho(k) \notin S_\ell$. First, we have

$$\xi\mathbf{m}_k^\mathsf{T}\mathbf{v}^* - s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i = \xi'\mathbf{m}_k^\mathsf{T}\mathbf{v}^* - s_k b'_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i \bmod p_1p_3p_4.$$

    We show the same relation also holds modp_2. Under our change of variables, we have

$$\begin{aligned} \xi\mathbf{m}_k^\mathsf{T}\mathbf{v}^* - s_k b_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i &= \left(\xi' + \sigma^{(\xi)}\right)\mathbf{m}_k^\mathsf{T}\mathbf{v}^* - s_k\left(b'_{\rho(k)} + \left(s_k \sum_{i\in[L]:\rho(k)\notin S_i} t_i\right)^{-1} \sigma^{(\xi)}\mathbf{m}_k^\mathsf{T}\mathbf{v}^*\right) \sum_{i\in[L]:\rho(k)\notin S_i} t_i \\ &= \xi'\mathbf{m}_k^\mathsf{T}\mathbf{v}^* - s_k b'_{\rho(k)} \sum_{i\in[L]:\rho(k)\notin S_i} t_i \pmod{p_2}. \end{aligned}$$

    Since $\rho(k) \notin S_\ell$, we have that $C_{4,k} = ((g_1g_2)^s g_4)^{s_k}$. We can now rewrite $C_{3,k}$ as

$$\begin{aligned} C_{3,k} &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi\mathbf{v}^*)} C_{4,k}^{-b_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k} \\ &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi\mathbf{v}^*)} ((g_1g_2)^s g_4)^{-s_k b_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k} \\ &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi\mathbf{v}^*) - s_k b_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k - s_k b_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} \\ &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi'\mathbf{v}^*) - s_k b'_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k - s_k b'_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} \\ &= ((g_1g_2)^s)^{\mathbf{m}_k^\mathsf{T}(\hat{\mathbf{v}}' + \xi'\mathbf{v}^*)} C_{4,k}^{-b'_{\rho(k)}\sum_{i\in[L]:\rho(k)\notin S_i} t_i} g_4^{\eta_k} \end{aligned}$$

Observe now with this relabeling of variables, we have recovered the ciphertext distribution in $\mathsf{cHyb}_{\ell,2}^{(\nu)}$ (with randomness β_2', ξ' and $b'_{\rho(k)}$). Thus, the distributions $\mathsf{cHyb}_{\ell,2}^{(\nu)}$ and $\mathsf{cHyb}_{\ell,1}^{(\nu)}$ are statistically indistinguishable. □

Lemma A.15. *For all $\ell \in [L]$, all adversaries $\mathcal{A}$, and all $\nu \in \{0,1\}$,* $\Pr[\mathsf{cHyb}_{\ell,3}^{(\nu)}(\mathcal{A}) = 1] = \Pr[\mathsf{cHyb}_{\ell,2}^{(\nu)}(\mathcal{A}) = 1]$.

Proof. This follows by the same argument as in the proof of Lemma A.13. □

Lemma A.16. *Suppose Assumption 5.2d holds with respect to* CompGroupGen. *Then, for all $\ell \in [L]$, all efficient adversaries $\mathcal{A}$, and all $\nu \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that*

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

Proof. This follows by a similar argument as in the proof of Lemma A.12. □

Combining Lemmas A.12 to A.16, Claim A.11 now follows by a hybrid argument. □

By Claims A.3 and A.11, Eqs. (A.4) and (A.5) both hold. Lemma 5.21 now follows by the triangle inequality. □

B Statistically-Secure Registered ABE without Progression-Free Sets

For completeness (and comparison purposes; see Section 6) we describe an analogue of our statically-secure registered ABE scheme (Construction 4.3) *without* progression-free sets. This allows us to base security on the less structured parallel Diffie-Hellman exponent assumption from [Wat11]; on the flip side, the size of the CRS is *quadratic* in the number of users (but still independent of the size of the attribute universe). This construction highlights our approach for reducing the CRS size by employing a partitioning-based proof strategy (which can be leveraged independently of using progression-free sets).

Parallel Diffie-Hellman exponent assumption. We start by reviewing the parallel Diffie-Hellman exponent assumption introduced by Waters [Wat11] in the context of constructing ciphertext-policy attribute-based encryption (in the centralized model).

Assumption B.1 (Parallel Diffie-Hellman Exponent Assumption [Wat11, adapted])**.** Let $\mathsf{PrimeGroupGen}$ be a prime-order group generator. For a security parameter λ and a bit $b \in \{0,1\}$, we define the (q_1, q_2)-parallel Diffie-Hellman exponent game between an adversary $\mathcal{A}$ and a challenger as follows:

1. The challenger starts by sampling $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$. It also samples exponents exponents $a, s, \beta_1, \dots \beta_{q_2} \xleftarrow{\text{R}} \mathbb{Z}_p$. Then the challenger computes the following quantities:
   - Let $Y = g^s$.
   - For each $i \in [2q_1] \setminus \{q_1\}$, let $X_i = g^{a^i}$.
   - For each $j \in [q_2]$, let $Y^{(j)} = g^{s\beta_j}$, and for each $i \in [2q_1] \setminus \{q_1\}$, let $X_i^{(j)} = g^{a^i/\beta_j}$.
   - For each $i \in [2q_1] \setminus \{q_1\}$ and $j, k \in [q_2]$ where $j \neq k$, let $Z_i^{(j,k)} = g^{a^i s \beta_k / \beta_j}$.

   Finally the challenger computes $T_0 = e(g,g)^{a^{q_1} s}$ and samples $T_1 \xleftarrow{\text{R}} \mathbb{G}_T$. It then gives the following challenge to the adversary
   $$\left(\mathcal{G}\,,\, g\,,\, Y\,,\, \{X_i\}_{i\in[2q_1]\setminus\{q_1\}}\,,\, \{Y^{(j)}, X_i^{(j)}\}_{j\in[q_2], i\in[2q_1]\setminus\{q_1\}}\,,\, \{Z_i^{(j,k)}\}_{j\neq k, i\in[2q_1]\setminus\{q_1\}}\,,\, \boxed{T_b}\right).$$

2. The adversary outputs a bit $b' \in \{0,1\}$ which is the output of the experiment.

We say that the (q_1, q_2)-Parallel Diffie-Hellman Exponent assumption holds with respect to $\mathsf{PrimeGroupGen}$ if for all efficient adversaries $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\lambda \in \mathbb{N}$,

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

in the (q_1, q_2)-parallel Diffie-Hellman exponent game.

Remark B.2 (Comparison to [Wat11])**.** Note that the original assumption formulation from [Wat11, §2.4.1] corresponds to the particular case where $q_1 = q_2$. Allowing different values q_1, q_2 simplifies the analysis of our construction. For completeness, we prove that our variant of the parallel BDHE assumption holds in the generic bilinear group model in Appendix D (Lemma D.7).

Slotted registered ABE without progression-free sets. We now give the analog of Construction 4.3 without relying on progression-free sets. This construction can also be viewed as a prime-order analog of the composite-order scheme from [HLWW23].

Construction B.3 (Slotted Attribute-Based Registration-Based Encryption)**.** Let $\mathsf{PrimeGroupGen}$ be a prime-order bilinear group generator, let $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda\in\mathbb{N}}$ be a (polynomial-size) attribute space, and let $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda\in\mathbb{N}}$ be a set of policies that can be described by a linear secret sharing scheme (Definition 2.2) over $\mathcal{U}$, where each policy $P \in \mathcal{P}_\lambda$ is defined over a maximum of $K = K(\lambda)$ attributes. We construct a slotted attribute-based registration-based encryption scheme $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{IsValid}, \mathsf{Aggregate}, \mathsf{Encrypt}, \mathsf{Decrypt})$ with attribute space $\mathcal{U}$ and policy space $\mathcal{P}$ as follows:

- $\mathsf{Setup}(1^\lambda, 1^L)$: On input the security parameter λ, and the number of slots L, the setup algorithm starts by sampling $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$. The setup algorithm now constructs the following quantities:
  - Sample random exponents $a \xleftarrow{R} \mathbb{Z}_p$ and set $\alpha = -a^{L+1}$. Compute $h = \prod_{i \in [L]} g^{a^{L+1-i}}$.
  - For each slot index $i \in [L]$, sample exponents $u_i, \delta_i \xleftarrow{R} \mathbb{Z}_p$ and let $t_i = a^i$. Then, define the following group elements:
$$A_i = g^{t_i} \quad , \quad B_i = g^\alpha h^{t_i} \quad , \quad P_i = g^{\delta_i} \quad , \quad U_i = g^{u_i}.$$
    Then, for each $i, j \in [L]$ where $i \neq j$, compute the "cross term" $W_{i,j} = g^{t_i u_j}$.
  - Finally, let $Z = e(g, g)^\alpha$. Output the common reference string
$$\mathsf{crs} = \left(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}\,,\, \{W_{i,j}\}_{j \neq i}\right) \tag{B.1}$$

  The associated message space $\mathcal{M}_\lambda$ is defined to be $\mathcal{M}_\lambda := \mathbb{G}_T$.

- $\mathsf{KeyGen}(\mathsf{crs}, i)$: This is the same algorithm as in Construction 4.3. Namely, on input the common reference string crs (with components given by Eq. (B.1)) and a slot index $i \in [L]$, the key-generation algorithm samples $r_i \xleftarrow{R} \mathbb{Z}_p$ and computes
$$T_i = g^{r_i} \quad , \quad Q_i = P_i^{r_i}.$$
  Then for each $j \neq i$, it computes the cross terms $V_{j,i} = A_j^{r_i}$. Finally, it outputs the public key pk_i and the secret key sk_i defined as follows:
$$\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i}) \quad \text{and} \quad \mathsf{sk}_i = r_i.$$
  Note that this key-generation algorithm does *not* depend on the set of attributes.

- $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$: This is the same algorithm as in Construction 4.3. Namely, on input the common reference string crs (with components given by Eq. (B.1)), a slot index $i \in [L]$, and a purported public key $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$, the key-validation algorithm first affirms that each of the components in pk_i is a valid group element (i.e., an element in $\mathbb{G}$). If so, it then checks
$$e(T_i, P_i) = e(g, Q_i)$$
  Next, for each $j \neq i$, the algorithm checks that
$$e\left(g, V_{j,i}\right) = e(T_i, A_j)$$
  If all checks pass, it outputs 1; otherwise, it outputs 0.

- $\mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$: On input the common reference string crs (with components given by Eq. (B.1)), a collection of L public keys $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$ together with their attribute sets $S_i \subseteq \mathcal{U}_\lambda$, the aggregation algorithm starts by computing the attribute-independent public key $\hat{T}$ and the attribute-independent slot key $\hat{V}_i$ for each $i \in [L]$:
$$\hat{T} = \prod_{j \in [L]} T_j \quad \text{and} \quad \hat{V}_i = \prod_{j \neq i} V_{i,j}.$$
  Next, for each attribute $w \in \mathcal{U}_\lambda$, it computes the attribute-specific public key $\hat{U}_w$ and the attribute-specific slot key $\hat{W}_{i,w}$ for each $i \in [L]$:
$$\hat{U}_w = \prod_{j \in [L] : w \notin S_j} U_j \quad \text{and} \quad \hat{W}_{i,w} = \prod_{j \neq i : w \notin S_j} W_{i,j}.$$
  Finally, it outputs the master public key mpk and the slot-specific helper decryption keys hsk_i where
$$\mathsf{mpk} = \left(\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}\right) \quad \text{and} \quad \mathsf{hsk}_i = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}_\lambda}\right).$$

- $\mathsf{Encrypt}(\mathsf{mpk}, (\mathbf{M}, \rho), \mu)$: On input the master public key $\mathsf{mpk} = (\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda})$, a policy $(\mathbf{M}, \rho)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function, and a message $\mu \in \mathbb{G}_T$, the encryption algorithm starts by sampling a secret exponent $s \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and $h_1, h_2 \xleftarrow{\mathrm{R}} \mathbb{G}$ such that $h = h_1 h_2$. Then, it constructs the ciphertext components as follows:
  - **Message-embedding components:** First, let $C_1 = \mu \cdot Z^s$ and $C_2 = g^s$.
  - **Attribute-specific components:** Sample $v_2, \ldots, v_n \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ for the linear secret sharing scheme and let $\mathbf{v} = [1, v_2, \ldots, v_n]^{\mathsf{T}}$. Then, for each $k \in [K]$, sample $s_k \xleftarrow{\mathrm{R}} \mathbb{Z}_p$ and let $C_{3,k} = h_2^{s \mathbf{m}_k^{\mathsf{T}} \mathbf{v}} \hat{U}_{\rho(k)}^{-s_k}$ and $C_{4,k} = g^{s_k}$, where $\mathbf{m}_k^{\mathsf{T}} \in \mathbb{Z}_p^n$ denotes the kth row of $\mathbf{M}$.
  - **Slot-specific component:** Set $C_5 = (h_1 \hat{T}^{-1})^s$

  It then outputs the ciphertext

$$\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big).$$

- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input the secret key $\mathsf{sk} = r$, the helper key $\mathsf{hsk} = \big(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w \in \mathcal{U}_\lambda}\big)$, where $\mathsf{mpk} = (\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w \in \mathcal{U}_\lambda})$, and the ciphertext $\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big)$ where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function, the decryption algorithm proceeds as follows:
  - If the set of attributes S_i is not authorized by $(\mathbf{M}, \rho)$, then the decryption algorithm outputs $\perp$.
  - Otherwise, let $I = \{k \in [K] : \rho(k) \in S_i\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes $S_i \subseteq \mathcal{U}_\lambda$. Write the elements as $I = \{k_1, \ldots, k_{|I|}\}$.
  - Let $\mathbf{M}_{S_i}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I. Since S_i is authorized, let $\boldsymbol{\omega}_{S_i} \in \mathbb{Z}_p^{|I|}$ be a vector such that $\boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} = \mathbf{e}_1^{\mathsf{T}}$.
  - Then, compute and output

$$\frac{C_1}{e(C_2, B_i)} \cdot \underbrace{e(C_5, A_i) \cdot e(C_2, A_i^r \hat{V}_i)}_{D_{\mathsf{slot}}} \cdot \underbrace{\prod_{1 \le j \le |I|} \Big(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)})\Big)^{\omega_{S_i,j}}}_{D_{\mathsf{attrib}}}. \tag{B.2}$$

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

Correctness. We now show that Construction B.3 satisfies completeness, correctness, compactness, and incremental aggregation.

Theorem B.4 (Completeness). *Construction B.3 is complete.*

Proof. Take any security parameter $\lambda \in \mathbb{N}$ and the number of slots $L \in \mathbb{N}$. Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$. Then, we can write

$$\mathsf{crs} = \big(\mathcal{G}\,,\ Z\,,\ g\,,\ h\,,\ \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}\,,\ \{W_{i,j}\}_{j \neq i}\big)\,.$$

Take any index $i \in [L]$ and let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. By construction of KeyGen, we can write $\mathsf{pk}_i = \big(T_i, Q_i, \{V_{j,i}\}_{j \neq i}\big)$, where

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

for some $r_i \in \mathbb{Z}_N$. We now consider each of the pairing checks in IsValid:

- $e(T_i, P_i) = e(g^{r_i}, P_i) = e(g, P_i^{r_i}) = e(g, Q_i)$.
- $e(g, V_{j,i}) = e(g, A_j^{r_i}) = e(g^{r_i}, A_j) = e(T_i, A_j)$.

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

Theorem B.5 (Correctness)**.** *Construction B.3 is correct.*

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

- Let $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L)$ where $\mathsf{crs} = \big(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i\in[L]}\,,\, \{W_{i,j}\}_{j\neq i}\big)$. By construction, the slot components can be written as $A_i = g^{t_i}$, $B_i = g^{\alpha} h^{t_i}$, and $P_i = g^{\delta_i}$. The attribute components can be written as $U_i = g^{u_i}$ and $W_{i,j} = g^{t_i u_j}$ (where $t_i = a^i$).
- Let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyGen}(\mathsf{crs}, i)$. Then, we can write $\mathsf{sk}_i = r_i$ and $\mathsf{pk}_i = \big(T_i, Q_i, \{V_{j,i}\}_{j\neq i}\big)$ where
$$T_i = g^{r_i} \quad , \quad Q_i = P_i^{r_i} \quad , \quad V_{j,i} = A_j^{r_i} = g^{t_j r_i}. \tag{B.3}$$
- Take any set of public keys $\{\mathsf{pk}_j\}_{j\neq i}$ where $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$ holds. Since pk_j satisfies the $\mathsf{IsValid}$ predicate, we can write $\mathsf{pk}_j = \big(T_j, Q_j, \{V_{\ell,j}\}_{\ell\neq j}\big)$.
- For each $j \in [L]$, let $S_j \subseteq \mathcal{U}_\lambda$ be the attributes associated with pk_j.
- Let $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) \leftarrow \mathsf{Aggregate}(\mathsf{crs}, (\mathsf{pk}_1, S_1), \ldots, (\mathsf{pk}_L, S_L))$. Then, the master public key mpk and the i^{th} slot-specific helper decryption key hsk_i can then be written as follows:
$$\mathsf{mpk} = \big(\mathcal{G}, g, h, Z, \hat{T}, \{\hat{U}_w\}_{w\in\mathcal{U}_\lambda}\big) \quad \text{and} \quad \mathsf{hsk}_i = \big(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i, \{\hat{W}_{i,w}\}_{w\in\mathcal{U}_\lambda}\big),$$
where $\hat{T} = \prod_{j\in[L]} T_j$, $\hat{V}_i = \prod_{j\neq i} V_{i,j}$, and
$$\hat{U}_w = \prod_{j\in[L]:w\notin S_j} U_j = \prod_{j\in[L]:w\notin S_j} g^{u_j}$$
$$\hat{W}_{i,w} = \prod_{j\neq i:w\notin S_j} W_{i,j} = \prod_{j\neq i:w\notin S_j} g^{t_i u_j}$$
- Let $(\mathbf{M}, \rho)$ be the challenge policy where $\mathbf{M} \in \mathbb{Z}_p^{K\times n}$ and $\rho\colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function. Take any message $\mu \in \mathbb{G}_T$. The challenge ciphertext ct can be written as
$$\mathsf{ct} = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k\in[K]}, C_5\big)$$
where $C_1 = \mu \cdot Z^s$, $C_2 = g^s$, $C_{3,k} = h_2^{\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s_k}$, $C_{4,k} = g^{s_k}$, and $C_5 = h_1^s \hat{T}^{-s}$.

We now show that $\mathsf{Decrypt}(\mathsf{sk}_i, \mathsf{hsk}_i, \mathsf{ct})$ outputs μ. Let $I = \{k \in [K] : \rho(k) \in S_i\}$ be the indices of the rows of $\mathbf{M}$ associated with the attributes S_i. Write the elements of I as $I = \{k_1, \ldots, k_{|I|}\}$. Let $\mathbf{M}_{S_i}$ be the matrix formed by taking the subset of rows in $\mathbf{M}$ indexed by I, and let $\boldsymbol{\omega}_{S_i} \in \mathbb{Z}_N^{|I|}$ be a vector such that $\boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} = \mathbf{e}_1^{\mathsf{T}}$. We break up the decryption relation (Eq. (B.2)) into several pieces and analyze them individually:

- **Policy check:** First, consider $D_{\mathsf{attrib}} = \prod_{1\leq j\leq|I|} \big(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)})\big)^{\omega_{S_i,j}}$. By definition,
$$e(C_{3,k_j}, A_i) = e\left(h_2^{s\mathbf{m}_{k_j}^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k_j)}^{-s_{k_j}}, g^{t_i}\right) = e(h_2, g)^{st_i\mathbf{m}_{k_j}^{\mathsf{T}}\mathbf{v}} \prod_{\ell\in[L]:\rho(k_j)\notin S_\ell} e(g,g)^{-s_{k_j} t_i u_\ell}$$
$$e\big(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}\big) = \prod_{\ell\neq i:\rho(k_j)\notin S_\ell} e\left(g^{s_{k_j}}, W_{i,\ell}\right) = \prod_{\ell\neq i:\rho(k_j)\notin S_\ell} e(g,g)^{t_i u_\ell s_{k_j}}$$
By construction, $\rho(k_j) \in S_i$, so the latter terms cancel out and we can write
$$e(C_{3,k_j}, A_i) e\big(C_{4,k_j}, \hat{W}_{i,\rho(k_j)}\big) = e(h_2, g)^{st_i\mathbf{m}_{k_j}^{\mathsf{T}}\mathbf{v}}.$$

Finally noting that $\mathbf{e}_1^{\mathsf{T}}\mathbf{v} = 1$, we have

$$\begin{aligned} D_{\mathsf{attrib}} &= \prod_{1 \le j \le |I|} \left(e(C_{3,k_j}, A_i) \cdot e(C_{4,k_j}, \hat{W}_{i,\rho(k_j)})\right)^{\omega_{S_i,j}} \\ &= e(h_2, g)^{st_i \sum_{1 \le j \le |I|} \omega_{S_i,j} \mathbf{m}_{k_j}^{\mathsf{T}} \mathbf{v}} \\ &= e(h_2, g)^{st_i \boldsymbol{\omega}_{S_i}^{\mathsf{T}} \mathbf{M}_{S_i} \mathbf{v}} \\ &= e(h_2, g)^{st_i \mathbf{e}_1^{\mathsf{T}} \mathbf{v}} = e(h_2, g)^{st_i}. \end{aligned}$$

- **Slot check:** Next, consider the component $D_{\mathsf{slot}} = e(C_5, A_i) e(C_2, A_i^{r_i} \hat{V}_i)$. By definition

$$e(C_5, A_i) = e\left(h_1^s \hat{T}^{-s}, g^{t_i}\right) = e(h_1, g)^{st_i} \prod_{j \in [L]} e(T_j, g)^{-st_i} = e(h_1, g)^{st_i} \prod_{j \in [L]} e(T_j, A_i)^{-s}$$
$$e(C_2, A_i^{r_i} \hat{V}_i) = e\left(g^s, g^{r_i t_i} \hat{V}_i\right) = e(g, g)^{s r_i t_i} \prod_{j \neq i} e(g, V_{i,j})^s.$$

  Now, since we know for all $j \in [L]$, $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$, we have that for all $j \neq i$, $e(g, V_{i,j}) = e(T_j, A_i)$. Thus, using Eq. (B.3), we can now write

$$\begin{aligned} D_{\mathsf{slot}} = e(C_5, A_i) e(C_2, A_i^{r_i} \hat{V}_i) &= \left(e(h_1, g)^{st_i} e(T_i, A_i)^{-s} \prod_{j \neq i} e(T_j, A_i)^{-s}\right) \left(e(g, g)^{s r_i t_i} \prod_{j \neq i} e(g, V_{i,j})^s\right) \\ &= e(h_1, g)^{st_i} e(T_i, A_i)^{-s} e(g, g)^{s r_i t_i} \\ &= e(h_1, g)^{st_i} e(g^{r_i}, g^{t_i})^{-s} e(g, g)^{s r_i t_i} = e(h_1, g)^{st_i} \end{aligned}$$

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

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

  Next, we can see that have

$$e(C_2, B_i) = e(g^s, g^\alpha h^{t_i}) = e(g, g)^{\alpha s} e(h, g)^{st_i}.$$

  Thus, putting everything together, Eq. (B.2) becomes

$$\frac{C_1 \cdot D_{\mathsf{slot}} \cdot D_{\mathsf{attrib}}}{e(C_2, B_i)} = \frac{\mu \cdot e(g, g)^{\alpha s} e(h, g)^{st_i}}{e(g, g)^{\alpha s} e(h, g)^{st_i}} = \mu$$

□

Theorem B.6 (Compactness). *Construction B.3 is compact.*

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

Theorem B.7 (Incremental Aggregation). *Construction B.3 supports f-incremental aggregation for $f(L, |\mathcal{U}_\lambda|) = O(L \cdot |\mathcal{U}_\lambda|)$.*

Proof. We construct the AggregateUpdate algorithm as follows:

- $\mathsf{AggregateUpdate}(\mathsf{crs}, \mathsf{st}, (\mathsf{pk}, S))$: On input the common reference string

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

  a state st (which could be $\perp$), and a public key (pk, S) (or the special symbol $\perp$), the update algorithm proceeds as follows:

1. If st $= \perp$, then the update algorithm initializes $k = 0$ and $\hat{T}^{(k)} = 1$, $\hat{V}_i^{(k)} = 1$ for all $i \in [L]$, $\hat{U}_w^{(k)} = 1$ for all $w \in \mathcal{U}_\lambda$, and $\hat{W}_{i,w}^{(k)} = 1$ for all $i \in [L]$ and $w \in \mathcal{U}_\lambda$. Otherwise, the update algorithm parses

$$\mathsf{st} = \left(k\ ,\ \hat{T}^{(k)}\ ,\ \{\hat{V}_i^{(k)}\}_{i\in[L]}\ ,\ \{\hat{U}_w^{(k)}\}_{w\in\mathcal{U}_\lambda}\ ,\ \{\hat{W}_{i,w}^{(k)}\}_{i\in[L],w\in\mathcal{U}_\lambda}\right)$$

2. If $(\mathsf{pk}, S) = \perp$, then the algorithm outputs

$$\mathsf{mpk} = \left(\mathcal{G}, g, h, Z, \hat{T}^{(k)}, \{\hat{U}_w^{(k)}\}_{w\in\mathcal{U}_\lambda}\right) \quad , \quad \forall i \in [L] : \mathsf{hsk}_i = \left(\mathsf{mpk}, i, S_i, A_i, B_i, \hat{V}_i^{(k)}, \{\hat{W}_{i,w}^{(k)}\}_{w\in\mathcal{U}_\lambda}\right).$$

3. Otherwise, the update algorithm parses $\mathsf{pk} = \left(T_{k+1}, Q_{k+1}, \{V_{i,k+1}\}_{i\neq k+1}\right)$ and updates the state as follows:
   - $\hat{T}^{(k+1)} = \hat{T}^{(k)} \cdot T_{k+1}$.
   - For each $i \in [L]$, if $i \neq k+1$ then $\hat{V}_i^{(k+1)} = \hat{V}_i^{(k)} \cdot V_{i,k+1}$. Otherwise, if $i = k+1$, then set $\hat{V}_i^{(k+1)} = \hat{V}_i^{(k)}$.
   - For each $w \in \mathcal{U}_\lambda$, if $w \notin S_{k+1}$, then $\hat{U}_w^{(k+1)} = \hat{U}_w^{(k)} \cdot U_{k+1}$. Otherwise, if $w \in S_{k+1}$, then $\hat{U}_w^{(k+1)} = \hat{U}_w^{(k)}$.
   - For each $i \in [L]$ and $w \in \mathcal{U}_\lambda$, if $i \neq k+1$ and $w \notin S_{k+1}$, then $\hat{W}_{i,w}^{(k+1)} = \hat{W}_{i,w}^{(k)} \cdot W_{i,k+1}$. Otherwise, set $\hat{W}_{i,w}^{(k+1)} = \hat{W}_{i,w}^{(k)}$.
4. Output the updated state

$$\mathsf{st} = \left(k+1\ ,\ \hat{T}^{(k+1)}\ ,\ \{\hat{V}_i^{(k+1)}\}_{i\in[L]}\ ,\ \{\hat{U}_w^{(k+1)}\}_{w\in\mathcal{U}_\lambda}\ ,\ \{\hat{W}_{i,w}^{(k+1)}\}_{i\in[L],w\in\mathcal{U}_\lambda}\right).$$

To complete the proof, we show that this incremental aggregation procedure implements the same behavior as the standard aggregation procedure. Specifically, we show inductively that for all $k \leq L$, the following properties hold for the elements in the AggregateUpdate algorithm:

- $\hat{T}^{(k)} = \prod_{j\in[k]} T_j$.
- For all $i \in [L]$, $\hat{V}_i^{(k)} = \prod_{j\in[k]\setminus\{i\}} V_{i,j}$.
- For all $w \in \mathcal{U}_\lambda$, $\hat{U}_w^{(k)} = \prod_{j\in[k]:w\notin S_j} U_j$.
- For all $i \in [L]$ and $w \in \mathcal{U}_\lambda$, $\hat{W}_{i,w}^{(k)} = \prod_{j\in[k]\setminus\{i\}:w\notin S_j} W_{i,j}$.

By construction, all of these properties hold for $k = 0$. Moreover, the inductive step follows by inspection: namely, each of the updates in Step 3 simply multiplies in the next component into the product (if present). When $k = L$, the components $\hat{T}^{(L)}$, $\hat{V}_i^{(L)}$, $\hat{U}_w^{(L)}$, and $\hat{W}_{i,w}^{(L)}$ precisely coincide with the quantities in the Aggregate algorithm. Finally, the intermediate state st always contains $O(L \cdot |\mathcal{U}_\lambda|)$ group elements, which proves the claim. □

Theorem B.8 (Static Security). *Let L be a bound on the number of slots. Let $q_1 = L+1$ and $q_2 = L \cdot K$. If the (q_1, q_2)-parallel bilinear Diffie-Hellman exponent assumption (Assumption B.1) holds with respect to* PrimeGroupGen, *then Construction B.3 is statically secure (for up to L slots).*

Proof. Similar to Section 4, our security proof relies on a partitioning strategy where we program the indices of the corrupted slots into the common reference string. We begin by defining a sequence of hybrid experiments. Each of our experiments is parameterized by a bit $\nu \in \{0,1\}$ (and implicitly, by the security parameter λ).

- $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$: This is the real security game where the challenger encrypts message μ_b^*. We recall the main steps here:
  - **Setup phase:** In the setup phase, the adversary $\mathcal{A}$ specifies the number of slots 1^L and the indices of the corrupted slots $C \subseteq [L]$. In the following, we also define the indices of the non-corrupted slots as $\mathcal{N} := [L] \setminus C$. The challenger then samples the common reference string crs according to the specification of Setup:
    - The challenger initializes a counter ctr $= 0$ and an (empty) dictionary Dict.
    - The challenger samples $\mathcal{G} = (\mathbb{G}, \mathbb{G}_T, p, g, e) \leftarrow \mathsf{PrimeGroupGen}(1^\lambda)$.

* The challenger samples $a \xleftarrow{R} \mathbb{Z}_p$ and set $\alpha = -a^{L+1}$. It also computes $h = \prod_{i \in [L]} g^{a^{L+1-i}}$. Then, for each slot $i \in [L]$, the challenger samples $u_i, \delta_i \xleftarrow{R} \mathbb{Z}_p$, and sets $t_i = a^i$. Then, it defines the following group elements:

$$A_i = g^{t_i} \quad , \quad B_i = g^{\alpha} h^{t_i} \quad , \quad P_i = g^{\delta_i} \quad , \quad U_i = g^{u_i}$$

  Then, for each $i, j \in [L]$ where $i \neq j$, it also sets $W_{i,j} = g^{t_i u_j}$.
* Finally, compute $Z \leftarrow e(g,g)^{\alpha}$. The challenger constructs the common reference string

$$\mathsf{crs} = \big(\mathcal{G}\,,\, Z\,,\, g\,,\, h\,,\, \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}\,,\, \{W_{i,j}\}_{j \neq i}\big) \tag{B.4}$$

  and gives crs to $\mathcal{A}$.

– **Query phase:** The challenger responds to the adversary's key-generation queries as follows:
  * **Key-generation query:** Whenever algorithm $\mathcal{A}$ makes a key-generation query on a non-corrupted slot index $i \in \mathcal{N}$, the challenger starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i \xleftarrow{R} \mathbb{Z}_p$. It then computes $T_i = g^{r_i}$, $Q_i = P_i^{r_i}$, and $V_{j,i} = A_j^{r_i}$ for $j \neq i$. The challenger sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}})$ to the dictionary Dict.

  Recall that in the static security game, the adversary is not allowed to make any corruption queries.

– **Challenge phase:** In the challenge phase, the adversary specifies a challenge policy $P^* = (\mathbf{M}, \rho) \in \mathcal{P}_\lambda$, where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function and two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$.[11] In addition, the adversary specifies a key for for each slot $i \in [L]$ as follows:
  * For each corrupted slot $i \in C$ the adversary specifies a public key $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$ and an attribute set S_i. The challenger checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$ and halts with output $\perp$ if not. Specifically, the challenger checks that $e(T_i, P_i) = e(g, Q_i)$ and for each $j \neq i$, that $e(g, V_{j,i}) = e(T_i, A_j)$.
  * For each non-corrupted slot $i \in \mathcal{N}$, the adversary specifies an index $c_i \in [\mathsf{ctr}]$. The challenger looks up the entry $\mathsf{Dict}[c_i] = (i', \mathsf{pk}')$. If $i = i'$, the challenger sets $\mathsf{pk}_i = \mathsf{pk}'$. If $\mathsf{pk}_i \neq \mathsf{pk}'$, the challenger halts with output $\perp$.

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

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

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

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

  The challenger then constructs the challenge ciphertext by sampling a secret exponent $s \xleftarrow{R} \mathbb{Z}_p$ and $h_1, h_2 \xleftarrow{R} \mathbb{G}$ such that $h = h_1 h_2$. It constructs the ciphertext components as follows:
  * **Message-embedding components:** First, let $C_1 = \mu_v^* \cdot Z^s$ and $C_2 = g^s$.
  * **Attribute-specific components:** Sample $v_2, \dots, v_n \xleftarrow{R} \mathbb{Z}_p$ for the linear secret sharing scheme and let $\mathbf{v} = [1, v_2, \dots, v_n]^{\mathsf{T}}$. Then, for each $k \in [K]$, sample $s_k \xleftarrow{R} \mathbb{Z}_p$, let $C_{3,k} = h_2^{s \mathbf{m}_k^{\mathsf{T}} \mathbf{v}} \hat{U}_{\rho(k)}^{-s_k}$ and $C_{4,k} = g^{s_k}$, where $\mathbf{m}_k^{\mathsf{T}}$ denotes the kth row of $\mathbf{M}$.
  * **Slot-specific component:** Let $C_5 = (h_1 \hat{T}^{-1})^s$.

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

$$\mathsf{ct}^* = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big).$$

[11] As in the proof of Theorem 4.8, we will assume that $\mathbf{M}$ has exactly K rows (which we can ensure by padding $\mathbf{M}$ with all-zero rows).

- **Output phase:** At the end of the game, the adversary outputs a bit $v' \in \{0,1\}$, which is also the output of the experiment.

- $\mathsf{Hyb}_1^{(v)}$: Same as $\mathsf{Hyb}_0^{(v)}$, except the challenger makes the sfollowing *syntactic* changes:
  - **Setup phase:** In the setup phase, the challenger samples $\beta_{i,k} \xleftarrow{\text{R}} \mathbb{Z}_p$ for each $i \in [L], k \in [K]$. Then it sets
  $$u_i = \sum_{k \in [K]} \frac{1}{\beta_{i,k}} a^{L+1-i}$$
  for all $i \in [L]$. Finally, instead of sampling the encryption randomness $s \in \mathbb{Z}_p$ in the challenge phase, the challenger now samples $s \xleftarrow{\text{R}} \mathbb{Z}_p$ in the setup phase. For the corrupted slots $i \in C$, the challenger now sets $P_i = g^{s\delta_i}$ (instead of $P_i = g^{\delta_i}$).
  - **Query phase:** When responding to a key-generation query for a slot $i \in \mathcal{N}$, instead of sampling $r_i \xleftarrow{\text{R}} \mathbb{Z}_p$, the challenger samples $r_i' \xleftarrow{\text{R}} \mathbb{Z}_p$ and sets $r_i = a^{L+1-i} + r_i'$.
  - **Challenge phase:** After the adversary outputs its challenge policy $P^* = (\mathbf{M}, \rho)$, the challenger computes for each $i \in C$ a vector $\mathbf{v}_i^* \in \mathbb{Z}_p^n$ with first entry 1 and which is orthogonal to every row $\mathbf{m}_k^{\mathsf{T}}$ of $\mathbf{M}$ where $\rho(k) \in S_i$. Note that such a vector exists (see also Definition 2.2) since the attributes in S_i (for a corrupted slot) do *not* satisfy the challenge policy $P^* = (\mathbf{M}, \rho)$. When generating the challenge ciphertext, the challenger generates the attribute-specific components $C_{3,k}$ and $C_{4,k}$ as well as the slot-specific component C_5 using the following modified procedure:
    - **Attribute-specific components:** The challenger sets
    $$s_k^* = s \cdot \sum_{i \in C : \rho(k) \notin S_i} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*$$
    and constructs the attribute-specific components as
    $$C'_{3,k} \leftarrow g^{s\mathbf{m}_k^{\mathsf{T}} \sum_{i \in C} a^{L+1-i} \mathbf{v}_i^*} \cdot \hat{U}_{\rho(k)}^{-s_k^*} \quad \text{and} \quad C'_{4,k} = g^{s_k^*}$$
    - **Slot-specific component:** The challenger sets the slot-specific component as
    $$C'_5 = g^{s \cdot \sum_{i \in \mathcal{N}} a^{L+1-i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}}$$

  Finally, the challenger rerandomizes the attribute-specific and slot-specific ciphertext components using the following rerandomization procedure:

  $\mathsf{Rerand}\big(\{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k \in [K]}, C'_5\big)$:

  1. Sample $\gamma, v_2', v_3', \ldots v_n' \xleftarrow{\text{R}} \mathbb{Z}_p$ and set $\mathbf{v}' = [1, v_2', v_3', \ldots, v_n']$ and $s_k' \xleftarrow{\text{R}} \mathbb{Z}_p$ for each $k \in [K]$.
  2. Compute the rerandomized ciphertext:
  $$C_{3,k} = C'_{3,k} \cdot g^{\gamma \mathbf{m}_k^{\mathsf{T}} \mathbf{v}'} \cdot \hat{U}_{\rho(k)}^{-s_k'} \quad \text{and} \quad C_{4,k} = C'_{4,k} \cdot g^{s_k'} \quad \text{and} \quad C_5 = C'_5 g^{-\gamma}.$$
  3. Output $\big(\{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big)$.

  Figure 2: Ciphertext rerandomization algorithm.

  The challenger then computes
  $$\big(\{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big) = \mathsf{Rerand}\big(\{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k \in [K]}, C'_5\big)$$
  and gives the rerandomized ciphertext to the adversary:
  $$\mathsf{ct}^* = \big((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\big).$$

  - **Output phase:** At the end of the game, the adversary outputs a bit $v' \in \{0,1\}$, which is also the output of the experiment.

- $\mathsf{Hyb}_{\mathsf{rand}}^{(v)}$: Same as $\mathsf{Hyb}_1^{(v)}$ except when constructing the challenge ciphertext, the challenger samples $C_1 \xleftarrow{R} \mathbb{G}_T$. Importantly, this distribution is *independent* of the message.

For a hybrid experiment Hyb and an adversary $\mathcal{A}$, we write $\mathsf{Hyb}(\mathcal{A})$ to denote the output distribution of an execution of Hyb with adversary $\mathcal{A}$. In the following, we argue that each the output distribution of each adjacent pair of hybrid is indistinguishable.

Lemma B.9. *For all adversaries $\mathcal{A}$ and all $v \in \{0,1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. We show that $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$ and $\mathsf{Hyb}_1^{(v)}$ are statistically close by showing that the adversary's view (i.e., the crs from the setup phase, the public keys in the query phase, and the challenge ciphertext ct^* in the challenge phase) in the two distributions is statistically close. We consider each phase separately.

Setup phase. The only difference is how the challenger samples u_i and P_i. In $\mathsf{Hyb}_1^{(v)}$, the challenger sets

$$u_i = \sum_{k \in [K]} \frac{1}{\beta_{i,k}} a^{L+1-i}.$$

where $\beta_{i,k} \xleftarrow{R} \mathbb{Z}_p$ for all $i \in [L]$ and $k \in [K]$. We consider the distribution of u_i:

- Since the challenger samples $a \xleftarrow{R} \mathbb{Z}_p$ and p is prime, the probability that $a^{L+1-i} = 0 \bmod p$ is at most $(L+1-i)/p$ (since a polynomial of degree $L+1-i$ can have at most $L+1-i$ roots over $\mathbb{Z}_p$). Since $p = 2^{-\Omega(\lambda)}$, we conclude that $(L+1-i)/p = \mathsf{negl}(\lambda)$ and so a^{L+1-i} is non-zero with overwhelming probability.
- Since each $\beta_{i,k}$ is uniform over $\mathbb{Z}_p$, they are non-zero with overwhelming probability. In this case, the distribution of each $\beta_{i,k}^{-1}$ is independent and uniform over $\mathbb{Z}_p$.

Thus, with overwhelming probability, $a^{L+1-i} \neq 0$ for all i and each $\beta_{i,k}^{-1}$ is an independent and uniform (non-zero) value over $\mathbb{Z}_p$. We conclude that the distribution of u_i is statistically close to independently uniform over $\mathbb{Z}_p$, which is the distribution of u_i in $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$. Next, consider the distribution of P_i for $i \in C$. In $\mathsf{Hyb}_0^{(v)}$, the challenger sets $P_i = g^{\delta_i}$ while in $\mathsf{Hyb}_1^{(v)}$, the challenger sets $P_i = g^{s\delta_i}$, where $s \xleftarrow{R} \mathbb{Z}_p$ and $\delta_i \xleftarrow{R} \mathbb{Z}_p$. As long as $s \neq 0$, these two distributions are identical. Since s is sampled uniformly, these two distributions are statistically close.

Query phase. The only change is how the challenger samples r_i for $i \in \mathcal{N}$. In $\mathsf{Hyb}_{\mathsf{real}}^{(v)}$, the challenger samples $r_i \xleftarrow{R} \mathbb{Z}_p$. In $\mathsf{Hyb}_1^{(v)}$, the challenger samples $r_i' \xleftarrow{R} \mathbb{Z}_p$ and sets

$$r_i = a^{L+1-i} + r_i'.$$

These two distributions are identical.

Challenge phase. In $\mathsf{Hyb}_1^{(v)}$, the distribution of the attribute-specific and slot-specific ciphertext components can be written as follows:

$$\begin{aligned}
C_{3,k} &= g^{s\mathbf{m}_k^\top \sum_{i \in C} a^{L+1-i}\mathbf{v}_i^*} \cdot \hat{U}_{\rho(k)}^{-s_k^*} \cdot g^{\gamma \mathbf{m}_k^\top \mathbf{v}'} \cdot \hat{U}_{\rho(k)}^{-s_k'} = g^{s\mathbf{m}_k^\top\left((\gamma/s)\cdot\mathbf{v}' + \sum_{i \in C} a^{L+1-i}\mathbf{v}_i^*\right)} \hat{U}_{\rho(k)}^{-(s_k^*+s_k')} \\
C_{4,k} &= g^{s_k^*} g^{s_k'} = g^{s_k^*+s_k'} \\
C_5 &= g^{s\cdot\sum_{i \in \mathcal{N}} a^{L+1-i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} g^{-\gamma} = g^{s\left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{L+1-i}\right)} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}},
\end{aligned}$$

where $\gamma, v_2', \ldots, v_n' \xleftarrow{R} \mathbb{Z}_p$, $s_k' \xleftarrow{R} \mathbb{Z}_p$ for all $k \in [K]$, and $\mathbf{v}' = [1, v_2', \ldots, v_n']$. For each $i \in C$, let (pk_i, S_i) be the public key and set of attributes the adversary chooses for slot $i \in C$. Parse $\mathsf{pk}_i = (T_i, Q_i, R_i, \{V_{j,i}\}_{j \neq i})$, and let $r_i \in \mathbb{Z}_p$ be the discrete log of T_i (i.e., $T_i = g^{r_i}$). Without loss of generality, we can assume that for all $i \in C$, $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$. Otherwise, the output in both experiments is $\perp$. In $\mathsf{Hyb}_1^{(\nu)}$, the challenger sets $P_i = g^{s\delta_i}$, so by construction of IsValid,

$$e(g, Q_i) = e(T_i, P_i) = e(g,g)^{s\delta_i r_i}.$$

In particular, $Q_i = g^{s\delta_i r_i}$. Thus, we can rewrite C_5 as

$$\begin{aligned} C_5 = g^{s\left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{L+1-i}\right)} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} &= g^{s\left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{L+1-i}\right)} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} g^{-sr_i} \\ &= g^{s\left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{L+1-i}\right)} \prod_{j \in [L]} T_j^{-s} \\ &= g^{s\left(-\gamma/s + \sum_{i \in \mathcal{N}} a^{L+1-i}\right)} \hat{T}^{-s}. \end{aligned}$$

We claim now that the distribution in $\mathsf{Hyb}_1^{(\nu)}$ is equivalent to an execution of $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$ with the following variable assignments:

$$h_1 := g^{-\gamma/s + \sum_{i \in \mathcal{N}} a^{L+1-i}} \quad \text{and} \quad h_2 := g^{\gamma/s + \sum_{i \in C} a^{L+1-i}},$$

and for all $k \in [K]$, $s_k := s_k^* + s_k'$, and

$$\mathbf{v} := \frac{(\gamma/s)\mathbf{v}' + \sum_{i \in C} a^{L+1-i} \mathbf{v}_i^*}{\gamma/s + \sum_{i \in C} a^{L+1-i}}.$$

For this assignment of variables, observe that

$$\begin{aligned} h_2^{s\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-s_k} &= g^{s\left(\gamma/s + \sum_{i \in C} a^{L+1-i}\right)\mathbf{m}_k^{\mathsf{T}}\mathbf{v}} \hat{U}_{\rho(k)}^{-(s_k^* + s_k')} \\ &= g^{s\mathbf{m}_k^{\mathsf{T}}\left((\gamma/s)\mathbf{v}' + \sum_{i \in C} a^{L+1-i}\mathbf{v}_i^*\right)} \cdot \hat{U}_{\rho(k)}^{-(s_k^* + s_k')} = C_{3,k}, \end{aligned}$$

$g^{s_k} = g^{s_k^* + s_k'} = C_{4,k}$, and $(h_1 \hat{T}^{-1})^s = C_5$, which coincides with the definitions in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. To complete the proof, it suffices to argue that this choice of assignments are distributed according to the specification in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. We analyze each component as follows:

- In $\mathsf{Hyb}_1^{(\nu)}$, the challenger samples $\gamma, s \xleftarrow{R} \mathbb{Z}_p$. As long as $s \neq 0$ (which happens with overwhelming probability), then over the random choice of γ, the distribution of γ/s is uniform. Thus, with overwhelming probability over the choice of s, the distribution of h_1 is uniform over $\mathbb{G}$. Moreover,

$$h_1 h_2 = g^{\sum_{i \in \mathcal{N}} a^{L+1-i} + \sum_{i \in C} a^{L+1-i}} = g^{\sum_{i \in [L]} a^{L+1-i}} = h,$$

  since C and $\mathcal{N}$ are a partition of $[L]$.

- Since the challenger samples $s_k' \xleftarrow{R} \mathbb{Z}_p$, the distribution of s_k is also uniform over $\mathbb{Z}_p$, which matches the distribution in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$.

- Write $\mathbf{v} = [v_1, v_2, \ldots, v_n]$ and $\mathbf{v}' = [1, v_2', \ldots, v_n']$. By construction, the first component of $\mathbf{v}'$ and $\mathbf{v}_i^*$ for all $i \in C$ is 1. This means $v_1 = 1$, just as in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. For $i > 1$, the challenger in $\mathsf{Hyb}_2^{(\nu)}$ samples $v_i' \xleftarrow{R} \mathbb{Z}_p$. Thus, as long as $\gamma, s \neq 0$, the distribution of $\gamma/s \cdot v_i'$ is uniformly random (and independent of all other components). Correspondingly, this means that the distributions of $v_2, \ldots, v_n$ are independent and uniform over $\mathbb{Z}_p$, exactly as required in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. Since the challenger samples $\gamma, s \xleftarrow{R} \mathbb{Z}_p$, they are non-zero with overwhelming probability.

Thus, with overwhelming probability over the choice of a, γ, and s, the challenge ciphertext $\mathsf{Hyb}_1^{(\nu)}$ is distributed exactly according to the distribution in $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. We conclude that the adversary's view in the two experiments are statistically indistinguishable, and the claim holds. ◻

Lemma B.10. *Let $q_1 = L + 1$ and $q_2 = L \cdot K$. Suppose the (q_1, q_2)-parallel bilinear Diffie-Hellman exponent assumption (Assumption B.1) holds with respect to* PrimeGroupGen*. Then, for all efficient adversaries $\mathcal{A}$, and all $v \in \{0, 1\}$, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all $\lambda \in \mathbb{N}$,*

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

Proof. Take any $v \in \{0, 1\}$ and suppose there exists an efficient adversary $\mathcal{A}$ where

$$\left|\Pr[\mathsf{Hyb}_1^{(v)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_{\mathsf{rand}}^{(v)}(\mathcal{A}) = 1]\right| = \varepsilon$$

for some non-negligible ε. Without loss of generality, we assume that for each security parameter λ, algorithm $\mathcal{A}$ always chooses a *fixed* number of slots $L = L(\lambda)$. We can formally model this by viewing the value of L as non-uniform advice. We use $\mathcal{A}$ to construct an adversary $\mathcal{B}$ for the (q_1, q_2)-parallel bilinear Diffie-Hellman exponent assumption, where $q_1 = L + 1$ and $q_2 = L \cdot K$. In the following, we will refer to elements of the set $[q_2] = \{1, \ldots, q_2\}$ by a *pair* of indices $(i, k) \in [L] \times [K]$. We now give the description of $\mathcal{B}$:

- **Initialization:** At the beginning of the game, the challenger gives $\mathcal{B}$ the parallel bilinear Diffie-Hellman challenge:
  - $\mathcal{G}$, g, Y, $\{X_j\}_{j \in [2(L+1)] \setminus \{L+1\}}$, $\{Y^{(i,k)}, X_j^{(i,k)}\}_{j \in [2(L+1)] \setminus \{L+1\}, (i,k) \in [L] \times [K]}$; and
  - $\{Z_j^{((i,k),(i',k'))}\}_{j \in [2(L+1)] \setminus \{L+1\}, (i,k) \neq (i',k') \in [L] \times [K]}$, T

  For emphasis, we color the components from the challenge in green.

- **Setup phase:** Algorithm $\mathcal{B}$ starts running algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ starts by specifying the number of slots 1^L and the indices of the corrupted slots $C \subseteq [L]$. Algorithm $\mathcal{B}$ then initializes a counter $\mathsf{ctr} \leftarrow 0$ and an (empty) dictionary Dict to keep track of the key-generation queries. Next, algorithm $\mathcal{B}$ computes $h = \prod_{i \in [L]} X_{L+1-i}$. Then, for each slot $i \in [L]$, it computes

$$A_i = X_i \quad \text{and} \quad B_i = \prod_{j \in [L] : j \neq i} X_{L+1-j+i} \quad \text{and} \quad U_i = \prod_{k \in [K]} X_{L+1-i}^{(i,k)}$$

  Next, for each $i \in [L]$ it samples $\delta_i \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. If $i \in \mathcal{N}$, it sets $P_i = g^{\delta_i}$, and if $i \in C$, it sets $P_i = Y^{\delta_i}$. For each $j \in [L]$ where $j \neq i$, algorithm $\mathcal{B}$ computes $W_{i,j} = \prod_{k \in [K]} X_{L+1-j+i}^{(j,k)}$. Finally, algorithm $\mathcal{B}$ sets $Z = e(X_1, X_L)^{-1}$ and defines the common reference string to be

$$\mathsf{crs} = \left(\mathcal{G}, Z, g, h, \{(A_i, B_i, P_i, U_i)\}_{i \in [L]}, \{W_{i,j}\}_{j \neq i}\right). \tag{B.5}$$

  Algorithm $\mathcal{B}$ gives crs to $\mathcal{A}$.

- **Query phase:** During the query phase, whenever algorithm $\mathcal{A}$ makes a key-generation query on a non-corrupted slot index $i \in \mathcal{N}$, algorithm $\mathcal{B}$ starts by incrementing the counter $\mathsf{ctr} = \mathsf{ctr} + 1$ and samples $r_i' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$. It then sets

$$T_i = X_{L+1-i} g^{r_i'} \quad \text{and} \quad Q_i = \left(X_{L+1-i} g^{r_i'}\right)^{\delta_i} \quad \text{and} \quad V_{j,i} = X_{L+1-i+j} \cdot X_j^{r_i'},$$

  for all $j \neq i$. Then $\mathcal{B}$ sets the public key to be $\mathsf{pk}_{\mathsf{ctr}} = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$ and responds with $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$. It adds the mapping $\mathsf{ctr} \mapsto (i, \mathsf{pk}_{\mathsf{ctr}})$ to the dictionary Dict.

- **Challenge phase:** In the challenge phase, algorithm $\mathcal{A}$ specifies a challenge policy $P^* = (\mathbf{M}, \rho)$, where $\mathbf{M} \in \mathbb{Z}_p^{K \times n}$ and $\rho \colon [K] \to \mathcal{U}_\lambda$ is a row-labeling function, along with two messages $\mu_0^*, \mu_1^* \in \mathbb{G}_T$. In addition, algorithm $\mathcal{A}$ specifies a key for for each slot $i \in [L]$ as follows:
  - For each corrupted slot $i \in C$, algorithm $\mathcal{A}$ specifies a public key pk_i and an attribute set S_i. Algorithm $\mathcal{B}$ checks that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$ and halts with output $\perp$ if not.

- For each non-corrupted slot $i \in \mathcal{N}$, the adversary specifies an index $c_i \in [\mathsf{ctr}]$. Algorithm $\mathcal{B}$ looks up the entry $\mathsf{Dict}[c_i] = (i', \mathsf{pk}')$. If $i = i'$, algorithm $\mathcal{B}$ sets $\mathsf{pk}_i = \mathsf{pk}'$. If $i \neq i'$, then algorithm $\mathcal{B}$ halts with output $\perp$.

For each slot $i \in [L]$, algorithm $\mathcal{B}$ parses the associated public key pk_i as $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j \neq i})$. Algorithm $\mathcal{B}$ then computes the attribute-independent public key $\hat{T}$ and attribute-independent slot key $\hat{V}_i$ for each $i \in [L]$ as follows:

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

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

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

Next, algorithm $\mathcal{B}$ constructs the challenge ciphertext. Since $\mathcal{A}$ is admissible, the attributes S_i for all corrupted indices $i \in \mathcal{C}$ do *not* satisfy the challenge policy P^*. Thus, for each $i \in \mathcal{C}$, there exists a vector $\mathbf{v}_i^*$ with first entry 1 and which is orthogonal to every row $\mathbf{m}_k^\top$ of $\mathbf{M}$ where $\rho(k) \in S_i$. Algorithm $\mathcal{B}$ now proceeds as follows:

- **Message-embedding components:** First, algorithm $\mathcal{B}$ sets $C_1 = \mu_\nu^* / T$ and $C_2 = Y$.
- **Attribute-specific components:** For ease of notation, for each $k \in [K]$, we define the following sets of indices $\Upsilon_1^{(k)}$ and $\Upsilon_2^{(k)}$:

$$\Upsilon_1^{(k)} = \{i \in [L] : \rho(k) \notin S_i\} \quad \text{and} \quad \Upsilon_2^{(k)} = \{i \in \mathcal{C} : \rho(k) \notin S_i\} \tag{B.6}$$

  Then, algorithm $\mathcal{B}$ computes $C'_{3,k}$ as

$$C'_{3,k} = \prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)}} \prod_{\substack{k' \in [K] \\ (i,k') \neq (j,k)}} \left(Z_{L+1-i}^{((i,k'),(j,k))} \right)^{-\mathbf{m}_k^\top \mathbf{v}_j^*}$$

  and $C'_{4,k}$ as

$$C'_{4,k} = \prod_{i \in \mathcal{C} : \rho(k) \notin S_i} \left(Y^{(i,k)} \right)^{\mathbf{m}_k^\top \mathbf{v}_i^*}.$$

- **Slot-specific component:** Algorithm $\mathcal{B}$ computes C'_5 as

$$C'_5 = \prod_{i \in \mathcal{N}} Y^{-r'_i} \prod_{i \in \mathcal{C}} Q_i^{-\delta_i^{-1}}.$$

Finally, algorithm $\mathcal{B}$ computes

$$\left(\{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\right) = \mathsf{Rerand}\left(\{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k \in [K]}, C'_5\right).$$

Algorithm $\mathcal{B}$ responds to $\mathcal{A}$ with the challenge ciphertext

$$\mathsf{ct}^* = \left((\mathbf{M}, \rho), C_1, C_2, \{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5\right).$$

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

We start by showing that algorithm $\mathcal{B}$ is able to simulate all of the parameters for $\mathcal{A}$ using the group elements from the (q_1, q_2)-parallel bilinear Diffie-Hellman exponent assumption.

- **CRS components:** First, we consider the components of the CRS.
  - Computing h, A_i, and B_i for $i \in [L]$ requires knowledge of the elements X_i, X_{L+1-i}, and $X_{L+1-i+j}$ for all $i \in [L]$ and $j \in [L] \setminus \{i\}$. Since $j \neq i$, the required indices are all contained in the set $[2(L+1)] \setminus \{L+1\}$.
  - Next, the component U_i depends on $X_{L+1-i}^{(i,k)}$ for all $i \in [L]$ and $k \in [K]$. Since $L+1-i \in [L]$, all of these components are contained in the challenge.
  - The component $P_i = Y^{\delta_i}$ can be simulated using Y.
  - Next, recall component $W_{i,j}$ is constructed as $\prod_{k \in [K]} X_{L+1-j+i}^{(j,k)}$. By construction, algorithm $\mathcal{B}$ only needs to construct $W_{i,j}$ for $i \neq j$. Since $i, k \in [L]$, it follows that $L+1-i+j \in [2(L+1)] \setminus \{L+1\}$, so these components are also contained in the challenge.
  - Finally, algorithm $\mathcal{B}$ sets $Z = e(X_1, X_L)^{-1}$. Both X_1 and X_L are contained in the challenge.
- **Key-generation queries:** Next, we consider the elements algorithm $\mathcal{B}$ uses to simulate public keys when responding to the adversary's key-generation queries. For each $i \in \mathcal{N}$, the elements T_i, Q_i, and $V_{j,i}$ for $j \neq i$ require knowledge of X_{L+1-i}, X_j, and $X_{L+1-i+j}$, where $i, j \in [L]$. By the same analysis as above (for the CRS components), all of these components are included in the challenge.
- **Challenge ciphertext:** Finally, we consider the components of the challenge ciphertext:
  - To construct C_1 and C_2, algorithm $\mathcal{B}$ requires T and Y, which are part of the challenge.
  - To construct $C'_{3,k}$, algorithm $\mathcal{B}$ requires $Z_{L+1-i}^{((i,k'),(j,k))}$ for all $i \in \Upsilon_1^{(k)}$, $j \in \Upsilon_2^{(k)}$, and $k' \in [K]$ where $(i,k') \neq (j,k)$. First, $L+1-i \in [L] \subseteq [2(L+1)] \setminus \{L+1\}$. As long as $(i,k') \neq (j,k)$, then this component is contained in the challenge.
  - To construct $C'_{4,k}$, algorithm $\mathcal{B}$ requires $Y^{(i,k)}$ for all $i \in C$ where $\rho(k) \notin S_i$. Since the challenge contains $Y^{(i,k)}$ for all $i \in [L]$ and $k \in [K]$, algorithm $\mathcal{B}$ can construct this term.
  - To construct C'_5, algorithm $\mathcal{B}$ needs Y, which is included in the challenge.

We conclude that the challenge contains all of the components algorithm $\mathcal{B}$ needs for simulating the CRS, the key-generation queries, and the challenge ciphertext. To complete the proof, we show that depending on the distribution of T, algorithm $\mathcal{B}$ either simulates an execution of $\mathsf{Hyb}_1^{(\nu)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ for $\mathcal{A}$. Let $a, s, \beta_{i,k} \in \mathbb{Z}_p$ for $i \in [L]$ and $k \in [K]$ be the exponents sampled by the (q_1, q_2)-parallel bilinear Diffie-Hellman exponent challenger. Then, the challenge components are defined as follows:

$$Y = g^s \ , \ X_i = g^{a^i} \ , \ Y^{(i,k)} = g^{s\beta_{i,k}} \ , \ X_j^{(i,k)} = g^{a^j/\beta_{i,k}} \ , \ Z_j^{(i,k),(i',k')} = g^{a^j s \beta_{i',k'}/\beta_{i,k}} .$$

We claim that algorithm $\mathcal{B}$ simulates an execution of $\mathsf{Hyb}_1^{(\nu)}$ or $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ where the exponents $a, s, \beta_{i,k}$ are the corresponding ones sampled by the (q_1, q_2)-parallel bilinear Diffie-Hellman challenger.

CRS components. Consider first the components of the CRS. Then in an execution of $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ where the randomness is $a, s, \{\beta_{i,k}\}_{i \in [L], k \in [K]}$, the challenger constructs the components of the CRS as follows. First, the challenger sets $\alpha = -a^{L+1}$ and $t_i = a^{d_i}$. It also computes

$$h = \prod_{i \in [L]} g^{a^{L+1-i}} = \prod_{i \in [L]} X_{L+1-i},$$

which matches the behavior of algorithm $\mathcal{B}$. Then, for each $i \in [L]$, the challenger would compute

$$\begin{aligned} A_i &= g^{t_i} = g^{a^i} = X_i \\ B_i &= g^\alpha h^{t_i} = g^{-a^{L+1}} \left(\prod_{j \in [L]} g^{a^{L+1-j}} \right)^{a^i} = \prod_{j \in [L]: j \neq i} g^{a^{L+1-j+i}} = \prod_{j \in [L]: j \neq i} X_{L+1-j+i}, \end{aligned}$$

which matches the behavior of algorithm $\mathcal{B}$. In both $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger sets

$$u_i = \sum_{k \in [K]} \frac{1}{\beta_{i,k}} a^{L+1-i} \tag{B.7}$$

The challenger then computes

$$U_i = g^{u_i} = g^{\left(\sum_{k\in[K]} \frac{1}{\beta_{i,k}} a^{L+1-i}\right)} = \prod_{k\in[K]} g^{\frac{1}{\beta_{i,k}}\left(a^{L+1-i}\right)} = \prod_{k\in[K]} X_{L+1-i}^{(i,k)},$$

which again matches the behavior of $\mathcal{B}$. Next, for each $i \in [L]$, the challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ sets $P_i = g^{s\delta_i} = Y^{\delta_i}$ if $i \in \mathcal{C}$ and $P_i = g^{\delta_i}$ if $i \in \mathcal{N}$. This is the same procedure used by algorithm $\mathcal{B}$. Next, for each $j \neq i \in [L]$, the challenger would set

$$W_{i,j} = g^{t_i u_j} = g^{\left(\sum_{k\in[K]} \frac{1}{\beta_{j,k}} a^{L+1-j}\right) a^i} = \prod_{k\in[K]} g^{\frac{1}{\beta_{j,k}}\left(a^{L+1-j+i}\right)} = \prod_{k\in[K]} X_{L+1-j+i}^{(j,k)}.$$

Finally, the challenger sets

$$Z = e(g,g)^{\alpha} = e(g,g)^{-a^{L+1}} = e(g^a, g^{a^L})^{-1} = e(X_1, X_L)^{-1}.$$

We conclude that algorithm $\mathcal{B}$ constructs the components in the CRS using the identical procedure as the challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$.

Key-generation queries. For the key-generation queries on indices $i \in \mathcal{N}$, the challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ generates $\mathsf{pk}_i = (T_i, Q_i, \{V_{j,i}\}_{j\neq i})$ by first sampling $r_i' \xleftarrow{\mathrm{R}} \mathbb{Z}_p$, setting $r_i = a^{L+1-i} + r_i'$ and then setting

- $T_i = g^{r_i} = g^{a^{L+1-i}+r_i'} = X_{L+1-i} g^{r_i'}$.
- $Q_i = P_i^{r_i} = g^{\delta_i(a^{L+1-i}+r_i')} = (X_{L+1-i} g^{r_i'})^{\delta_i}$. Recall that $i \in \mathcal{N}$ so $P_i = g^{\delta_i}$.
- $V_{j,i} = A_j^{r_i} = g^{a^j \cdot (a^{L+1-i}+r_i')} = X_{L+1-i+j} X_j^{r_i'}$.

Again, algorithm $\mathcal{B}$ perfectly simulates the responses to the key-generation queries.

Challenge ciphertext. We now analyze the challenge ciphertext components. First, we consider the distribution of C_1. We have two possibilities:

- Suppose $T = e(g,g)^{a^{(L+1)s}}$. Then algorithm $\mathcal{B}$ sets $C_1 = \mu_\nu^* / T = \mu_\nu^* \cdot e(g,g)^{-a^{L+1}s} = \mu_\nu^* \cdot Z^s$, which matches the distribution of C_1 in $\mathsf{Hyb}_1^{(\nu)}$.
- Suppose the challenger samples $T \xleftarrow{\mathrm{R}} \mathbb{G}_T$. Then, $C_1 = \mu_\nu^* / T$ is also uniform over $\mathbb{G}_T$, and algorithm $\mathcal{B}$ simulated the distribution of C_1 in $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$.

To complete the proof, it suffices to argue that the remaining components in the challenge ciphertext are simulated exactly according to the specification of $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. First, in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger would set $C_2 = g^s = Y$. which coincides with the behavior of algorithm $\mathcal{B}$. Next, consider $C_{3,k}'$ for $k \in [K]$. In $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger would first set

$$s_k^* = s \cdot \sum_{i\in\mathcal{C}:\rho(k)\notin S_i} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* = s \cdot \sum_{i \in \Upsilon_2^{(k)}} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*, \tag{B.8}$$

using the definition of $\Upsilon_2^{(k)} = \{i \in C : \rho(k) \notin S_i\}$ from Eq. (B.6). Then, the challenger computes

$$C'_{3,k} = g^{s\mathbf{m}_k^{\mathsf{T}} \sum_{i \in C} a^{L+1-i} \mathbf{v}_i^*} \cdot \hat{U}_{\rho(k)}^{-s_k^*} = \hat{U}_{\rho(k)}^{-s_k^*} \prod_{i \in C} g^{a^{L+1-i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} = \hat{U}_{\rho(k)}^{-s_k^*} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{L+1-i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s}, \tag{B.9}$$

using the fact that in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$, the challenger chooses $\mathbf{v}_i^*$ such that $\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* = 0$ for all $k \in [K]$ where $\rho(k) \in S_i$. Consider the term $\hat{U}_{\rho(k)}^{-s_k^*}$. By definition,

$$\hat{U}_{\rho(k)} = \prod_{i \in [L] : \rho(k) \notin S_i} U_i = \prod_{i \in \Upsilon_1^{(k)}} g^{u_i}$$

using the definition of $\Upsilon_1^{(k)} = \{i \in [L] : \rho(k) \notin S_i\}$ from Eq. (B.6). For ease of notation, let $\hat{U}_{\rho(k)}^{-s_k^*} = g^{\xi}$ for some $\xi \in \mathbb{Z}_p$. Then, substituting in the definitions of u_i from Eq. (B.7) and s_k^* from Eq. (B.8), we have

$$\begin{aligned} \xi &= \sum_{i \in \Upsilon_1^{(k)}} -u_i s_k^* \\ &= \sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} -\beta_{j,k} u_i \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s && \text{by Eq. (B.8)} \\ &= \sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} \sum_{k' \in [K]} -\frac{\beta_{j,k}}{\beta_{i,k'}} a^{L+1-i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s && \text{by Eq. (B.7).} \end{aligned}$$

We decompose ξ into the terms ξ_1 where $(i,k') \neq (j,k)$ and the terms ξ_2 where $(i,k') = (j,k)$ (also meaning $i = j$). Then, we have

$$\xi = \underbrace{\sum_{i \in \Upsilon_1^{(k)}} \sum_{j \in \Upsilon_2^{(k)}} \sum_{\substack{k' \in [K] \\ (i,k') \neq (j,k)}} \left(-\frac{\beta_{j,k}}{\beta_{i,k'}} a^{L+1-i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^* s \right)}_{\xi_1} + \underbrace{\sum_{i \in \Upsilon_1^{(k)} \cap \Upsilon_2^{(k)}} -a^{L+1-i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s}_{\xi_2}.$$

From Eq. (B.6), we have that $\Upsilon_2^{(k)} \subseteq \Upsilon_1^{(k)}$, so we can write

$$\xi_2 = \sum_{i \in \Upsilon_1^{(k)} \cap \Upsilon_2^{(k)}} -a^{L+1-i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s = \sum_{i \in \Upsilon_2^{(k)}} -a^{L+1-i} \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s.$$

Substituting back into Eq. (B.9) and using the fact that $\hat{U}_{\rho(k)}^{-s_k^*} = g^{\xi} = g^{\xi_1 + \xi_2}$, we conclude that

$$\begin{aligned} C'_{3,k} = \hat{U}_{\rho(k)}^{-s_k^*} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{L+1-i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} &= g^{\xi_1} g^{\xi_2} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{L+1-i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} \\ &= g^{\xi_1} \left(\prod_{i \in \Upsilon_2^{(k)}} g^{a^{L+1-i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} \right)^{-1} \prod_{i \in \Upsilon_2^{(k)}} g^{a^{L+1-i} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^* s} \\ &= \prod_{i \in \Upsilon_1^{(k)}} \prod_{j \in \Upsilon_2^{(k)}} \prod_{\substack{k' \in [K] \\ (i,k') \neq (j,k)}} \left(Z_{L+1-i}^{((i,k'),(j,k))} \right)^{-\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_j^*}. \end{aligned}$$

This is precisely how algorithm $\mathcal{B}$ constructs $C'_{3,k}$. Next, consider $C'_{4,k}$. The challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ sets

$$C'_{4,k} = g^{s_k^*} = g^{s \cdot \sum_{i \in C : \rho(k) \notin S_i} \beta_{i,k} \cdot \mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*} = \prod_{i \in C : \rho(k) \notin S_i} \left(Y^{(i,k)} \right)^{\mathbf{m}_k^{\mathsf{T}} \mathbf{v}_i^*},$$

which is how algorithm $\mathcal{B}$ constructs $C'_{4,k}$. Finally, consider C'_5. The challenger in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ sets

$$C'_5 = g^{s \cdot \sum_{i \in \mathcal{N}} a^{L+1-i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}}. \tag{B.10}$$

By construction, for all $i \in \mathcal{N}$, the challenger sets $T_i = g^{r_i} = g^{a^{L+1-i} + r'_i}$. Thus, we can write

$$\prod_{i \in \mathcal{N}} T_i^{-s} = \prod_{i \in \mathcal{N}} g^{-sa^{L+1-i} - sr'_i} = g^{-s \sum_{i \in \mathcal{N}} a^{L+1-i}} \prod_{i \in \mathcal{N}} g^{-r'_i s}.$$

Substituting back into Eq. (B.10), this means

$$\begin{aligned} C'_5 &= g^{s \cdot \sum_{i \in \mathcal{N}} a^{L+1-i}} \prod_{i \in \mathcal{N}} T_i^{-s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} \\ &= g^{s \cdot \sum_{i \in \mathcal{N}} a^{L+1-i}} \left(g^{s \cdot \sum_{i \in \mathcal{N}} a^{L+1-i}} \right)^{-1} \prod_{i \in \mathcal{N}} g^{-r'_i s} \prod_{i \in C} Q_i^{-\delta_i^{-1}} \\ &= \prod_{i \in \mathcal{N}} Y^{-r'_i} \prod_{i \in C} Q_i^{-\delta_i^{-1}}, \end{aligned}$$

which is how algorithm $\mathcal{B}$ constructs C'_5. Finally, algorithm $\mathcal{B}$ computes

$$\left(\{C_{3,k}, C_{4,k}\}_{k \in [K]}, C_5 \right) = \mathsf{Rerand}\left(\{\hat{U}_w\}_{w \in \mathcal{U}_\lambda}, (\mathbf{M}, \rho), \{C'_{3,k}, C'_{4,k}\}_{k \in [K]}, C'_5 \right),$$

which exactly coincides with the challenger's behavior in $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_{\mathsf{real}}^{(\nu)}$. We conclude that if $T = e(g,g)^{a^{L+1}s}$, then algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_1^{(\nu)}$ whereas if $T \xleftarrow{\mathrm{R}} \mathbb{G}_T$, algorithm $\mathcal{B}$ perfectly simulates an execution of $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$. Thus, algorithm $\mathcal{B}$ succeeds with the same advantage of $\mathcal{A}$, and the claim follows. □

By construction the adversary's view in $\mathsf{Hyb}_{\mathsf{rand}}^{(\nu)}$ is independent of ν. As such, for all adversaries $\mathcal{A}$, the output distributions $\mathsf{Hyb}_{\mathsf{rand}}^{(0)}(\mathcal{A})$ and $\mathsf{Hyb}_{\mathsf{rand}}^{(1)}(\mathcal{A})$ are identically distributed. The claim now follows from Lemmas B.9 and B.10. □

C Incremental Aggregation for Registered ABE

As discussed in Appendix C, a naïve application from a slotted registered ABE scheme to a standard registered ABE scheme (that supports dynamic user registrations; see Definition 3.1) would require the key curator to keep track of every user's public key. In our slotted registered ABE scheme, the size of each user's key contains $O(L)$ group elements, where L is the number of slots. As such, if the key curator has to keep store every users' public key, then its storage requirement would scale quadratically as $O(L^2)$. Here, we show that if we apply the [HLWW23] transformation to a slotted registered ABE scheme that supports incremental aggregation, then we can reduce the size of the key curator state. In particular, transforming our particular slotted registered ABE schemes (Constructions 4.3 and 5.5) to a full registered ABE scheme that supports up to L users only requires the key curator to store $\tilde{O}(L)$ group elements. We start by recalling the transformation (taken nearly verbatim) from [HLWW23]:

Construction C.1 (Slotted Registered ABE to Registered ABE [HLWW23, Construction 6.1])**.** Let λ be a security parameter. Let $\Pi_{\mathsf{sRABE}} = (\mathsf{sRABE.Setup}, \mathsf{sRABE.KeyGen}, \mathsf{sRABE.IsValid}, \mathsf{sRABE.Aggregate}, \mathsf{sRABE.Encrypt}, \mathsf{sRABE.Decrypt})$ be a slotted registered ABE scheme with attribute universe $\mathcal{U} = \{\mathcal{U}_\lambda\}_{\lambda \in \mathbb{N}}$ and policy space $\mathcal{P} = \{\mathcal{P}_\lambda\}_{\lambda \in \mathbb{N}}$. We now construct a registered ABE scheme $\Pi_{\mathsf{RABE}} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{RegPK}, \mathsf{Encrypt}, \mathsf{Update}, \mathsf{Decrypt})$ that supports a bounded number of users and over the same attribute space $\mathcal{U}$ and policy space $\mathcal{P}$. In the description, we adopt the following conventions:

- Without loss of generality, we assume that the bound on the number of users $L = 2^\ell$ is a power of two. Rounding the bound to the next power of two incurs at most a factor of 2 overhead.

- The registered ABE scheme will internally maintain $\ell + 1$ slotted ABE schemes, where the k^{th} scheme is a slotted scheme with 2^k slots (for $k \in [0, \ell]$). We assume that the message space $\mathcal{M}_\lambda$ associated with Π_{sRABE} is a deterministic function of the security parameter $\lambda \in \mathbb{N}$.
- The auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{Dict}_1, \mathsf{Dict}_2, \mathsf{mpk})$ consists of the following components:
  - A counter ctr that keeps track of the number of registered users in the system.
  - A dictionary Dict_1 that maps a scheme index $k \in [0, \ell]$ and a slot index $i \in [2^k]$ to a pair (pk, S) which specifies the public key and attribute set currently assigned to slot i of scheme k.
  - A dictionary Dict_2 that maps a scheme index $k \in [0, \ell]$ and a user index $i \in [L]$ to the helper decryption key associated with scheme k and user i.
  - The current master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \dots, \mathsf{mpk}_\ell)$.

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

We construct our registered ABE scheme as follows:

- $\mathsf{Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^L)$: On input the security parameter λ the size of the attribute universe $\mathcal{U}_\lambda$, and a bound on number of users $L = 2^\ell$, the setup algorithm runs the setup algorithm for $\ell + 1$ copies of the slotted RBE scheme. Specifically, for each $k \in [0, \ell]$, it samples $\mathsf{crs}_k \leftarrow \mathsf{sRABE.Setup}(1^\lambda, 1^{|\mathcal{U}_\lambda|}, 1^{2^k})$ and outputs $\mathsf{crs} = (\mathsf{crs}_0, \dots, \mathsf{crs}_\ell)$. The message space associated with crs is the message space $\mathcal{M}_\lambda$ associated with $\mathsf{crs}_0, \dots, \mathsf{crs}_\ell$ (recall that we assume that the message space $\mathcal{M}_\lambda$ associated with each output of $\mathsf{sRABE.Setup}$ is a deterministic function of the security parameter λ).
- $\mathsf{KeyGen}(\mathsf{crs}, \mathsf{aux})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \dots, \mathsf{crs}_\ell)$ and the auxiliary data $\mathsf{aux} = (\mathsf{ctr}, \mathsf{Dict}_1, \mathsf{Dict}_2, \mathsf{mpk})$, the key-generation algorithm generates a public/secret key-pair for each of the $\ell + 1$ underlying schemes. Specifically, for each $k \in [0, \ell]$, let $i_k = (\mathsf{ctr} \bmod 2^k) + 1 \in [2^k]$ be a slot index for the k^{th} scheme, and sample a key $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{sRABE.KeyGen}(\mathsf{crs}_k, i_k)$. Output $\mathsf{pk} = (\mathsf{ctr}, \mathsf{pk}_0, \dots, \mathsf{pk}_\ell)$ and $\mathsf{sk} = (\mathsf{ctr}, \mathsf{sk}_0, \dots, \mathsf{sk}_\ell)$.
- $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S_{\mathsf{pk}})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \dots, \mathsf{crs}_\ell)$, the auxiliary data $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{Dict}_1, \mathsf{Dict}_2, \mathsf{mpk})$, where $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{mpk}_0, \dots, \mathsf{mpk}_\ell)$, a public key $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_0, \dots, \mathsf{pk}_\ell)$, and an associated set of attributes S_{pk}), the registration algorithm proceeds as follows:
  - For each $k \in [0, \ell]$, let $i_k = (\mathsf{ctr}_{\mathsf{aux}} \bmod 2^k) + 1 \in [2^k]$ be the slot index for the k^{th} scheme.
  - For each $k \in [0, \ell]$, check that $\mathsf{sRABE.IsValid}(\mathsf{crs}_k, i_k, \mathsf{pk}_k) = 1$. In addition, check that $\mathsf{ctr}_{\mathsf{aux}} = \mathsf{ctr}_{\mathsf{pk}}$. If any check fails, the algorithm halts and outputs the current auxiliary data aux and master public key mpk.
  - Then for each $k \in [0, \ell]$, the registration algorithm updates $\mathsf{Dict}_1[k, i_k] \leftarrow (\mathsf{pk}, S_{\mathsf{pk}})$. In addition, if $i_k = 2^k$ (i.e., all of the slots in scheme k are filled), the registration algorithm additionally does the following:
    * Compute
    $$\left(\mathsf{mpk}'_k, \mathsf{hsk}'_{k,1}, \dots, \mathsf{hsk}'_{k,2^k}\right) \leftarrow \mathsf{sRABE.Aggregate}\left(\mathsf{crs}_k, \mathsf{Dict}_1[k, 1], \dots, \mathsf{Dict}_1[k, 2^k]\right).$$
    * Update $\mathsf{Dict}_2[\mathsf{ctr} + 1 - 2^k + i, k] = \mathsf{hsk}'_{k,i}$ for each $i \in [2^k]$.
    * If $i_k \neq 2^k$, $\mathsf{mpk}'_k = \mathsf{mpk}_k$ is unchanged.
  - Define the new master public key $\mathsf{mpk}' = (\mathsf{ctr}_{\mathsf{aux}} + 1, \mathsf{mpk}'_1, \dots, \mathsf{mpk}'_\ell)$.
  - Finally, the registration algorithm outputs the new master public key mpk' and auxiliary data $\mathsf{aux}' = (\mathsf{ctr}_{\mathsf{aux}} + 1, \mathsf{Dict}_1, \mathsf{Dict}_2, \mathsf{mpk}')$.
- $\mathsf{Encrypt}(\mathsf{mpk}, P, \mu)$: On input the master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \dots, \mathsf{mpk}_\ell)$, the access policy $P \in \mathcal{P}_\lambda$, and a message $\mu \in \mathcal{M}_\lambda$, the encryption algorithm computes $\mathsf{ct}_k \leftarrow \mathsf{sRABE.Encrypt}(\mathsf{mpk}_k, P, \mu)$ for each $k \in [0, \ell]$; if $\mathsf{mpk}_k = \bot$, then it sets $\mathsf{ct}_k = \bot$. Then it outputs $\mathsf{ct} = (\mathsf{ctr}, \mathsf{ct}_0, \dots, \mathsf{ct}_\ell)$.

- $\mathsf{Update}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$, the auxiliary data $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{Dict}_1, \mathsf{Dict}_2, \mathsf{mpk})$, and a public key $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$, the update algorithm outputs $\perp$ if $\mathsf{ctr}_{\mathsf{pk}} \geq \mathsf{ctr}_{\mathsf{aux}}$. Otherwise, for each $k \in [0, \ell]$, it sets $\mathsf{hsk}_k = \mathsf{Dict}_2[\mathsf{ctr}_{\mathsf{pk}} + 1, k]$ and replies with $\mathsf{hsk} = (\mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell)$.
- $\mathsf{Decrypt}(\mathsf{sk}, \mathsf{hsk}, \mathsf{ct})$: On input a secret key $\mathsf{sk} = (\mathsf{ctr}_{\mathsf{sk}}, \mathsf{sk}_0, \ldots, \mathsf{sk}_\ell)$, a helper key $\mathsf{hsk} = (\mathsf{hsk}_0, \ldots, \mathsf{hsk}_\ell)$, and a ciphertext $\mathsf{ct} = (\mathsf{ctr}_{\mathsf{ct}}, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$, the decryption algorithm outputs $\perp$ if $\mathsf{ctr}_{\mathsf{ct}} \leq \mathsf{ctr}_{\mathsf{sk}}$. Otherwise, it computes the largest index k on which ctr and ctr' differ (where bits are 0-indexed starting from the least significant bit). If $\mathsf{hsk}_k = \perp$, then the decryption algorithm outputs $\mathsf{GetUpdate}$. Otherwise, it outputs $\mathsf{sRABE}.\mathsf{Decrypt}(\mathsf{sk}_k, \mathsf{hsk}_k, \mathsf{ct}_k)$.

Correctness and security. We refer to [HLWW23, §6] for the correctness and security analysis of Construction C.1. While [HLWW23] only analyzing the transformation as applied to fully secure slotted registered ABE scheme, the same analysis also applies to a statically-secure slotted registered ABE scheme. In this case, the transformed scheme inherits the same security properties as the slotted scheme (i.e., if we apply Construction C.1 to a statically-secure slotted registered ABE scheme, then we obtain a statically-secure registered ABE scheme).

Leveraging incremental aggregation. We now show that if the slotted registered ABE scheme supports incremental aggregation (Definition 3.8), then the key curator can incrementally update its state as users join (instead of leading to store all of the users' public keys until one of the underlying slotted schemes fills up). As noted earlier, when applied to our constructions, incremental aggregation brings the storage requirements of the key curator from $\Omega(L^2)$ to $\tilde{O}(L)$.

Lemma C.2 (Incremental Aggregation)**.** *Suppose* sRABE *supports f-incremental aggregation for some function $f(L, |\mathcal{U}_\lambda|)$. Then, we can modify Construction C.1 to only require maintaining an auxiliary state* aux *of size at most*

$$|\mathsf{aux}| = f(L, |\mathcal{U}_\lambda|) \cdot \mathsf{poly}(\lambda, \log L) + L \cdot \mathsf{poly}(\lambda, |\mathcal{U}_\lambda|, \log L).$$

Proof. We consider a functionally-equivalent version of Construction C.1 where we replace Dict_1 in aux (that maps indices to public keys) with the aggregation state. Namely, the structure of aux is now:

$$\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, (\mathsf{st}_0, \ldots, \mathsf{st}_\ell), \mathsf{Dict}_2, \mathsf{mpk}).$$

The internal states are all initialized to $\perp$: $\mathsf{st}_0 = \mathsf{st}_1 = \cdots = \mathsf{st}_\ell = \perp$. We now define the $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S_{\mathsf{pk}})$ algorithms as follows:

- $\mathsf{RegPK}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, S_{\mathsf{pk}})$: On input the common reference string $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$, the auxiliary data $\mathsf{aux} = (\mathsf{ctr}_{\mathsf{aux}}, (\mathsf{st}_0, \ldots, \mathsf{st}_\ell), \mathsf{Dict}_2, \mathsf{mpk})$, where $\mathsf{mpk} = (\mathsf{ctr}_{\mathsf{aux}}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$, a public key $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$, and an associated set of attributes S_{pk}, the registration algorithm proceeds as follows:
  - For each $k \in [0, \ell]$, let $i_k = (\mathsf{ctr}_{\mathsf{aux}} \bmod 2^k) + 1 \in [2^k]$ be the slot index for the k^{th} scheme.
  - For each $k \in [0, \ell]$, check that $\mathsf{sRABE}.\mathsf{IsValid}(\mathsf{crs}_k, i_k, \mathsf{pk}_k) = 1$. In addition, check that $\mathsf{ctr}_{\mathsf{aux}} = \mathsf{ctr}_{\mathsf{pk}}$. If any check fails, the algorithm halts and outputs the current auxiliary data aux and master public key mpk.
  - Then for each $k \in [0, \ell]$, compute $\mathsf{st}_k \leftarrow \mathsf{sRABE}.\mathsf{AggregateUpdate}(\mathsf{crs}_k, \mathsf{st}_k, (\mathsf{pk}_k, S_{\mathsf{pk}}))$. In addition, if $i_k = 2^k$ (i.e., all of the slots in scheme k are filled), the registration algorithm additionally does the following:
    - Compute
    $$\left(\mathsf{mpk}'_k, \mathsf{hsk}'_{k,1}, \ldots, \mathsf{hsk}'_{k,2^k}\right) \leftarrow \mathsf{sRABE}.\mathsf{AggregateUpdate}\left(\mathsf{crs}_k, \mathsf{st}_k, \perp\right).$$
    - Update $\mathsf{Dict}_2[\mathsf{ctr} + 1 - 2^k + i, k] \leftarrow \mathsf{hsk}'_{k,i}$ for each $i \in [2^k]$. Finally, set $\mathsf{st}_k = \perp$.

    If $i_k \neq 2^k$, then $\mathsf{mpk}'_k = \mathsf{mpk}_k$ is unchanged.
  - Define the new master public key as $\mathsf{mpk}' = (\mathsf{ctr}_{\mathsf{aux}} + 1, \mathsf{mpk}'_1, \ldots, \mathsf{mpk}'_\ell)$.
  - Finally, the registration algorithm outputs the new master public key mpk' and auxiliary data $\mathsf{aux}' = (\mathsf{ctr}_{\mathsf{aux}} + 1, (\mathsf{st}_0, \ldots, \mathsf{st}_\ell), \mathsf{Dict}_2, \mathsf{mpk}')$.

Essentially, we have simply replaced the dictionary Dict_1 with the intermediate state of each of the underlying slotted schemes. We now argue the correctness and efficiency properties:

- **Correctness:** Correctness follows by the incremental aggregation property: namely, the modified incremental registration algorithm implements exactly the same functionality as the original non-incremental registration algorithm (Definition 3.8).
- **Size of auxiliary data:** First, by compactness of registered ABE, the size of each master public key mpk_i and helper decryption key $\mathsf{hsk}_{k,i}$ have size bounded by $\mathsf{poly}(\lambda, |\mathcal{U}_\lambda|, \log L)$. There are $O(L)$ such keys at any point in time. Moreover, each of the underlying states st_i has size bounded by $f(L, |\mathcal{U}_\lambda|) \cdot \mathsf{poly}(\lambda)$. Thus, the total size of the auxiliary data is at most $L \cdot \mathsf{poly}(\lambda, |\mathcal{U}_\lambda|, \log L) + f(L, |\mathcal{U}_\lambda|) \cdot \mathsf{poly}(\lambda, \log L)$. □

D Validating Assumptions in Generic Group Model

We now show that the complexity assumptions we use in this work (Assumption 4.2, Assumption 5.4, Assumption 7.7,Assumption B.1) hold in the generic (bilinear) group model. First, we recall the generic (bilinear) group model [Sho97, BBG05, Boy08].

Definition D.1 (Generic Bilinear Group Model)**.** For a positive integer $N \in \mathbb{Z}$, let $\mathcal{L} \subseteq \{0,1\}^*$ be a set of strings of bounded length and cardinality at least N. The generic (symmetric) bilinear group model is initialized with two *random* injective mappings $\sigma, \sigma_T : \mathbb{Z}_N \to \mathcal{L}$ (which map a discrete log over $\mathbb{Z}_N$ to an associated label in $\mathcal{L}$). Here σ corresponds to the labeling function associated with the base group while σ_T is the labeling function associated with the target group. In the generic group model, we assume that the parties have oracle access to the generic bilinear group oracle which supports the following operations:

- **Base group encoding:** On input $x \in \mathbb{Z}_N$, the oracle responds with $\sigma(x)$.
- **Base group operation:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, if ℓ_1, ℓ_2 are in the image of σ, then the oracle replies with $\sigma(\sigma^{-1}(\ell_1) + \sigma^{-1}(\ell_2))$. If either ℓ_1 or ℓ_2 are not in the image of σ, then the oracle replies with $\perp$.
- **Target group encoding:** On input $x \in \mathbb{Z}_N$, the oracle responds with receives $\sigma_T(x)$.
- **Target group operation:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, if ℓ_1, ℓ_2 are in the image of σ_T, then the oracle replies with $\sigma_T(\sigma_T^{-1}(\ell_1) + \sigma_T^{-1}(\ell_2))$. If either ℓ_1 or ℓ_2 are not in the image of σ_T, then the oracle replies with $\perp$.
- **Pairing:** On input two labels $\ell_1, \ell_2 \in \mathcal{L}$, if ℓ_1, ℓ_2 are in the image of σ, then the oracle replies with $\sigma_T(\sigma^{-1}(\ell_1) \cdot \sigma^{-1}(\ell_2))$. If either ℓ_1 or ℓ_2 are not in the image of σ, then the oracle replies with $\perp$.

Notation. We will write g to denote the label for $\sigma(1)$ and g^x to denote $\sigma(x)$. Similarly, we write $e(g,g)$ to denote $\sigma_T(1)$ and $e(g,g)^x$ to denote $\sigma_T(x)$. We write $\mathbb{G}$ and $\mathbb{G}_T$ to denote the groups induced by the labeling functions σ and σ_T, respectively (i.e., $\mathbb{G} = \{\sigma(x) : x \in \mathbb{Z}_N\}$ and $\mathbb{G}_T = \{\sigma_T(x) : x \in \mathbb{Z}_N\}$). To analyze our assumptions, we follow the methodology from [BBG05, KSW08, KSW13] by enumerating a set of sufficient conditions for security to hold unconditionally in the generic bilinear group model. We begin with a notion of independence and then give the theorem statements we use in our analysis.

Definition D.2 (Independence of Polynomials)**.** Let $N = \prod_{i \in [m]} p_i$ be a positive integer that is a product of $m \geq 1$ distinct primes p_i. Let $\mathcal{P} = \{P_i\}_{i \in [k]}$ be a collections of polynomials where each $P_i \in \mathbb{Z}_N[X_1, \ldots, X_n]$ is an n-variate polynomials over $\mathbb{Z}_N$. By the Chinese Remainder Theorem, we can view each polynomial P_i as defining a tuple of m polynomials $P_{i,1} \in \mathbb{Z}_{p_1}[X_1, \ldots, X_n], \ldots, P_{i,m} \in \mathbb{Z}_{p_m}[X_1, \ldots, X_n]$ and where $P_{i,j}(x_1, \ldots, x_n) = P_i(x_1, \ldots, x_n) \bmod p_j$ for all $j \in [m]$. We say that a polynomial $f \in \mathbb{Z}_N[X_1, \ldots, X_n]$ (with associated components $f_1, \ldots, f_m$) is dependent on $\mathcal{P}$ if there exists coefficients $\alpha_i \in \mathbb{Z}_N$ such that

$$\forall j \in [m] : f_j(X_1, \ldots, X_N) = \sum_{i \in [k]} \alpha_i P_{i,j}(X_1, \ldots, X_N) \bmod p_j.$$

We say f is independent on $\mathcal{P}$ if f is not dependent on $\mathcal{P}$.

Theorem D.3 (Generic Hardness in Prime-Order Groups [BBG05, Theorem A.2]). *Let p be a prime. Let $\mathcal{P} = \{P_i\}_{i \in [k]}$ and $\mathcal{Q} = \{Q_j\}_{j \in [\ell]}$ be collections of polynomials where each $P_i, Q_j \in \mathbb{Z}_p[X_1, \ldots, X_n]$ is an n-variate polynomial over $\mathbb{Z}_N$ and $P_1 = Q_1 = 1$. Let $T_0, T_1 \in \mathbb{Z}_p[X_1, \ldots, X_n]$ be two challenge polynomials. Then, for a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, define the following experiment in the generic bilinear group model 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 $\ell_i = \sigma(P_i(x_1, \ldots, x_n))$, $\ell'_j = \sigma_T(Q_j(x_1, \ldots, x_n))$, $\tau_0 = \sigma_T(T_0(x_1, \ldots, x_n))$, and $\tau_1 = \sigma_T(T_1(x_1, \ldots, x_n))$.*
- *The challenger gives $\big(p, \{\ell_i\}_{i \in [k]}, \{\ell'_j\}_{j \in [\ell]}, \tau_b\big)$ to the adversary.*
- *The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.*

Let $\mathcal{P}^2 := \{P_i P_j : i, j \in [k]\}$. Suppose that the total degree of P_i, Q_j, T_0, T_1 is at most d and the polynomials T_0, T_1 are independent of $\mathcal{P}^2 \cup \mathcal{Q}$. Then, for all adversaries $\mathcal{A}$ making at most q queries to the generic group oracle, it holds that

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

in the above distinguishing experiment.

Theorem D.4 (Generic Hardness in Composite-Order Groups [KSW13, Theorem A.2, adapted]). *Let $N = \prod_{j \in [m]} p_j$ be a product of distinct primes where each $p_j \geq 2^\lambda$. Let $\mathcal{P} = \{P_i\}_{i \in [k]}$ and $\mathcal{Q} = \{Q_i\}_{i \in [\ell]}$ be collections of linearly independent polynomials where each $P_i, Q_i \in \mathbb{Z}_N[X_1, \ldots, X_n]$ is an n-variate polynomial over $\mathbb{Z}_N$. We assume that $P_1 = Q_1 = 1$. As in Definition D.2, we write $P_{i,j}$ and $Q_{i,j}$ to denote the action of the polynomial P_i and Q_i, respectively, modulo p_j. Let $T_0, T_1 \in \mathbb{Z}_N[X_1, \ldots, X_n]$ be two challenge polynomials. Then for a bit $b \in \{0,1\}$ and an adversary $\mathcal{A}$, define the following experiment in the generic bilinear group model of order N:*

- *At the beginning of the game, the challenger samples $x_1, \ldots, x_n \xleftarrow{R} \mathbb{Z}_N$. For each $i \in [k]$ and $j \in [\ell]$, it computes*

$$\begin{aligned} \ell_i &= \sigma(P_i(x_1, \ldots, x_n)) \\ \ell'_j &= \sigma_T(Q_j(x_1, \ldots, x_n)) \\ \tau_0 &= \sigma(T_0(x_1, \ldots, x_n)) \\ \tau_1 &= \sigma(T_1(x_1, \ldots, x_n)). \end{aligned}$$

- *The challenger gives $\big(N, \{\ell_i\}_{i \in [k]}, \{\ell'_j\}_{j \in [\ell]}, \tau_b\big)$ to the adversary.*
- *The adversary outputs a bit $b' \in \{0,1\}$, which is the output of the experiment.*

For an adversary $\mathcal{A}$, define its advantage $\delta_{\mathcal{A}}$ to be

$$\delta_{\mathcal{A}} := |\Pr[b' = 1 : b = 0] - \Pr[b' = 1 : b = 1]|$$

in the above distinguishing experiment. Let $\mathcal{P}^2 := \{P_i P_j : i, j \in [k]\}$. For a bit $b \in \{0,1\}$, let

$$\mathcal{S}^{(b)} := \mathcal{P}^2 \cup \mathcal{Q} \cup \{T_b P_i : i \in [k]\}.$$

For an index $i \in [k]$, define $\mathcal{S}_i^{(b)} := \mathcal{S}^{(b)} \setminus \{T_b P_i\}$. Suppose now the following properties hold:

- *The total degree of P_i, Q_j, T_0, T_1 is at most d.*
- *For all $i \in [k]$ and $b \in \{0,1\}$, the polynomial T_b is independent of $\mathcal{P}$.*
- *For all $i \in [k]$, if $T_0 P_i \neq T_1 P_i$, then for all $b \in \{0,1\}$, the polynomial $T_b P_i$ is independent of $\mathcal{S}_i^{(b)}$.*
- *For all $b \in \{0,1\}$, the polynomial T_b^2 is independent of $\mathcal{S}^{(b)}$.*

Then, for all adversaries $\mathcal{A}$ making at most q queries to the generic group oracle, if $\mathcal{A}$ has distinguishing advantage $\delta_{\mathcal{A}}$ in the above distinguishing experiment, then there is an algorithm that runs in time polynomial in λ and the running time of $\mathcal{A}$ that outputs a non-trivial factor of N with success probability at least $\delta_{\mathcal{A}} - O((q+k+\ell)^2 d/2^{\lambda})$.

Remark D.5 (Comparison with [KSW13, Theorem A.2])**.** The conditions we give in Theorem D.4 differs slightly from the corresponding theorem statement from [KSW13, Theorem A.2].[12] The corresponding theorem statement from [KSW13] only requires that

$$\{T_b^2\} \cup \{T_b P_i\}_{i : T_0 P_i \neq T_1 P_i} \text{ is independent of } \{T_b P_i\}_{i : T_0 P_i = T_1 P_i} \cup \mathcal{P}^2 \cup \mathcal{Q} \tag{D.1}$$

whereas we *additionally* require that T_b^2 to be independent of $T_b P_i$ for all $i \in [k]$ (in addition to $\mathcal{P}^2 \cup \mathcal{Q}$) rather than just the indices $i \in [k]$ where $T_0 P_i = T_1 P_i$. This condition is necessary, as the presence of such a dependence can be used to break the assumption. For instance, consider the following distribution in two-prime composite-order group:

$$P_1(X_1, X_2) = [X_1, X_1] \quad , \quad T_0(X_1, X_2) = [X_1, 0] \quad , \quad T_1(X_1, X_2) = [X_2, 0].$$

Here, we write each polynomial in terms of its decomposition under the Chinese Remainder Theorem. Namely, in this example, $T_0(X_1, X_2) = X_1 \bmod p_1$ and $T_0(X_1, X_2) = 0 \bmod p_2$. By construction, $T_0 P_1 \neq T_1 P_1$. Thus, to satisfy Eq. (D.1) from [KSW13], it suffices to show that $\{T_b^2, T_b P_1\}$ is independent of P_1^2, but this is immediate since P_1^2 is a non-zero polynomial modulo p_2 while T_b^2 and $T_b P_1$ are the identically-zero polynomial modulo p_2. At the same time, given the challenge (N, ℓ_1, τ_b), the adversary can simply use the pairing oracle to check whether $e(\ell_1, \tau_b) = e(\tau_b, \tau_b)$. When $b = 0$, this check always passes whereas if $b = 1$, this check passes with probability $1/p_2 \leq 2^{-\lambda}$. For completeness, we give a proof of our amended theorem in Appendix D.1.

Theorem D.6 (Generic Hardness of Assumption 4.2)**.** *Let λ be a security parameter. Then, for every polynomial $q = q(\lambda)$, and every prime $p \geq 2^{\lambda}$, Assumption 4.2 holds in the generic bilinear group model of order p.*

Proof. We will use Theorem D.3. Take any set $S \subseteq [q-1]$. The components given out in Assumption 4.2 can be expressed as polynomials over the formal variables $\hat{s}, \hat{a}, \hat{r}$. Specifically, the polynomials in the base group are defined as follows:

$$1 \,,\; \hat{s} \,,\; \{\hat{a}^i\}_{i \in S \cup [q+1, 2q]} \,,\; \{\hat{s}\hat{a}^{q-i}\}_{i \in [q-1] \setminus S} \,,\; \{\hat{s}\hat{a}^i\}_{i \in [q+1, 2q]}. \tag{D.2}$$

The assumption also gives out a polynomial $\hat{a}^q$ in the target group. The challenge polynomials are

$$T_0(\hat{s}, \hat{a}, \hat{r}) := \hat{s}\hat{a}^q \quad \text{and} \quad T_1(\hat{s}, \hat{a}, \hat{r}) := \hat{r},$$

Certainly, T_0 and T_1 are independent of $\hat{a}^q$ (since they are different monomials). It suffices to check that T_0 and T_1 are independent of the pairwise products of the monomials in Eq. (D.2). Since none of the monomials in Eq. (D.2) depend on $\hat{r}$, the claim holds for T_1. We now claim that we cannot none of the pairs of monomials in Eq. (D.2) multiply to $\hat{s}\hat{a}^q$. We can partition the elements in Eq. (D.2) into two sets:

- The monomials of the form $\hat{a}^i$ for $i \in \{0\} \cup S \cup [q+1, 2q]$.
- The monomials of the form $\hat{s}\hat{a}^{q-i}$ for $i \in [q-1] \setminus S$ and $\hat{s}\hat{a}^i$ for $i \in [q+1, 2q]$.

The only possible way to form the monomial $\hat{s}\hat{a}^q$ is to multiply a monomial from the first set (possibly 1) with a monomial from the second set. Let $\hat{a}^{i_1}$ and $\hat{s}\hat{a}^{i_2}$ be these two monomials, where $i_1 \in \{0\} \cup S$ and $q - i_2 \in [q-1] \setminus S$. Moreover, it must be the case that $i_1 + i_2 = q$, which means that $i_1 \in \{0\} \cup S$ and $i_1 \in [q-1] \setminus S$. Since the sets $\{0\} \cup S$ and $[q-1] \setminus S$ are disjoint, this is a contradiction. We conclude that T_0 is independent of the pairwise products of the polynomials in Eq. (D.2) so the claim now follows from Theorem D.3. □

Lemma D.7 (Generic Hardness of Assumption B.1)**.** *Let λ be a security parameter. Then, for all polynomials $q_1 = q_1(\lambda)$, $q_2 = q_2(\lambda)$ and every prime $p \geq 2^{\lambda}$, Assumption B.1 holds in the generic bilinear group model of order p.*

[12] Note that we are specifically referring to the journal version [KSW13], which corrects an earlier version of the theorem statement from [KSW08].

Proof. As written, Assumption 4.10 requires publishing inverses of group elements, which does not fall under the structure of Theorem D.3. However, we can apply a simple change of basis to avoid having to give out inverses. First, recall that the challenge components in Assumption 4.10 consists of the group elements

$$\left(g\,,\, g^s\,,\, \{g^{a^i}\}_{i\in[2q_1]\setminus\{q_1\}}\,,\, \{g^{s\beta_j}, g^{a^i/\beta_j}\}_{j\in[q_2],i\in[2q_1]\setminus\{q_1\}}\,,\, \{g^{sa^i\beta_k/\beta_j}\}_{j\neq k,i\in[2q_1]\setminus\{q_1\}}\right)$$

and the challenge terms $T_0 = e(g,g)^{sa^{q_1}}$ or $T_1 = e(g,g)^r$. Consider the assumption where we replace the generator g with $\tilde{g} = g^{u\prod_{j\in[q_2]}\beta_j}$, where $u \xleftarrow{\text{R}} \mathbb{Z}_{p_1}$. Note that $\tilde{g}$ is a random generator (since $u \xleftarrow{\text{R}} \mathbb{Z}_{p_1}$). With respect to the new generator $\tilde{g}$, the challenge components can now be written as

$$\left(\tilde{g}\,,\, \tilde{g}^s\,,\, \{\tilde{g}^{a^i}\}_{i\in[2q_1]\setminus\{q_1\}}\,,\, \{\tilde{g}^{s\beta_j}, g^{ua^i\prod_{t\notin j}\beta_t}\}_{j\in[q_2],i\in[2q_1]\setminus\{q_1\}}\,,\, \{g^{usa^i\beta_k\prod_{t\neq j}\beta_t}\}_{j\neq k,i\in[2q_1]\setminus\{q_1\}}\right),$$

and the challenge terms are $T_0 = e(\tilde{g},\tilde{g})^{sa^{q_1}}$ and $T_1 = e(\tilde{g},\tilde{g})^r$. We now use Theorem D.3 to analyze this version of the assumption. The components given out in Assumption B.1 can be expressed as polynomials over the formal variables $\hat{u}, \hat{s}, \hat{a}, \hat{\beta}_1, \ldots, \hat{\beta}_{q_2}, \hat{r}$. Specifically, let $\hat{\xi} = \hat{u}\prod_{j\in[q_2]}\hat{\beta}_j$. Then the polynomials in the base group are defined as follows:

$$\hat{\xi}\,,\, \hat{\xi}\hat{s}\,,\, \{\hat{\xi}\hat{a}^i\}_{i\in[2q_1]\setminus\{q_1\}}\,,\, \left\{\hat{\xi}\hat{s}\hat{\beta}_j, \hat{u}\hat{a}^i\prod_{t\neq j}\hat{\beta}_t\right\}_{j\in[q_2],i\in[2q_1]\setminus\{q_1\}}\,,\, \left\{\hat{u}\hat{s}\hat{a}^i\hat{\beta}_k\prod_{t\neq j}\hat{\beta}_t\right\}_{j\neq k,i\in[2q_1]\setminus\{q_1\}}. \tag{D.3}$$

as well as the challenge polynomials

$$T_0\left(\hat{u}, \hat{s}, \hat{a}, \hat{\beta}_1, \ldots, \hat{\beta}_{q_2}, \hat{r}\right) := \hat{\xi}^2\hat{s}\hat{a}^{q_1} \quad \text{and} \quad T_1\left(\hat{u}, \hat{s}, \hat{a}, \hat{\beta}_1, \ldots, \hat{\beta}_{q_2}, \hat{r}\right) := \hat{\xi}^2\hat{r}.$$

By construction, T_1 is independent of all of the monomials in Eq. (D.3) (since none of them depend on $\hat{r}$), so it suffices to show that T_0 is independent of the pairwise products of the monomials in Eq. (D.3). By construction, at least one of the monomials in the pairwise product must contain $\hat{s}$. We now consider the possibilities:

- Suppose we use $\hat{\xi}\hat{s}$. To obtain the monomial $\hat{\xi}^2\hat{s}\hat{a}^{q_1}$, we would need to multiply by the monomial $\hat{\xi}\hat{a}^{q_1}$, which is not given out in the assumption.
- Suppose we use $\hat{\xi}\hat{s}\hat{\beta}_j$ for some $j \in [q_2]$. To obtain the monomial $\hat{\xi}^2\hat{s}\hat{a}^{q_1}$, we would need to multiply by the monomial
$$\hat{\xi}\hat{a}^{q_1}/\hat{\beta}_j = \hat{u}\hat{a}^{q_1}\prod_{t\neq j}\hat{\beta}_t,$$
which is not given out in the assumption.
- Suppose we use $\hat{u}\hat{s}\hat{a}^i\hat{\beta}_k\prod_{t\neq j}\hat{\beta}_t$ for some $i \in [2q_1] \setminus \{q_1\}$ and $j \neq k$. Then, to obtain the monomial $\hat{\xi}^2\hat{s}\hat{a}^{q_1}$ we need to multiply by the monomial
$$\hat{\xi}^2\hat{a}^{q_1-i}/\hat{u}\hat{\beta}_k\prod_{t\neq j}\hat{\beta}_t = \hat{\xi}\hat{a}^{q_1-i}\hat{\beta}_j/\hat{\beta}_k = \hat{u}\hat{a}^{q_1-i}\hat{\beta}_j\prod_{t\neq k}\hat{\beta}_t,$$
which is not given out in the assumption.

We conclude that T_0 is linearly independent of the pairwise products of the monomials in Eq. (D.3). Finally, the polynomials in Eq. (D.3) have degree at most $O(q_1+q_2)$. Since $q_1, q_2 = \text{poly}(\lambda)$, and the number of terms is $O(q_1+q_2)$, the claim follows from Theorem D.3. □

Lemma D.8 (Generic Hardness of Assumption 5.4)**.** *If factoring a product of four primes (each of size 2^λ) is computationally hard, then Assumption 5.4 holds in the generic bilinear group model of order N where N is a product of four primes (each of size 2^λ).*

Proof. We will use Theorem D.4. Take any polynomial input length $L = L(\lambda)$, any progression-free and double-free set $\mathcal{D} = \{d_i\}_{i\in[L]}$ and a challenge index $i^* \in [L]$. We start by enumerating the component given out by Assumption 5.4. We express these components in their representation under the Chinese Remainder Theorem. Specifically, since we are working over a composite-order group with modulus $N = p_1p_2p_3p_4$, we write $[\hat{x}_1, \hat{x}_2, \hat{x}_3, \hat{x}_4]$ to denote the variable $\hat{x}$ over $\mathbb{Z}_N^*$ where $\hat{x} = \hat{x}_i \bmod p_i$ for all $i \in \{1,2,3,4\}$. The components given out in Assumption 5.4 can be expressed as polynomials over the formal variables $\hat{s}, \hat{a}, \hat{b}, \hat{\tau}, \hat{s}_{23}$:

- for all $i \in [L] \backslash \{i^*\}$, $A'_i = [\hat{a}^{d_i}, 0, 0, 0]$;
- for all $i \in [L]$, $U'_i = [\hat{b}\hat{a}^{d_i}, 0, 0, 0]$;
- for all $z \in \mathcal{E}$, $W'_z = [\hat{b}\hat{a}^z, 0, 0, 0]$;
- $X = [\hat{s}, \hat{s}, 0, 0]$;
- for all $i \in [L] \backslash \{i^*\}$, $Y_i = [\hat{s}\hat{b}\hat{a}^{d_i}, 0, 0, 0]$ and $Y_{i^*} = [\hat{s}\hat{b}\hat{a}^{d_{i^*}}, \hat{s}\hat{b}\hat{a}^{d_{i^*}}, 0, 0]$;
- generators $G_1 = [1, 0, 0, 0]$, $G_3 = [0, 0, 1, 0]$, $G_4 = [0, 0, 0, 1]$, and $G_{23} = [0, \hat{s}_{23}, \hat{s}_{23}, 0]$.

Finally, the challenge polynomials are constructed as

$$T_0(\hat{s}, \hat{a}, \hat{b}, \hat{\tau}, \hat{s}_{23}) = [\hat{a}^{d_{i^*}}, 0, \hat{\tau}, 0] \quad \text{and} \quad T_1(\hat{s}, \hat{a}, \hat{b}, \hat{\tau}, \hat{s}_{23}) = [\hat{a}^{d_{i^*}}, \hat{\tau}, \hat{\tau}, 0].$$

We now consider the conditions in Theorem D.4. First, it is easy to see that T_0 and T_1 are independent of the other monomials given out in the challenge (since T_0 and T_1 are the only polynomials that depend on the formal variable $\hat{\tau}$). Next, we consider the pairwise products T_bP (where P is one of the other polynomials in the challenge). We only consider T_bP when $T_0P \neq T_1P$. By construction, T_0 and T_1 are identical in the p_1, p_3, and p_4 subgroups and only differ in the p_2 subgroup. As such, the only element P for which $T_0P \neq T_1P$ are the challenge polynomials that are nonzero in the $\mathbb{G}_2$ subgroup. These are the components X, Y_{i^*}, and G_{23}. We check each of the linear independence requirements:

1. T_bX is independent of $\mathcal{S}_X^{(b)}$ (where $\mathcal{S}_X^{(b)}$ is the set as defined in Theorem D.4).
   - First $T_0X = [\hat{s}\hat{a}^{d_{i^*}}, 0, 0, 0]$ Observe that since this element does not include $\hat{b}$, so it is independent of any pairwise product that involves U'_i, W'_z, Y_i, Y_{i^*}. Since X is the only remaining element with $\hat{s}$ in $\mathbb{G}_1$, the only non-independent pairwise products must contain X. Since $XA_i = [\hat{s}\hat{a}^{d_i}, 0, 0, 0]$ for $i \neq i^*$, these are linearly independent of T_0X, as required.
   - Next, $T_1, X = [\hat{s}\hat{a}^{d_{i^*}}, \hat{s}\hat{\tau}, 0, 0]$. This follows analogously to the above case; in particular, none of the pairwise products produce $\hat{s}\hat{a}^{d_i}$ in the p_1 subgroup, linear independence holds.
2. $T_bY_{i^*}$ is independent of $\mathcal{S}^b_{Y_{i^*}}$
   - First, $T_0Y_{i^*} = [\hat{s}\hat{b}\hat{a}^{2d_{i^*}}, 0, 0, 0]$. The monomial $\hat{s}\hat{b}\hat{a}^{2d_{i^*}}$ in the p_1 subgroup is degree 1 in $\hat{s}$ and $\hat{b}$. The only pairwise products with this property are XU'_i, XW'_z, $Y_iA'_j$, and T_0Y_i. We consider each term separately:
     – For all $i \in [L]$, in the p_1 subgroup, $XU'_i = \hat{s}\hat{b}\hat{a}^{d_i}$. Since $d_i, d_{i^*} \in \mathcal{D}$ and $\mathcal{D}$ is double-free, there does not exist any $i \in [L]$ where $d_i = 2d_{i^*}$. As such, XU'_i is independent of T_0Y_i.
     – For all $z \in \mathcal{E}$, in the p_1 subgroup, $XW'_z = \hat{s}\hat{b}\hat{a}^z$, where $z = d_i + d_j$ for some $i \neq j$. Since $d_i, d_j, d_{i^*} \in \mathcal{D}$ and $\mathcal{D}$ is progression-free, it follows that $z = d_i + d_j \neq 2d_{i^*}$. Thus, $XW'_z \neq \hat{s}\hat{b}\hat{a}^{2d_{i^*}}$, and XW'_z is independent of T_0Y_i.
     – For all $i \in [L]$ and $j \neq i^*$, in the p_1 subgroup, $Y_iA'_j = \hat{s}\hat{b}\hat{a}^{d_i+d_j}$. Since $\mathcal{D}$ is double-free if $i \neq j$, then $d_i + d_j \neq 2d_{i^*}$. If $i = j$, then $i, j \neq d_{i^*}$. Since the elements of $\mathcal{D}$ are distinct, once again $d_i + d_j \neq 2d_{i^*}$. Thus, $Y_iA'_j$ is independent of T_0Y_i.
     – Finally, T_0Y_i in the p_1 subgroup has value $\hat{s}\hat{b}\hat{a}^{d_i+d_{i^*}}$ for $i \neq i^*$. Since $d_i \neq d_{i^*}$ when $i \neq i^*$, this value is again independent of $T_0Y_{i^*}$.
   - Next, $T_1Y_{i^*} = [\hat{s}\hat{b}\hat{a}^{2d_{i^*}}, \hat{s}\hat{b}\hat{a}^{d_{i^*}}\hat{\tau}, 0, 0]$. This analysis follows as in the previous case. Namely, none of the allowed pairwise products produce the monomial $\hat{s}\hat{b}\hat{a}^{2d_{i^*}}$ in the p_1 subgroup.
3. T_bG_{23} is independent of $\mathcal{S}^b_{G_{23}}$.
   - First, $T_0G_{23} = [0, \hat{s}_{23}, \hat{s}_{23}\hat{\tau}, 0]$. By construction, the only pairwise product that contains the monomial $\hat{s}_{23}\hat{\tau}$ is T_0G_{23}, so the claim holds.

- Next, $T_1 G_{23} = [0, \hat{s}_{23}\hat{\tau}, \hat{s}_{23}\hat{\tau}, 0]$. This follows analogously to the the above case.

To complete the proof, we argue that T_b^2 is independent of the other pairwise products. In both cases, T_b^2 contains the monomial $\hat{\tau}^2$ in the p_3 subgroup. None of the other challenge components depend on $\hat{\tau}$, so linear independence is immediate. Finally, the maximum degree of the monomials appearing in the challenge is bounded by $O(\max(\mathcal{D}))$. Since $\max(\mathcal{D})$ is polynomially-bounded, the claim now follows. □

Lemma D.9 (Generic Hardness of Assumption 7.7)**.** *If factoring a product of two primes (each of size 2^λ) is computationally hard, then Assumption 7.7 holds in the generic group model of order N where N is a product of two primes (each of size 2^λ).*

Proof. We will use Theorem D.4. Take any polynomial input length $L = L(\lambda)$, any progression-free and double-free set $\mathcal{D} = \{d_i\}_{i \in [L]}$ and a challenge index $i^* \in [L]$. We start by enumerating the component given out by Assumption 7.7. We express these components in their representation under the Chinese Remainder Theorem. Specifically, since we are working over a composite-order group with modulus $N = p_1 p_2$, we write $[\hat{x}_1, \hat{x}_2]$ to denote the variable $\hat{x}$ over $\mathbb{Z}_N^*$ where $\hat{x} = \hat{x}_i \bmod p_i$ for all $i \in \{1, 2\}$. The components given out in Assumption 7.7 can be expressed as polynomials over the formal variables $\hat{s}, \hat{a}$:

- for all $i \in [L] \backslash \{i^*\}$, $A'_i = [\hat{s}\hat{a}^{d_i}, 0]$;
- for all $i \neq j \in [L]$, $B'_{i,j} = [\hat{s}^2 \hat{a}^{d_i + d_j}, 0]$;
- generator $G_1 = [1, 0]$.

Finally, the challenge polynomials are constructed as

$$T_0(\hat{s}, \hat{a}) = [\hat{s}\hat{a}^{d_{i^*}}, 0] \quad \text{and} \quad T_1(\hat{s}, \hat{a}) = [\hat{s}\hat{a}^{d_{i^*}}, \hat{s}\hat{a}^{d_{i^*}}].$$

We now consider the conditions in Theorem D.4. First, it is easy to see that T_0 and T_1 are independent of A'_i for all $i \neq i^*$, and also with $B_{i,j}$ since each $B_{i,j}$ is quadratic in $\hat{s}$ while T_0, T_1 are linear in $\hat{s}$. They are also independent of G_1. Next, $T_0 P = T_1 P$ for all monomials P appearing in the challenge. Thus, to invoke Theorem D.4, it suffices to show that T_b^2 is independent of the other pairwise products.

1. Consider $T_0^2 = [\hat{s}^2 \hat{a}^{2d_{i^*}}, 0]$. The only pairwise products of A'_i and $B_{i,j}$ that are degree-2 in $\hat{s}$ are of the form $A'_i A'_j$ for $i, j \neq i^*$ and those of the form $G_1 B'_{i,j}$ for $i \neq j$ We consider each case separately.
   - Take any $i, j \neq i^*$. Then, $A'_i A'_j = [\hat{s}^2 \hat{a}^{d_i + d_j}, 0]$. If $i = j \neq i^*$, this is $[\hat{s}^2 \hat{a}^{2d_i}, 0]$. Since the elements of $\mathcal{D}$ are distinct and $i \neq i^*$, $\hat{s}^2 \hat{a}^{2d_i} \neq \hat{s}^2 \hat{a}^{2d_{i^*}}$. If $i \neq j$, then $A'_i A'_j = [\hat{s}^2 \hat{a}^{d_i + d_j}, 0]$. Since $\mathcal{D}$ is progression-free, this means that $d_i + d_j \neq 2d_{i^*}$ and linear independence holds.
   - Take any $i \neq j \in [L]$. Then, $G_1 B'_{i,j} = [\hat{s}^2 \hat{a}^{d_i + d_j}, 0]$. Since $\mathcal{D}$ is progression-free, it follows that $d_i + d_j \neq 2d_{i^*}$, so linear independence again holds.
2. Consider $T_1^2 = [\hat{s}^2 \hat{a}^{2d_{i^*}}, \hat{s}^2 \hat{a}^{2d_{i^*}}]$. Since T_1^2 is the only element that has a non-trivial p_2 component, it is linearly independent of the pairwise products of all other components.

The maximum degree of the monomials appearing in the challenge is bounded by $O(\max(\mathcal{D}))$. Since $\max(\mathcal{D})$ is polynomially-bounded, the claim now follows via Theorem D.4. □

D.1 Proof of Theorem D.4

In this theorem, we provide a proof of Theorem D.4. Our analysis follows an identical strategy as [KSW13, Theorem A.2], except we incorporate the final condition in Theorem D.4 into the analysis. As discussed in Remark D.5, the extra condition is required to argue hardness in the generic group model.

Proof of Theorem D.4. Let N be a product of m distinct primes $p_1, \ldots, p_m$. We define a sequence of hybrid experiments:

- $\mathsf{Hyb}_0^{(\nu)}$: This is the real game with bit $\nu \in \{0,1\}$ in Theorem D.4. Specifically, in this experiment, the challenger samples random variables $x_1, \ldots, x_n \xleftarrow{R} \mathbb{Z}_N$, and constructs the group elements as polynomials of $(x_1, \ldots, x_n)$ as described in Theorem D.4. The challenge component is set to be τ_ν. The challenger associates a fresh handle with each distinct group element and answers generic group queries according to the specification of the generic bilinear group oracle (Definition D.1).

- $\mathsf{Hyb}_1^{(\nu)}$: In this hybrid, rather than sampling $x_1, \ldots, x_n \xleftarrow{R} \mathbb{Z}_N$, the challenger instead associates each variable x_i with a formal variable $\hat{x}_i$. Each label now maps onto a vector of m formal polynomials on $\hat{x}_1, \ldots, \hat{x}_n$, where the i^{th} formal polynomial is an element of $\mathbb{Z}_{p_i}[\hat{x}_1, \ldots, \hat{x}_n]$. The group operation computes an element-wise sum of formal polynomials while a pairing operation computes an element-wise product of formal polynomials. The challenger associates distinct labels to each distinct *vector* of formal polynomials (in the variables $\hat{x}_1, \ldots, \hat{x}_n$).

- $\mathsf{Hyb}_2^{(\nu)}$: In this hybrid, instead of associating labels with a vector of polynomials $\mathbb{Z}_{p_1}[\hat{x}_1, \ldots, \hat{x}_k] \times \cdots \times \mathbb{Z}_{p_m}[\hat{x}_1, \ldots, \hat{x}_k]$, the challenger associated each label with a vector of polynomials over $\mathbb{Z}_N[\hat{x}_1, \ldots, \hat{x}_k] \times \cdots \times \mathbb{Z}_N[\hat{x}_1, \ldots, \hat{x}_k]$. Specifically, each of the polynomials has coefficients over $\mathbb{Z}_N$ rather than $\mathbb{Z}_{p_i}$. As before, the challenger associates distinct labels to each distinct vector of formal polynomials.

Lemma D.10. *Let $\mathcal{A}$ be an adversary that makes at most Q queries to the generic group oracles. Then for all bits $\nu \in \{0,1\}$,*

$$\left|\Pr[\mathsf{Hyb}_0^{(\nu)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_1^{(\nu)}(\mathcal{A}) = 1]\right| \leq O((q+k+\ell)^2 t/2^\lambda).$$

Proof. Observe that if two polynomials are equal, they will be equal at all points. Thus, these two hybrids only differ when two distinct polynomials f_1, f_2 (in the variables $\hat{x}_1, \ldots, \hat{x}_n$) evaluate to the same value modulo *every* p_i. In this case, the challenger would return the same handle in $\mathsf{Hyb}_0^{(\nu)}$, but different handles in $\mathsf{Hyb}_1^{(\nu)}$. Since the maximum degree of the polynomials is t, the maximum degree of any polynomial appearing in the generic bilinear group encoding table is $2t$ (specifically, the only operation that can increase the degree of the polynomial is the pairing operation). Since $x_1, \ldots, x_n \xleftarrow{R} \mathbb{Z}_N$ are sampled uniformly over $\mathbb{Z}_N$ (and thus, over each $\mathbb{Z}_{p_i}$), the probability that a pair of non-identical polynomials f_1, f_2 (over $\mathbb{Z}_{p_i}$) satisfy $f_1(x_1, \ldots, x_n) = f_2(x_1, \ldots, x_n) \bmod p_i$ is at most $\frac{2t}{p_i} < \frac{2t}{2^\lambda}$, since $p_i > 2^\lambda$. Since there are at most $q + k + \ell$ polynomials in the table (from the ones introduced by the adversary's queries and the ones from the assumption), we can union bound over all $\binom{q+k+\ell}{2}$ pairs of polynomials. Thus, the statistical distance between these two experiments is at most $O((q+k+\ell)^2 t/2^\lambda)$. □

Lemma D.11. *Let $\mathcal{A}$ be an adversary where*

$$\left|\Pr[\mathsf{Hyb}_1^{(\nu)}(\mathcal{A}) = 1] - \Pr[\mathsf{Hyb}_2^{(\nu)}(\mathcal{A}) = 1]\right| = \varepsilon.$$

Then there exists an algorithm $\mathcal{B}$ which finds a nontrivial factor of N with probability ε.

Proof. Since $\mathsf{Hyb}_1^{(\nu)}$ and $\mathsf{Hyb}_2^{(\nu)}$ differ only in the setting where the table contains a pair of labels mapping onto two vectors $(f_{1,1}, \ldots, f_{1,n})$ and $(f_{2,1}, \ldots, f_{2,n})$ where for some index $j \in [n]$, it holds that $f_{1,j} = f_{2,j} \bmod p_j$ but $f_{1,j} \neq f_{2,j} \bmod N$. When this happens, $f_{1,j} - f_{2,j} = 0 \bmod p_j$ but $f_{1,j} - f_{2,j} \neq 0 \bmod N$, so computing the greatest common division between the coefficients of the difference $f_{1,j} - f_{2,j}$ and N will yield a non-trivial factor of N. □

Lemma D.12. *For all adversaries $\mathcal{A}$,* $\Pr[\mathsf{Hyb}_2^{(0)}(\mathcal{A}) = 1] = \Pr[\mathsf{Hyb}_2^{(1)}(\mathcal{A}) = 1]$.

Proof. By definition, these two experiments are identical unless the adversary $\mathcal{A}$ is able to construct a formal polynomial involving the challenge polynomial T_ν that is identical to the all-zeroes polynomial for one value of $\nu \in \{0,1\}$ but not for the other value $1 - \nu$. Let $\hat{x}_1, \ldots, \hat{x}_n$ be the formal variables in an execution of $\mathsf{Hyb}_2^{(0)}$ and $\mathsf{Hyb}_2^{(1)}$. Initially, the tables $\mathcal{D}$ contain the polynomials $P_1, \ldots, P_k$ as well as the challenge polynomial T_ν. The table $\mathcal{D}_T$ contain the polynomials $Q_1, \ldots, Q_\ell$. Define the sets $\mathcal{P} = \{P_i\}_{i \in [k]}$ and $\mathcal{Q} = \{Q_i\}_{i \in [\ell]}$ and $\mathcal{S}^{(\nu)} = \mathcal{P}^2 \cup \mathcal{Q} \cup \{T_b P_i : i \in [k]\}$ as in Theorem D.4. We now consider two possibilities:

- Consider a polynomial f_v in the base group. By construction, we can write f_v as

$$f_v(\hat{x}_1, \ldots, \hat{x}_n) = \sum_{i \in [k]} \alpha_i P_i(\hat{x}_1, \ldots, \hat{x}_n) + \beta T_v(\hat{x}_1, \ldots, \hat{x}_n),$$

  where $\alpha_i, \beta \in \mathbb{Z}_N$ (note that we do not need an affine term since $P_1 = 1$). If $f_v = 0$ for some value of $v \in \{0,1\}$, then T_v is not linearly independent of $\mathcal{P}$ which contradicts the requirement in Theorem D.4.

- Consider a polynomial f_v in the target group. By construction, we can write f_v as

$$f = \sum_{i,j \in [k]} \alpha_{i,j} P_i P_j + \sum_{i \in [\ell]} \beta_i Q_i + \sum_{i \in [k]} \gamma_i P_i T_v + \delta T_v^2,$$

  where $\alpha_{i,j}, \beta_i, \gamma_i, \delta \in \mathbb{Z}_N$. Suppose that $f_v = 0$ for some $v \in \{0,1\}$ and $f_{1-v} \neq 0$. We consider two cases. If $\delta \neq 0$, then we can write T_v^2 as a linear combination of the elements of $\mathcal{S}^{(v)}$. This contradicts the requirements in Theorem D.4. Suppose instead that $\delta = 0$. Then,

$$f_v = \sum_{i,j \in [k]} \alpha_{i,j} P_i P_j + \sum_{i \in [\ell]} \beta_i Q_i + \sum_{i \in [k]} \gamma_i P_i T_v = 0.$$

  Let i^* be the first index for which $T_0 P_{i^*} \neq T_1 P_{i^*} \bmod N$. Such an index must exist since $f_v = 0$ but $f_{1-v} \neq 0$. In this case, we have expressed $P_{i^*} T_v$ as a linear combination of polynomials in the set $\mathcal{S}^{(v)} \setminus T_v P_{i^*} = \mathcal{S}_{i^*}^{(v)}$, which contradicts the premise in Theorem D.4.

Thus, if $\mathcal{P}, \mathcal{Q}, T_0, T_1$ satisfy the requirements in Theorem D.4, then algorithm $\mathcal{A}$ is not able to construct a polynomial f_v where $f_v = 0$ and $f_{1-v} \neq 0$. The claim holds. □

Combining Lemmas D.10 to D.12, Theorem D.4 holds. □

E Analyzing the (q_1, q_2)-Intermediate Set-Consistent BDHE Assumption

In this section, we show that the (q_1, q_2)-intermediate set-consistent bilinear Diffie-Hellman exponent assumption (Assumption 4.10) is implied by the q-set-consistent bilinear Diffie-Hellman exponent assumption (Assumption 4.2). Specifically, we prove the following statement, which implies Lemma 4.11 as a special case.

Lemma E.1. *Let $q = q(\lambda)$ be any polynomial. Suppose the q-set-consistent bilinear Diffie-Hellman exponent assumption (Assumption 4.2) holds with respect to* PrimeGroupGen. *Then, the (q_1, q_2)-intermediate set-consistent bilinear Diffie-Hellman exponent assumption (Assumption 4.10) holds with respect to* PrimeGroupGen *for any q_1, q_2 where $4q_1q_2 = q$.*

Proof. Take any $q_1, q_2 \in \mathbb{N}$ where $q = 4q_1q_2$. Suppose $\mathcal{A}$ is an efficient adversary for the (q_1, q_2)-intermediate set-consistent BDHE assumption (Assumption 4.10) that succeeds with non-negligible advantage ε. We use $\mathcal{A}$ to construct an algorithm $\mathcal{B}$ for the q-set-Consistent BDHE game. At a high level, algorithm $\mathcal{B}$ simulates the "parallel" terms containing β_j using *disjoint* intervals of the powers of a. In the following description, we write $i \bmod 4q_2$ to refer to the (unique) representative of i in the interval $[-2q_2, 2q_2 - 1]$. We now describe the algorithm $\mathcal{B}$:

1. Algorithm $\mathcal{B}$ start by running algorithm $\mathcal{A}$. Algorithm $\mathcal{A}$ outputs a set $S \subseteq [q_1 - 1]$.
2. Algorithm $\mathcal{B}$ constructs the sets

$$S_0' = \{4q_2 \cdot i \mid i \in S\} \quad \text{and} \quad S_1' = \{i \in [q - 2q_2] \mid i \bmod 4q_2 \text{ is odd and negative}\}.$$

   It sets $S' = S_0' \cup S_1'$ and gives S' to the challenger.
3. The challenger replies with the challenge

$$\mathcal{G}\,,\ g\,,\ Y'\,,\ \left\{X_i'\right\}_{i \in S' \cup [q+1,2q]}\,,\ \left\{Z_{q-i}'\right\}_{i \in [q-1] \setminus S'}\,,\ \left\{Z_i'\right\}_{[q+1,2q]}\,,\ Q'\,,\ T'. \tag{E.1}$$

   For emphasis, we color these terms in green.

4. Algorithm $\mathcal{B}$ samples random exponents $a_0, \beta'_1, \ldots, \beta'_{q_2} \xleftarrow{\text{R}} \mathbb{Z}_p$. It then constructs the following elements:

   - It sets $Y = Y'$.
   - For each $i \in S \cup [q_1+1, 2q_1]$, it sets $X_i = \left(X'_{4q_2 i}\right)^{a_0^i}$.
   - For each $i \in [q_1 - 1] \setminus S$, it sets $Z_{q_1-i} = \left(Z'_{q-4q_2 i}\right)^{a_0^{q_1-i}}$. For each $i \in [q_1+1, 2q_1]$, it sets $Z_i = \left(Z'_{4q_2 i}\right)^{a_0^i}$.
   - It sets $Q = \left(Q'\right)^{a_0^{q_1}}$.

   To construct the parallel terms containing β_j, the challenger proceeds as follows:

   - For each $j \in [q_2]$, the challenger sets $Y^{(j)} = \left(Z'_{2j-1}\right)^{\beta'_j}$.
   - For each $i \in [2q_1] \setminus \{q_1\}$ and $j \in [q_2]$, the challenger sets $X_i^{(j)} = \left(X'_{4q_2 i-(2j-1)}\right)^{a_0^i/\beta'_j}$.
   - For each $i \in [2q_1] \setminus \{q_1\}$ and $j \neq k \in [q_2]$, the challenger sets $Z_i^{(j,k)} = \left(Z'_{4q_2 i+2(k-j)}\right)^{a_0^i \beta'_k/\beta'_j}$.

   Finally, algorithm $\mathcal{B}$ sets $T = T'^{a_0^{q_1}}$ and gives the following components to $\mathcal{A}$:

   - $\mathcal{G}$, g, Y, $\{X_i\}_{i \in S \cup [q_1+1,2q_1]}$, $\{Z_{q_1-i}\}_{i \in [q_1-1]\setminus S}$, $\{Z_i\}_{i \in [q_1+1,2q_1]}$; and
   - $\{Y^{(j)}\}_{j \in [q_2]}$, $\{X_i^{(j)}\}_{i \in [2q_1]\setminus\{q_1\}, j \in [q_2]}$, $\{Z_i^{(j,k)}\}_{i \in [2q_1]\setminus\{q_1\}, j \neq k}$, Q, T.

5. Algorithm $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$, which $\mathcal{B}$ also outputs.

First, we argue that the components algorithm $\mathcal{B}$ uses to simulate the challenge are indeed provided by the challenger. We consider each set of components.

- First, consider the components X_i and Z_i when $i \in [q_1+1, 2q_1]$. These depend on the values of $X'_{4q_2 i}$, $Z'_{4q_2 i}$. Since $i \in [q_1+1, 2q_1]$, it follows that $4q_2 i \geq 4q_2(q_1+1) = q + 4q_2 \geq q+1$ and $4q_2 i \leq 8q_1 q_2 = 2q$. Thus, $4q_2 i \in [q+1, 2q]$, and correspondingly, $X'_{4q_2 i}$ and $Z'_{4q_2 i}$ are included in Eq. (E.1).

- Consider the components X_i where $i \in S$. These depend on the values of $X'_{4q_2 i}$. Since $i \in S$, this means $4q_2 i \in S'_0 \subseteq S'$, and so the component is included in Eq. (E.1).

- Consider the components Z_{q_1-i} for $i \in [q_1-1] \setminus S$. These depend on the values of $Z'_{q-4q_2 i}$. Since $4q_2 i < q-1$, it suffices to show that $4q_2 i \notin S'$. Since $q = 4q_1 q_2$, it follows that $q - 4q_2 i = 0 \bmod 4q_2$, so $q - 4q_2 i \notin S'_1$. Since $i \notin S$, it follows that $4q_2 i \notin S'_0$, and the claim holds.

- Consider the components $Y^{(j)}$ for $j \in [q_2]$. These depend on the values of $Z'_{2j-1} = Z'_{q-(q-(2j-1))}$. Since $q-(2j-1) \in [q-1]$, it suffices to show that $q-(2j-1) \notin S'$. Since $2j-1$ is odd, $q-(2j-1) \notin S'_0$. Moreover, since $j \in [q_2]$, $2j-1 < 2q_2$. This means that $q-(2j-1) > q - 2q_2$, so $q-(2j-1) \notin S'_1$, and the claim holds.

- Consider the components $X_i^{(j)}$ for $i \in [2q_1] \setminus \{q_1\}$ and $j \in [q_2]$. These depend on the values of $X'_{4q_2 i-(2j-1)}$.
  - When $i \in [q_1+1, 2q_1]$, it follows that $4q_2 i - (2j-1) \geq 4q_1 q_2 + 4q_2 - (2j-1) > q+1$ for all $j \in [q_2]$. As such $X'_{4q_2 i-(2j-1)}$ is included in Eq. (E.1).
  - When $i \in [q_1 - 1]$, then
  $$4q_2 i - (2j-1) \leq 4q_2(q_1-1) - (2j-1) = q - 4q_2 - 2j + 1 \leq q - 2q_2.$$
  We will show then that $4q_2 i - (2j-1) \in S'_1 \subseteq S'$. First,
  $$4q_2 i - (2j-1) = -(2j-1) \bmod 4q_2.$$
  Since $2j-1 < 2q_2$, we conclude that $4q_2 i-(2j-1) \bmod 4q_2$ is odd and negative, so $4q_2 i-(2j-1) \in S'_1 \subseteq S'$, as required.

- Consider the components $Z_i^{(j,k)}$ for $i \in [2q_1] \setminus \{q_1\}$ and $j \neq k \in [q_2]$. These depend on the values of $Z'_{4q_2 i + 2(k-j)}$.
  - When $i \in [q_1+1, 2q_1]$, it follows that $4q_2 i + 2(k-j) \geq 4q_1q_2 + 4q_2 + 2(k-j) > q+1$ for all $j,k \in [q_2]$. As such $Z'_{4q_2i+2(k-j)}$ is included in Eq. (E.1).
  - When $i \in [q_1 - 1]$, then
$$4q_2 i + 2(k-j) \leq 4q_1q_2 - 4q_2 + 2(k-j) \leq q - 4q_2 + 2(q_2 - 1) \leq q - 2q_2.$$
    Thus, we need to show that $q - 4q_2 i - 2(k-j) \notin S'$. First,
$$q - 4q_2 i - 2(k-j) = -2(k-j) \bmod 4q_2.$$
    This is always even so $q - 4q_2 i - 2(k-j) \notin S'_1$, and moreover, since $k \neq j$ and $j,k \in [q_2]$, this means that $q - 4q_2 i - 2(k-j) \neq 0 \bmod 4q_2$. Hence, $q - 4q_2 i - 2(k-j) \notin S'_0$, as required.

Thus, we conclude that algorithm $\mathcal{B}$ is able to simulate all of the components for $\mathcal{A}$ using its challenge. Let $a', s' \in \mathbb{Z}_p$ be the exponents the challenger samples to construct the q-set-consistent BDHE challenge. Then

$$X'_i = g^{(a')^i} \quad \text{and} \quad Z'_i = g^{(a')^i s} \quad \text{and} \quad Y' = g^s \quad \text{and} \quad Q' = e(g,g)^{(a')^q}$$

We claim that $\mathcal{B}$ perfectly simulates an instance of the (q_1,q_2)-intermediate set-consistent BDHE challenge with randomness

$$a = a_0 \cdot (a')^{4q_2} \quad \text{and} \quad s = s' \quad \text{and} \quad \forall j \in [q_2] : \beta_j = \beta'_j \cdot (a')^{2j-1}. \tag{E.2}$$

Since algorithm $\mathcal{B}$ samples $a', \beta'_j \xleftarrow{R} \mathbb{Z}_p$, the resulting distributions of a and $\beta_1, \ldots, \beta_{q_2}$ are uniform and independent over $\mathbb{Z}_p$, exactly as required in the (q_1,q_2)-intermediate set-consistent BDHE game. It suffices to argue that the challenge constructed by algorithm $\mathcal{B}$ is consistent with the variable assignment in Eq. (E.2). We consider each components separately:

- For all $i \in S \cup [q_1+1, 2q_1]$, we have $X_i = \left(X'_{4q_2 i}\right)^{a_0^i} = g^{a_0^i (a')^{4q_2 i}} = g^{a^i}$.
- For all $i \in [q-1] \setminus S$, we have
$$Z_{q_1 - i} = \left(Z'_{q-4q_2 i}\right)^{a_0^{q_1 - i}} = g^{(a')^{(q-4q_2 i)} a_0^{q_1-i} s} = g^{(a')^{(4q_1q_2 - 4q_2 i)} a_0^{a_1 - i} s} = g^{((a')^{4q_2} a_0)^{q_1 - i} s} = g^{a^{q_1 - i} s}.$$
  For $i \in [q_1+1, 2q_1]$, we have $Z_i = \left(Z'_{4q_2 i}\right)^{a_0^i} = g^{(a')^{4q_2 i} a_0^i s} = g^{a^i s}$
- We have $Y = Y' = g^s$ and $Q = (Q')^{a_0^{q_1}} = e(g,g)^{(a')^q a_0^{q_1}} = e(g,g)^{((a')^{4q_2} a_0)^{q_1}} = e(g,g)^{a^{q_1}}$.
- For all $j \in [q_2]$, we have $Y^{(j)} = \left(Z'_{2j-1}\right)^{\beta'_j} = g^{(a')^{2j-1} s \beta'_j} = g^{s\beta_j}$.
- For all $i \in [2q_1] \setminus \{q_1\}$ and $j \in [q_2]$, we have
$$X_i^{(j)} = \left(X'_{4q_2 i - (2j-1)}\right)^{a_0^i / \beta'_j} = g^{(a')^{4q_2 i - (2j-1)} a_0^i / \beta'_j} = g^{\left((a')^{4q_2} a_0\right)^i / \left((a')^{2j-1} \beta'_j\right)} = g^{a^i / \beta_j}$$
- For all $i \in [2q_1] \setminus \{q_1\}$ and $j \neq k \in [q_2]$,
$$Z_i^{(j,k)} = \left(Z'_{4q_2 i + 2(k-j)}\right)^{a_0^i \beta'_k / \beta'_j} = g^{(a')^{4q_2 i + 2(k-j)} s a_0^i \beta'_k / \beta'_j} = g^{((a')^{4q_2} a_0)^i s \cdot \left((a')^{2k-1} \beta'_k\right) / \left((a')^{2j-1} \beta'_j\right)} = g^{a^i s \beta_k / \beta_j}.$$

Finally, if $T' = e(g,g)^{(a')^q s}$, then

$$T = (T')^{a_0^{q_1}} = e(g,g)^{(a')^q a_0^{q_1} s} = e(g,g)^{(a')^{4q_1q_2} a_0^{q_1} s} = e(g,g)^{a^{q_1} s},$$

which corresponds to the (q_1,q_2)-intermediate set-consistent BDHE distribution where $b = 0$. If the challenger sampled $T' \xleftarrow{R} \mathbb{G}_T$, then $T = (T')^{a_0^{q_1}}$ is also uniform over $\mathbb{G}_T$, so long as $a_0^{q_1} \neq 0$. Since $q_1 = \mathsf{poly}(\lambda)$ and algorithm $\mathcal{B}$ samples $a_0 \xleftarrow{R} \mathbb{Z}_p$, the probability that $a_0^{q_1} = 0$ is at most $q_1/p = \mathsf{negl}(\lambda)$. In this case, algorithm $\mathcal{B}$ perfectly simulates the (q_1,q_2)-intermediate set-consistent BDHE distribution where $b = 1$. We thus conclude that the advantage of algorithm $\mathcal{B}$ is negligibly close to the advantage of $\mathcal{A}$, and the claim follows. □