Registered Functional Encryption for Circuits with Dynamic Bounded Collusion*

Yijian Zhang, Rupeng Yang✉, Zuoxia Yu, and Willy Susilo

Institute of Cybersecurity and Cryptology, School of Computing and Information Technology, University of Wollongong, Wollongong NSW, Australia.
{yijianz,rupengy,zyu,wsusilo}@uow.edu.au

Abstract. *Registered Functional Encryption* (Reg-FE) eliminates the key escrow issue in traditional functional encryption by allowing users to generate their own public/secret keys without the help of a central authority. The public key is then registered to the system along with a function, and all the registered public keys will be aggregated to a compact master public key by a transparent key curator. Messages are encrypted under the master public key, and users can learn functions of encrypted data from the ciphertext using their own secret keys.

Prior constructions of Reg-FE either become insecure when more than one malicious users collude or require the heavy machinery of indistinguishability obfuscation. In this work, we construct the first collusion-resistant Reg-FE without obfuscations. The schemes are secure if the number of colluded users is bounded, and the collusion bound can be either a global parameter or specified by the encryption algorithm dynamically. In both settings, we provide generic constructions of Reg-FE from registered attribute-based encryption and instantiate them from either the standard k-Lin assumption or the succinct LWE assumption.

1 Introduction

Functional Encryption (FE) [BSW11] is an advanced encryption scheme that enables fine-grained access control over the encrypted data. In FE, there exists a central authority that sets up the system and generates a pair of master keys $(\mathsf{mpk}, \mathsf{msk})$. The master public key mpk is used for encryption, and the master secret key msk is used to generate the function secret key sk_f, which is associated with a function f. Given a ciphertext ct that encrypts a message x under mpk, the user holding sk_f can learn $f(x)$ but nothing else.

We can define either the indistinguishability-based (IND) security or the simulation-based (SIM) security for a FE scheme. The IND security requires that it is hard to distinguish encryption of x_0 and encryption of x_1 given a function secret key sk_f, if $f(x_0) = f(x_1)$; while the SIM security requires that an encryption of x and a function secret key sk_f can be simulated given only $f(x)$. The SIM security is stronger and captures the intuition that nothing but the function evaluation result is revealed.

Early constructions of FE focused on restricted functions like predicates [BF01,BB04,SW05,BGW05,LOS+10] and inner-products [ABDP15,ALS16]. Then in [SS10], Sahai and Seyalioglu constructed the first functional encryption scheme for general functions that can be described as any polynomial-sized circuit. However, the construction is only proved to have a weak *single-key security* and the scheme is not secure if users holding different function secret keys collude. To solve the problem, Gorbunov et al. [GVW12] (and a series of subsequent works [GKP+13,ALS16,AR17,AW17,Agr17,AV19]) constructed FE schemes for general circuits in the *static bounded collusion model*, which allows a prior bounded number of users to collude. Recently, Agrawal et al. [AMVY21] and Garg

* We appreciate the anonymous reviewers for their valuable comments. Yijian Zhang and Zuoxia Yu are partially supported by the Australian Research Council (ARC) Discovery Early Career Research (DECRA) DE240100282. Rupeng Yang is partially supported by ARC DECRA DE250101041. Willy Susilo is supported by ARC Laureate Fellowship FL230100033.

et al. [GGLW22] constructed FE schemes in the *dynamic bounded collusion model*, where the bound on the number of colluded users is specified during the encryption phase instead of the setup phase. In a parallel line of work [GGH+13,JLS21,JLS22,RVV24], FE schemes were constructed in the *full collusion model*, where an unbounded number of users may collude. However, the schemes only have IND security due to the strong impossibility results [BSW11,AGVW13]; in addition, the constructions are complex with high costs, and the complexity seems essential given that fully collusion resistant FE implies *indistinguishability obfuscation* (*iO*) [AJ15,BV15].

A key restriction of existing FE schemes is the so-called *key-escrow issue*: the central authority holding the master secret key can essentially decrypt all the ciphertexts in the system and thus must be fully trusted. Such strong trust assumptions are usually difficult to satisfy in practice.

Registered FE. To resolve the key-escrow issue, Francati et al. [FFM+23] and Datta et al. [DPY24] proposed the notion of *registered functional encryption* (Reg-FE), which completely mitigates the need of a central authority.

In particular, each user in a Reg-FE scheme can locally generate his own public/secret key pair $(\mathsf{pk}, \mathsf{sk})$. The user will register the public key pk to the system along with a function f. Then a key curator will aggregate all the public keys into a master public key mpk and will generate the helper secret key hsk for each user. The system supports dynamic registration and the key curator will update the master public key and the helper secret keys once a new user comes. Moreover, the aggregation can be run publicly and deterministically, thus the key curator will not store any secrets and we do not have to fully trust it. The message x will be encrypted under the master public key mpk; a user who has registered her public key pk with a function f can then recover $f(x)$ from the ciphertext using sk and the helper secret key hsk assigned to her.

Following previous studies on FE schemes, one can define either the IND security or the SIM security for a Reg-FE scheme, and we focus on the stronger SIM security in this work. Besides, for a system with L users, the sizes of the master public key and the helper secret key are required to be polynomial in $\log L$. This rules out the trivial solution that simply sets $\mathsf{mpk} = (\mathsf{pk}_i, f_i)_{i\in[L]}$ and encrypts x by encrypting $f_i(x)$ under pk_i for all $i \in \{1, \ldots, L\}$.

The notion of registered FE implies several previously proposed registration-based cryptographic primitives, such as registration-based encryption [GHMR18,GHM+19] and registered attribute-based encryption (Reg-ABE) [HLWW23,GLWW24,AT24,CHW25a,WW25], and we can view them as Reg-FE for predicate functions. Besides, [ZLZ+24,DPY24] constructed Reg-FE for linear/quadratic functions. Reg-FE for general functions are also constructed in the literature. Francati et al. [FFM+23] and Datta et al. [DPY24] constructed Reg-FE schemes for general circuits in the full collusion model, but the constructions only achieve the weaker IND security and rely on the heavy machinery of *iO*. Branco et al. [BLM+24] showed that Reg-FE for circuits can be constructed from Reg-FE for linear functions and provide instantiations from pairing. However, the scheme only has single-key security, i.e., the adversary can only register at most one public key in the system. Then in [ZCHZ24], Zhang et al. constructed single-key secure Reg-FE schemes from either pairing or lattice; they also upgraded the schemes to be secure in the static bounded collusion model, where the adversary may register a prior bounded number of public keys. However, their latter scheme is only secure against a semi-honest adversary, which must register honestly generated public keys to the system.

To summarize, if we hope to have a registered FE for general circuits with SIM security against a fully malicious adversary, the only known constructions are single-key secure. Even if we consider the weaker IND security, collusion resistance is only achieved by constructions from *iO*. This stands in sharp contrast to standard FE, where constructions with SIM security from simple assumptions are known in the static/dynamic bounded collusion model. This raises the following natural question:

How to construct static/dynamic bounded collusion Reg-FE for general circuits with SIM security?

Table 1. Comparison of existing Reg-FEs for general circuits

	Model	Collusion	Malicious	Security	Assumption
[FFM+23,DPY24]	Full	–	✓	AD-IND	*iO*
[PST25]	Full	–	✓	SEL-IND	*priO*
[BLM+24]	Static	1	✓	SEL-IND	q-type DDH
[ZLZ+24]†	Static	1	✓	AD-IND	k-Lin
[ZCHZ24]	Static	1	✓	AD-SIM	k-Lin/scLWE
	Static	Q	✗	AD-SIM	k-Lin/scLWE
§B	Static	Q	✓	NA-SIM	k-Lin/scLWE
§C	Static	Q	✓	AD-SIM	k-Lin/scLWE
§D	Dynamic	Q	✓	NA-SIM	k-Lin/scLWE
§4-7	Dynamic	Q	✓	AD-SIM	k-Lin

* The column "Collusion" refers to the collusion bound on the number of registered users who collude with the adversary. Here, Q can be selected as any value within $\{1, 2, \ldots, L\}$ and the symbol "–" denotes non-applicability.

** The column "Malicious" indicates whether the scheme is secure against malicious users.

*** In the column of "Security", "SEL", "NA" and "AD" stand for "selective", "non-adaptive" and "adaptive", which will be explained in Section 2. In the column of "Assumption", "*iO*" and "*priO*" denote indistinguishability obfuscation for general circuits and pseudorandom functions, respectively; "scLWE" denotes the succinct LWE assumption [Wee24].

† While [ZLZ+24] only provides Reg-FE for linear functions, it can be transformed into a Reg-FE for general circuits via the framework in [BLM+24].

1.1 Our Results

In this work, we answer the above question affirmatively by constructing registered FE for general circuits in both the static bounded collusion model and the dynamic bounded collusion model. The constructions start with a registered ABE scheme, which can be instantiated from either the k-Lin assumption [EHK+13] or the succinct learning with errors (LWE) assumption [Wee24]. We then upgrade the Reg-ABE scheme to Reg-FE schemes in the static/dynamic bounded collusion model by applying standard cryptographic tools. We show the roadmap from Reg-ABE to Reg-FE in Figure 1, and we compare the resulting schemes with existing Reg-FE schemes for general circuits in Table 1. Besides, our techniques are useful in applications beyond constructing Reg-FE. As an example, our techniques can be applied to construct new multi-authority ABE. Next, we describe our constructions in different models in detail.

Static Bounded Collusion Reg-FE. In this model, the adversary can register some maliciously generated public keys to the system or corrupt some honest users to get their secret keys, where the total number of malicious public keys and corrupted secret keys is bounded by a number Q determined when setting up the system. Let $f_1, \ldots, f_Q$ be the functions associated with these keys mastered by the adversary, then the security requires that encryption of a message x chosen by the adversary can be simulated given only $f_1(x), \ldots, f_Q(x)$.

Our construction of Reg-FE in this model proceeds in two steps. First, we construct a single-key secure Reg-FE (denoted as 1-bound Reg-FE hereafter) from Reg-ABE. The construction follows the construction of single-key secure Reg-FE in [ZCHZ24], and it only requires a very simple Reg-ABE scheme that simply accepts all attributes. Then we compile the 1-bound Reg-FE to Q-bound Reg-FE. This construction is inspired by the well known paradigm presented in [GVW12], which upgrades a single-key secure FE to a Q-bound FE using MPC techniques. However, the original construction is not secure in the Reg-FE setting since the adversary may act maliciously when generating

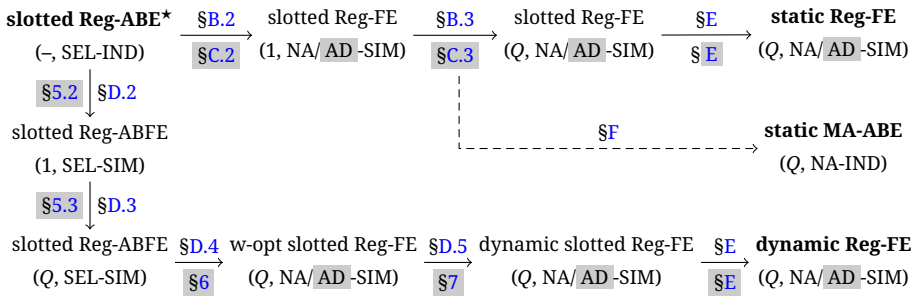

Fig. 1. The roadmap for constructing static Reg-FE and dynamic Reg-FE. For each construction, we will consider both the non-adaptive (NA) security and the adaptive (AD) security, and the constructions towards adaptive security are highlighted by gray boxes. The terms below each primitive represent the notion of (collusion bound, security level) that can be achieved by it. We use $\star$ to denote that the scheme may require a stronger property called equivocality when constructing dynamic Reg-FE schemes. We use "slotted" to denote Reg-FE with a stateless aggregator (rather than a stateful curator) and "w-opt" to denote Reg-FE with the weakly optimal compactness. The dashed arrow is used to denote that our derandomization technique is also applicable to upgrade 1-bound security into Q-bound security for static MA-ABE.

the public key. We solve the problem by applying a novel *derandomization* technique developed in this work to derandomize the generation of (parts of) the public keys.

The above construction is only secure against a non-adaptive adversary, where the adversary must register public keys or corrupt honest users before seeing the simulated ciphertext, and the simulated ciphertext may not be compatible with secret keys corrupted by the adversary after the challenge phase. We solve the problem by using a chameleon hash function and a non-interactive zero-knowledge (NIZK) proof. Besides, we require the underlying Reg-ABE to additionally have rerandomized public keys. We note that existing Reg-ABE schemes from k-Lin and succinct LWE have this additional property. Besides, the other primitives, including chameleon hash and NIZK proofs, can be instantiated from either the k-Lin assumption or the standard LWE assumption. Thus, we have:

Theorem 1 (Informal). *Assuming the k-Lin assumption or the succinct LWE assumption, there exist registered FE for circuits with non-adaptive/adaptive simulation-based security in the static bounded collusion model.*

Dynamic Bounded Collusion Reg-FE. In the dynamic bounded collusion model, the adversary can still register some malicious public keys and corrupt some users to get their secret keys. Also, we will pose an upper bound on the number of keys mastered by the adversary. However, the upper bound Q is now determined by the encryption algorithm (for each ciphertext) instead of the setup algorithm (for all ciphertexts in the system).

The Reg-FE constructed above does not fit the dynamic bounded collusion model. This is because their parameters, even including the user's public key and secret key, are polynomial in the collusion bound Q. This dependence forces the system to set a global upper bound on the number of collusions. We solve the problem by applying the techniques in the previous construction of dynamic bounded collusion FE schemes. However, these techniques are not compatible with our derandomization technique used in the construction of Reg-FE with static bounded collusion. We then develop new *pseudorandomization* and *rerandomization* techniques to defend against a fully malicious adversary in the dynamic bounded collusion setting. We also provide two constructions of Reg-FE in this model: a construction with non-adaptive security from a standard Reg-ABE and a construction with adaptive security from a stronger version of Reg-ABE, which is denoted as equivocal Reg-ABE in this work. Unfortunately, we only know how to construct equivocal Reg-ABE from pairing, and therefore we have:

Theorem 2 (Informal). *Assuming the k-Lin assumption or the succinct LWE assumption, there exist registered FE for circuits with non-adaptive simulation-based security in the dynamic bounded collusion model.*

Theorem 3 (Informal). *Assuming the k-Lin assumption, there exist registered FE for circuits with adaptive simulation-based security in the dynamic bounded collusion model.*

Discussion. In an independent and concurrent work [GY25], Goyal and Yadugiri construct the first multi-authority FE (MA-FE) [BCG+17] that are secure against malicious authorities without obfuscation. Similar to Reg-FE, MA-FE is also proposed to solve the key-escrow issue in traditional FE schemes, but it decentralizes the trust of a single central authority to a number of independent (sub-)authorities. The constructions provided in [GY25] (with security against malicious authority) also achieve adaptive SIM security in the static/dynamic collusion model, and security of their schemes rely on the standard LWE or SXDH assumptions. However, their scheme is only secure when the number of malicious authorities is within $O(\log \lambda)$ due to a strong upper bound shown in the same work. In contrast, our Reg-FE schemes do not have such unnatural limit on the number of colluded users, but our constructions rely on either the standard k-Lin assumption or the succinct LWE assumption, where the latter assumption is stronger than the standard LWE assumption.

Beyond the above comparison with MA-FE, another important aspect concerns the practical efficiency of Reg-FE. To our knowledge, all known Reg-FE schemes for circuits that tolerate more than one malicious user, including ours, make non-black-box use of the underlying primitives. Such techniques pose challenges to practical efficiency, and we leave developing more practical constructions, ideally avoiding such non-black-box techniques, as future work.

1.2 Organization

We organize the rest of the paper as follows:

- We give an overview of all our constructions in Section 2. The preliminaries for all cryptographic tools employed in this work are provided in Section 3. For simplicity, we only focus on the construction of dynamic Reg-FE with adaptive security in the main body. We begin with the notion of equivocal slotted Reg-ABE and present a concrete instantiation from the k-Lin assumption in Section 4. Then we show how to transform it into a dynamic slotted Reg-FE with adaptive security step by step from Section 5 to Section 7. The slotted Reg-FE can be transformed into a full-fledged Reg-FE using standard techniques, where the transform is recalled in Appendix E. Some missing proofs for the constructions in the main body are given in Appendix A.
- Then, we present other results of this work: we give the constructions of static slotted Reg-FE achieving non-adaptive/adaptive security in Appendix B and Appendix C, respectively; we give the construction of dynamic slotted Reg-FE with non-adaptive security in Appendix D.
- All slotted Reg-FE constructed in this work can be transformed into full-fledged Reg-FE following the compiler given in Appendix E. Besides, in Appendix F, we show how our techniques can be used in constructing strong MA-ABE.

2 Technical Overview

We will follow the blueprint of existing registered primitives, i.e., first construct a slotted version of Reg-FE and then generically transform slotted Reg-FE schemes into full-fledged Reg-FE schemes.

Slotted Reg-FE. As noted in [HLWW23,FFM+23,ZCHZ24], Reg-FE can be generically derived from slotted Reg-FE (sReg-FE) via a standard "power-of-two" method. In slotted Reg-FE, the key curator is replaced by a stateless aggregator who aggregates all public keys and circuits to generate mpk and hsk's at once. In the following, we adopt the notion of Q-bound sReg-FE that is secure against at most Q corrupted and malicious users. Let L denote the total number of registered users, after collecting all $\{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}$, the aggregator would publish master public key mpk and helper secret keys $\{\mathsf{hsk}_j\}_{j\in[L]}$. With $(\mathsf{sk}_i, \mathsf{hsk}_i)$, the user under slot i can properly decrypt the ciphertext ct encrypted by (mpk, x) and obtain $\mathsf{C}_i(x)$. For sReg-FE with static collusion, all system overheads depend on Q, so mpk and hsk would be of size $\mathsf{poly}(\lambda, C, Q, \log L)$ where C is the circuit size. As for dynamic collusion, it instead requires mpk and hsk of size $\mathsf{poly}(\lambda, C, \log L)$.

On the other hand, we also consider sReg-FE for predicate functions. For example, in the identity-based setting, x and C_i can be represented as a tag-message pair $(\mathsf{tag}, \mathsf{m})$ and a tag tag_i, respectively. It has $\mathsf{C}_i(\mathsf{tag}, \mathsf{m}) = (\mathsf{tag}, \mathsf{m})$ if $\mathsf{tag} = \mathsf{tag}_i$ and $\mathsf{C}_i(\mathsf{tag}, \mathsf{m}) = (\mathsf{tag}, \perp)$ otherwise. This is the so-called slotted registration-based encryption (sRBE). Similarly, we can define slotted registered attribute-based encryption (sReg-ABE) in which x and C_i can be represented by a policy-message pair (P, m) and an attribute att_i, respectively. It has $\mathsf{C}_i(\mathsf{P}, \mathsf{m}) = (\mathsf{P}, \mathsf{m})$ if $\mathsf{P}(\mathsf{att}_i) = 1$ and $\mathsf{C}_i(\mathsf{P}, \mathsf{m}) = (\mathsf{P}, \perp)$ otherwise.

Security definitions. We aim at the strongest adaptive SIM security, in which any adversary cannot distinguish the following two worlds:

- *Setup phase:* With input $(1^L, 1^Q, C)$ specified by the adversary, where 1^λ denotes the security parameter, the challenger returns a common reference string crs to the adversary. Note that if we consider a dynamic collusion, 1^Q will be specified in the challenge phase.
- *Pre-challenge phase:* The challenger initialises a corruption/malicious set $\mathcal{K} = \emptyset$. In this phase, the adversary can freely make the following queries:
  - Key generation query: it specifies a slot index $i \in [L]$ and obtains a public key pk_i from the challenger.
  - Corruption query: it specifies a slot index $i \in [L]$ and a public key pk_i, then obtains a secret key sk_i from the challenger.
- *Challenge phase:* The adversary specifies all keys and circuits $\{(\mathsf{pk}_i^*, \mathsf{C}_i^*)\}_{i\in[L]}$ and a challenge message x^*. For each $j \in [L]$, if pk_j^* is either previously corrupted or maliciously created by the adversary, then update $\mathcal{K} = \mathcal{K} \cup \{j\}$. The challenger will return a challenge ciphertext ct^* that is either normally generated from x^* (real world) or simulated from $\{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}}$ (ideal world). We require $|\mathcal{K}| \le Q$.
- *Post-challenge phase:* The adversary continues to make corruption queries on (j, pk_j^*): the challenger returns the secret key sk_j^* that is either normally generated (real world) or simulated from $(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))$ (ideal world). Update $\mathcal{K} = \mathcal{K} \cup \{j\}$ and we always require $|\mathcal{K}| \le Q$.
- *Output phase:* It outputs a bit to indicate the real/ideal world.

The above security can be weakened into several forms: (1) *non-adaptive* security: the adversary cannot make any key corruption queries in the post-challenge phase; (2) *selective* security: the adversary must specify x^* before seeing crs. In sReg-ABE, policy-selective security is often considered, meaning that the challenge policy is revealed before seeing crs. Since Reg-FE will inherit the security of underlying sReg-FE, we only discuss sReg-FE in the following sections.

2.1 Warm-up

We present a strawman construction of Q-bound sReg-FE by simply applying Gorbunov et al.'s technique [GVW12] to a 1-bound sReg-FE scheme; this process is similar to Zhang et al.'s construction with "very weak" Q-bound security [ZCHZ24].

We start with a 1-bound sReg-FE scheme for general circuits and try to upgrade it into a Q-bound sReg-FE scheme for NC^1 circuits (of depth D). Concretely, the encryptor splits a message x into N shares $(x_1, \ldots, x_N)$ via classic t-out-of-N Shamir's secret sharing. Then, he utilizes N sub-systems of 1-bound sReg-FE to encrypt each share and hence obtain the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$. Here, we require each user to sample and register their public keys into $Dt+1$ sub-systems (denoted by a subset $\Gamma \subseteq [N]$ of size $Dt+1$). When a user who owns a circuit C wants to learn $\mathsf{C}(x)$, he can use secret keys to decrypt $\{\mathsf{ct}_i\}_{i\in\Gamma}$ and hence acquire $\{\mathsf{C}(x_i)\}_{i\in\Gamma}$. Then, by the homomorphism of Shamir's secret sharing, he can recover $\mathsf{C}(x)$ from $\{\mathsf{C}(x_i)\}_{i\in\Gamma}$. To utilize the security of secret sharing, it requires randomizing encrypted shares for distinct circuits. Therefore, the encryptor can encrypt S random (zero-constant) polynomials together with shares of the message x, and each user picks v polynomials (denoted by a subset $\Delta \subseteq [S]$ of size v) to complete such randomization.

Security analysis. The most important part of security analysis is the distribution of $\{(\Gamma_i, \Delta_i)\}_{i\in[L]}$. Assuming that there are Q corrupted/malicious users whose slot indices are stored in the set $\mathcal{K}$, since the user under slot i only has decryption access to $tD+1$ sub-systems of 1-bound sReg-FE, the security of the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$ is guaranteed by the following facts:

- *Collusion resistance against a single user.* For those sub-systems where there is at most one corrupted/malicious user who has decryption access, it can rely on the 1-bound security of sReg-FE to hide the encrypted shares.
- *Small pairwise intersections.* For those sub-systems where there are two or more corrupted/malicious users who have decryption access, the 1-bound security would be compromised, and the encrypted shares are hence leaked to the adversary. We stress that if the number of these sub-systems reaches the threshold t, the adversary can trivially recover the entire message x. To avoid this case, the probability $\Pr\left[\left|\bigcup_{i\in\mathcal{K}, j\in\mathcal{K}\setminus\{i\}}(\Gamma_i \cap \Gamma_j)\right| \geq t\right]$ is required to be negligible in λ.
- *Cover freeness.* As we mentioned before, the decryption results of corrupted/ malicious users should be randomized with at least a unique random polynomial. Namely, the probability $\Pr\left[\Delta_i \setminus \left(\bigcup_{j\in\mathcal{K}\setminus\{i\}} \Delta_j\right) = \emptyset\right]$ is required to be negligible in λ for all $i \in \mathcal{K}$.

Gorbunov et al. recommended to set $t = \Theta(Q^2\lambda), N = \Theta(D^2Q^2t), v = \Theta(\lambda)$ and $S = \Theta(Q^2v)$. Recently, Lai et al. [LLW21] further improved the parameter choice S of cover freeness to allow it to be linear in Q. Nevertheless, we note that in all these analyses, small pairwise intersections and cover freeness hold as long as all sets $\{(\Gamma_i, \Delta_i)\}_{i\in\mathcal{K}}$ are *uniformly and independently* sampled.

The above strawman construction only enables the computation of NC^1 circuits, rather than general circuits. Indeed, this construction is designed by following a reusable dynamic MPC protocol for NC^1 circuits [WOG88], and we can instead employ a reusable dynamic MPC protocol for general circuits [AV19] to obtain sReg-FE for general circuits. Since both protocols construct sReg-FE in a similar way, i.e., they require both small pairwise intersections and cover freeness, we will focus on these two probability arguments in the following. The formal definition of reusable dynamic MPC is available in Section 3.7.

2.2 The Constructions of Static Slotted Reg-FE

Next, we want to adapt the strawman construction into static bounded collusion sReg-FE with Q-bound adaptive SIM security. However, there are two critical challenges that arise.

Challenge 1: Generating the Sets for Q-Bound Security. The Q-bound security calls for the sets $\{(\Gamma_i, \Delta_i)\}_{i\in[L]}$ subject to small pairwise intersections and cover freeness, whereas it cannot appropriately ensure this point due to the existence of malicious users and the determinism of the aggregator. Next, we discuss the issues that appear when the sets are generated by different parties.

What will happen if users generate the sets? If users can sample the sets, we argue that the security will be immediately compromised, even with powerful tools such as NIZK.

A simple but effective attack is: let the adversary register two malicious users with the same Γ set and the same Δ set, then it will immediately break small pairwise intersections and cover freeness! Even if we force users to attach NIZK proofs to show the well-formness of public keys as in [ZZGQ23,BLM+24], this attack is still possible because NIZK only works for deterministic operations. An alternative method is to employ *Pseudorandom Function* (PRF) to sample (Γ, Δ) in a deterministic manner, but sharing a PRF key among users will be a problem.

What will happen if aggregator generates the sets? Given that allowing users to sample the sets is risky, this task is naturally transferred to the aggregator. As the aggregator is transparent and deterministic, we sample all sets at the setup stage. However, this leads to new issues:

(i) All $\{\Gamma_i\}_{i\in[L]}$ are fixed before seeing the challenge ciphertext, so the reduction cannot locate the sets corrupted by the adversary in the post-challenge phase. For our strawman construction, the reduction cannot tell if the security of ct_j for $j \in [N]$ should be reduced to the 1-bound security or the security of underlying MPC protocols, so that it is unlikely to achieve adaptive security.

(ii) Previous probability analysis and parameter choices for (t, N, v, S) no longer hold. We turn to prove that, for any adversary, it is infeasible to find Q sets among $L \gg Q$ sets such that either small pairwise intersections or cover freeness does not hold.

Attempt: Random Oracle Model. Adopting a random oracle seems to resolve issue (ii). In particular, the aggregator can query a random oracle on input $\{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}$ to obtain the sets $\{(\Gamma_i, \Delta_i)\}_{i\in[L]}$ that are uniformly sampled. Then, by the extended probability arguments in [GY24], the corrupted/malicious sets are still subject to small pairwise intersections and cover freeness.

However, the above measure still leads to non-adaptive security, i.e., it cannot handle issue (i). The readers familiar with registered cryptography might know that [HLWW23,FFM+23,DPY24] have shown an intrinsic implication between non-adaptive security and adaptive security for sReg-FE: a simple reduction can be made to demonstrate the equivalence for them. However, this implication only works for full-collusion IND security because it critically relies on the condition that the adversary can only corrupt users whose registered functions satisfy $f(x_0) = f(x_1)$, a condition that is admissible in the IND case but is not guaranteed in the SIM case. A similar problem also appears in [GY24], and they suggested exploiting the standard complexity leveraging technique to achieve adaptive security, but it incurs a sub-exponential reduction loss. Next, we present our solution towards more standard security.

Our Solution: New Derandomization Technique. Now we place more insights on small pairwise intersections and cover freeness. An interesting observation is that these two arguments are essentially the same. More precisely, we can reduce them to a new argument, i.e., the maximal intersection length of any two distinct sets is smaller than a threshold value. Along this way, it is possible to enable the choices of (t, N, v, S) to still rely on Q even given a large number of sets, and we can hence resolve issue (ii) in the standard model! Concretely, for small pairwise intersections, we have:

$$\left|\bigcup_{i\in\mathcal{K}, j\in\mathcal{K}\setminus\{i\}} (\Gamma_i \cap \Gamma_j)\right| < Q^2 \cdot \max_{i\in\mathcal{K}, j\in\mathcal{K}\setminus\{i\}} \left|\Gamma_i \cap \Gamma_j\right|.$$

For cover freeness and all corrupted/malicious slots $i \in \mathcal{K}$, we have:

$$\left|\Delta_i \setminus \left(\bigcup_{j\in\mathcal{K}\setminus\{i\}} \Delta_j\right)\right| = \left|\Delta_i \setminus \bigcup_{j\in\mathcal{K}\setminus\{i\}} (\Delta_i \cap \Delta_j)\right| > v - Q \cdot \max_{j\in\mathcal{K}\setminus\{i\}} \left|\Delta_i \cap \Delta_j\right|.$$

It thus suffices to show $t \geq Q^2 \cdot \max_{i \neq j} |\Gamma_i \cap \Gamma_j|$ and $v \geq Q \cdot \max_{i \neq j} |\Delta_i \cap \Delta_j|$. Our next task is to prove $\max_{i \neq j} |\Gamma_i \cap \Gamma_j|$ and $\max_{i \neq j} |\Delta_i \cap \Delta_j|$ are only related to some threshold values such as λ.

We observe that this question is quite similar to the design of error-correcting codes that always assume two distinct codes with minimal distance. Therefore, inspired by the Reed-Solomon error-correcting code [RS60], we find a novel method that enables $\max_{i \neq j} |\Gamma_i \cap \Gamma_j|$ and $\max_{i \neq j} |\Delta_i \cap \Delta_j|$ to only depend on the security parameter λ, without involving any non-standard assumptions. Take small pairwise intersections as an example, we can use a collision-resistant hash function to *deterministically* generate a set Γ_i under slot i:

- Given (i, pk_i), the aggregator would define a polynomial of degree $\lambda - 1$:

$$f_i(x) = a_{i,\lambda-1} x^{\lambda-1} + \cdots + a_{i,1} x + a_{i,0}$$

  where it hashes the input (i, pk_i) to coefficients $(a_{i,0}, a_{i,1}, \ldots, a_{i,\lambda-1})$ that are defined over $\mathbb{Z}_p$.
- Then, the aggregator evaluates $f_i(1), \ldots, f_i(Dt+1)$, and injectively maps each point $(k, f_i(k))$ to distinct values $\gamma_{i,k} \in [N]$. Set $\Gamma_i = \{\gamma_{i,k}\}_{k \in [Dt+1]}$.

We claim that there are at most $\lambda - 1$ intersection points between f_i and f_j for any $i \neq j$ because if there exist λ intersection points, f_i and f_j can share the same coefficients based on well-known Lagrange interpolation, which obviously violates the collision resistance of hash functions. Therefore, for all $i \neq j$, the probability $\Pr\left[|\Gamma_i \cap \Gamma_j| \geq \lambda\right]$ is negligible, so we can just set $\max_{i \neq j} |\Gamma_i \cap \Gamma_j| = \lambda$. Similarly, it would also have $\max_{i \neq j} |\Delta_i \cap \Delta_j| = \lambda$.

Parameter regime. Regarding our derandomization technique, we recommend to set $t = \Theta(Q^2\lambda)$ for small pairwise intersections and $v = \Theta(Q\lambda)$ for cover freeness. Besides, to *injectively* map evaluated points to the values over $[N]$ (resp. $[S]$), we recommend to set $p = \Theta(Dt)$ and $N = \Theta(Dtp) = \Theta(D^2Q^4\lambda^2)$ (resp. $S = \Theta(v^2) = \Theta(Q^2\lambda^2)$).

Challenge 2: Predicting the Sets in the Adaptive Corruption Setting. The above construction is just non-adaptive secure, as all sets are still fixed before the challenge phase. Fortunately, a key difference is that now we only require collision-resistance. In particular, to simulate the challenge ciphertext ct^* from $\{(\mathsf{C}_j, \mathsf{C}_j(x^*))\}_{j \in \mathcal{K}}$, we rely on the following two facts (recall that ct^* consists of N sub-ciphertexts of 1-bound sReg-FE):

(i) *The adaptive SIM security of 1-bound sReg-FE*. This property can be guaranteed for 1-bound sReg-FE by utilizing non-committing techniques [DN00,GVW12,GGLW22] to upgrade non-adaptive security into adaptive security;

(ii) *The ability of predicting the sets* $\{\Gamma_j\}_{j \in \mathcal{K}}$. In the challenge phase, there may exist some corrupted slots that have not been queried, so we can only locate $\{\Gamma_j\}_{j \in \mathcal{K}'}$, where $\mathcal{K}'$ only stores the corrupted/malicious slots queried in the pre-challenge phase. This is not a problem in the proof of previous centralized FE schemes, as they allow the reduction algorithm to pre-sample these sets. However, our construction publicly binds the slot j to a unique set Γ_j, so that we cannot appropriately simulate ct^*.

Our Solution: Private Registration. Our idea is to register all sets $\{\Gamma_i\}_{i \in [L]}$ privately. To do so, we allow each user to sample sets by themselves and keep these sets secret. Specifically, we let the user i sample Γ_i as in our non-adaptive construction but adopt a *chameleon hash* function, instead of a general hash function. This is because chameleon hash additionally provides the invertible property that can be used to answer adaptive corruptions from the adversary. By this, the reduction can program all corrupted sets to simulate the challenge ciphertext. A key point is that, as we warned before, allowing users to provide the set is risky. Fortunately, now all we want is to ensure the collision-resistance, which can be done by forcing each user to attach a NIZK proof to claim the usage of distinct hash inputs.

On the other hand, the aggregation procedure needs to hide all sets. Recall that in our non-adaptive construction, user i openly registered his public keys into multiple 1-bound sReg-FE sub-systems indexed by Γ_i, so that Γ_i is

publicly known. Therefore, we require user i to register *rerandomized* public keys into the other sub-systems that are not in Γ_i. Here, a rerandomized public key is generated based on a normally generated public key, but no one can tell if it is rerandomized or normally generated. Existing 1-bound sReg-FE schemes can support this property. Putting all these together, we manage to hide the binding relationship between user slots and registered sets, hence completing the whole proof.

Above all, we resolve all challenges towards static bounded collusion sReg-FE with adaptive SIM security. It can be built from 1-bound sReg-FE, which is known from existing sReg-ABE schemes that simply accept all attributes.

2.3 The Constructions of Dynamic Slotted Reg-FE

Now we pay attention to dynamic bounded collusion sReg-FE for general circuits. It is conceptually similar to static sReg-FE except that the declaration time of the collusion bound Q is postponed to the encryption stage. Besides, the overheads of all system algorithms (except for encryption and decryption) are independent of Q, so it naturally supports $(\mathsf{mpk}, \mathsf{hsk})$ of shorter size $\mathsf{poly}(\lambda, C, \log L)$. To construct dynamic bounded collusion sReg-FE, we will define an important intermediate primitive called *weakly optimal slotted Reg-FE*.

Weakly Optimal Slotted Reg-FE. Following Garg et al.'s blueprint [GGLW22] of designing dynamic bounded collusion FE, we observe that dynamic bounded collusion sReg-FE can be implicitly derived from weakly optimal sReg-FE via the standard power-of-two method. Weakly optimal sReg-FE is defined the same as static bounded collusion sReg-FE, but additionally requires weakly optimal compactness, i.e., all overheads except for encryption and decryption are bounded by $\mathsf{poly}(\log Q)$, instead of $\mathsf{poly}(Q)$. In a nutshell, dynamic bounded collusion sReg-FE can be constructed by invoking $\log L + 1$ sub-systems of weakly optimal sReg-FE that specify different collusion bounds ranging from $2^0, 2^1$ to $2^{\log L}$. To flexibly choose a desired bound Q, the encryptor just selects the $\lceil \log Q \rceil$-th sub-system (with a collusion bound at most $2Q$) and then generates his ciphertext as in weakly optimal sReg-FE.

To explore how to construct weakly optimal sReg-FE, we first recap a mild adaptation of Garg et al.'s technique. Concretely, we can build weakly optimal sReg-FE from *slotted Registered Identity-Based Functional Encryption* (sReg-IBFE for short); the latter can be viewed as a combination of sRBE and static sReg-FE. In sReg-IBFE, user i registers his circuit C_i and a tag tag_i into $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$, and the message x is encrypted under mpk and a tag tag. The circuit evaluation $\mathsf{C}_i(x)$ can be recovered from the ciphertext only if $\mathsf{tag}_i = \mathsf{tag}$. As for security, the ciphertext leaks no further information about x even when the adversary colludes with Q users identified by tag and an arbitrary number of users identified by other tags. Based on our derandomization technique, it would be rather easy to construct sReg-IBFE from sRBE. For a Q-bound sReg-IBFE over tag space $\{0,1\}^m$, the compactness requires $(\mathsf{mpk}, \mathsf{hsk})$ of sizes $\mathsf{poly}(\lambda, m, C, Q, \log L)$. However, we remark that the problem lies in the compilation from sReg-IBFE to weakly optimal sReg-FE.

Challenge 3: Generating the Tags in the (Non-)Adaptive Corruption Setting. To build a weakly optimal sReg-FE, it usually initializes a λ-bound sReg-IBFE scheme where we set the security parameter λ as the collusion bound and $\{0,1\}^{\log Q}$ as the tag space (it often assumes $\lambda < Q \ll L$). Suppose that each user is allowed to pick their own tags, after registering all circuits and tags, the final ciphertext consists of Q sub-ciphertexts of sReg-IBFE, each of which encrypts x for $\mathsf{tag} \in [Q]$. For decryption, user i will pick the sub-ciphertext associated with tag_i and perform the decryption of sReg-IBFE to recover the desired evaluation result. It is easy to see that this construction exactly satisfies weakly optimal compactness with parameter sizes $\mathsf{poly}(\lambda, C, \log Q, \log L)$.

To claim the security of the above weakly optimal sReg-FE scheme, we need to analyze the distribution of the tags $\{\mathsf{tag}_j\}_{j\in\mathcal{K}}$ from corrupted/malicious users. Based on the probability argument concluded by [AV19,GGLW22],

the security can be preserved as long as all these tags are ensured to be *uniformly* sampled. On the contrary, if an adversary colludes with more than λ corrupted/malicious users under the same tag, the λ-bound security of sReg-IBFE will be trivially compromised! This is quite similar to the problem we met in **Challenge 1**, but now it becomes infeasible to perform a derandomization trick again, as there must exist more than λ same registered tags (according to the pigeonhole principle), no matter how these tags are sampled. Even if we employ a random oracle to sample $\{\mathsf{tag}_i\}_{i\in[L]}$ as in **Attempt**, it cannot even achieve non-adaptive security, as all tags are publicly generated so that the adversary can always find λ same tags. Therefore, we need to handle this issue more carefully.

First Step: Pseudorandomization Technique. We introduce a new pseudorandomization technique to get around the above inherent limitation. This is inspired by Goyal et al.'s technique [GY24] to construct MA-FE with weakly selective security[1]. Next, we show how to apply it in a different way to achieve stronger (non-)adaptive security.

The high-level idea is to let the encryptor dominate the tag generation. To do so, our idea is to rely on sReg-ABE, instead of simple sRBE. More specifically, we try to construct a *slotted Registered Attribute-Based Functional Encryption* (sReg-ABFE) from sReg-ABE, and then compile sReg-ABFE to weakly optimal sReg-FE. In sReg-ABFE, user i registers his circuit C_i along with attribute att_i into $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$, and the ciphertext is associated with x and a policy P. For decryption, $\mathsf{C}_i(x)$ can be recovered only if $\mathsf{P}(\mathsf{att}_i) = 1$. The Q-bound security is guaranteed as long as the total number of corrupted/malicious users whose attributes satisfy the policy P does not exceed Q. The compactness requires $(\mathsf{mpk}, \mathsf{hsk})$ of size $\mathsf{poly}(\lambda, P, C, Q, \log L)$, where P denotes the policy size.

Next, we introduce a new compilation from sReg-ABFE to weakly optimal sReg-FE, where it just requires a weak policy family represented by NC^1 circuits. Concretely, we first initialize a λ-bound sReg-ABFE scheme and let each user treat their slot indices as attributes, i.e., set $\mathsf{att}_i = a_i$ to be a randomly sampled attribute for $i \in [L]$. After registering all circuits and attributes, the ciphertext is generated as follows:

- Sample a weak PRF key $\mathsf{k} \in \{0,1\}^\lambda$, where the output space of weak PRF is $[Q]$ and the evaluation algorithm is efficiently computable in NC^1 circuits. This is a mild assumption that can be known from the standard k-Lin/LWE assumption [NR97,BPR12,BP14,DS15,Zha25]; [2];
- For all $\mathsf{tag} \in [Q]$, it specifies the policy

$$\mathsf{P}_{\mathsf{tag}}(a_i) = 1 \iff \mathsf{wPRF.Eval}(\mathsf{k}, a_i) = \mathsf{tag}. \tag{1}$$

  where $i \in [L]$. Then generate Q sub-ciphertexts of sReg-ABFE regarding $(\mathsf{P}_{\mathsf{tag}}, x)$ as the final ciphertext of weakly optimal sReg-FE.

User i can now decrypt one of Q sub-ciphertexts to obtain $\mathsf{C}_i(x)$. Since the size of $\mathsf{P}_{\mathsf{tag}}$ is bounded by $\mathsf{poly}(\lambda, \log Q, \log L)$, it remains weakly optimal compactness.

As for security proof, since k keeps hidden before seeing the challenge ciphertext, the adversary cannot know how to select i to obtain λ same tags, as otherwise we can exploit it to break the pseudorandomness of weak PRF. Therefore, the resulting weakly optimal sReg-FE scheme is non-adaptively secure. In addition, since k is randomly sampled, the challenge policy of underlying sReg-ABE can be determined in advance, so our construction just relies on the existence of sReg-ABE for NC^1 with policy-selective security [FWW23,CHW25b,ZZC+25].

Second Step: Rerandomization Technique. A natural question is whether we can improve above construction to achieve fully adaptive security. Recall that, upon receiving a challenge ciphertext, it will immediately disclose

[1] We note that this is also an approach towards weakly optimal sReg-FE with weakly selective security, only assuming the existence of sRBE.

[2] It is also feasible to employ a standard PRF from the standard DDH assumption [NR97].

the PRF key to the adversary, so that all user tags will be publicly known in the post-challenge phase. To resolve this issue, we introduce the *rerandomization* technique.

Our idea is to *rerandomize* the pseudorandom tag in the policy part. More specifically, the policy $\mathsf{P}_{\mathsf{tag}}$ in Eq. (1) can be modified into the following form:

$$\mathsf{P}_{\mathsf{tag}}(\underbrace{a_i, \widehat{\mathsf{tag}}_i}_{\text{private}}) = 1 \iff \underbrace{\boxed{\mathsf{wPRF.Eval}(k, a_i)}}_{\text{public}} \oplus \widehat{\mathsf{tag}}_i = \mathsf{tag}. \tag{2}$$

Here, we set $\mathsf{att}_i = (a_i, \widehat{\mathsf{tag}}_i)$ as the whole attribute part. The key point is that $\widehat{\mathsf{tag}}_i \leftarrow [Q]$ is locally and uniformly sampled by user i, and att_i should be privately registered[3], i.e., the adversary has no idea about $\widehat{\mathsf{tag}}_i$ until it colludes with user i. Therefore, we can exploit the pseudorandomness from weak PRF and the randomness from $\widehat{\mathsf{tag}}_i$ to claim the pre-challenge and post-challenge security, respectively.

In this construction, we need to hide $\widehat{\mathsf{tag}}_i$ from the adversary. However, this cannot be satisfied by standard sReg-ABE, as att_i is disclosed in the clear for aggregation. We overcome this barrier by formalizing a new variant of sReg-ABE, namely *equivocal sReg-ABE*.

Equivocal sReg-ABE. This primitive is the same as standard sReg-ABE, except that (1) it does not publicly register the attribute att_i but allows user i to embed att_i into pk_i, then only register pk_i; (2) it additionally requires a new property called *equivocal* public key. Similar to our solution to **Challenge 2**, the adversary cannot learn any information about att_i from a public key pk_i that is generated in the equivocal mode, while it can still perfectly simulate the corrupted secret key sk_i in the reduction proof. Therefore, we define this property as follows:

- *Equivocal public key.* It has two key generation modes that are indistinguishable from each other: in the normal mode, it computes $(\mathsf{pk}_i, \mathsf{sk}_i)$ from att_i; in the equivocal mode, it simulates a public key pk_i without involving att_i, then use some trapdoor information to equivocate pk_i to any $(\mathsf{att}_i, \mathsf{sk}_i)$.

How to achieve equivocality? We observe that the ingenious structure of existing sReg-ABE schemes provides a feasible path to equivocal sReg-ABE. Concretely, we observe that many schemes enjoy a *decomposable* aggregation: the aggregator can first generate L pieces of aggregation shares $\{\mathsf{ash}_i\}_{i\in[L]}$, each of which can be independently and deterministically computed from $(\mathsf{pk}_i, \mathsf{att}_i)$; then it combines all $\{\mathsf{ash}_i\}_{i\in[L]}$ to derive $(\mathsf{mpk}, \mathsf{hsk}_j)$. At first glance, this feature is unrelated to the functionality described in Eq. (2), while we can now change the registration of all attribute-related parts. In particular, to hide att_i at the aggregation stage, we can transfer the task of computing ash_i from the aggregator to user i. Then, the aggregator just needs to collect and combine all ash_i from users, without direct access to att_i. The next task is to generate ash_i that can equivocate att_i.

This can be done by deploying a well-structured equivocal commitment scheme. Concretely, user i can normally generate a share ash_i and then adapt it into an equivocal commitment to att_i. We require that the derived commitment must be compatible with the aggregation, encryption, and decryption algorithms of the original sReg-ABE. Such a sReg-ABE scheme can be found in [ZZGQ23], which is proved under the standard k-Lin assumption in the standard model, and we can apply a classic Pedersen-style equivocal commitment[4]. Then, we manage to achieve the desired functionality in Eq. (2), and the equivocality naturally comes from the underlying equivocal commitment. This leads to a weakly optimal sReg-FE with adaptive SIM security, which further implies a dynamic sReg-FE with adaptive SIM security. Finally, we obtain a dynamic bounded collusion Reg-FE with adaptive security under k-Lin assumption in the standard model.

[3] In fact, only $\widehat{\mathsf{tag}}_i$ should be private. For the convenience of expression, we require the whole attribute part to be privately registered.

[4] Other pairing-based sReg-ABE schemes [HLWW23,GLWW24,AT24,DPY24] are also compatible with this equivocal commitment, which may lead to more equivocal sReg-ABE schemes.

3 Preliminaries

For a finite set S, we write $s \leftarrow S$ to denote that s is picked uniformly from finite set S. Then, we use $|S|$ to denote the size of S. For integers ℓ and n such that $\ell < n$, $[n]$ and $[\ell, n]$ denote sets $\{1, \ldots, n\}$ and $\{\ell, \ldots, n\}$, respectively. Let $\approx_s$ denote two distributions being statistically indistinguishable, and $\approx_c$ denote two distributions being computationally indistinguishable. For any string $x \in \{0,1\}^n$, we use $x[w]$ to denote the w-th bit of x.

3.1 Hardness Assumptions

For matrices $\mathbf{A} := (a_{i,j}) \in \mathbb{Z}^{n \times m}$ and $\mathbf{B} \in \mathbb{Z}^{k \times \ell}$, the tensor (Kronecker) product is defined as

$$\mathbf{A} \otimes \mathbf{B} = \begin{pmatrix} a_{1,1}\mathbf{B} & \cdots & a_{1,m}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{n,1}\mathbf{B} & \cdots & a_{n,m}\mathbf{B} \end{pmatrix} \in \mathbb{Z}^{nk \times m\ell}.$$

For matrices $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}$ with compatible dimensions, we have $(\mathbf{A} \otimes \mathbf{B})(\mathbf{C} \otimes \mathbf{D}) = (\mathbf{AC}) \otimes (\mathbf{BD})$.

Pairing Setting. A generator $\mathcal{G}$ takes as input a security parameter 1^λ and outputs a group description $\mathbb{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$, where p is a prime, $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ are cyclic groups of order p, and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a non-degenerate bilinear map. Group operations in $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ and bilinear map e are computable in deterministic polynomial time in λ. Let $g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2$ and $g_T = e(g_1, g_2) \in \mathbb{G}_T$ be the respective generators, we employ *implicit representation* of group elements: for a matrix $\mathbf{M}$ over $\mathbb{Z}_p$, we define $[\mathbf{M}]_s = g_s^{\mathbf{M}}, \forall s \in \{1, 2, T\}$, where exponentiation is carried out component-wise. Given $[\mathbf{A}]_1, [\mathbf{B}]_2$ where $\mathbf{A}$ and $\mathbf{B}$ have proper sizes, we let $e([\mathbf{A}]_1, [\mathbf{B}]_2) = [\mathbf{AB}]_T$. Next, we review *Matrix Decisional Diffie-Hellman (MDDH) assumption*, which is implied by the k-Lin assumption [EHK+13].

Assumption 1 ((k, ℓ, d)-MDDH over $\mathbb{G}_s$, $s \in \{1, 2\}$ [EHK+13]) *Let $k, \ell, d \in \mathbb{N}$ with $k < \ell$. We say that the (k, ℓ, d)-MDDH assumption holds in $\mathbb{G}_s$ if for all efficient adversaries $\mathcal{A}$, the following advantage function is negligible in λ:*

$$\mathsf{Adv}^{MDDH}_{\mathcal{A},s,k,\ell,d}(\lambda) = \left| \Pr[\mathcal{A}(\mathbb{G}, [\mathbf{M}]_s, [\mathbf{SM}]_s) = 1] - \Pr[\mathcal{A}(\mathbb{G}, [\mathbf{M}]_s, [\mathbf{U}]_s) = 1] \right|$$

where $\mathbb{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$, $\mathbf{M} \leftarrow \mathbb{Z}_p^{k \times \ell}$, $\mathbf{S} \leftarrow \mathbb{Z}_p^{d \times k}$ and $\mathbf{U} \leftarrow \mathbb{Z}_p^{d \times \ell}$.

Lattice Setting. We use $\mathcal{D}_{\mathbb{Z},\sigma}$ to denote the discrete Gaussian distribution over $\mathbb{Z}$ with standard deviation $\sigma > 0$. Let $q \in \mathbb{N}$ be an integer modulus, for a matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ and a target vector $\mathbf{y} \in \mathbb{Z}_q^n$ in the image space of $\mathbf{A}$, we write $\mathbf{A}_\sigma^{-1}(\mathbf{y})$ to denote the random variable $\mathbf{x} \leftarrow \mathcal{D}_{\mathbb{Z},\sigma}^m$ conditioned on $\mathbf{Ax} = \mathbf{y}$. For any $n, m, q \in \mathbb{N}$ such that $m \geq n\lceil \log q \rceil$, let $\mathbf{G} = \mathbf{I}_n \otimes (1, 2, \ldots, 2^{\lceil \log q \rceil - 1}, \mathbf{0}_{m'}) \in \mathbb{Z}_q^{n \times m}$ be the gadget matrix [MP12] where $\mathbf{I}_n$ is the identity matrix of dimension n, and $\mathbf{0}'_m$ is a row zero vector such that $m = n\lceil \log q \rceil + nm'$.

Assumption 2 (ℓ-succinct LWE [Wee24]) *Given the security parameter λ and LWE parameters $n, m, q, \chi \in \mathbb{N}$, we say the ℓ-succinct LWE assumption with Gaussian width parameter σ holds if*

$$(\mathbf{B}, \mathbf{sB} + \mathbf{e}, \mathbf{W}, \mathbf{T}) \approx_c (\mathbf{B}, \mathbf{c}, \mathbf{W}, \mathbf{T})$$

where $\mathbf{B} \leftarrow \mathbb{Z}_q^{n \times m}, \mathbf{s} \leftarrow \mathbb{Z}_q^{1 \times n}, \mathbf{e} \leftarrow \mathcal{D}_{\mathbb{Z},\chi}^{1 \times m}, \mathbf{W} \leftarrow \mathbb{Z}_q^{\ell n \times m}, \mathbf{T} \leftarrow [\mathbf{I}_\ell \otimes \mathbf{B} \mid \mathbf{W}]_\sigma^{-1}(\mathbf{I}_\ell \otimes \mathbf{G})$, and $\mathbf{c} \leftarrow \mathbb{Z}_q^{1 \times m}$.

3.2 Collision-Resistant Hash Function

Let $H : \{0,1\}^* \to Z$ be a collision-resistant hash function family, then we define the following algorithms:

- $\mathsf{Gen}(1^\lambda) \to \mathsf{hk}$. It takes as input the security parameter 1^λ, then outputs a hash key hk[5].
- $\mathsf{H}(\mathsf{hk}, \mathsf{m}) \to h$. It takes as input the hash key hk and a message $\mathsf{m} \in \{0,1\}^*$, then deterministically outputs a hash value $h \in Z$.

The collision resistance is defined as follows:

- **Collision Resistance.** For all efficient adversaries $\mathcal{A}$, the probability

$$\Pr\left[\mathsf{m} \neq \mathsf{m}' \ \wedge \ \mathsf{H}(\mathsf{hk},\mathsf{m}) = \mathsf{H}(\mathsf{hk},\mathsf{m}') \,\middle|\, \begin{array}{l} \mathsf{hk} \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{m},\mathsf{m}') \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right]$$

  is negligible in λ.

3.3 Chameleon Hash Function

Let $H : \{0,1\}^* \to Z$ be a chameleon hash function (CHF for short) family, then we define the following algorithms:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{hk}, \mathsf{td})$. It takes as input the security parameter 1^λ, then outputs a hash key hk and trapdoor information td.
- $\mathsf{H}(\mathsf{hk}, \mathsf{m}; r) \to h$. It takes as input hk, a message $\mathsf{m} \in \{0,1\}^*$, then outputs a hash value $h \in Z$. Here, r is the randomness used for this algorithm.
- $\mathsf{H}^{-1}(\mathsf{hk}, \mathsf{td}, (\mathsf{m}, r), \widetilde{\mathsf{m}}) \to \widetilde{r}$. It takes as input the trapdoor td, a pair of message and randomness (m, r), and a message $\widetilde{\mathsf{m}}$, then output a randomness $\widetilde{r}$.

The above scheme is required to satisfy:

- **Collision Resistance.** For all efficient adversaries $\mathcal{A}$, the probability

$$\Pr\left[(\mathsf{m}, r) \neq (\mathsf{m}', r') \ \wedge \ \mathsf{H}(\mathsf{hk},\mathsf{m};r) = \mathsf{H}(\mathsf{hk},\mathsf{m}';r') \,\middle|\, \begin{array}{l} (\mathsf{hk},\mathsf{td}) \leftarrow \mathsf{Gen}(1^\lambda) \\ (\mathsf{m}, r, \mathsf{m}', r') \leftarrow \mathcal{A}(\mathsf{hk}) \end{array}\right]$$

  is negligible in λ.
- **Chameleon Property.** For all $\lambda \in \mathbb{N}$, all $(\mathsf{hk}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$, all messages $\mathsf{m}, \widetilde{\mathsf{m}} \in \{0,1\}^*$, and all randomness r, we have

$$\mathsf{H}(\mathsf{hk}, \mathsf{m}; r) = \mathsf{H}(\mathsf{hk}, \widetilde{\mathsf{m}}; \widetilde{r})$$

  where $\widetilde{r} \leftarrow \mathsf{H}^{-1}(\mathsf{hk}, \mathsf{td}, (\mathsf{m}, r), \widetilde{\mathsf{m}})$.

3.4 Garbled Circuits

A garbled circuit scheme for the circuit family C over input space $\{0,1\}^n$ [Yao86,BHR12] works as follows:

- $\mathsf{Garble}(1^\lambda, \mathsf{C}) \to (\tilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}})$. It takes as input security parameter 1^λ and a circuit $\mathsf{C} \in C$, then outputs a garbled circuit $\tilde{\mathsf{C}}$ and labels $\{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}$. Without loss of generality, we assume $\mathsf{lab}_{w,b} \in \{0,1\}^\lambda$.

[5] In this work, we adopt the notion of keyed hash functions. Here, one can view hk as an index to specify a particular hash function $\mathsf{H}_{\mathsf{hk}} \in H$.

– $\mathsf{Eval}(\tilde{\mathsf{C}}, \{\mathsf{lab}_{w,x[w]}\}_{w\in[n]}) \to z$. It takes as input a garbled circuit $\tilde{\mathsf{C}}$ and a sequence of input labels $\{\mathsf{lab}_{w,x[w]}\}_{w\in[n]}$, then outputs a value z.

The above scheme is required to satisfy:

– **Correctness.** For all $\lambda \in \mathbb{N}$, all circuits $\mathsf{C} \in \mathcal{C}$, and all inputs $x \in \{0,1\}^n$,

$$\Pr[\mathsf{Eval}(\tilde{\mathsf{C}}, \{\mathsf{lab}_{w,x[w]}\}_{w\in[n]}) = \mathsf{C}(x)] = 1$$

where $(\tilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{C})$.

– **Security.** For all circuits C and all inputs $x \in \{0,1\}^n$, there exists a simulator $\widetilde{\mathsf{Garble}}$ such that

$$\left(\tilde{\mathsf{C}}, \{\mathsf{lab}_{w,x[w]}\}_{w\in[n]}\right) \approx_c \widetilde{\mathsf{Garble}}(1^\lambda, \mathsf{C}(x))$$

where $(\tilde{\mathsf{C}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}) \leftarrow \mathsf{Garble}(1^\lambda, \mathsf{C})$.

3.5 Non-Interactive Zero Knowledge Argument

A non-interactive zero knowledge argument (NIZK for short) for a NP language $\mathcal{L}$ associated with witness relation $\mathcal{R} : S \times W \to \{0,1\}$ works as follows:

– $\mathsf{Setup}(1^\lambda) \to (\mathsf{crs}, \mathsf{td})$. It takes as input the security parameter 1^λ, then outputs a common random string crs and trapdoor information td.
– $\mathsf{Prv}(\mathsf{crs}, s, w) \to \pi$. It takes as input the common random string crs, a statement $s \in S$, and a witness $w \in W$, then outputs a proof π.
– $\mathsf{Ver}(\mathsf{crs}, s, \pi) \to 0/1$. It takes as input the common random string crs, a statement $s \in S$, and a proof π, then outputs a bit indicating whether accept π.
– $\mathsf{SPrv}(\mathsf{crs}, \mathsf{td}, s) \to \widetilde{\pi}$. It takes as input the common random string crs, a trapdoor td, and a statement $s \in S$, then outputs a simulated proof $\widetilde{\pi}$.

The above scheme is required to satisfy:

– **Completeness.** For all λ and all $(s, w) \in S \times W$ such that $\mathcal{R}(s, w) = 1$ we have

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, s, \pi) = 1 \mid \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda);\ \pi \leftarrow \mathsf{Prv}(\mathsf{crs}, s, w)\right] = 1.$$

– **Zero Knowledge.** For all λ, all $(\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda)$ and all $(s, w) \in S \times W$ such that $\mathcal{R}(s, w) = 1$, we have

$$\left| \Pr\left[b = b' \wedge \mathcal{R}(s^*, w^*) = 1 \;\middle|\; \begin{array}{l} (\mathsf{crs}, \mathsf{td}) \leftarrow \mathsf{Setup}(1^\lambda) \\ (s^*, w^*) \leftarrow \mathcal{A}(\mathsf{crs}) \\ b \leftarrow \{0,1\}; \pi_0 \leftarrow \mathsf{Prv}(\mathsf{crs}, s^*, w^*) \\ \pi_1 \leftarrow \mathsf{SPrv}(\mathsf{crs}, \mathsf{td}, s^*) \\ b' \leftarrow \mathcal{A}(\pi_b) \end{array} \right] - \frac{1}{2} \right|$$

is negligible in λ.

– **Simulation-Sound Extractability.** For all λ and all efficient adversaries $\mathcal{A}$, there exists an extractor $(\mathsf{ExtSetup}, \mathsf{Ext})$ such that the following probability

$$\Pr\left[\begin{array}{c} (s^*, \pi^*) \notin Q \ \wedge\ \mathsf{Ver}(\mathsf{crs}, s^*, \pi^*) = 1 \wedge \\ \mathcal{R}(s^*, w^*) = 0 \end{array} \;\middle|\; \begin{array}{l} (\mathsf{crs}, \mathsf{td}, \mathsf{ex}) \leftarrow \mathsf{ExtSetup}(1^\lambda) \\ (s^*, \pi^*) \leftarrow \mathcal{A}^{\mathsf{SPrv}(\mathsf{crs},\mathsf{td},\cdot)}(\mathsf{crs}) \\ w^* \leftarrow \mathsf{Ext}(\mathsf{crs}, \mathsf{ex}, s^*, \pi^*) \end{array} \right]$$

is negligible in λ, where Q records all queries to $\mathsf{SPrv}(\mathsf{crs}, \mathsf{td}, \cdot)$ along with responses. Besides, $(\mathsf{crs}, \mathsf{td})$ output by ExtSetup are identical to those from Setup.

3.6 Weak Pseudorandom Function

Let $F : K \times X \to Y$ be a weak pseudorandom function (PRF for short) family, then we define

- $\mathsf{Gen}(1^\lambda) \to \mathsf{k}$. It takes as input the security parameter 1^λ, then outputs a key $\mathsf{k} \in K$.
- $\mathsf{Eval}(\mathsf{k}, x) \to y$. It takes as input a key k and an input value $x \in X$, then outputs a value $y \in Y$.

We define the pseudorandomness as follows:

- **Pseudorandomness.** For all $\lambda \in N$ and all efficient adversaries $\mathcal{A}$, it requires that

$$\left|\Pr\left[b = 1 | b \leftarrow \mathcal{A}(\{x_i, \mathsf{Eval}(k, x_i)\}_{i\in[L]})\right] - \Pr\left[b = 1 |\, b \leftarrow \mathcal{A}(\{x_i, R(x_i)\})\right]\right|$$

  is negligible in λ, where $x_1, \ldots, x_L \leftarrow X$, $\mathsf{k} \leftarrow \mathsf{Gen}(1^\lambda)$, and $R(\cdot)$ is a random function.

Theorem 4 (Weak PRF computed in NC^1 [NR97,BPR12,BP14,DS15,Zha25]). *Under the standard k-Lin or the LWE assumption [Reg05], there exist weak PRFs in which the algorithm* Eval *can be computed in the class of NC^1 circuits.*

3.7 Reusable Dynamic Multi-Party Computation

Assuming parameters $N = N(\lambda, Q)$, $S = S(\lambda, Q)$ $t = t(\lambda, Q)$, $m = m(\lambda, Q)$, and $v = v(\lambda, Q)$, a reusable dynamic multi-party computation protocol (reusable dynamic MPC for short) for circuit family $C : X \to Z$ works as follows:

- $\mathsf{CktEnc}(1^\lambda, 1^Q, \mathsf{C}, \Delta) \to (\widehat{\mathsf{C}}_1, \ldots, \widehat{\mathsf{C}}_N)$. It takes as input the security parameter 1^λ, the number of sessions Q, a circuit $\mathsf{C} \in C$, and a set $\Delta \subseteq [S]$ of size v, then outputs an encoding $\widehat{\mathsf{C}} = (\widehat{\mathsf{C}}_1, \ldots, \widehat{\mathsf{C}}_N)$. This algorithm is deterministic.
- $\mathsf{InpEnc}(1^\lambda, 1^Q, x) \to (\widehat{x}_1, \ldots, \widehat{x}_N)$. It takes as input the security parameter 1^λ, the number of sessions Q and an input $x \in X$, then outputs an encoding $(\widehat{x}_1, \ldots, \widehat{x}_N)$.
- $\mathsf{Local}(\widehat{\mathsf{C}}_k, \widehat{x}_k) \to \widehat{y}_k$. It takes as input the k-th encoding $\widehat{\mathsf{C}}_k$ of C, and k-th encoding $\widehat{x}_k$ of x, then outputs $\widehat{y}_k$. This algorithm is deterministic.
- $\mathsf{Decode}(\{\widehat{y}_k\}_{k\in\Gamma}, \Gamma) \to z \backslash \perp$. It takes as input a set of encodings $\{\widehat{y}_k\}_{k\in S}$ and a set $\Gamma \subseteq [N]$, then outputs $z \in Z$ or an empty symbol $\perp$.

A reusable dynamic MPC protocol is required to satisfy:

- **Correctness.** For all $\lambda, Q \in \mathbb{N}$, all circuits $\mathsf{C} \in C$, all sets $\Delta \subseteq [S]$ of size v, all inputs $x \in X$, and all sets $\Gamma \subseteq [N]$ of size m, we have

$$\Pr\left[\mathsf{Decode}(\{\mathsf{Local}(\widehat{\mathsf{C}}_k, \widehat{x}_k)\}_{k\in\Gamma}, \Gamma) = \mathsf{C}(x)\right] = 1.$$

  where $(\widehat{\mathsf{C}}_1, \ldots, \widehat{\mathsf{C}}_N) \leftarrow \mathsf{CktEnc}(1^\lambda, 1^Q, \mathsf{C}, \Delta)$, $(\widehat{x}_1, \ldots, \widehat{x}_N) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, x)$.
- **Security.** For all efficient adversaries $\mathcal{A}$, there exists a simulator $(\widetilde{\mathsf{Local}}, \widetilde{\mathsf{InpEnc}})$ such that it is computationally hard to distinguish the following two worlds:
  - *Setup phase:* Initially, $\mathcal{A}$ specifies the number of sessions Q, the total number of parties N, the number of involved parties m, and a threshold t. Besides, $\mathcal{A}$ specifies a corruption set $\mathcal{T} \subseteq [N]$, a series of sets $\Gamma_1, \ldots, \Gamma_Q \subseteq [N]$ of size n, circuits $\mathsf{C}_1, \ldots, \mathsf{C}_q \in C$ and sets $\Delta_1, \ldots, \Delta_q \subseteq [S]$ of size v, where $q \le Q$.
    For each $i \in [q]$, it computes the circuit encodings

$$(\widehat{\mathsf{C}}_{i,1}, \ldots, \widehat{\mathsf{C}}_{i,N}) = \mathsf{CktEnc}(1^\lambda, 1^Q, \mathsf{C}_i, \Delta_i)$$

    where it requires $\Delta_i \setminus (\bigcup_{j\in[q]\setminus\{i\}} \Delta_j) \neq \emptyset$ (i.e., cover freeness).

- *Challenge phase:* In this phase, $\mathcal{A}$ specifies a challenge message x^*. The challenger checks whether $|\mathcal{T}| \leq t$. If not, abort and output 0; otherwise,
  * In the real world, it computes

$$(\widehat{x}_1^*, \ldots, \widehat{x}_N^*) \leftarrow \mathsf{InpEnc}(1^\lambda, 1^Q, x^*),$$
$$\forall i \in [q] \text{ and } k \in \Gamma_i,\ \widehat{y}_{i,k} \leftarrow \mathsf{Local}(\widehat{\mathsf{C}}_{i,k}, \widehat{x}_k^*).$$

  * In the ideal world, it simulates

$$(\{\widehat{x}_k^*\}_{k\in\mathcal{T}}, \{\widehat{y}_{i,k}\}_{i\in[q],k\in\Gamma_i}) \leftarrow \widetilde{\mathsf{InpEnc}}(1^{|x^*|}, \mathcal{T}, \{(\mathsf{C}_i, \mathsf{C}_i(x^*))\}_{i\in[q]}).$$

    where it holds that $\widehat{y}_{i,k} = \mathsf{Local}(\widehat{\mathsf{C}}_{i,k}, \widehat{x}_{i,k}^*)$ for each $k \in \Gamma_i \cap \mathcal{T}$.

  Return $(\{\widehat{x}_i^*\}_{i\in\mathcal{T}}, \{\widehat{y}_{i,k}\}_{i\in[q],k\in\Gamma_i})$ to $\mathcal{A}$.
- *Post-challenge phase:* When $\mathcal{A}$ makes i-th query where $i = q+1, \ldots, Q$,
  * $\mathsf{OCor}(\mathsf{C}_i, \Gamma_i, \Delta_i)$: It requires that $\Gamma_i \subseteq [N]$ and $\Delta_i \subseteq [S]$ such that $|\Gamma_i| = m$ and $|\Delta| = v$. The challenger computes $(\widehat{\mathsf{C}}_{i,1}, \ldots, \widehat{\mathsf{C}}_{i,N}) = \mathsf{CktEnc}(1^\lambda, 1^Q, \mathsf{C}_i, \Delta_i)$. Then, in the real world, it computes $\widehat{y}_{i,k} \leftarrow \mathsf{Local}(\widehat{\mathsf{C}}_{i,k}, \widehat{x}_{i,k}^*)$ for all $k \in \Gamma_i$; in the ideal world, it simulates

$$\{\widehat{y}_{i,k}\}_{k\in\Gamma_i} \leftarrow \widetilde{\mathsf{Local}}\,(i, \mathsf{C}_i, \mathsf{C}_i(x^*))\,.$$

    Return $\widehat{\mathsf{C}}_i = (\widehat{\mathsf{C}}_{i,1}, \ldots, \widehat{\mathsf{C}}_{i,N})$ and $\{\widehat{y}_{i,k}\}_{k\in\Gamma_i}$ to $\mathcal{A}$.

  For all $i \in [Q]$, it requires $\Delta_i \setminus (\bigcup_{j\in[Q]\setminus\{i\}} \Delta_j) \neq \emptyset$ (i.e., cover freeness).
- *Output phase:* $\mathcal{A}$ outputs a bit to indicate the real/ideal world.

Remark. Our definition stems from the work of Ananth and Vaikuntanathan [AV19]. Despite a few differences, the reusable dynamic MPC scheme for general circuits proposed in [AV19] can be slightly adapted to satisfy our definition:

- For syntax, our algorithm CktEnc additionally takes a set Δ and is required to be deterministic. Recall that the original algorithm CktEnc actually samples a random set Δ and outputs $\widehat{\mathsf{C}}_k = (\mathsf{C}, \Delta)$ for $k \in [N]$. Therefore, if Δ is provided from outside, our definition would make no difference.
- For security, the original definition requires simulating all queried circuit encodings $\{\widehat{\mathsf{C}}_i\}_{i\in[Q]}$ in the ideal world, while ours does not. This is because our algorithm CktEnc is deterministic. We observe that in the original scheme, the encodings generated in the ideal world are identical to those in the real world, except that the sets $\{\Delta_i\}_{i\in[Q]}$ are subject to cover freeness. Therefore, our definition additionally requires that all sets $\{\Delta_i\}_{i\in[Q]}$ provided by $\mathcal{A}$ are subject to cover freeness.

Theorem 5 (Reusable Dynamic MPC for P/Poly [AV19]). *Let $N > \Theta(Q^2\lambda)$, $t < \lfloor m/3 \rfloor$ and $m < N$. Assuming the existence of pseudorandom generators, there exists a secure reusable dynamic MPC scheme for general circuits.*

4 Equivocal Slotted Reg-ABE

In this section, we formalize a novel variant of *slotted registered attribute-based encryption* (sReg-ABE), namely equivocal sReg-ABE, as the basic building block of 1-bound equivocal slotted Reg-ABFE (in Section 5.2). Then we provide a concrete instantiation of equivocal sReg-ABE for NC^1 policy from the k-Lin assumption.

4.1 Definition

An equivocal slotted Reg-ABE for policy family $P : A \to \{0,1\}$ is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, P) \to \mathsf{crs}$. It takes as input the security parameter 1^λ, the upper bound 1^L on the number of user slots, and the description of policy family P, then outputs a common reference string crs.
- $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. It takes as input the common reference string crs, a slot index $i \in [L]$, and an attribute $\mathsf{att}_i \in A$, then outputs a public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
- $\mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i\}_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$. It takes as input the common reference string crs and a series of public keys $\{\mathsf{pk}_i\}_{i\in[L]}$, then outputs master public key mpk and helper keys $\{\mathsf{hsk}_j\}_{j\in[L]}$. This algorithm is deterministic.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, \mathsf{m}) \to \mathsf{ct}$. It takes as input the master public key mpk, a policy $\mathsf{P} \in P$, and a message $\mathsf{m} \in M$, then outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct}) \to \mathsf{m}/\perp$. It takes as input a helper key hsk_{i^*}, a secret key sk_{i^*}, an attribute $\mathsf{att}_{i^*} \in A$, and a ciphertext ct, then outputs a message $\mathsf{m} \in M$ or an empty symbol $\perp$.

We require the following properties:

- **Compactness.** For all $\lambda, L \in \mathbb{N}$, all P, and all $i \in [L]$, it requires

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, P, \log L) \text{ and } |\mathsf{hsk}_j| = \mathsf{poly}(\lambda, P, \log L).$$

- **Correctness.** For all $\lambda, L \in \mathbb{N}$, all P, all $i^* \in [L]$ and all $\mathsf{att}_{i^*} \in A$, all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, P)$, all $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i^*, \mathsf{att}_{i^*})$, all $\{\mathsf{pk}_i\}_{i\in[L]\setminus\{i^*\}}$ such that $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$, all $\mathsf{P} \in P$ such that $\mathsf{P}(\mathsf{att}_{i^*}) = 1$ and all $\mathsf{m} \in M$, and computing $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i\}_{i\in[L]})$, we have

$$\Pr\left[\mathsf{Dec}\left(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, \mathsf{m})\right) = \mathsf{m}\right] = 1.$$

- **Equivocal Public Key.** We define the following algorithms:
  - $\mathsf{ESetup}(1^\lambda, 1^L, P) \to (\mathsf{crs}, \mathsf{ek})$. Same as Setup, except that it additionally outputs an equivocal key ek. We require that the output crs is indistinguishable from that produced by Setup.
  - $\mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek}) \to (\mathsf{pk}_i, \mathsf{st}_i)$. Same as Gen, except that it does not involve att_i and outputs a public key pk_i and internal state st_i.
  - $\mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek}) \to \mathsf{sk}_i$. It takes as input an internal state st_i, an attribute att_i, and an equivocal key ek, then outputs a secret key sk_i.

  This property requires $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i) \approx_c \{\mathsf{pk}_i, \mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek})\}$, where $(\mathsf{pk}_i, \mathsf{st}_i) \leftarrow \mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek})$.
- **Policy-Selective IND Security.** For all efficient adversaries $\mathcal{A}$, we define:
  - *Setup phase:* Initially, $\mathcal{A}$ specifies $(1^L, P)$ and a challenge policy $\mathsf{P}^* \in P$. The challenger runs $(\mathsf{crs}, \mathsf{ek}) \leftarrow \mathsf{ESetup}(1^\lambda, 1^L, P)$ and return crs to $\mathcal{A}$.
  - *Pre-challenge phase:* The challenger initializes a dictionary $\{\mathcal{D}_i = \emptyset\}_{i\in[L]}$ and a set $C = \emptyset$. In this phase, $\mathcal{A}$ can make generation/corruption queries:
    - $\mathsf{OGen}(i)$: The challenger samples $(\mathsf{pk}, \mathsf{st}) \leftarrow \mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek})$. Update $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}$ and return pk to $\mathcal{A}$.
    - $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: Parse $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}$. It computes $\mathsf{sk} \leftarrow \mathsf{Equiv}(\mathsf{st}, \mathsf{att}, \mathsf{ek})$. Return the secret key sk to $\mathcal{A}$ and update $C = C \cup \{(i, \mathsf{pk}, \mathsf{att})\}$.
  - *Challenge phase:* In this phase, $\mathcal{A}$ specifies a pair of challenge messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$, L pairs of public keys and attributes $\{(\mathsf{pk}_i^*, \mathsf{att}_i^*)\}_{i\in[L]}$ together with a list of randomness $\{r_i^*\}$. The challenger runs $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i^*\}_{i\in[L]})$. Then, the challenger checks:

* For each slot $i \in [L]$, it must fall in one of the following three cases:

$$\begin{aligned}
&\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \notin C && \text{(honest slot)}\\
&\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \in C \implies \mathsf{P}^*(\mathsf{att}_i^*) = 0 && \text{(corrupted slot)}\\
&\mathcal{D}_i[\mathsf{pk}_i^*] = \perp \implies \mathsf{P}^*(\mathsf{att}_i^*) = 0 && \text{(malicious slot)}
\end{aligned}$$

* For each malicious slot i, it must hold that $\mathsf{pk}_i^* \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i^*; r_i^*)$, where r_i^* is the randomness used by the algorithm Gen.

It aborts if either of the above two conditions is not satisfied; otherwise, the challenger flips a random coin $b \in \{0,1\}$ and replies $\mathcal{A}$ with a challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}^*, \mathsf{m}_b^*)$.

- *Post-challenge phase:* $\mathcal{A}$ can make corruption queries to the oracle OCor as in the pre-challenge phase. For each corruption query $(i, \mathsf{pk}_i^*, \mathsf{att}_i^*)$, it requires $\mathsf{P}^*(\mathsf{att}_i^*) = 0$. Besides, $\mathcal{A}$ can corrupt any non-registered keys.
- *Output phase:* $\mathcal{A}$ outputs a bit b' as the guess of b.

We say that an equivocal sReg-ABE scheme achieves policy-selective IND security if $|\Pr[b' = b] - \frac{1}{2}|$ is negligible in λ.

Unlike standard registered notions, the above definition is somewhat weak in that it requires the adversary to provide all randomness and attributes for maliciously created public keys. Nevertheless, this is sufficient for constructing our dynamic bounded sReg-FE.

4.2 Scheme

We present a concrete equivocal sReg-ABE scheme for NC^1 policies based on the k-Lin assumption. In particular, we show that some slight modifications to the standard sReg-ABE scheme proposed by Zhu et al. [ZZGQ23] can lead to the desired equivocal sReg-ABE. We first review the notion of predicate encoding as follows.

Predicate Encoding for NC^1. Since NC^1 circuits can be captured by (monotone) span programs, we use the predicate encoding of monotone span programs denoted by P, which is defined by a matrix $\mathbf{P} \in \mathbb{Z}_p^{m \times \ell}$ such that

$$\mathsf{P}(\mathbf{x}) = 1 \iff \mathbf{x} \in \{0,1\}^{1\times m} \text{ satisfies } \mathbf{P} \iff \exists \boldsymbol{\omega} \in \mathbb{Z}_p^{1\times m} \text{ s.t. } \mathbf{e}_1 = \boldsymbol{\omega} \cdot \mathsf{diag}(\mathbf{x})\mathbf{P}$$

Here we use notation: $\mathsf{diag}(\mathbf{x}) := \begin{pmatrix} x_1 & & \\ & \ddots & \\ & & x_m \end{pmatrix} \in \mathbb{Z}_p^{m\times m}$ for any attribute vector $\mathbf{x} = (x_1, \ldots, x_m)$. Note that $\mathbf{e}_1^\top = (1, 0, \ldots, 0)$ and $\mathsf{diag}(\mathbf{x}) = \mathsf{diag}(\mathbf{x})^\top$.

Recall the predicate encoding for span program predicate (ciphertext-policy variant) in [CGW15]:

$$\begin{aligned}
&\mathbf{C}_\mathbf{P} = \begin{pmatrix} \mathbf{I}_m \\ \mathbf{P}^\top \end{pmatrix}, \quad \mathbf{K}_\mathbf{x} = \begin{pmatrix} \mathbf{0}_m & \mathsf{diag}(\mathbf{x}) \\ \mathbf{e}_1^\top & \mathbf{0}_{\ell\times m}^\top \end{pmatrix}, \quad \mathbf{a} = (1 \| \mathbf{0}_m),\\
&\mathbf{d}_{\mathbf{P},\mathbf{x}} = (1 \| \boldsymbol{\omega} \| - \mathsf{diag}\,(\mathbf{x}) \cdot \boldsymbol{\omega}^\top)
\end{aligned} \tag{3}$$

where $\mathbf{I}_m$ is an identity matrix of dimension m and $\mathbf{0}_m$ is a *row* zero vector of size m. It defines the matrix $\mathbf{M}_{\mathbf{P},\mathbf{x}} = \begin{pmatrix} \mathbf{a} & \mathbf{0}_m \\ \mathbf{K}_\mathbf{x} & \mathbf{C}_\mathbf{P} \end{pmatrix}$. The predicate encoding requires:

– **Correctness.** For any attributes $\mathbf{x}$ and policies P such that $\mathsf{P}(x) = 1$, it must hold that $\mathbf{M}_{\mathbf{P},\mathbf{x}}\mathbf{d}_{\mathbf{P},\mathbf{y}}^\top = \mathbf{e}_1^\top$.

– α-**Privacy.** For any attributes $\mathbf{x}$ and policies P such that $\mathsf{P}(x) = 1$, and for all $\alpha \in \mathbb{Z}_p$, we have

$$\{\mathbf{P}, \mathbf{x}, \alpha, (\alpha \| \mathbf{w})\mathbf{M}_{\mathbf{P},\mathbf{x}}\} \approx_s \{\mathbf{P}, \mathbf{x}, \alpha, (0 \| \mathbf{w})\mathbf{M}_{\mathbf{P},\mathbf{x}}\}.$$

where $\mathbf{w} \leftarrow \mathbb{Z}_p^n$.

Then our equivocal sReg-ABE for NC^1 policies works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, P)$: Compute the parameters (m, ℓ) from P, where m is the length of the attribute vector and let $n = m + \ell$. Generate bilinear groups $\mathbb{G} := (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e) \leftarrow \mathcal{G}(1^\lambda)$ and sample

$$\mathbf{A} \leftarrow \mathbb{Z}_p^{k\times(k+1)},\ \mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k},\ \mathbf{k} \leftarrow \mathbb{Z}_p^{1\times(k+1)},$$
$$\forall i \in [L],\ \mathbf{V}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)},\ \mathbf{W}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)n},\ \mathbf{r}_i \leftarrow \mathbb{Z}_p^{1\times k}.$$

Output the common reference string

$$\mathsf{crs} = \begin{pmatrix} [\mathbf{A}]_1, [\mathbf{A}\mathbf{k}^\top]_T, \{[\mathbf{A}\mathbf{V}_i, \mathbf{A}\mathbf{W}_i]_1\}_{i\in[L]}, \\ \left\{[\mathbf{B}\mathbf{r}_i^\top, \mathbf{V}_i\mathbf{B}\mathbf{r}_i^\top + \mathbf{k}^\top]_2\right\}_{i\in[L]}, \\ \left\{[\mathbf{V}_i\mathbf{B}\mathbf{r}_j^\top, \mathbf{W}_i(\mathbf{I}_m \otimes \mathbf{B}\mathbf{r}_j^\top)]_2\right\}_{i\in[L], j\in[L]\setminus\{i\}} \end{pmatrix}.$$

– $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$: Parse $\mathsf{att}_i := \mathbf{x}_i \in \{0,1\}^{1\times m}$. Sample

$$\mathbf{U}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)},\ \mathbf{T}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)\cdot 2m}.$$

Define the matrix $\mathbf{K}_{\mathbf{x}_i}$ as in Eq. (3) and output the secret key $\mathsf{sk}_i = (\mathbf{U}_i, \mathbf{T}_i)$ and the public key

$$\mathsf{pk}_i = \begin{pmatrix} [\underbrace{\mathbf{A}\mathbf{U}_i}_{\mathbf{F}_i}, \underbrace{\mathbf{A}\mathbf{W}_i(\mathbf{K}_{\mathbf{x}_i} \otimes \mathbf{I}_{k+1}) + \mathbf{A}\mathbf{T}_i}_{\mathbf{H}_i}]_1, \{\underbrace{[\mathbf{U}_i\mathbf{B}\mathbf{r}_j^\top]_2}_{\mathbf{f}_{i,j}^\top}\}_{j\in[L]\setminus\{i\}} \\ \{\underbrace{[\mathbf{W}_i(\mathbf{I}_n \otimes \mathbf{B}\mathbf{r}_j^\top)\mathbf{K}_{\mathbf{x}_i} + \mathbf{T}_i(\mathbf{I}_{2m} \otimes \mathbf{B}\mathbf{r}_j^\top)]_2}_{\mathbf{Q}_{i,j}^\top}\}_{j\in[L]\setminus\{i\}} \end{pmatrix}.$$

– $\mathsf{Agg}(\mathsf{crs}, \{\mathsf{pk}_i\}_{i\in L})$: For each $i \in [L]$, parse $\mathsf{pk}_i = ([\mathbf{F}_i, \mathbf{H}_i]_1, \{[\mathbf{f}_{i,j}^\top, \mathbf{Q}_{i,j}^\top]_2\}_{j\in[L]\setminus\{i\}})$. Define the vector $\mathbf{a}$ as in Eq. (3) Output the master public key

$$\mathsf{mpk} = \left([\mathbf{A}]_1, \left[\sum_{i\in[L]} ((\mathbf{A}\mathbf{V}_i + \mathbf{F}_i)(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{H}_i)\right]_1, \left[\sum_{i\in[L]} \mathbf{A}\mathbf{W}_i\right]_1, [\mathbf{A}\mathbf{k}^\top]_T\right),$$

and for all $j \in [L]$, output the helper key

$$\mathsf{hsk}_j = \left([\underbrace{\mathbf{B}\mathbf{r}_j^\top}_{\mathbf{k}_0^\top}]_2, \underbrace{[\mathbf{V}_j\mathbf{B}\mathbf{r}_j^\top + \mathbf{k}^\top]_2}_{\mathbf{k}_1^\top}, \underbrace{\left[\sum_{i\in[L]\setminus\{j\}} \left((\mathbf{V}_i\mathbf{B}\mathbf{r}_j^\top + \mathbf{f}_{i,j}^\top)\mathbf{a} + \mathbf{Q}_{i,j}^\top\right)\right]_2}_{\mathbf{K}_2^\top}, \underbrace{\left[\sum_{i\in[L]\setminus\{j\}} \mathbf{W}_i(\mathbf{I}_n \otimes \mathbf{B}\mathbf{r}_j^\top)\right]_2}_{\mathbf{K}_3^\top}\right).$$

– $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, \mathsf{m})$: Parse $\mathsf{mpk} = \left([\mathbf{A}]_1, [\mathbf{A}\mathbf{k}^\top]_T, \left[\sum_{j\in[L]}(\mathbf{A}\mathbf{V}_j + \mathbf{A}\mathbf{U}_j)\right]_1\right)$. Compute the matrix $\mathbf{P} \in \mathbb{Z}_p^{m\times\ell}$ from the policy P and then define the matrix $\mathbf{C}_\mathbf{P}$ as in Eq. (3). Sample $\mathbf{s} \leftarrow \mathbb{Z}_p^{1\times k}$ and output the ciphertext

$$\mathsf{ct} = \left([\underbrace{\mathbf{s}\mathbf{A}}_{\mathbf{c}_0}]_1, \underbrace{\left[\sum_{i\in[L]} ((\mathbf{s}\mathbf{A}\mathbf{V}_i + \mathbf{s}\mathbf{F}_i)(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{s}\mathbf{H}_i)\right]_1}_{\mathbf{c}_1}, \underbrace{\left[\sum_{i\in[L]} \mathbf{s}\mathbf{A}\mathbf{W}_i(\mathbf{C}_\mathbf{P} \otimes \mathbf{I}_{k+1})\right]_1}_{\mathbf{c}_2}, \underbrace{[\mathbf{s}\mathbf{A}\mathbf{k}^\top]_T \cdot \mathsf{m}}_{c}\right).$$

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct})$: Parse $\mathsf{hsk}_{i^*} = ([\mathbf{k}_0^\top]_2, [\mathbf{k}_1^\top]_2, [\mathbf{K}_2^\top]_2, [\mathbf{K}_3^\top]_2)$, $\mathsf{sk}_{i^*} = (\mathbf{U}_{i^*}, \mathbf{T}_{i^*})$, $\mathsf{att}_{i^*} = \mathbf{x}_{i^*}$, and $\mathsf{ct} = ([\mathbf{c}_0]_1, [\mathbf{c}_1]_1, [\mathbf{c}_2]_1, C)$. Compute

$$
\begin{aligned}
&[\mathbf{z}_1]_T = e([\mathbf{c}_1 \| \mathbf{c}_2]_1, [(\mathbf{I}_{2m+1} \otimes \mathbf{k}_0^\top)]_2), && [\mathbf{z}_2]_T = e([\mathbf{c}_0]_1, [\mathbf{K}_2^\top \| \mathbf{K}_3^\top \mathbf{C}_{\mathbf{P}}]_2),\\
&[\mathbf{z}_3] = e([\mathbf{c}_0 \mathbf{T}_{i^*} \| \mathbf{0}_{(k+1)m}]_1, [(\mathbf{I}_{2m+1} \otimes \mathbf{k}_0^\top)]_2), && [z_4]_T = e([\mathbf{c}_0 \mathbf{U}_{i^*}]_1, [\mathbf{k}_0^\top]_2),\\
&[z_5]_T = e([\mathbf{c}_0]_1, [\mathbf{k}_1^\top]_2).
\end{aligned}
$$

Then compute the vector $\mathbf{d}_{\mathbf{P},\mathbf{x}_{i^*}}$ as in Eq. (3) and $[z_6]_T = [(\mathbf{z}_1 - \mathbf{z}_2 - \mathbf{z}_3)\mathbf{d}_{\mathbf{P},\mathbf{x}_{i^*}} - z_4 - z_5]_T$. Output the message $\mathsf{m} = C \cdot [z_6]_T$.

Equivocal Public Key. We define the following algorithms:

– $\mathsf{ESetup}(1^\lambda, 1^L, P)$: It samples the common reference string crs as in Setup. In addition, it sets the equivocal key $\mathsf{ek} = \{\mathbf{W}_i\}_{i\in[L]}$.

– $\mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek})$: Sample $\mathbf{U}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$, $\widetilde{\mathbf{T}}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)\cdot 2m}$. Output the internal state $\mathsf{st}_i = (\mathbf{U}_i, \widetilde{\mathbf{T}}_i)$ and public key

$$
\mathsf{pk}_i = \left([\mathbf{A}\mathbf{U}_i, \mathbf{A}\widetilde{\mathbf{T}}_i]_1, \left\{ [\mathbf{U}_i \mathbf{B}\mathbf{r}_j^\top]_2 \right\}_{j\in[L]\setminus\{i\}}, \left\{ [\widetilde{\mathbf{T}}_i (\mathbf{I}_{2m} \otimes \mathbf{B}\mathbf{r}_j^\top)]_2 \right\}_{j\in[L]\setminus\{i\}} \right).
$$

– $\mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek})$: Parse $\mathsf{st}_i = (\mathbf{U}_i, \widetilde{\mathbf{T}}_i)$, $\mathsf{att}_i = \mathbf{x}_i$, and $\mathsf{ek} = \{\mathbf{W}_i\}_{i\in[L]}$. Define the matrix $\mathbf{K}_{\mathbf{x}_i}$ as in Eq. (3) and compute $\mathbf{T}_i = \widetilde{\mathbf{T}}_i - \mathbf{W}_i(\mathbf{K}_{\mathbf{x}_i} \otimes \mathbf{I}_{k+1})$. Output the secret key $\mathsf{sk}_i = (\mathbf{U}_i, \mathbf{T}_i)$.

The above key generation mode is statistically indistinguishable from the normal mode, hence satisfying the property of equivocal public keys.

Compactness. The above construction is compact, i.e., $|\mathsf{mpk}| = \mathsf{poly}(\lambda, m, \ell, \log L)$ and $|\mathsf{hsk}_i| = \mathsf{poly}(\lambda, m, \ell, \log L)$, where (m, ℓ) depend on the policy family P.

Correctness. We define the matrix $\mathbf{M}_{\mathbf{P},\mathbf{x}_{i^*}} = \begin{pmatrix} \mathbf{a} & \mathbf{0}_m \\ \mathbf{K}_{\mathbf{x}_{i^*}} & \mathbf{C}_{\mathbf{P}} \end{pmatrix}$ as in Eq. (3). Let $\mathbf{t}_{i,i^*} = (\mathbf{s}\mathbf{A}\mathbf{T}_i \| \mathbf{0}_{(k+1)m})(\mathbf{I}_{2m+1} \otimes \mathbf{B}\mathbf{r}_{i^*}^\top)$ for each $i \in [L]$, we hence obtain

$$
\begin{aligned}
\mathbf{z}_1 &= \sum_{i\in[L]} \left((\mathbf{s}\mathbf{A}\mathbf{V}_i + \mathbf{s}\mathbf{F}_i)(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{s}\mathbf{H}_i \| \mathbf{s}\mathbf{A}\mathbf{W}_i(\mathbf{C}_{\mathbf{P}} \otimes \mathbf{I}_{k+1}) \right) (\mathbf{I}_{2m+1} \otimes \mathbf{B}\mathbf{r}_{i^*}^\top),\\
&= \sum_{i\in[L]} (\mathbf{s}\mathbf{A}\mathbf{V}_i + \mathbf{s}\mathbf{F}_i \| \mathbf{s}\mathbf{A}\mathbf{W}_i)(\mathbf{M}_{\mathbf{P},\mathbf{x}_{i^*}} \otimes \mathbf{I}_{k+1})(\mathbf{I}_{2m+1} \otimes \mathbf{B}\mathbf{r}_{i^*}^\top) + \sum_{i\in[L]} \mathbf{t}_{i,i^*}\\
&= \sum_{i\in[L]} (\mathbf{s}\mathbf{A}\mathbf{V}_i + \mathbf{s}\mathbf{F}_i \| \mathbf{s}\mathbf{A}\mathbf{W}_i)(\mathbf{I}_{n+1} \otimes \mathbf{B}\mathbf{r}_{i^*}^\top)\mathbf{M}_{\mathbf{P},\mathbf{x}_{i^*}} + \sum_{i\in[L]} \mathbf{t}_{i,i^*}\\
&= \sum_{i\in[L]} (\mathbf{s}\mathbf{A}\mathbf{V}_i \mathbf{B}\mathbf{r}_{i^*}^\top + \mathbf{s}\mathbf{F}_i \mathbf{B}\mathbf{r}_{i^*}^\top \| \mathbf{s}\mathbf{A}\mathbf{W}_i(\mathbf{I}_n \otimes \mathbf{B}\mathbf{r}_{i^*}^\top))\mathbf{M}_{\mathbf{P},\mathbf{x}_{i^*}} + \sum_{i\in[L]} \mathbf{t}_{i,i^*},\\
\mathbf{z}_2 &= \sum_{i\in[L]\setminus\{i^*\}} \left((\mathbf{s}\mathbf{A}\mathbf{V}_i \mathbf{B}\mathbf{r}_{i^*}^\top + \mathbf{s}\mathbf{A}\mathbf{f}_{i,i^*}^\top)\mathbf{a} + \mathbf{s}\mathbf{A}\mathbf{Q}_{i,i^*}^\top \| \mathbf{s}\mathbf{A}\mathbf{W}_i(\mathbf{I}_n \otimes \mathbf{B}\mathbf{r}_j^\top)\mathbf{C}_{\mathbf{P}} \right)\\
&= \sum_{i\in[L]\setminus\{i^*\}} \left(\mathbf{s}\mathbf{A}\mathbf{V}_i \mathbf{B}\mathbf{r}_{i^*}^\top + \mathbf{s}\mathbf{A}\mathbf{f}_{i,i^*}^\top \| \mathbf{s}\mathbf{A}\mathbf{W}_i(\mathbf{I}_n \otimes \mathbf{B}\mathbf{r}_j^\top) \right) \mathbf{M}_{\mathbf{P},\mathbf{x}_{i^*}} + \sum_{i\in[L]\setminus\{i^*\}} \mathbf{t}_{i,i^*},\\
\mathbf{z}_3 &= (\mathbf{s}\mathbf{A}\mathbf{T}_{i^*}(\mathbf{I}_{2m} \otimes \mathbf{B}\mathbf{r}_{i^*}^\top) \| \mathbf{0}_{(k+1)m}) = \mathbf{t}_{i,i^*},\\
z_4 &= \mathbf{s}\mathbf{A}\mathbf{U}_{i^*}\mathbf{B}\mathbf{r}_{i^*}^\top,\\
z_5 &= \mathbf{s}\mathbf{A}\mathbf{V}_{i^*}\mathbf{B}\mathbf{r}_{i^*}^\top + \mathbf{s}\mathbf{A}\mathbf{k}^\top.
\end{aligned}
$$

If $\mathsf{P}(\mathbf{x}_{i^*}) = 1$, i.e., $\mathbf{M}_{\mathbf{P},\mathbf{x}_{i^*}}\mathbf{d}_{\mathbf{P},\mathbf{x}_{i^*}} = \mathbf{e}_1$, then we have

$$\begin{aligned}
z_6 &= (\mathbf{z}_1 - \mathbf{z}_2 - \mathbf{z}_3)\mathbf{d}_{\mathbf{P},\mathbf{x}_{i^*}} - z_4 - z_5 \\
&= (\mathbf{sAV}_i\mathbf{Br}_{i^*}^\top + \mathbf{sF}_{i^*}\mathbf{Br}_{i^*}^\top \| \mathbf{sAW}_{i^*}(\mathbf{I}_n \otimes \mathbf{Br}_{i^*}^\top))\mathbf{M}_{\mathbf{P},\mathbf{x}_{i^*}}\mathbf{d}_{\mathbf{P},\mathbf{x}_{i^*}} \\
&\quad - \mathbf{sAU}_{i^*}\mathbf{Br}_{i^*}^\top - \mathbf{sAV}_{i^*}\mathbf{Br}_{i^*}^\top - \mathbf{sAk}^\top \\
&= \mathbf{sAV}_i\mathbf{Br}_{i^*}^\top + \mathbf{sF}_{i^*}\mathbf{Br}_{i^*}^\top - \mathbf{sAU}_{i^*}\mathbf{Br}_{i^*}^\top - \mathbf{sAV}_{i^*}\mathbf{Br}_{i^*}^\top - \mathbf{sAk}^\top \\
&= -\mathbf{sAk}^\top.
\end{aligned}$$

where $\mathbf{F}_{i^*} = \mathbf{AU}_{i^*}$ for each $i^* \in [L]$. Finally, we have $z = C \cdot [z_6]_T = \mathsf{m}$. This proves the correctness.

4.3 Proof

Theorem 6. *Under the MDDH assumption (ref. Assumption 1), our construction satisfies policy-selective IND security as defined in Section 4.1.*

The proof strategy is analogous to [ZZGQ23]. Recall that our security definition requires sampling equivocal keys to answer oracle queries made by the adversary, but this does not affect the overall proof. Next, we give the formal proof.

Game Sequence. We prove the above theorem via the following game sequences. Let L be the number of slots, $\mathsf{att}_i^* = \mathbf{x}_i^*$ be the challenge attribute, P^* be the challenge policy, and m^* be the challenge message; Suppose $\{\mathsf{pk}_i^*\}_{i\in[L]}$ are challenge public keys to be registered. For all $i \in [L]$, $\mathcal{D}_i = \{\mathsf{pk}_i : \mathcal{D}_i[\mathsf{pk}_i] = \mathsf{sk}_i \neq \perp\}$ stores the response to $\mathsf{OGen}(i)$; $C_i = \{\mathsf{pk}_i : (i, \mathsf{pk}_i) \in C\}$ stores the response to $\mathsf{OCor}(i, \cdot)$.

- Game_0 : Real Game. Recall that:
  - The common reference string is that

$$\mathsf{crs} = \begin{pmatrix} [\mathbf{A}]_1, [\mathbf{Ak}^\top]_T, \{[\mathbf{AV}_i, \mathbf{AW}_i]_1\}_{i\in[L]}, \\ \left\{[\mathbf{Br}_i^\top, \mathbf{V}_i\mathbf{Br}_i^\top + \mathbf{k}^\top]_2\right\}_{i\in[L]}, \\ \left\{[\mathbf{V}_i\mathbf{Br}_j^\top, \mathbf{W}_i(\mathbf{I}_m \otimes \mathbf{Br}_j^\top)]_2\right\}_{i\in[L], j\in[L]\setminus\{i\}} \end{pmatrix}.$$

    In addition, it sets the equivocal key $\mathsf{ek} = \{\mathbf{W}_i\}_{i\in[L]}$.
  - For each $i \in [L]$, each public key $\mathsf{pk}_i \in \mathcal{D}_i$ is that

$$\mathsf{pk}_i = \left([\mathbf{AU}_i, \mathbf{A}\widetilde{\mathbf{T}}_i]_1, \left\{[\mathbf{U}_i\mathbf{Br}_j^\top]_2\right\}_{j\in[L]\setminus\{i\}}, \left\{[\widetilde{\mathbf{T}}_i(\mathbf{I}_{2m} \otimes \mathbf{Br}_j^\top)]_2\right\}_{j\in[L]\setminus\{i\}}\right).$$

    It corresponds to the internal state $\mathsf{st}_i = (\mathbf{U}_i, \widetilde{\mathbf{T}}_i)$, where $\mathbf{U}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$, $\widetilde{\mathbf{T}}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)\cdot 2m}$.
  - The challenge ciphertext is

$$\mathsf{ct} = \left([\underbrace{\mathbf{sA}}_{\mathbf{c}_0}]_1, \underbrace{\left[\sum_{i\in[L]}((\mathbf{sAV}_i + \mathbf{sAU}_i)(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{sH}_i)\right]_1}_{\mathbf{c}_1}, \underbrace{\left[\sum_{i\in[L]}\mathbf{sAW}_i(\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{I}_{k+1})\right]_1}_{\mathbf{c}_2}, \underbrace{[\mathbf{sAk}^\top]_T \cdot \mathsf{m}_b}_{c}\right).$$

    where $\mathbf{s} \leftarrow \mathbb{Z}_p^{1\times k}$. Besides, $\mathbf{H}_i = \mathbf{AW}_i(\mathbf{K}_{\mathbf{x}_i^*} \otimes \mathbf{I}_{k+1}) + \mathbf{AT}_i$ if slot i is malicious/corrupted, and $\mathbf{H}_i = \mathbf{A}\widetilde{\mathbf{T}}_i$ otherwise.

– Game_1 : Same as Game_0 except that we replace $\mathbf{sA}$ in challenge ciphertext with $\mathbf{c} \leftarrow \mathbb{Z}_p^{1\times(k+1)}$. Then the challenge ciphertext is

$$\mathsf{ct} = \left(\underbrace{[\ \mathbf{c}\ ]_1}_{\mathbf{c}_0}, \underbrace{\left[\sum_{i\in[L]} \left((\mathbf{c}\mathbf{V}_i + \mathbf{c}\mathbf{U}_i)(\mathbf{a}\otimes\mathbf{I}_{k+1}) + \mathbf{c}\widetilde{\mathbf{H}}_i \right) \right]_1}_{\mathbf{c}_1}, \underbrace{\left[\sum_{i\in[L]} \mathbf{c}\mathbf{W}_i(\mathbf{C}_\mathbf{P}\otimes\mathbf{I}_{k+1}) \right]_1}_{\mathbf{c}_2}, \underbrace{[\mathbf{c}\mathbf{k}^\top]_T \cdot \mathsf{m}_b}_{c} \right).$$

where $\widetilde{\mathbf{H}}_i = \mathbf{W}_i(\mathbf{K}_{\mathbf{x}_i^*}\otimes\mathbf{I}_{k+1}) + \mathbf{T}_i$ if slot i is malicious/corrupted, and $\widetilde{\mathbf{H}}_i = \widetilde{\mathbf{T}}_i$ otherwise. Observe that we have $\mathsf{Game}_0 \approx_c \mathsf{Game}_1$, which follows the MDDH assumption, ensuring that $([\mathbf{A}]_1, [\mathbf{sA}]_1) \approx_c ([\mathbf{A}]_1, [\mathbf{c}]_1)$ where $\mathbf{A} \leftarrow \mathbb{Z}_p^{k\times(k+1)}, \mathbf{s} \leftarrow \mathbb{Z}_p^{1\times k}$ and $\mathbf{c} \leftarrow \mathbb{Z}_p^{1\times(k+1)}$. See Lemma 1 for more details.

– $\mathsf{Game}_{2,\ell,1}(\ell \in [L])$: Same as $\mathsf{Game}_{2,\ell-1,3}$ except that we replace $[\mathbf{B}\mathbf{r}_\ell^\top, \mathbf{V}_\ell\mathbf{B}\mathbf{r}_\ell^\top + \mathbf{k}^\top]_2$ in crs with the following form

$$[\ \mathbf{d}_\ell^\top, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \mathbf{k}^\top]_2,$$

where $\mathbf{d}_\ell \leftarrow \mathbb{Z}_p^{1\times(k+1)}$. We have $\mathsf{Game}_{2,\ell-1,3} \approx_c \mathsf{Game}_{2,\ell,1}$, which follows the MDDH assumption $([\mathbf{B}]_2, [\mathbf{B}\mathbf{r}_\ell^\top]_2) \approx_c ([\mathbf{B}]_2, [\mathbf{d}_\ell^\top]_2)$, where $\mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}, \mathbf{r}_\ell \leftarrow \mathbb{Z}_p^{1\times(k+1)}$ and $\mathbf{d}_\ell \leftarrow \mathbb{Z}_p^{1\times k}$. See Lemma 2 for more details.

– $\mathsf{Game}_{2,\ell,2}$: Same as $\mathsf{Game}_{2,\ell,1}$ except that we change replace $[\mathbf{d}_\ell^\top, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \mathbf{k}^\top]_2$ in crs with the following form

$$[\mathbf{d}_\ell^\top, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \alpha\mathbf{c}^\perp + \mathbf{k}^\top]_2,$$

where $\alpha_\ell \leftarrow \mathbb{Z}_p$ and $\mathbf{c}^\perp \leftarrow \mathbb{Z}_p^{2k+1}$ such that $\mathbf{A}\mathbf{c}^\perp = 0, \mathbf{c}\mathbf{c}^\perp = 1$. We have $\mathsf{Game}_{2,\ell,1} \approx_s \mathsf{Game}_{2,\ell,2}$, see Lemma 3 for more details.

– $\mathsf{Game}_{2,\ell,3}$: Same as $\mathsf{Game}_{2,\ell,2}$ except that we change replace $[\mathbf{d}_\ell^\top, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \alpha\mathbf{c}^\perp + \mathbf{k}^\top]_2$ in crs with the following form

$$[\ \mathbf{B}\mathbf{r}_\ell^\top, \mathbf{V}_\ell\mathbf{B}\mathbf{r}_\ell^\top + \alpha\mathbf{c}^\perp + \mathbf{k}^\top]_2.$$

Notice that we can prove $\mathsf{Game}_{2,\ell,2} \approx_c \mathsf{Game}_{2,\ell,3}$ via Lemma 2 as well.

– Game_3 : Same as $\mathsf{Game}_{2,L}$ except that we replace the ciphertext into the following form:

$$\mathsf{ct}^* = \left(\underbrace{[\ \mathbf{c}\ ]_1}_{\mathbf{c}_0}, \underbrace{\left[\sum_{j\in[L]} (\mathbf{c}\mathbf{V}_j + \mathbf{c}\mathbf{U}_j) \right]_1}_{\mathbf{c}_1}, \underbrace{g_T^*}_{c} \right).$$

where g_T^* is sampled uniformly over $\mathbb{G}_T$. We claim that $\mathsf{Game}_{2,L} \approx_s \mathsf{Game}_3$ which follows the following the statistical argument:

$$(\mathbf{A}\mathbf{k}^\top, \mathbf{k}^\top + \alpha\mathbf{c}^\perp, \mathbf{c}\mathbf{k}^\top) \approx_s (\mathbf{A}\mathbf{k}^\top, \mathbf{k}^\top, \mathbf{c}\mathbf{k}^\top - \alpha)$$

where $[\mathbf{c}\mathbf{k}^\top - \alpha]_T$ is uniform, namely g_T^*.

We see that in Game_3, the challenge ciphertext ct^* is independent of b, so that $\mathcal{A}$ cannot learn any information about b. Thus, the advantage of $\mathcal{A}$ is exactly 0.

In the following, we use $\mathsf{Adv}^i_{\mathcal{A}}(\lambda)$ to denote the advantage of $\mathcal{A}$ in Game_i.

Lemma 1. *By the $(k, k+1, 1)$-MDDH assumption, we claim $\mathsf{Game}_0 \approx_c \mathsf{Game}_1$.*

Proof. Recall that the difference between two games is that we replace $[\mathbf{sA}]_1$ in Game_0 with $[\mathbf{c}]_1$, where $\mathbf{A} \leftarrow \mathbb{Z}_p^{k\times(k+1)}, \mathbf{s} \leftarrow \mathbb{Z}_p^{1\times k}$ and $\mathbf{c} \leftarrow \mathbb{Z}_p^{1\times(k+1)}$. This follows from $(k, k+1, 1)$-MDDH assumption, which ensures that:

$$([\mathbf{A}]_1, [\mathbf{sA}]_1) \approx_c ([\mathbf{A}]_1, [\mathbf{c}]_1).$$

On input $([\mathbf{A}]_1, [\mathbf{t}]_1)$ where $\mathbf{t} = \mathbf{sA}$ or $\mathbf{t} = \mathbf{c}$, we can define algorithm $\mathcal{B}$ that interacts with the adversary $\mathcal{A}$ as follows:

– *Setup phase:* Sample

$$\mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k},\ \mathbf{k} \leftarrow \mathbb{Z}_p^{1\times(k+1)},$$
$$\forall i \in [L],\ \mathbf{V}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)},\ \mathbf{W}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)n},\ \mathbf{r}_i \leftarrow \mathbb{Z}_p^{1\times k}.$$

Output the equivocal key $\mathsf{ek} = \{\mathbf{W}_i\}_{i\in[L]}$ and the common reference string

$$\mathsf{crs} = \begin{pmatrix} [\mathbf{A}]_1, [\mathbf{A}\mathbf{k}^\top]_T, \{[\mathbf{A}\mathbf{V}_i, \mathbf{A}\mathbf{W}_i]_1\}_{i\in[L]}, \\ \left\{[\mathbf{B}\mathbf{r}_i^\top, \mathbf{V}_i\mathbf{B}\mathbf{r}_i^\top + \mathbf{k}^\top]_2\right\}_{i\in[L]}, \\ \left\{[\mathbf{V}_i\mathbf{B}\mathbf{r}_j^\top, \mathbf{W}_i(\mathbf{I}_m \otimes \mathbf{B}\mathbf{r}_j^\top)]_2\right\}_{i\in[L], j\in[L]\setminus\{i\}} \end{pmatrix}.$$

– *Query phase:* Here, for all $i \in [L]$ and each $(\mathsf{pk}_i, \mathsf{st}_i) \in \mathcal{D}_i$ is generated honestly as :

$$\mathsf{pk}_i = \left([\mathbf{A}\mathbf{U}_i, \mathbf{A}\widetilde{\mathbf{T}}_i]_1, \left\{[\mathbf{U}_i\mathbf{B}\mathbf{r}_j^\top]_2\right\}_{j\in[L]\setminus\{i\}}, \left\{[\widetilde{\mathbf{T}}_i(\mathbf{I}_{2m} \otimes \mathbf{B}\mathbf{r}_j^\top)]_2\right\}_{j\in[L]\setminus\{i\}}\right).$$

and the internal state $\mathsf{st}_i = (\mathbf{U}_i, \widetilde{\mathbf{T}}_i)$, where $\mathbf{U}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$, $\widetilde{\mathbf{T}}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)\cdot 2m}$.

For a corruption query on $(i, \mathsf{pk}_i, \mathbf{x}_i)$, it returns the secret key $\mathsf{sk}_i = (\mathbf{U}_i, \mathbf{T}_i)$, where $\mathbf{T}_i = \widetilde{\mathbf{T}}_i - \mathbf{W}_i(\mathbf{K}_{\mathbf{x}_i} \otimes \mathbf{I}_{k+1})$.

– *Challenge phase:* On input challenge m_b^*, output

$$\mathsf{ct}^* = \left([\mathbf{t}]_1, \left[\sum_{i\in[L]}\left((\mathbf{t}\mathbf{V}_i + \mathbf{t}\mathbf{U}_i)(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{t}\widetilde{\mathbf{H}}_i\right)\right]_1, \left[\sum_{i\in[L]} \mathbf{t}\mathbf{W}_i(\mathbf{C}_\mathbf{P} \otimes \mathbf{I}_{k+1})\right]_1, [\mathbf{t}\mathbf{k}^\top]_T \cdot \mathsf{m}_b^*\right).$$

Note that $\widetilde{\mathbf{H}}_i = \mathbf{W}_i(\mathbf{K}_{\mathbf{x}_i^*} \otimes \mathbf{I}_{k+1}) + \mathbf{T}_i$ if slot i is malicious/corrupted, and $\widetilde{\mathbf{H}}_i = \widetilde{\mathbf{T}}_i$ otherwise. Since $\mathcal{B}$ have known all challenge attributes $\mathsf{att}_i^* = \mathbf{x}_i^*$ together with randomness for each pk_i^*, it can compute $\widetilde{\mathbf{H}}_i$ properly.

Observe that when $\mathbf{t} = \mathbf{s}\mathbf{A}$, $\mathcal{B}$ exactly simulates Game_0; when $\mathbf{t} = \mathbf{c}$, $\mathcal{B}$ exactly simulates Game_1. This readily proves the lemma. □

Lemma 2. *By the* $(k, k+1, 1)$*-MDDH assumption, we claim* $\mathsf{Game}_{2,\ell-1,3} \approx_c \mathsf{Game}_{2,\ell,1}$.

Proof. This follows from the $(k, k+1, 1)$-MDDH assumption:

$$([\mathbf{B}]_2, [\mathbf{B}\mathbf{r}_\ell^\top]_2) \approx_c ([\mathbf{B}]_2, [\mathbf{d}_\ell^\top]_2)$$

where $\mathbf{B} \leftarrow \mathbb{Z}_p^{(k+1)\times k}$, $\mathbf{r}_\ell \leftarrow \mathbb{Z}_p^{1\times k}$ and $\mathbf{d}_\ell \leftarrow \mathbb{Z}_p^{1\times(k+1)}$. On input $[\mathbf{B}]_2, [\mathbf{t}_\ell^\top]_2$ where $\mathbf{t}_\ell^\top = \mathbf{B}\mathbf{r}_\ell^\top$ or $\mathbf{t}_\ell^\top = \mathbf{d}_\ell^\top$, we can define algorithm $\mathcal{B}$ that interacts with the adversary $\mathcal{A}$ as follows:

– *Setup phase:* Sample

$$\mathbf{A} \leftarrow \mathbb{Z}_p^{k\times(k+1)},\ \mathbf{k} \leftarrow \mathbb{Z}_p^{1\times(k+1)},$$
$$\forall i \in [L],\ \mathbf{V}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)},\ \mathbf{W}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)n},$$
$$\forall i \in [L]\setminus\{\ell\},\ \mathbf{r}_i \leftarrow \mathbb{Z}_p^{1\times k}.$$

Output the equivocal key $\mathsf{ek} = \{\mathbf{W}_i\}_{i\in[L]}$ and the common reference string

$$\mathsf{crs} = \begin{pmatrix} [\mathbf{A}]_1, [\mathbf{A}\mathbf{k}^\top]_T, \{[\mathbf{A}\mathbf{V}_i, \mathbf{A}\mathbf{W}_i]_1\}_{i\in[L]}, \\ \left\{[\mathbf{B}\mathbf{r}_i^\top, \mathbf{V}_i\mathbf{B}\mathbf{r}_i^\top + \mathbf{k}^\top + \alpha\mathbf{c}^\perp]_2\right\}_{i<\ell}, \boxed{[\mathbf{t}_\ell, \mathbf{V}_\ell\mathbf{t}_\ell^\top + \mathbf{k}^\top]_2}, \left\{[\mathbf{B}\mathbf{r}_i^\top, \mathbf{V}_i\mathbf{B}\mathbf{r}_i^\top + \mathbf{k}^\top]_2\right\}_{i>\ell}, \\ \left\{[\mathbf{V}_i\mathbf{B}\mathbf{r}_j^\top, \mathbf{W}_i(\mathbf{I}_m \otimes \mathbf{B}\mathbf{r}_j^\top)]_2\right\}_{i\in[L]\setminus\{j\}, j\in[L]\setminus\{\ell\}}, \boxed{\left\{[\mathbf{V}_i\mathbf{t}_\ell^\top, \mathbf{W}_i(\mathbf{I}_m \otimes \mathbf{t}_\ell^\top)]_2\right\}_{i\in[L]\setminus\{\ell\}}} \end{pmatrix}.$$

– *Query phase:* Here, for all $i \in [L]$ and each $(\mathsf{pk}_i, \mathsf{st}_i) \in \mathcal{D}_i$ is generated as follows:

- if $i \neq \ell$, the pk_i is that

$$\begin{pmatrix} [\mathbf{A}\mathbf{U}_i, \mathbf{A}\widetilde{\mathbf{T}}_i]_1, \left\{[\mathbf{U}_i\mathbf{B}\mathbf{r}_j^\top]_2\right\}_{j\in[L]\setminus\{i,\ell\}}, \boxed{[\mathbf{U}_i\mathbf{t}_\ell^\top]_2} \\ \left\{[\widetilde{\mathbf{T}}_i(\mathbf{I}_{2m}\otimes\mathbf{B}\mathbf{r}_j^\top)]_2\right\}_{j\in[L]\setminus\{i,\ell\}}, \boxed{[\widetilde{\mathbf{T}}_i(\mathbf{I}_{2m}\otimes\mathbf{t}_\ell^\top)]_2} \end{pmatrix}.$$

- if $i = \ell$, the pk_ℓ is that

$$\left([\mathbf{A}\mathbf{U}_\ell, \mathbf{A}\widetilde{\mathbf{T}}_\ell]_1, \left\{[\mathbf{U}_\ell\mathbf{B}\mathbf{r}_j^\top]_2\right\}_{j\in[L]\setminus\{\ell\}}, \left\{[\widetilde{\mathbf{T}}_\ell(\mathbf{I}_{2m}\otimes\mathbf{B}\mathbf{r}_j^\top)]_2\right\}_{j\in[L]\setminus\{\ell\}}\right).$$

and $\mathsf{st}_i = (\mathbf{U}_i, \widetilde{\mathbf{T}}_i)$, where $\mathbf{U}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)}$, $\widetilde{\mathbf{T}}_i \leftarrow \mathbb{Z}_p^{(k+1)\times(k+1)\cdot 2m}$.

For a corruption query on $(i, \mathsf{pk}_i, \mathbf{x}_i)$, it returns the secret key $\mathsf{sk}_i = (\mathbf{U}_i, \mathbf{T}_i)$, where $\mathbf{T}_i = \widetilde{\mathbf{T}}_i - \mathbf{W}_i(\mathbf{K}_{\mathbf{x}_i}\otimes\mathbf{I}_{k+1})$.

– *Challenge phase:* On input challenge m^*, output ct^* as

$$\mathsf{ct}^* = \left([\mathbf{c}]_1, \left[\sum_{i\in[L]}\left((\mathbf{c}\mathbf{V}_i + \mathbf{c}\mathbf{U}_i)(\mathbf{a}\otimes\mathbf{I}_{k+1}) + \mathbf{c}\widetilde{\mathbf{H}}_i\right)\right]_1, \left[\sum_{i\in[L]}\mathbf{c}\mathbf{W}_i(\mathbf{C}_\mathbf{P}\otimes\mathbf{I}_{k+1})\right]_1, [\mathbf{c}\mathbf{k}^\top]_T\cdot\mathsf{m}_b^*\right).$$

where $\mathbf{c} \leftarrow \mathbb{Z}_p^{1\times(k+1)}$. Note that $\widetilde{\mathbf{H}}_i = \mathbf{W}_i(\mathbf{K}_{\mathbf{x}_i^*}\otimes\mathbf{I}_{k+1}) + \mathbf{T}_i$ if slot i is malicious/corrupted, and $\widetilde{\mathbf{H}}_i = \widetilde{\mathbf{T}}_i$ otherwise. Since $\mathcal{B}$ have known all challenge attributes $\mathsf{att}_i^* = \mathbf{x}_i^*$ together with randomness for each pk_i^*, it can compute $\widetilde{\mathbf{H}}_i$ properly.

Observe that when $\mathbf{t}_\ell^\top = \mathbf{B}\mathbf{r}_\ell^\top$, the simulation is identical to $\mathsf{Game}_{2,\ell-1,3}$; when $\mathbf{t}_\ell^\top = \mathbf{d}_\ell^\top$, the simulation is identical to $\mathsf{Game}_{2,\ell,1}$. □

Lemma 3. *We claim* $\mathsf{Game}_{2,\ell,1} \approx_s \mathsf{Game}_{2,\ell,2}$.

Proof. Here, we argue $\mathsf{Game}_{2,\ell,1} \approx_s \mathsf{Game}_{2,\ell,2}$ by considering two cases:

– If slot ℓ is honest, then we have

$$\begin{aligned}
&\begin{cases} \mathbf{A}, \mathbf{B}, \mathbf{c}^\perp, \mathbf{d}_\ell^\top, \mathbf{A}\mathbf{V}_\ell, \mathbf{V}_\ell\mathbf{B}, \mathbf{V}_\ell\mathbf{d}_\ell^\top + b\mathbf{c}^\perp\alpha & //\mathsf{crs} \\ \mathbf{A}\mathbf{U}_\ell, \mathbf{U}_\ell\mathbf{B}; \quad \mathbf{c}, \mathbf{c}\mathbf{V}_\ell + \mathbf{c}\mathbf{U}_\ell & //\,\mathsf{pk}_i^*;\ \mathsf{ct}^* \end{cases} \\
\approx_s &\begin{cases} \mathbf{A}, \mathbf{B}, \mathbf{c}^\perp, \mathbf{d}_\ell^\top, \mathbf{A}\mathbf{V}_\ell, \mathbf{V}_\ell\mathbf{B}, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \boxed{\nu_\ell\mathbf{c}^\perp} + b\mathbf{c}^\perp\alpha \\ \mathbf{A}\mathbf{U}_\ell; \quad \mathbf{c}, \mathbf{c}\mathbf{V}_\ell + \mathbf{c}\mathbf{U}_\ell + \boxed{\nu_\ell\mathbf{d}^\perp + u_\ell\mathbf{d}^\perp} \end{cases} \\
\approx_s &\begin{cases} \mathbf{A}, \mathbf{B}, \mathbf{c}^\perp, \mathbf{d}_\ell^\top, \mathbf{A}\mathbf{V}_\ell, \mathbf{V}_\ell\mathbf{B}, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \nu_\ell\mathbf{c}^\perp + \cancel{b\mathbf{c}^\perp\alpha} \\ \mathbf{A}\mathbf{U}_\ell; \quad \mathbf{c}, \mathbf{c}\mathbf{V}_\ell + \mathbf{c}\mathbf{U}_\ell + \nu_\ell\mathbf{d}^\perp + u_\ell\mathbf{d}^\perp \end{cases}
\end{aligned}$$

where $b \in \{0,1\}$. Here, we have

- The first $\approx_s$ follows that:

$$\mathbf{V}_\ell \mapsto \mathbf{V}_\ell + \mathbf{c}^\perp\nu_\ell\mathbf{d}^\perp \quad \text{and} \quad \mathbf{U}_\ell \mapsto \mathbf{U}_\ell + \mathbf{c}^\perp u_\ell\mathbf{d}^\perp$$

where $\nu_\ell, u_\ell \leftarrow \mathbb{Z}_p$. Here, it sets $\mathbf{c}^\perp \in \mathbb{Z}_p^{k+1}$ and $\mathbf{d}^\perp \in \mathbb{Z}_p^{1\times(k+1)}$ such that $\mathbf{A}\mathbf{c}^\perp = \mathbf{0}, \mathbf{c}\mathbf{c}^\perp = 1, \mathbf{d}^\perp\mathbf{B} = \mathbf{0}, \mathbf{d}^\perp\mathbf{d}_\ell = 1$.

- The second $\approx_s$ holds by the fact that ν_ℓ in crs also seems to be sampled randomly because u_ℓ hides extra ν_ℓ in ct^*.

– If slot ℓ is malicious/corrupted, then we have

$$\begin{aligned}
&\begin{cases}
\mathbf{A}, \mathbf{B}, \mathbf{c}^\perp, \mathbf{d}_\ell^\top, \mathbf{A}\mathbf{V}_\ell, \mathbf{A}\mathbf{W}_\ell, \mathbf{W}_\ell(\mathbf{I}_m \otimes \mathbf{B}), \mathbf{V}_\ell\mathbf{B}, \mathbf{V}_\ell\mathbf{d}_\ell^\top + b\mathbf{c}^\perp\alpha & //\mathsf{crs} \\
\mathbf{c}, \mathbf{c}\mathbf{V}_\ell(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{c}\mathbf{W}_\ell(\mathbf{K}_{\mathbf{x}_\ell^*} \otimes \mathbf{I}_{k+1}), \mathbf{c}\mathbf{W}_\ell(\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{I}_{k+1}) & //\mathsf{ct}^*
\end{cases} \\
\approx_s &\begin{cases}
\mathbf{A}, \mathbf{B}, \mathbf{c}^\perp, \mathbf{d}_\ell^\top, \mathbf{A}\mathbf{V}_\ell, \mathbf{A}\mathbf{W}_\ell, \mathbf{W}_\ell(\mathbf{I}_m \otimes \mathbf{B}), \mathbf{V}_\ell\mathbf{B}, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \boxed{\nu_\ell\mathbf{c}^\perp} + b\mathbf{c}^\perp\alpha \\
\mathbf{c}, \mathbf{c}\mathbf{V}_\ell(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{c}\mathbf{W}_\ell(\mathbf{K}_{\mathbf{x}_\ell^*} \otimes \mathbf{I}_{k+1}) + \boxed{\nu_\ell\mathbf{a} \otimes \mathbf{d}^\perp + \mathbf{w}_\ell^\top\mathbf{K}_{\mathbf{x}_\ell^*} \otimes \mathbf{d}^\perp}, \\
\mathbf{c}\mathbf{W}_\ell(\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{I}_{k+1}) + \boxed{\mathbf{w}_\ell^\top\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{d}^\perp}
\end{cases} \\
\approx_s &\begin{cases}
\mathbf{A}, \mathbf{B}, \mathbf{c}^\perp, \mathbf{d}_\ell^\top, \mathbf{A}\mathbf{V}_\ell, \mathbf{A}\mathbf{W}_\ell, \mathbf{W}_\ell(\mathbf{I}_m \otimes \mathbf{B}), \mathbf{V}_\ell\mathbf{B}, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \nu_\ell\mathbf{c}^\perp + b\mathbf{c}^\perp\alpha \\
\mathbf{c}, \mathbf{c}\mathbf{V}_\ell(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{c}\mathbf{W}_\ell(\mathbf{K}_{\mathbf{x}_\ell^*} \otimes \mathbf{I}_{k+1}) + \cancel{\nu_\ell\mathbf{a} \otimes \mathbf{d}^\perp} + \mathbf{w}_\ell^\top\mathbf{K}_{\mathbf{x}_\ell^*} \otimes \mathbf{d}^\perp, \\
\mathbf{c}\mathbf{W}_\ell(\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{I}_{k+1}) + \mathbf{w}_\ell^\top\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{d}^\perp
\end{cases} \\
\approx_s &\begin{cases}
\mathbf{A}, \mathbf{B}, \mathbf{c}^\perp, \mathbf{d}_\ell^\top, \mathbf{A}\mathbf{V}_\ell, \mathbf{A}\mathbf{W}_\ell, \mathbf{W}_\ell(\mathbf{I}_m \otimes \mathbf{B}), \mathbf{V}_\ell\mathbf{B}, \mathbf{V}_\ell\mathbf{d}_\ell^\top + \nu_\ell\mathbf{c}^\perp + \cancel{b\mathbf{c}^\perp\alpha} \\
\mathbf{c}, \mathbf{c}\mathbf{V}_\ell(\mathbf{a} \otimes \mathbf{I}_{k+1}) + \mathbf{c}\mathbf{W}_\ell(\mathbf{K}_{\mathbf{x}_\ell^*} \otimes \mathbf{I}_{k+1}) + \mathbf{w}_\ell^\top\mathbf{K}_{\mathbf{x}_\ell^*} \otimes \mathbf{d}^\perp, \\
\mathbf{c}\mathbf{W}_\ell(\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{I}_{k+1}) + \mathbf{w}_\ell^\top\mathbf{C}_{\mathbf{P}^*} \otimes \mathbf{d}^\perp
\end{cases}
\end{aligned}$$

where $b \in \{0,1\}$. Here, we have

- The first $\approx_s$ follows that:

$$\mathbf{V}_\ell \mapsto \mathbf{V}_\ell + \mathbf{c}^\perp\nu_\ell\mathbf{d}^\perp \quad \text{and} \quad \mathbf{W}_\ell \mapsto \mathbf{W}_\ell + \mathbf{c}^\perp(\mathbf{w}_\ell^\top \otimes \mathbf{d}^\perp)$$

  where $\nu_\ell \leftarrow \mathbb{Z}_p, \mathbf{w}_\ell \leftarrow \mathbb{Z}_p^n$. Here, it sets $\mathbf{c}^\perp \in \mathbb{Z}_p^{k+1}$ and $\mathbf{d}^\perp \in \mathbb{Z}_p^{1\times(k+1)}$ such that $\mathbf{A}\mathbf{c}^\perp = \mathbf{0}, \mathbf{c}\mathbf{c}^\perp = 1, \mathbf{d}^\perp\mathbf{B} = \mathbf{0}, \mathbf{d}^\perp\mathbf{d}_\ell = 1$.
- The second $\approx_s$ uses the fact $\mathsf{P}^*(\mathbf{x}_\ell^*) = 0$ and the α-privacy of predicate encodings.
- The third $\approx_s$ holds by the fact that ν_ℓ in crs is sampled randomly, hence hiding the bit b.

This readily proves the lemma. □

Discussion on Post-Quantum Security. In this work, we discover that an equivocal sReg-ABE scheme can be utilized to constructed dynamic bounded collusion Reg-FE with fully adaptive SIM security. We combine a pairing-based sReg-ABE scheme with a well-structured equivocal commitment to derive equivocal sReg-ABE. Since several concrete lattice-based Reg-ABE schemes have recently been proposed [WW25,CHW25b,AMR25], one may expect that a similar idea can be applied to obtain new equivocal sReg-ABE schemes with post-quantum security, hence leading to dynamic bounded collusion Reg-FE with both adaptive security and post-quantum security. However, we argue that it is still unclear whether there exists an equivocal commitment scheme that is compatible with existing lattice-based Reg-ABE schemes. Thus, we leave constructing dynamic bounded collusion Reg-FE with both adaptive SIM security and post-quantum security as a future work.

5 From Equivocal Slotted Reg-ABE to Equivocal Slotted Reg-ABFE

In this section, we formally define equivocal slotted Reg-ABFE and construct it from equivocal slotted Reg-ABE. In particular, we first consider the construction with 1-bound security, then upgrade it into Q-bound adaptive security.

5.1 Definition

An equivocal slotted Reg-ABFE (equivocal sReg-ABFE for short) for policy family $P : A \to \{0,1\}$ and circuit family $C : X \to Z$ is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C) \to \mathsf{crs}$. It takes as input the security parameter 1^λ, slot amount 1^L, collusion bound 1^Q, policy family P, and circuit family C, then outputs a common reference string crs.
- $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. It takes as input the common reference string crs, a slot index $i \in [L]$, and an attribute $\mathsf{att}_i \in A$, then outputs a public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$. It takes as input crs and a series of public keys and circuits $\{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}$, then outputs the master public key mpk and helper keys $\{\mathsf{hsk}_j\}_{j\in[L]}$. This algorithm is deterministic.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x) \to \mathsf{ct}$. It takes as input the master public key mpk, a policy $\mathsf{P} \in P$, and a message $x \in X$, then outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct}) \to z/\perp$. It takes as input a helper key hsk_{i^*}, a secret key sk_{i^*}, an attribute $\mathsf{att}_{i^*} \in A$ and a ciphertext ct, then outputs $z \in Z$ or an empty symbol $\perp$.

The above scheme should satisfy the following properties:

- **Compactness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all P, all C, and all $i \in [L]$, it requires

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, P, C, Q, \log L) \text{ and } |\mathsf{hsk}_i| = \mathsf{poly}(\lambda, P, C, Q, \log L).$$

- **Correctness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all P and all C, all $\mathsf{C}_1, \ldots, \mathsf{C}_L \in C$ and $\mathsf{att}_1, \ldots, \mathsf{att}_L$, all $i^* \in [L]$, all $\mathsf{att}_{i^*} \in A$, all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C)$, all $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i^*, \mathsf{att}_{i^*})$, all public keys $\{\mathsf{pk}_i\}_{i\in[L]\setminus\{i^*\}}$ such that $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$, all $\mathsf{P} \in P$ such that $\mathsf{P}(\mathsf{att}_{i^*}) = 1$, and all $x \in X$, and computing $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) \leftarrow \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$, we have

$$\Pr\left[\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x)) = \mathsf{C}_{i^*}(x)\right] = 1.$$

- **Equivocal Public Key.** We define the following algorithms:
  - $\mathsf{ESetup}(1^\lambda, 1^L, 1^Q, P, C) \to (\mathsf{crs}, \mathsf{ek})$. Same as Setup, except that it additionally outputs an equivocal key ek. We require that the output crs is indistinguishable from that produced by Setup.
  - $\mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek}) \to (\mathsf{pk}_i, \mathsf{st}_i)$. Same as Gen, except that it does not involve att_i and outputs a public key pk_i and internal state st_i.
  - $\mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek}) \to \mathsf{sk}_i$. It takes as input an internal state st_i, an attribute att_i, and an equivocal key ek, then outputs a secret key sk_i.

  This property requires $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i) \approx_c \{\mathsf{pk}_i, \mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek})\}$, where $(\mathsf{pk}_i, \mathsf{st}_i) \leftarrow \mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek})$.
- **(Non-)Adaptive SIM Security with Selective Policies.** For all efficient adversaries $\mathcal{A}$, there exists a simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ such that it is computationally hard to distinguish the following two worlds:
  - *Setup phase:* $\mathcal{A}$ specifies $(1^L, 1^Q, P, C)$ and $\mathsf{P}_1, \ldots, \mathsf{P}_{Q_{\mathsf{Enc}}} \in P$, where Q_{Enc} is the number of encryption queries. In the real world (left hand) or the ideal world (right hand), the challenger returns crs to $\mathcal{A}$, where

$$(\mathsf{crs}, \mathsf{ek}) \leftarrow \mathsf{ESetup}(1^\lambda, 1^L, 1^Q, P, C) \text{ or } (\mathsf{crs}, \mathsf{td}) \leftarrow \widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, P, C).$$

  - *Pre-challenge phase:* The challenger initializes a dictionary $\{\mathcal{D}_i = \emptyset\}_{i\in[L]}$ and a set $C = \emptyset$. In this phase, $\mathcal{A}$ can freely make key generation/corruption queries to the following oracles:
    * $\mathsf{OGen}(i)$: In the real world (left hand) or the ideal world (right hand), the challenger runs

$$(\mathsf{pk}, \mathsf{st}) \leftarrow \mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek}) \text{ or } (\mathsf{pk}, \mathsf{st}) \leftarrow \widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td}).$$

      Then, the challenger sets $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}$. Return pk to $\mathcal{A}$.

* $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: Parse $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}$. The challenger returns the secret key sk to $\mathcal{A}$, where

$$\mathsf{sk} \leftarrow \mathsf{Equiv}(\mathsf{st}, \mathsf{att}, \mathsf{ek}) \ \text{ or } \ \mathsf{sk} \leftarrow \widetilde{\mathsf{Gen}}_2(\mathsf{st}, \mathsf{att}, \mathsf{td}, (\perp, \perp)).$$

  Then, the challenger updates $C = C \cup \{(i, \mathsf{pk}, \mathsf{att})\}$.

- *Challenge phase:* In this phase, $\mathcal{A}$ specifies L tuples of public keys, attributes and circuits $\{(\mathsf{pk}_i^*, \mathsf{att}_i^*, \mathsf{C}_i^*)\}_{i\in[L]}$ together with a list of randomness $\{r_i^*\}$. It computes $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i^*, \mathsf{C}_i^*)\}_{i\in[L]})$. Then, the challenger checks:
  * For each slot $i \in [L]$, it must fall in one of the following three cases:

$$\begin{aligned}
&\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \notin C && \text{(honest slot)}\\
&\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \in C && \text{(corrupted slot)}\\
&\mathcal{D}_i[\mathsf{pk}_i^*] = \perp && \text{(malicious slot)}
\end{aligned}$$

  * For each malicious slot i, it must hold that $\mathsf{pk}_i^* \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i^*; r_i^*)$, where r_i^* is the randomness used by the algorithm Gen.

  It aborts if either of the above two conditions is not satisfied.
- *Post-challenge phase:* In this phase, $\mathcal{A}$ can adaptively make encryption/corruption queries as follows:
  * $\mathsf{OEnc}(\mathsf{P}_d, x_d)$: We assume that this is the d-th encryption query made by $\mathcal{A}$, where $d = 1, \ldots, Q_{\mathsf{Enc}}$. Record the corrupted/malicious slot indices in the set $\mathcal{K}_d$, where it requires $|\mathcal{K}_d| \leq Q$ and $\mathsf{P}_d(\mathsf{att}_j^*) = 1$ for each $j \in \mathcal{K}_d$. Next, in the real world (left hand) or the ideal world (right hand), it replies $\mathcal{A}$ with a ciphertext ct_d by sampling either

$$\mathsf{ct}_d \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}_d, x_d) \ \text{ or } \ \mathsf{ct}_d \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_d, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x))\}_{j\in\mathcal{K}_d}).$$

  * $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: Parse $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}$. If pk is not registered into the system or it satisfies $\mathsf{P}_d(\mathsf{att}) = 0$ for all $d \in [Q_{\mathsf{Enc}}]$, the challenger acts as in the pre-challenge phase; otherwise, there exists a unique index $d \in [Q_{\mathsf{Enc}}]$ such that $\mathsf{P}_d(\mathsf{att}) = 1$ and $\mathsf{P}_k(\mathsf{att}) = 0$ for any $k \neq d$, where (P_d, x_d) has been queried to OEnc before. Then, in the real world (left hand) or the ideal world (right hand), it computes

$$\mathsf{sk} \leftarrow \mathsf{Equiv}(\mathsf{st}, \mathsf{att}, \mathsf{ek}) \ \text{ or } \ \mathsf{sk} \leftarrow \widetilde{\mathsf{Gen}}_2(\mathsf{st}, \mathsf{att}, \mathsf{td}, (\mathsf{C}_i^*, \mathsf{C}_i^*(x_d))).$$

    Then, the challenger returns sk to $\mathcal{A}$. Update $\mathcal{K}_d = \mathcal{K}_d \cup \{i\}$.
- *Output phase:* $\mathcal{A}$ outputs a bit to indicate the real/ideal world.

For the above experiment, we require

(i) For each $d \in [Q_{\mathsf{Enc}}]$, it always has $|\mathcal{K}_d| \leq Q$.

(ii) One can express the policies $\mathsf{P}_1, \ldots, \mathsf{P}_{Q_{\mathsf{Enc}}}$ as a new policy P^* that is of size independent of Q_{Enc}. Given any index $d \in [Q_{\mathsf{Enc}}]$, P_d can be efficiently derived from P^*, which is also independent of Q_{Enc}.

(iii) If $\mathcal{A}$ does not make any query to OEnc, it is hard to distinguish above two worlds, even when $\mathcal{A}$ colludes with more than Q registered users with authorized attributes.

The above definition aims at adaptive SIM security. If considering non-adaptive SIM security, $\mathcal{A}$ is not allowed to make a post-challenge corruption query on any pk_i^* such that $\mathsf{P}_d(\mathsf{att}_i^*) = 1$ for $d \in [Q_{\mathsf{Enc}}]$.

Similar to equivocal sReg-ABE, the above definition is also somewhat weak in that it forces the adversary to reveal all randomness and attributes for maliciously created public keys. In Section 6, we will employ NIZK to guarantee this condition.

5.2 From Equivocal Slotted Reg-ABE to 1-Bound Equivocal Slotted Reg-ABFE

For some circuit family $C : X \to Z$ of circuit size n, let $\mathsf{U}(\cdot,\cdot)$ be the universal circuit such that $\mathsf{U}(\mathsf{C}, x) = \mathsf{C}(x)$ for any circuit $\mathsf{C} \in C$ and input $x \in X$. Let $P : A \to \{0,1\}$ be a policy family that is computable in NC^1 circuits.

Assuming a garbled circuit scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$, and an equivocal sReg-ABE for NC^1 circuits $\mathsf{esRABE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, then our construction works as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C)$: Set $Q = 1$ as default. Then, it proceeds as follows:
  - Sample $\mathsf{crs}_{\mathsf{esRABE}} \leftarrow \mathsf{esRABE.Setup}(1^\lambda, 1^L, P)$.
  - For all $i \in [L]$, $w \in [n]$ and $b \in \{0,1\}$, it samples dummy public keys $\widehat{\mathsf{pk}}_{i,w,b} \leftarrow \mathsf{esRABE.Gen}(\mathsf{crs}_{w,b}, i, 1)$ (omit secret keys).

  Output the common reference string
  $$\mathsf{crs} = (\mathsf{crs}_{\mathsf{esRABE}}, \{\widehat{\mathsf{pk}}_{i,w,b}\}_{i\in[L],w\in[n],b\in\{0,1\}}).$$
- $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$: Sample $(\mathsf{pk}_{i,w,b}, \mathsf{sk}_{i,w,b}) \leftarrow \mathsf{esRABE.Gen}(\mathsf{crs}_{\mathsf{esRABE}}, i, \mathsf{att}_i)$ for all $w \in [n]$ and $b \in \{0,1\}$. Output the public key and secret key
  $$\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}},\ \mathsf{sk}_i = \{\mathsf{sk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}.$$
- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: Parse public key $\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$ and circuit $\mathsf{C}_i = (\mathsf{C}_i[1], \ldots, \mathsf{C}_i[n]) \in \{0,1\}^n$, then it proceeds as follows:
  - For all $i \in [L]$, $w \in [n]$ and $b \in \{0,1\}$, set
    $$\overline{\mathsf{pk}}_{i,w,b} := \begin{cases} \mathsf{pk}_{i,w,b}, & \text{when } \mathsf{C}_i[w] = b, \\ \widehat{\mathsf{pk}}_{i,w,b}, & \text{otherwise.} \end{cases}$$
  - Run $(\mathsf{mpk}_{w,b}, \{\mathsf{hsk}_{j,w,b}\}_{j\in[L]}) \leftarrow \mathsf{esRABE.Agg}(\mathsf{crs}_{\mathsf{esRABE}}, \{\overline{\mathsf{pk}}_{i,w,b}\}_{i\in[L]})$.

  Output the master public key and helper keys
  $$\mathsf{mpk} = \{\mathsf{mpk}_{w,b}\}_{w\in[n],b\in\{0,1\}},\ \mathsf{hsk}_j = (\mathsf{C}_j, \{\mathsf{hsk}_{j,w,\mathsf{C}_j[w]}\}_{w\in[n]}).$$
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x)$: Let $\mathsf{U}[x]$ be a universal circuit with x hard-wired. Run
  $$(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{U}[x]).$$
  For all $w \in [n], b \in \{0,1\}$, it computes $\mathsf{ct}_{w,b} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}, \mathsf{lab}_{w,b})$. Output the ciphertext $\mathsf{ct} = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$.
- $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct})$: If $\mathsf{P}(\mathsf{att}_{i^*}) = 1$, for each $w \in [n]$, it can recover
  $$\mathsf{lab}_{w,\mathsf{C}_{i^*}[w]} \leftarrow \mathsf{esRABE.Dec}(\mathsf{hsk}_{i^*,w,\mathsf{C}_{i^*}[w]}, \mathsf{sk}_{i^*,w,\mathsf{C}_{i^*}[w]}, \mathsf{att}_{i^*}, \mathsf{ct}_{w,\mathsf{C}_{i^*}[w]}).$$
  Output the evaluation result $z \leftarrow \mathsf{GC.Eval}(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,b_w}\}_{w\in[n]})$.

Compactness and Correctness. By the compactness of esRABE, we have $|\mathsf{mpk}| = 2n \cdot \mathsf{poly}(\lambda, P, \log L)$ and $|\mathsf{hsk}_j| = 2n \cdot \mathsf{poly}(\lambda, P, \log L)$, where n denotes the circuit size. Therefore, our construction meets the compactness requirement.

For correctness, the correctness of equivocal sReg-ABE guarantees that the user can decrypt a ciphertext $\mathsf{ct}_{w,b}$ under a master public key $\mathsf{mpk}_{w,b}$ as long as she has one secret key of the public key registered to $\mathsf{mpk}_{w,b}$, and we therefore see that the user i^* can only decrypt $\mathsf{ct}_{w,\mathsf{C}_{i^*}[w]}$ to learn $\mathsf{lab}_{w,\mathsf{C}_{i^*}[w]}$. Then, by the correctness of garbled circuits, the user can evaluate
$$\mathsf{GC.Eval}(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,\mathsf{C}_{i^*}[w]}\}_{w\in[n]}) = \mathsf{U}(\mathsf{C}, x) = \mathsf{C}(x).$$

Note that the dummy public keys $\widehat{\mathsf{pk}}_{i,w,b}$ are also the normal public keys of equivocal sReg-ABE, which are only employed to fill slot gaps and will not affect the decryption correctness.

Equivocal Public Key. We define the following algorithms:

- $\mathsf{ESetup}(1^\lambda, 1^L, 1^Q, P, C)$:
  - Sample $(\mathsf{crs}_{\mathsf{esRABE}}, \mathsf{ek}) \leftarrow \mathsf{esRABE.ESetup}(1^\lambda, 1^L, P)$.
  - For all $i \in [L]$, $w \in [n]$ and $b \in \{0,1\}$, it samples dummy public keys $\widehat{\mathsf{pk}}_{i,w,b} \leftarrow \mathsf{esRABE.Gen}(\mathsf{crs}_{w,b}, i, 1)$ (omit secret keys).

  Output the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{esRABE}}, \{\widehat{\mathsf{pk}}_{i,w,b}\}_{i\in[L],w\in[n],b\in\{0,1\}})$. and the equivocal key ek.
- $\mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek})$: Sample $(\mathsf{pk}_{i,w,b}, \mathsf{st}_{i,w,b}) \leftarrow \mathsf{esRABE.EGen}(\mathsf{crs}_{\mathsf{esRABE}}, i, \mathsf{ek})$ for all $w \in [n]$ and $b \in \{0,1\}$. Output the public key and internal state

$$\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}},\ \mathsf{st}_i = \{\mathsf{st}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}.$$

- $\mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek})$: Run $\mathsf{sk}_{i,w,b} \leftarrow \mathsf{esRABE.Equiv}(\mathsf{st}_{i,w,b}, \mathsf{att}_i, \mathsf{ek})$ for all $w \in [n]$ and $b \in \{0,1\}$. Output the secret key $\mathsf{sk}_i = \{\mathsf{sk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$.

We see that if the underlying esRABE scheme satisfies the property of equivocal public keys, then so does the above construction.

Theorem 7. *Assuming* GC *is secure and* esRABE *achieves policy-selective IND security, our construction achieves 1-bound non-adaptive SIM security with selective policies as defined in Section 5.1.*

Proof. Informally, the ciphertext ct_d for $d \in [Q_{\mathsf{Enc}}]$ consists of a garbled circuit $\tilde{\mathsf{U}}$ and $2n$ sub-ciphertexts that encrypt evaluation labels. Since a single malicious/corrupted user can only access half of the labels, we can randomize the other half of the labels via the security of esRABE and then simulate $\tilde{\mathsf{U}}$ and the remaining labels via the security of GC. Since the proof strategy is analogous to [ZCHZ24], we omit more details. Here, we define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

- $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, P, C)$: This algorithm is identical to ESetup (set td = ek).
- $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: This algorithm is identical to EGen. Namely, it samples

$$\forall w \in [n] \text{ and } b \in \{0,1\},\ (\mathsf{pk}_{i,w,b}, \mathsf{st}_{i,w,b}) \leftarrow \mathsf{esRABE.EGen}(\mathsf{crs}_{\mathsf{esRABE}}, i, \mathsf{ek}).$$

  Output $\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$ and $\mathsf{st}_i = \{\mathsf{st}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$.
- $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{att}_i, \mathsf{td}, (\bot, \bot))$: This algorithm is identical to Equiv. Namely, it samples

$$\forall w \in [n] \text{ and } b \in \{0,1\},\ \mathsf{sk}_{i,w,b} \leftarrow \mathsf{esRABE.Equiv}(\mathsf{st}_{i,w,b}, \mathsf{att}_i, \mathsf{ek}).$$

  Output $\mathsf{sk}_i = \{\mathsf{sk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$.
- $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_d, (f, z))$: Parse $\mathsf{mpk} = \{\mathsf{mpk}_{w,b}\}_{w\in[n],b\in\{0,1\}}$. For the d-th encryption query on the policy P_d and the message x_d, it considers two cases:
  - If there does not exist any corrupt/malicious slot, it must have $f = \bot$ and $z = \bot$, so the challenger generates

$$\forall w \in [n] \text{ and } b \in \{0,1\},\ \mathsf{ct}_{w,b} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{m}_{w,b})$$

    where $\mathsf{m}_{w,b} \leftarrow \{0,1\}^\lambda$.
  - Otherwise, we have $f = \mathsf{C}^*_j$ and $z = \mathsf{C}^*_j(x_d)$ for some corrupt/malicious slot $j \in [L]$ that is known before. Then, it samples

$$\forall w \in [n],\ \mathsf{lab}_{w,1-\mathsf{C}^*_j[w]} \leftarrow \{0,1\}^\lambda,$$
$$\left(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,\mathsf{C}^*_j[w]}\}_{w\in[n]}\right) \leftarrow \mathsf{GC}.\widetilde{\mathsf{Garble}}(1^\lambda, \mathsf{C}^*_j(x_d)).$$

    For $w \in [n]$ and $b \in \{0,1\}$, run $\mathsf{ct}_{w,b} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{lab}_{w,b})$.

Output the ciphertext $\mathsf{ct}_d = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$.

Theorem 8. *Assuming* esRABE *achieves policy-selective IND security, the construction with 1-bound non-adaptive SIM security with selective policies can be upgraded to achieve 1-bound adaptive SIM security with selective policies.*

This can be done by following standard non-committing techniques [DN00,GVW12,GGLW22], and we present the upgraded construction along with the proof in Appendix A.

5.3 From 1-Bound Security to Q-Bound Security

For some circuit family $C : X \to Z$ of circuit size n, let $t = \Theta(Q^2\lambda), m = \Theta(t), v = \Theta(Q\lambda), p_1 = \Theta(m), p_2 = \Theta(v), N = \Theta(Q^2\lambda p_1)$ and $S = S(Q\lambda p_2)$, where p_1 and p_2 are primes. For any circuit $\mathsf{C} \in C$ and set $\Delta \subseteq [S]$ of size v, we define a circuit family G in which each circuit $\mathsf{G} \in G$ with input $\widehat{x}$ can compute

$$\mathsf{G}(\widehat{x}) = \mathsf{RDMPC.Local}((\mathsf{C}, \Delta), \widehat{x}) \tag{4}$$

where (C, Δ) is hard-wired in G. Let $n' = \mathsf{poly}(\lambda, n, Q)$ be the circuit size of G.

Let $P : A \to \{0,1\}$ be an NC^1 circuit family. Here, we define a new NC^1 policy family $T : A \times \{0,1\}^{\lceil 2\log N\rceil} \to \{0,1\}$ in which each policy $\mathsf{T}_i \in T$ with input att and $j \in [2N]$ works as follows:

$$\mathsf{T}_i(\mathsf{att}, j) = 1 \iff \mathsf{P}(\mathsf{att}) = 1 \ \wedge \ i = j. \tag{5}$$

Assuming a reusable dynamic MPC scheme $\mathsf{RDMPC} = (\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode})$, and a 1-bound equivocal sReg-ABFE scheme (with adaptive security) $\mathsf{oesRABFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, our construction works as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C)$: It proceeds as follows:
  - Sample the common reference string
  $$\mathsf{crs}_{\mathsf{oesRABFE}} \leftarrow \mathsf{oesRABFE.Setup}(1^\lambda, 1^{LN}, 1, T, G).$$
  - Let $\mathsf{CHF} : \{0,1\}^* \to \mathbb{Z}_{p_1}^\lambda$ be a chameleon hash function family and $\mathsf{Hash} : \{0,1\}^* \to \mathbb{Z}_{p_2}^\lambda$ be a collision-resistant hash function family. Sample
  $$(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) \leftarrow \mathsf{CHF.Gen}(1^\lambda), \ \mathsf{hk}_2 \leftarrow \mathsf{Hash.Gen}(1^\lambda).$$
  - Define two injective maps $\rho_1 : [m] \times \mathbb{Z}_{p_1} \to [N]$ and $\rho_2 : [v] \times \mathbb{Z}_{p_2} \to [S]$.

  Output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$.
- $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$: It proceeds as follows:
  - Sample the randomness $r_i \leftarrow \{0,1\}^\lambda$ for the chameleon hash function.
  - For $i \in [L]$, it computes coefficients $(a_{i,0}, \ldots, a_{i,\lambda-1}) = \mathsf{CHF.H}(\mathsf{hk}_1, i; r_i)$ and hence defines a polynomial function of degree $\lambda - 1$:
  $$f_i(x) = a_{i,\lambda-1}x^{\lambda-1} + \cdots + a_{i,1}x + a_{i,0}.$$
  For each $j \in [m]$, evaluate the point $f_i(j)$ and then set $\gamma_{i,j} = \rho_1(j, f_i(j))$. Set $\Gamma_i = \{\gamma_{i,j}\}_{j\in[m]}$.
  - Let $i_k = (i-1)N + k$ for $k \in [N]$, then sample
  $$\forall k \in \Gamma_i, \ (\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{oesRABFE.Gen}(\mathsf{crs}_{\mathsf{oesRABFE}}, i_k, (\mathsf{att}_i, k)),$$
  $$\forall k \in [N] \setminus \Gamma_i, \ (\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{oesRABFE.Gen}(\mathsf{crs}_{\mathsf{oesRABFE}}, i_k, (\mathsf{att}_i, k+N)).$$

Output $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ and $\mathsf{sk}_i = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i})$.

– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: It proceeds as follows:
  - For $i \in [L]$, it computes coefficients $(c_{i,0}, \ldots, c_{i,v-1}) = \mathsf{Hash.H}(\mathsf{hk}_2, i||\mathsf{pk}_i)$ and hence defines a polynomial function of degree $\lambda - 1$:
$$g_i(x) = c_{i,\lambda-1}x^{\lambda-1} + \cdots + c_{i,1}x + c_{i,0}.$$
  For each $j \in [v]$, evaluate the point $g_i(j)$ and then set $\delta_{i,j} = \rho_2(j, g_i(j))$. Set $\Delta_i = \{\delta_{i,j}\}_{j\in[v]}$.
  - For each $i \in [L]$ and $k \in [N]$, define circuits $\mathsf{G}_{i,k} = (\mathsf{G}_{i,k}[1], \ldots, \mathsf{G}_{i,k}[n']) \in \{0,1\}^{n'}$ as in Eq. (4), where $\mathsf{G}_{i,k}$ hardwires $\widehat{\mathsf{C}}_{i,k} = (\mathsf{C}_i, \Delta_i)$ such that
$$(\widehat{\mathsf{C}}_{i,1}, \ldots, \widehat{\mathsf{C}}_{i,N}) = \mathsf{RDMPC.CktEnc}(1^\lambda, 1^Q, \mathsf{C}_i, \Delta_i).$$
  - Compute $(\mathsf{mpk}, \{\mathsf{hsk}_{j,k}\}_{j\in[L],k\in[N]}) = \mathsf{oesRABFE.Agg}(\mathsf{crs}_{\mathsf{oesRABFE}}, \{(\mathsf{pk}_{i,k}, \mathsf{G}_i)\}_{i\in[L],k\in[N]})$.

  Output the master key mpk and helper keys $\mathsf{hsk}_j = \{\mathsf{hsk}_{j,k}\}_{k\in[N]}$.

– $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x)$: It uses the policy P to define multiple policies $\mathsf{T}_1, \ldots, \mathsf{T}_N$ as in Eq. (5). Compute
$$(\widehat{x}_1, \ldots, \widehat{x}_N) \leftarrow \mathsf{RDMPC.InpEnc}(1^\lambda, 1^Q, x),$$
$$\forall k \in [N],\ \mathsf{ct}_k \leftarrow \mathsf{oesRABFE.Enc}(\mathsf{mpk}, \mathsf{T}_k, \widehat{x}_k).$$
  Output the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct})$: Compute
$$\forall k \in \Gamma_{i^*},\ \widehat{y}_k \leftarrow \mathsf{oesRABFE.Dec}(\mathsf{hsk}_{i^*,k}, \mathsf{sk}_{i^*,k}, (\mathsf{att}_{i^*}, k), \mathsf{ct}_k).$$
  Output the evaluation result $z = \mathsf{RDMPC.Decode}(\{\widehat{y}_k\}_{k\in\Gamma_{i^*}}, \Gamma_{i^*})$.

Compactness and Correctness. By the compactness of oesRABFE,
$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, n', T, \log LN) = \mathsf{poly}(\lambda, n, Q, P, \log L),$$
$$|\mathsf{hsk}_j| = N \cdot \mathsf{poly}(\lambda, n', T, \log LN) = \mathsf{poly}(\lambda, n, Q, P, \log L),$$
where N depends on Q, T is defined over P, and $n' = \mathsf{poly}(\lambda, n, Q)$. Thus, our construction satisfies the compactness requirement. The correctness follows the underlying oesRABFE and RDMPC schemes.

Equivocal Public Key. We define the following algorithms:

– $\mathsf{ESetup}(1^\lambda, 1^L, 1^Q, P, C)$: Sample
$$(\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{ek}_{\mathsf{oesRABFE}}) \leftarrow \mathsf{oesRABFE.ESetup}(1^\lambda, 1^{LN}, 1, T, G)$$
  Then, it samples $(\mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$ as in Setup. Output the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$ and equivocal key $\mathsf{ek} = \mathsf{ek}_{\mathsf{oesRABFE}}$.

– $\mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek})$: Sample
$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{oesRABFE.EGen}(\mathsf{crs}_{\mathsf{oesRABFE}}, i_k, \mathsf{ek}_{\mathsf{oesRABFE}})$$
  where $i_k = (i-1)N + k$ and $s_i = (i, \mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{hk}_1, \rho_1, \{\mathsf{pk}_{i,k}\}_{k\in[N]})$. Then, it samples the set Γ_i as in Gen. Output the public key $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ and internal state $\mathsf{st}_i = (\Gamma_i, \{\mathsf{st}_{i,k}\}_{k\in\Gamma_i})$.

– $\mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek})$: Run $\mathsf{sk}_{i,k} \leftarrow \mathsf{oesRABFE.Equiv}(\mathsf{st}_{i,k}, (\mathsf{att}_i, k), \mathsf{ek}_{\mathsf{oesRABFE}})$ for all $k \in \Gamma_i$. Output the secret key $\mathsf{sk}_i = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i})$.

We see that if the underlying oesRABFE scheme satisfies equivocal public keys, then the above construction naturally satisfies equivocal public keys.

Theorem 9. *Assuming* RDMPC *is secure and* oesRABFE *achieves 1-bound adaptive SIM security with selective policies, then our construction achieves Q-bound adaptive SIM security with selective policies.*

Proof. Recall that $\mathcal{A}$ will make Q_{Enc} encryption queries, where it specifies a policy P_d and a message x_d during the d-th query and obtains a ciphertext $\mathsf{ct}_d = (\mathsf{ct}_{d,1}, \ldots, \mathsf{ct}_{d,N})$. For each $d \in [Q_{\mathsf{Enc}}]$, suppose the adversary $\mathcal{A}$ colludes with Q corrupted/malicious users whose attributes satisfy P_d. We store the slot indices of these users in the set $\mathcal{K}_d \subseteq [L]$ and then define the following set

$$\mathcal{T}_d = \bigcup_{i \in \mathcal{K}_d, j \in \mathcal{K}_d \setminus \{j\}} (\Gamma_i^* \cap \Gamma_j^*). \tag{6}$$

where Γ_i^* is the pairwise set owned by the registered user under slot i. Let $\mathcal{K}_d'$ be the set that stores the corrupted/malicious slots that are known before the d-th encryption query, so we have $\mathcal{K}_d' \subseteq \mathcal{K}_d$. For each $d \in [Q_{\mathsf{Enc}}]$ and $k \in [N]$, the proof is organized as follows:

- If $k \notin \mathcal{T}_d$, it means there exists at most one set Γ_i^* such that $k \in \Gamma_i^*$ and $k \notin \Gamma_j^*$ for any $j \neq i$. In particular, $\mathsf{ct}_{d,k}$ can only be decrypted using the secret key sk_i. Thus, it can rely on the 1-bound security of oesRABFE;
- Otherwise, it means that $\mathcal{A}$ holds more than one secret keys that correctly decrypt $\mathsf{ct}_{d,k}$. In this way, 1-bound security would be lost, and the security would in turn rely on the underlying RDMPC scheme.

Then, we define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

- $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, P, C)$: This algorithm samples

$$(\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{td}_{\mathsf{oesRABFE}}) \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^{LN}, 1, T, G),$$
$$(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) \leftarrow \mathsf{CHF}.\mathsf{Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash}.\mathsf{Gen}(1^\lambda).$$

  Then, it defines injective maps (ρ_1, ρ_2) as in algorithm Setup. Output

$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2),\ \mathsf{td} = \mathsf{td}_{\mathsf{oesRABFE}}.$$

- $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: This algorithm samples a set Γ_i as in algorithm Gen and then computes

$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_{\mathsf{oesRABFE}}, i_k, \mathsf{td}_{\mathsf{oesRABFE}}),$$

  where $i_k = (i-1)N + k$. Output $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k \in [N]}$ and $\mathsf{st}_i = (\Gamma_i, \{\mathsf{st}_{i,k}\}_{k \in [N]})$.
- $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{att}_i, \mathsf{td}, (f, z))$: Parse $\mathsf{st}_i = (\Gamma_i, \{\mathsf{st}_{i,k}\}_{k \in [N]})$. It considers the following two cases:
  - If $f = \perp$ and $z = \perp$, this means that $\mathcal{A}$ makes this corruption query in the pre-challenge phase, or the public key pk_i is not registered into the system, or it satisfies $\mathsf{P}_d(\mathsf{att}) = 0$ for all $d \in [Q_{\mathsf{Enc}}]$. Then, it computes

$$\forall k \in [N],\ \mathsf{sk}_{i,k} \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, (\mathsf{att}_i, k), \mathsf{td}_{\mathsf{oesRABFE}}, (\perp, \perp)).$$

    Output the secret key $\mathsf{sk}_i = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k \in \Gamma_i})$. Note that if the public key pk_i is registered into the system, we can treat Γ_i as Γ_i^*.
  - Otherwise, $\mathcal{A}$ makes this corruption query in the post-challenge phase, and we have $f = \mathsf{C}_i^*$ and $z = \mathsf{C}_i^*(x_d)$ for some $d \in [Q_{\mathsf{Enc}}]$ such that $\mathsf{att}_i = \mathsf{att}_i^*$ and $\mathsf{P}_d(\mathsf{att}_i) = 1$. Here, we assume that the challenger has known sets $(\Gamma_i^*, \Delta_i^*, \mathcal{T}_d)$ during the challenger phase. Therefore, it can simulate

$$\{\widehat{y}_{i,k}\}_{k \in \Gamma_i^*} \leftarrow \mathsf{RDMPC}.\widetilde{\mathsf{Local}}\left(i, \mathsf{C}_i^*, \mathsf{C}_i^*(x_d)\right),$$
$$\forall k \in \Gamma_i^* \cap \mathcal{T}_d,\ \mathsf{sk}_{i,k} \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, (\mathsf{att}_i, k), \mathsf{td}_{\mathsf{oesRABFE}}, (\perp, \perp)),$$
$$\forall k \in \Gamma_i^* \setminus \mathcal{T}_d,\ \mathsf{sk}_{i,k} \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, (\mathsf{att}_i, k), \mathsf{td}_{\mathsf{oesRABFE}}, (\mathsf{G}_i^*, \widehat{y}_{i,k})).$$

    where G_i^* is defined over C_i^* and Δ_i^*. Output $\mathsf{sk}_i = (\Gamma_i^*, \{\mathsf{sk}_{i,k}\}_{k \in \Gamma_i^*})$.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_d, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x_d))\}_{j\in\mathcal{K}'_d})$: Here, $\mathcal{K}'_d$ just stores the corrupted/malicious slots that are known before the d-th encryption query. We assume that in the challenge phase, the challenger has known sets $\{\Gamma_j^*\}_{j\in\mathcal{K}_d}, \Delta_1^*, \ldots, \Delta_L^*$ (note that $\mathcal{K}_d \setminus \mathcal{K}'_d$ is still unknown). Therefore, the challenger can compute the set $\mathcal{T}_d$ as in Eq. (6). It then proceeds as follows:

  - It simulates the following input and local encodings

$$(\{\widehat{x}_k\}_{k\in\mathcal{T}_d}, \{\widehat{y}_{j,k}\}_{j\in\mathcal{K}'_d, k\in\Gamma_j^*}) \leftarrow \mathsf{RDMPC}.\widetilde{\mathsf{InpEnc}}(1^{|x_d|}, \mathcal{T}_d, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x_d))\}_{j\in\mathcal{K}'_d})$$

    where $\widehat{y}_{j,k} = \mathsf{RDMPC}.\mathsf{Local}(\widehat{\mathsf{C}}_{j,k}^*, \widehat{x}_k)$ for all $k \in \Gamma_j^* \cap \mathcal{T}_d$.
  - For all $k \in \mathcal{T}_d$, it generates the ciphertexts

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{oesRABFE}.\mathsf{Enc}(\mathsf{mpk}, \mathsf{T}_{d,k}, \widehat{x}_k).$$

    where $\mathsf{T}_{d,k}$ is defined over P_d and $k \in [N]$.
  - For all $k \notin \mathcal{T}_d$, if there is a set Γ_j such that $k \in \Gamma_j$ and $j \in \mathcal{K}'_d$, it can simulate

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{T}_{d,k}, (\mathsf{G}_{c_i}, \widehat{y}_{j,k})).$$

    Otherwise, we simulate $\mathsf{ct}_{d,k} \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{T}_{d,k}, (\bot, \bot))$.

  Finally, output the ciphertext $\mathsf{ct}_d = (\mathsf{ct}_{d,1}, \ldots, \mathsf{ct}_{d,N})$ as the response to d-th encryption query.

Game Sequence. We prove Theorem 9 via a sequence of games as follows:

– Game_0: Real Game.
– $\mathsf{Game}_{1,d,1}$: Same as $\mathsf{Game}_{1,d-1,4}$ except that in the challenge phase, for each $i = |\mathcal{K}'_d|+1, \ldots, Q$, it samples a new set Γ_i as in Gen .
– $\mathsf{Game}_{1,d,2}$: Same as $\mathsf{Game}_{1,d,1}$ except that for each $c_i \in \mathcal{K}_d \setminus \mathcal{K}'_d$, where $i = |\mathcal{K}'_d| + 1, \ldots, Q$, it sets $\Gamma_{c_i}^* = \Gamma_i$.
– $\mathsf{Game}_{1,d,3}$: Same as $\mathsf{Game}_{1,d,2}$ except that for the d-th encryption query, if $\left|\bigcup_{i\in\mathcal{K}_d, j\in\mathcal{K}_d\setminus\{i\}} (\Gamma_i^* \cap \Gamma_j^*)\right| \geq t$, it aborts and outputs $\bot$.
– $\mathsf{Game}_{1,d,4}$: Same as $\mathsf{Game}_{1,d,3}$ except that for the d-th encryption query and each $i \in \mathcal{K}_d$, if $\Delta_i^* \setminus \left(\bigcup_{j\in\mathcal{K}_d\setminus\{i\}} \Delta_j^*\right) = \emptyset$, it aborts and outputs $\bot$.
– Game_2: Same as $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4}$ except that for each $d \in [Q_{\mathsf{Enc}}]$ and $k \notin \mathcal{T}_d$, it simulates $(\mathsf{crs}, \{(\mathsf{pk}_j, \mathsf{sk}_j)\}_{j\in\mathcal{K}_d}, \mathsf{ct}_{d,k})$ as in $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.
– $\mathsf{Game}_{3,d}$: Same as $\mathsf{Game}_{4,d-1}$ except that for all $k \in \mathcal{T}_d$, it simulates the cipheretxts $\mathsf{ct}_{d,k}$ as in $\widetilde{\mathsf{Enc}}$.
– Game_4: Same as $\mathsf{Game}_{3,Q_{\mathsf{Enc}}}$ except that it completely replaces $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc})$ with $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.

We see that in the final game Game_4, each ciphertext ct_d and the corrupted secret keys $\{\mathsf{sk}_i\}_{i\in\mathcal{K}_d}$ can be simulated only using $\{(\mathsf{C}_i^*, \mathsf{C}_i^*(x_d))\}_{i\in\mathcal{K}_d}$. This is exactly the simulation world.

Lemma 4. *We claim that* $\mathsf{Game}_{1,d-1,4} \equiv \mathsf{Game}_{1,d,1}$.

Proof. The additional sampling operation in $\mathsf{Game}_{1,d,1}$ does not affect any distributions. Therefore, the lemma follows readily. □

Lemma 5. *By the chameleon property of* CHF*, we claim* $\mathsf{Game}_{1,d,1} \approx_s \mathsf{Game}_{1,d,2}$.

Proof. In $\mathsf{Game}_{1,d,1}$, the challenger samples

$$\forall i = |\mathcal{K}'_d| + 1, \ldots, Q,\ r'_{d,i} \leftarrow \{0,1\}^\lambda.$$

Then, it computes coefficients $(a'_{d,i,0}, \ldots, a'_{d,i,\lambda-1}) = \mathsf{CHF.H}(\mathsf{hk}_1, i; r'_{d,i})$ and hence defines a polynomial function $f'_{d,i}(x)$ of degree $\lambda - 1$. For each $i = |\mathcal{K}'_d| + 1, \ldots, Q$, it deterministically sets $\Gamma_i = \{\rho_1(j, f'_{d,i}(j))\}_{j\in[m]}$.

The above sampling operation happens in the challenge phase. In the post-challenge phase, the challenger can respond to all corruption queries (on registered keys and authorized attributes) as follows:

- $\mathsf{OCor}(c_i, \mathsf{pk}^*_{c_i}, \mathsf{att}^*_{c_i})$: Here, it requires $\mathsf{P}_d(\mathsf{att}^*_{c_i}) = 1$ and $c_i \in \mathcal{K}_d \setminus \mathcal{K}'_d$, where $i = |\mathcal{K}'_d| + 1, \ldots, Q$. Parse $\mathsf{pk}^*_{c_i} = (\{\mathsf{pk}^*_{c_i,k}\}_{k\in[N]}, \pi^*_{c_i})$, the challenger can find corresponding secret keys $\{\mathsf{sk}^*_{c_i,k}\}_{k\in[N]}$ by equivocating $\mathcal{D}_{c_i}[\mathsf{pk}^*_{c_i}]$. Assume that $\Gamma^*_{c_i}$ is generated based on hash input c_i and randomness $r_{d,i}$, the challenger can compute

  $$r_{d,i} \leftarrow \mathsf{CHF.H}^{-1}(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}, (i, r'_{d,i}), c_i).$$

  By the chameleon property, we have $\mathsf{CHF.H}(\mathsf{hk}_1, i; r'_{d,i}) = \mathsf{CHF.H}(\mathsf{hk}_1, c_i; r_{d,i})$, meaning that $\Gamma_i = \Gamma^*_{c_i}$. Finally, return $\mathsf{sk}^*_{c_i} = (\Gamma_{c_i}, \{\mathsf{sk}^*_{c_i,k}\}_{k\in\Gamma^*_{c_i}})$ to $\mathcal{A}$.

In summary, it is reasonable to set $\Gamma^*_{c_i} = \Gamma_i$ for all post-challenge corrupted slots $c_i \in \mathcal{K}_d \setminus \mathcal{K}'_d$. This readily proves the lemma. □

Lemma 6. *By the collision resistance of* CHF*, we claim* $\mathsf{Game}_{1,d,2} \approx_s \mathsf{Game}_{1,d,3}$.

Proof. The only difference between two games lies in the distribution of sets $\{\Gamma^*_j\}_{j\in\mathcal{K}_d}$. In $\mathsf{Game}_{2,d,2}$, the challenger will abort if $\left|\bigcup_{i\in\mathcal{K}_d, j\in\mathcal{K}_d\setminus\{j\}}(\Gamma^*_i \cap \Gamma^*_j)\right| \geq t$. This suffices to prove the small pairwise intersection:

$$\Pr\left[\left|\bigcup_{i\in\mathcal{K}_d, j\in\mathcal{K}_d\setminus\{j\}}(\Gamma^*_i \cap \Gamma^*_j)\right| \geq t\right] \leq \mathsf{negl}(\lambda). \tag{7}$$

Recall that the set Γ^*_i actually consists of a series of points $\{(k, f_i(k)\}_{k\in[m]}$ evaluated on a degree $\lambda - 1$ polynomial f_i. By the collision-resistance of the chameleon hash function family CHF, we know that $f_i = f_j$ for any $i \neq j$ occurs with negligible probability in λ. Thus, based on the Lagrange interpolation method, it must hold that there are at most $\lambda - 1$ intersection points between f_i and f_j for any $i \neq j$. Namely, for all $i \neq j$ we have

$$\Pr\left[\left|\Gamma^*_i \cap \Gamma^*_j\right| \geq \lambda\right] \leq \mathsf{negl}(\lambda).$$

Thus, the following equality holds except with negligible probability

$$\left|\bigcup_{i\in\mathcal{K}_d, j\in\mathcal{K}_d\setminus\{j\}}(\Gamma^*_i \cap \Gamma^*_j)\right| \leq Q^2 \cdot \max_{i\neq j}\left|\Gamma^*_i \cap \Gamma^*_j\right| < Q^2\lambda.$$

Recall that $t = \Theta(Q^2\lambda)$, so the argument in Eq. (7) must holds. The lemma follows readily. □

Lemma 7. *By the collision resistance of* Hash*, we claim* $\mathsf{Game}_{1,d,3} \approx_s \mathsf{Game}_{1,d,4}$.

Proof. The proof is analogous to Lemma 6. It suffices to prove the cover freeness:

$$\forall i \in \mathcal{K}_d,\ \Pr\left[\Delta^*_i \setminus \left(\bigcup_{j\in\mathcal{K}_d\setminus\{i\}} \Delta^*_j\right) = \emptyset\right] \leq \mathsf{negl}(\lambda). \tag{8}$$

Recall that the set Δ^*_i is also a series of points $\{(k, g_i(k)\}_{k\in[v]}$ evaluated on a degree $\lambda - 1$ polynomial g_i. By the collision-resistance of the hash function family Hash_2, we know that $g_i = g_j$ for any $i \neq j$ occurs with negligible probability in λ. Thus, based on the Lagrange interpolation method, it must hold that there are at most $\lambda - 1$ intersection points between g_i and g_j for any $i \neq j$. Namely,

$$\Pr\left[\left|\Delta^*_i \cap \Delta^*_j\right| \geq \lambda\right] \leq \mathsf{negl}(\lambda).$$

Thus, for all $i \in \mathcal{K}$, the following equality holds except with negligible probability

$$\left|\Delta_i^* \setminus \left(\bigcup_{j\in\mathcal{K}_d\setminus\{j\}} \Delta_j^*\right)\right| = \left|\Delta_i^* \setminus \bigcup_{j\in\mathcal{K}_d\setminus\{i\}} \left(\Delta_i^* \cap \Delta_j^*\right)\right| > \nu - Q\lambda.$$

Recall that $\nu = \Theta(Q\lambda)$, so Eq. (8) must holds. The lemma follows readily. □

Lemma 8. *By the adaptive SIM security with selective policies of* oesRABFE, *we claim that* $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4} \approx_c \mathsf{Game}_2$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4}$ from Game_2, we can build an algorithm $\mathcal{B}$ to break the security of oesRABFE. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, P, C)$ and a policy P^* from $\mathcal{A}$, where the policy P_d for any $d \in [Q_{\mathsf{Enc}}]$ can be efficiently derived from P^*. For each policy $\mathsf{P}_d \in P$, it can also efficiently define a new policy $\mathsf{T}_{d,i} \in T$ for any $i \in [2N]$ such that

$$\mathsf{T}_{d,i}(\mathsf{att}, j) = 1 \iff \mathsf{P}_d(\mathsf{att}) = 1 \ \wedge \ i = j. \tag{9}$$

Thus, $\mathsf{T}_{d,i}$ can be efficiently derived from P^*, which is independent of Q_{Enc}. Then, $\mathcal{B}$ proceeds as follows:

– *Setup phase:* It sends $(1^L, 1, T, G)$ and the policy P^* to the challenger of oesRABFE to obtain $\mathsf{crs}_{\mathsf{oesRABFE}}$. Then, it samples

$$(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) \leftarrow \mathsf{CHF.Gen}(1^\lambda), \ \mathsf{hk}_2 \leftarrow \mathsf{Hash.Gen}(1^\lambda)$$

and defines injective maps (ρ_1, ρ_2). Reply $\mathcal{A}$ with

$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2).$$

– *Pre-challenge phase:* Here, $\mathcal{A}$ can query the oracles below:
  - $\mathsf{OGen}(i)$: $\mathcal{B}$ samples the set Γ_i as in algorithm Gen. For each $k \in [N]$, it queries the oracle oesRABFE.OGen on input i_k to obtain the public key $\mathsf{pk}_{i,k}$. Return $\mathsf{pk} = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ to $\mathcal{A}$.
  - $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: Parse $\mathsf{pk} = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$. To respond this query, for each $k \in \Gamma_i$, it queries the oracle oesRABFE.OCor on input $(i_k, \mathsf{pk}_{i,k}, (\mathsf{att}, k))$ to obtain the secret key $\mathsf{sk}_{i,k}$. Return $\mathsf{sk} = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i})$ to $\mathcal{A}$.

– *Challenge phase:* $\mathcal{A}$ specifies public keys $\{\mathsf{pk}_i^*\}_{i\in[L]}$, attributes $\{\mathsf{att}_i^*\}_{i\in[L]}$, and circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$, where $\mathsf{pk}_i^* = \{\mathsf{pk}_{i,k}^*\}_{k\in[N]}$. For each malicious slot j, $\mathcal{A}$ will submit the pair of randomness and attributes (r_j, att_j^*) for generating pk_j^*. Therefore, $\mathcal{B}$ can acquire the set Γ_j^* based on the randomness r_j. As in $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4}$, $\mathcal{B}$ can now compute the sets $\{\Gamma_j^*\}_{j\in\mathcal{K}_d}, \Delta_1^*, \ldots, \Delta_L^*$ and hence learn the set $\mathcal{T}_d$ for $d \in [Q_{\mathsf{Enc}}]$.
  Next, $\mathcal{B}$ sends challenge information $\{(\mathsf{pk}_{i,k}^*, \mathsf{att}_{i,k}^*, \mathsf{G}_i^*)\}_{i\in[L],k\in[N]}$ to the challenger. Here, the circuit G_i^* is defined over C_i^* and Δ_i^*. Besides, for each non-malicious slot i, it sets the attribute $\mathsf{att}_{i,k}^* = (\mathsf{att}_i^*, k)$ if $k \in [N]$; for each malicious slot i, it instead sets $\mathsf{att}_{i,k}^* = (\mathsf{att}_i^*, k)$ if $k \in \Gamma_i^*$ and $\mathsf{att}_{i,k}^* = (\mathsf{att}_i^*, k+N)$ otherwise. We note that even when $\mathcal{A}$ corrupts slot i in the post-challenge phase, the attribute $\mathsf{att}_{i,k}^*$ for $k \in [N] \setminus \Gamma_i^*$ is still hidden. Thus, it is reasonable to set $\mathsf{att}_{i,k}^* = (\mathsf{att}_i^*, k)$ for each non-malicious slot i and $k \in [N] \setminus \Gamma_i^*$. Next, $\mathcal{B}$ computes the master public key mpk as in algorithm Agg.

– *Post-challenge phase:*
  - $\mathsf{OEnc}(\mathsf{P}_d, x_d)$: It requires $d = 1, \ldots, Q_{\mathsf{Enc}}$. $\mathcal{B}$ derives the following input encodings

$$(\widehat{x}_{d,1}, \ldots, \widehat{x}_{d,N}) \leftarrow \mathsf{RDMPC.InpEnc}(1^\lambda, 1^Q, x_d).$$

  Next, $\mathcal{B}$ generates a ciphertext $\mathsf{ct}_d = (\mathsf{ct}_{d,1}, \ldots, \mathsf{ct}_{d,N})$ as follows:
  * For each $k \in \mathcal{T}_d$, $\mathcal{B}$ directly computes

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{oesRABFE.Enc}(\mathsf{mpk}, \mathsf{T}_{d,k}, \widehat{x}_{d,k}).$$

  where the policy $\mathsf{T}_{d,k}$ is as defined in Eq. (9).

* For each $k \notin \mathcal{T}_d$, $\mathcal{B}$ queries the oracle oesRABFE.OEnc on input $(\mathsf{T}_{d,k}, \widehat{x}_k)$ to obtain the ciphertext $\mathsf{ct}_{d,k}$.

- $\mathsf{OCor}(i, \mathsf{pk}_i)$: If $\mathsf{pk}_i = \mathsf{pk}_i^*$ and $\mathsf{P}_d(\mathsf{att}_i^*) = 1$ for some $d \in [Q_{\mathsf{Enc}}]$, as in $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4}$, $\mathcal{B}$ can exploit the trapdoor $\mathsf{td}_{\mathsf{CHF}}$ to acquire the randomness that is used to generate the corrupted set Γ_i^*. Then, for each $k \in \Gamma_i^*$, it queries the oracle oesRABFE.OCor on input $(i_k, \mathsf{pk}_{i,k}^*, (\mathsf{att}_i^*, k))$ to obtain the secret key $\mathsf{sk}_{i,k}$. Return $\mathsf{sk} = (\Gamma_i^*, \{\mathsf{sk}_{i,k}\}_{k \in \Gamma_i^*})$ to $\mathcal{A}$. (Note that if $\mathcal{A}$ makes corruption queries on non-registered/non-authorized keys, $\mathcal{B}$ can respond as in the pre-challenge phase.)

We remark that all above encryption/corruption queries are admissible for the 1-bound security of oesRABFE, because

- For $k \in \mathcal{T}_d$, $\mathcal{B}$ does not make any encryption queries.
- For $k \notin \mathcal{T}_d$, there is at most a single corrupted/malicious key $\mathsf{pk}_{j,k}^*$ such that $\mathsf{T}_{d,k}(\mathsf{att}_j^*, k) = 1$.

Therefore, we see that in the real world of oesRABFE, $\mathcal{B}$ simulates $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4}$; in the ideal world of oesRABFE, $\mathcal{B}$ exactly simulates Game_2. The lemma follows readily. □

Lemma 9. *By the security of* RDMPC*, we claim* $\mathsf{Game}_{3,d-1} \approx_c \mathsf{Game}_{3,d}$ *for each* $d \in [Q_{\mathsf{Enc}}]$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{3,d-1}$ from $\mathsf{Game}_{3,d}$, we can build an algorithm $\mathcal{B}$ to break the security of RDMPC. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, C)$ from $\mathcal{A}$. Then, $\mathcal{B}$ proceeds as follows:

– *Setup phase:* $\mathcal{B}$ samples

$$(\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{td}_{\mathsf{oesRABFE}}) \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^{LN}, 1, T, G),$$
$$(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) \leftarrow \mathsf{CHF.Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash.Gen}(1^\lambda).$$

Then, it defines injective maps (ρ_1, ρ_2) as in algorithm Setup. It replies $\mathcal{A}$ with the common reference string

$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2).$$

– *Pre-challenge phase:* Here, $\mathcal{A}$ can query the oracles below:

- $\mathsf{OGen}(i)$: $\mathcal{B}$ samples the set Γ_i as in algorithm Gen and then computes

$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_{\mathsf{oesRABFE}}, i_k, \mathsf{td}_{\mathsf{oesRABFE}}).$$

Return $\mathsf{pk} = \{\mathsf{pk}_{i,k}\}_{k \in [N]}$ to $\mathcal{A}$ and update $\mathcal{D}_i[\mathsf{pk}] = (\Gamma_i, \{\mathsf{st}_{i,k}\}_{k \in [N]})$.

- $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: With access to $\mathcal{D}_i[\mathsf{pk}]$, $\mathcal{B}$ can compute

$$\forall k \in \Gamma_i,\ \mathsf{sk}_{i,k} \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{att}, \mathsf{td}_{\mathsf{oesRABFE}}, (\perp, \perp)).$$

Return $\mathsf{sk} = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k \in [N]})$ to $\mathcal{A}$.

– *Challenge phase:* $\mathcal{B}$ receives challenge public keys $\{\mathsf{pk}_i^*\}_{i \in [L]}$ and circuits $\{\mathsf{C}_i^*\}_{i \in [L]}$. As in $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4}$, $\mathcal{B}$ can now compute $\{\Gamma_j^*\}_{j \in \mathcal{K}_d}, \Delta_1^*, \ldots, \Delta_L^*$ and hence know the set $\mathcal{T}_d$ for any $d \in [Q_{\mathsf{Enc}}]$. It computes the master public key mpk as in algorithm Agg. Then, $\mathcal{B}$ computes the master public key mpk as in algorithm Agg.

– *Post-challenge phase:*

- $\mathsf{OEnc}(\mathsf{P}_i, x_i)$: $\mathcal{B}$ proceeds as follows:
  * For $i < d$, it generates the ciphertext ct_d as in $\widetilde{\mathsf{Enc}}$.
  * For $i > d$, it generates the ciphertext ct_d as in Enc.

* For $i = d$, $\mathcal{B}$ invokes the challenger of RDMPC by specifying parameters (Q, N, m, t), corruption set $\mathcal{T}_d$, challenge message x_d, sets $\{\Gamma_j^*\}_{j\in\mathcal{K}_d}$, and $\{(\mathsf{C}_j, \Delta_j^*)\}_{j\in\mathcal{K}'_d}$, where $\mathcal{K}'_d \subseteq \mathcal{K}_d$. The challenger replies $\mathcal{B}$ with the following encodings

$$\{\widehat{x}_k\}_{k\in\mathcal{T}} \text{ and } \{\widehat{y}_{j,k}\}_{j\in\mathcal{K}'_d, k\in\Gamma_j^*}.$$

Then, for all $k \in \mathcal{T}$, $\mathcal{B}$ directly computes

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{oesRABFE.Enc}(\mathsf{mpk}, \mathsf{T}_{d,k}, \widehat{x}_k).$$

For all $k \notin \mathcal{T}$, $\mathcal{B}$ acts as in Game_2. Namely, if there exists some j such that $k \in \Gamma_j$ and $j \in \mathcal{K}'_d$, then $\mathcal{B}$ simulates

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{oesRABFE.}\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{T}_{d,k}, (\mathsf{G}_j^*, \widehat{y}_{j,k})).$$

otherwise, it simulates $\mathsf{ct}_{d,k} \leftarrow \mathsf{oesRABFE.}\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{T}_{d,k}, (\perp, \perp))$. Return $\mathsf{ct}_d = (\mathsf{ct}_{d,1}, \ldots, \mathsf{ct}_{d,N})$ to $\mathcal{A}$.

- $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: Parse $\mathcal{D}_i[\mathsf{pk}] = (\Gamma_i, \{\mathsf{st}_{i,k}\}_{k\in[N]})$. It considers the following two cases:
  * If $\mathsf{pk} \neq \mathsf{pk}_i^*$ or $\mathsf{P}_j(\mathsf{att}_i) = 0$ for all $j \in [Q_{\mathsf{Enc}}]$, it computes

$$\forall k \in \Gamma_i,\ \mathsf{sk}_{j,k} \leftarrow \mathsf{oesRABFE.}\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, (\mathsf{att}, k), \mathsf{td}_{\mathsf{oesRABFE}}, (\perp, \perp)).$$

  Return $\mathsf{sk} = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_{c_i}})$ to $\mathcal{A}$.
  * Otherwise, it must have $\mathsf{pk} = \mathsf{pk}_i^*$ and $\mathsf{P}_d(\mathsf{att}) = 1$ for some $d \in [Q_{\mathsf{Enc}}]$. As in $\mathsf{Game}_{1,Q_{\mathsf{Enc}},4}$, $\mathcal{B}$ can exploit the trapdoor $\mathsf{td}_{\mathsf{CHF}}$ to acquire the randomness that is used to generate the corrupted set Γ_i^*. Then, $\mathcal{B}$ queries the oracle RDMPC.OCor on input $(\mathsf{C}_i^*, \Delta_i^*)$ to obtain the encodings $\{\widehat{y}_{i,k}\}_{k\in\Gamma_i^*}$. It can simulate

$$\forall k \in \Gamma_i^* \cap \mathcal{T},\ \mathsf{sk}_{i,k} \leftarrow \mathsf{oesRABFE.}\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, (\mathsf{att}_i, k), \mathsf{td}_{\mathsf{oesRABFE}}, (\perp, \perp)),$$
$$\forall k \in \Gamma_i^* \setminus \mathcal{T},\ \mathsf{sk}_{i,k} \leftarrow \mathsf{oesRABFE.}\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, (\mathsf{att}_i, k), \mathsf{td}_{\mathsf{oesRABFE}}, (\mathsf{G}_i^*, \widehat{y}_{i,k})).$$

  where the circuit G_i^* is defined over C_i^* and Δ_i^*. Return the secret key $\mathsf{sk} = (\Gamma_i^*, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i^*})$ to $\mathcal{A}$.

We see that in the real world of RDMPC, $\mathcal{B}$ simulates $\mathsf{Game}_{3,d-1}$; in the ideal world of RDMPC, $\mathcal{B}$ simulates $\mathsf{Game}_{3,d}$. Therefore, the lemma follows readily. □

Lemma 10. *We claim* $\mathsf{Game}_{3,Q_{\mathsf{Enc}}} \equiv \mathsf{Game}_4$.

Proof. In $\mathsf{Game}_{3,Q_{\mathsf{Enc}}}$, we see that the scheme has completely employed the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$, which is the same as Game_4. The lemma follows readily. □

6 From Equivocal Slotted Reg-ABFE to Weakly Optimal Slotted Reg-FE

In this section, we formalize the notion of weakly optimal slotted Reg-FE. We provide a generic construction that achieves the strongest adaptive SIM security, assuming the existence of Q-bound equivocal sReg-ABFE for $\mathsf{NC}^1 \circ \mathsf{P/Poly}$.

6.1 Definition

A weakly optimal sReg-FE for circuit family $C : X \to Z$ is defined as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, C) \to \mathsf{crs}$. It takes as input the security parameter 1^λ, the upper bound 1^L on the number of slots, the collusion bound 1^Q, and the description of circuit family C, then outputs a common reference string crs.

- $\mathsf{Gen}(\mathsf{crs}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. It takes as input the common reference string crs and a slot index $i \in [L]$, then outputs a public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
- $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) \to 0/1$. It takes as input the common reference string crs, a slot index $i \in [L]$, and a public key pk_i, then outputs a bit indicating whether pk_i is valid.
- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$. It takes as input crs and a series of public keys and circuits $\{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}$, then outputs the master public key mpk and helper keys $\{\mathsf{hsk}_j\}_{j\in[L]}$. This algorithm is deterministic.
- $\mathsf{Enc}(\mathsf{mpk}, x) \to \mathsf{ct}$. It takes as input the master public key mpk and a message $x \in X$, then outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct}) \to z/\perp$. It takes as input a helper key hsk_{i^*}, a secret key sk_{i^*}, and a ciphertext ct, then outputs $z \in Z$ or an empty symbol $\perp$.

The above scheme is required to satisfy:

- **Weakly Optimal Compactness.** For all $\lambda, L \in \mathbb{N}$, all $Q \le L$, all C, and all $i \in [L]$, it requires

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, C, \log Q, \log L) \text{ and } |\mathsf{hsk}_i| = \mathsf{poly}(\lambda, C, \log Q, \log L).$$

  Besides, the overheads of $(\mathsf{Setup}, \mathsf{Gen})$ are independent of Q.
- **Completeness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all C, and all $i \in [L]$,

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, C) \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs}, i) \end{array}\right] = 1.$$

- **Correctness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all C, all $i^* \in [L]$, all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$, all $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i^*)$, all $\{\mathsf{pk}_i\}_{i\in[L]\setminus\{i^*\}}$ such that $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$, for all messages $x \in X$ and circuits $\mathsf{C}_1, \ldots, \mathsf{C}_L \in C$, and computing $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$, we have

$$\Pr\left[\mathsf{Dec}\left(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{Enc}(\mathsf{mpk}, x)\right) = \mathsf{C}_{i^*}(x)\right] = 1.$$

- **(Non-)Adaptive SIM Security.** For all efficient adversaries $\mathcal{A}$, there exists a simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ such that it is hard for $\mathcal{A}$ to distinguish the following two worlds:
  - *Setup phase:* Initially, $\mathcal{A}$ specifies the input $(1^L, 1^Q, C)$. In the real world (left hand) or the ideal world (right hand), the challenger samples

$$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, C) \;\text{ or }\; (\mathsf{crs}, \mathsf{td}) \leftarrow \widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C).$$

    Return crs to $\mathcal{A}$. Here, td is the trapdoor of crs.
  - *Pre-challenge phase:* The challenger initially sets $\mathcal{K} = \emptyset$, $\{\mathcal{D}_i = \emptyset\}_{i\in[L]}$, and $C = \emptyset$. $\mathcal{A}$ can make key generation/corruption queries to the following oracles:
    * $\mathsf{OGen}(i)$: in the real world (left hand) or the ideal world (right hand), the challenger runs

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i) \;\text{ or }\; (\mathsf{pk}, \mathsf{st}) \leftarrow \widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td}).$$

      Then, the challenger sets $\mathcal{D}_i[\mathsf{pk}] = \mathsf{sk}$ or $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}$, where st denotes the internal state of pk. Return pk to $\mathcal{A}$.
    * $\mathsf{OCor}(i, \mathsf{pk})$: The challenger returns the secret key sk to $\mathcal{A}$, where

$$\mathsf{sk} = \mathcal{D}_i[\mathsf{pk}] \;\text{ or }\; \mathsf{sk} \leftarrow \widetilde{\mathsf{Gen}}_2(\mathcal{D}_i[\mathsf{pk}], \mathsf{td}, (\perp, \perp)).$$

      Then, the challenger updates $C = C \cup \{(i, \mathsf{pk})\}$.

- *Challenge phase:* In this phase, $\mathcal{A}$ specifies L pairs of public key and circuit $\{(\mathsf{pk}_i^*, \mathsf{C}_i^*)\}_{i\in[L]}$ together with a challenge message x^*. It computes $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i^*, \mathsf{C}_i^*)\}_{i\in[L]})$. For each slot $i \in [L]$, it must fall in one of the following three cases:

$$\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*) \notin C \qquad \text{(honest slot)}$$
$$\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*) \in C \qquad \text{(corrupted slot)}$$
$$\mathcal{D}_i[\mathsf{pk}_i^*] = \perp \qquad \text{(malicious slot)}$$

  Record all corrupted/malicious slot indices in the set $\mathcal{K}$, where $|\mathcal{K}| = q \leq Q$. Next, in the real world (left hand) or the ideal world (right hand), it replies $\mathcal{A}$ with a challenge ciphertext ct^* by either sampling

$$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^*) \ \text{ or } \ \mathsf{ct}^* \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}}).$$

- *Post-challenge phase:* $\mathcal{A}$ continues to make corruption queries:
  * $\mathsf{OCor}(i, \mathsf{pk})$: If pk is not registered into the system, it acts as in the pre-challenge phase; otherwise, if $|\mathcal{K}| = Q$, $\mathcal{A}$ is not allowed to corrupt any registered key in this phase. In the real world (left hand) or the ideal world (right hand), it computes

$$\mathsf{sk} = \mathcal{D}_i[\mathsf{pk}] \ \text{ or } \ \mathsf{sk} \leftarrow \widetilde{\mathsf{Gen}}_2(\mathcal{D}_i[\mathsf{pk}], \mathsf{td}, (\mathsf{C}_i^*, \mathsf{C}_i(x^*))).$$

    Then, the challenger returns sk to $\mathcal{A}$ and updates $\mathcal{K} = \mathcal{K} \cup \{i\}$
- *Output phase:* $\mathcal{A}$ outputs a bit to indicate the real/ideal world.

If considering non-adaptive security, it requires $q = Q$.

6.2 Construction

Assuming a weak pseudorandom function family $\mathsf{wPRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to [Q]$ computable in NC^1, we define a policy family $P : \{0,1\}^\lambda \times [Q] \to \{0,1\}$. In particular, the policy $\mathsf{P} \in P$ with input $a \in \{0,1\}^\lambda$ and $\widehat{\mathsf{tag}} \in [Q]$ works as follows:

$$\mathsf{P}(a, \widehat{\mathsf{tag}}) = 1 \iff \mathsf{wPRF.Eval}(\mathsf{k}, a) \oplus \widehat{\mathsf{tag}} = \mathsf{tag} \tag{10}$$

where P hardwires a weak PRF evaluation key $\mathsf{k} \in \{0,1\}^\lambda$ and a tag $\mathsf{tag} \in [Q]$. Apparently, P is computable in NC^1, and its size is bounded by $\mathsf{poly}(\lambda, \log Q, \log L)$.

Assume an equivocal sReg-ABFE scheme $\mathsf{esRABFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$ and a simulation-sound extractable non-interactive zero-knowledge argument $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{SPrv})$, our construction works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$: Sample $a_i \leftarrow \{0,1\}^\lambda$ for each $i \in [L]$ and

$$\mathsf{crs}_{\mathsf{esRABFE}} \leftarrow \mathsf{esRABFE.Setup}(1^\lambda, 1^L, 1^{2\lambda}, P, C),$$
$$(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda).$$

  Output the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{crs}_{\mathsf{NIZK}}, \{a_i\}_{i\in[L]})$.

– $\mathsf{Gen}(\mathsf{crs}, i)$: Sample $\widehat{\mathsf{tag}}_i \leftarrow [Q]$ and randomness r_i. Run

$$(\mathsf{pk}_i', \mathsf{sk}_i') \leftarrow \mathsf{esRABFE.Gen}(\mathsf{crs}_{\mathsf{esRABFE}}, i, (a_i, \widehat{\mathsf{tag}}_i); r_i),$$
$$\pi_i \leftarrow \mathsf{NIZK.Prv}(\mathsf{crs}_{\mathsf{NIZK}}, s_i, w_i),$$

  where r_i is the randomness for generating $(\mathsf{pk}_i', \mathsf{sk}_i')$. We define the statement $s_i = (i, a_i, \mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{pk}_i')$ and the witness $w_i = (\mathsf{sk}_i', \widehat{\mathsf{tag}}_i, r_i)$ to prove the well-formness of pk_i'. Output $\mathsf{pk}_i = (\mathsf{pk}_i', \pi_i)$ and $\mathsf{sk}_i = (\mathsf{sk}_i', \widehat{\mathsf{tag}}_i)$.

– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i)$: It checks whether $\mathsf{NIZK.Ver}(\mathsf{crs}_{\mathsf{NIZK}}, s_i, \pi_i) \stackrel{?}{=} 1$. If not, output 0; otherwise, output 1.
– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: Output

$$(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) \leftarrow \mathsf{esRABFE.Agg}(\mathsf{crs}_{\mathsf{esRABFE}}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}).$$

– $\mathsf{Enc}(\mathsf{mpk}, x)$: Sample $\mathsf{k} \leftarrow \mathsf{wPRF.Gen}(1^\lambda)$ and generate

$$\forall \mathsf{tag} \in [Q],\ \mathsf{ct}_{\mathsf{tag}} \leftarrow \mathsf{esRABFE.Enc}(\mathsf{mpk}, \mathsf{P}_{\mathsf{k},\mathsf{tag}}, x)$$

where the policy $\mathsf{P}_{\mathsf{k},\mathsf{tag}}$ is as defined in Eq. (10) which takes $(\mathsf{k}, \mathsf{tag})$ as constants. Output the ciphertext

$$\mathsf{ct} = (\mathsf{k}, \mathsf{ct}_1, \ldots, \mathsf{ct}_Q).$$

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct})$: Compute $\mathsf{tag}_{i^*} = \mathsf{wPRF.Eval}(\mathsf{k}, a_{i^*}) \oplus \widehat{\mathsf{tag}}_{i^*}$, and output the evaluation result

$$z \leftarrow \mathsf{esRABFE.Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}'_{i^*}, (a_{i^*}, \widehat{\mathsf{tag}}_{i^*}), \mathsf{ct}_{\mathsf{tag}_{i^*}}).$$

Compactness, Completeness and Correctness. For compactness, since we set the collusion bound as $1^{2\lambda}$ and the policy P is of size $\mathrm{poly}(\lambda, \log Q, \log L)$, then $|\mathsf{mpk}| = \mathrm{poly}(\lambda, C, \log Q, \log L)$ and $|\mathsf{hsk}_j| = \mathrm{poly}(\lambda, C, \log Q, \log L)$. The completeness and correctness follow the underlying NIZK protocol and esRABFE scheme, respectively.

6.3 Proof

Theorem 10. *Assuming* esRABFE *achieves Q-bound adaptive SIM security with selective policies, then our construction achieves Q-bound adaptive SIM security.*

Proof. Let $\mathcal{K}'$ and $\mathcal{K}$ be the sets of the corrupted/malicious slot indices that are found in the challenge phase and the final output phase, respectively. For each $\mathsf{tag}^* \in [Q]$, let $\mathsf{cnt}_{\mathsf{tag}^*}$ be the accumulated number of tags such that $\mathsf{tag}^* = \mathsf{wPRF.Eval}(\mathsf{k}, a_j) \oplus \widehat{\mathsf{tag}}^*_j$ for any $j \in \mathcal{K}$, where $\widehat{\mathsf{tag}}^*_j$ is the tag associated with the registered public key pk^*_j. Here, we define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: Sample $a_i \leftarrow \{0,1\}^\lambda$ for each $i \in [L]$ and

$$\begin{aligned} \mathsf{k} &\leftarrow \mathsf{wPRF.Gen}(1^\lambda), \\ (\mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{td}_{\mathsf{esRABFE}}) &\leftarrow \mathsf{esRABFE.}\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^{2\lambda}, P, C), \\ (\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}) &\leftarrow \mathsf{NIZK.EGen}(1^\lambda). \end{aligned}$$

Output the common reference and trapdoor

$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{crs}_{\mathsf{NIZK}}, \{a_i\}_{i\in[L]}),\ \mathsf{td} = (\mathsf{td}_{\mathsf{esRABFE}}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}})$$

– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: Sample $\widehat{\mathsf{tag}}_i \leftarrow [Q]$ and

$$\begin{aligned} (\mathsf{pk}'_i, \mathsf{st}'_i) &\leftarrow \mathsf{esRABFE.}\widetilde{\mathsf{Gen}}_1(\widetilde{\mathsf{crs}}_{\mathsf{esRABFE}}, i, \mathsf{td}_{\mathsf{esRABFE}}), \\ \pi_i &\leftarrow \mathsf{NIZK.SPrv}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, s_i) \end{aligned}$$

where we define the statement $s_i = (i, \widetilde{\mathsf{crs}}_{\mathsf{esRABFE}}, \mathsf{pk}'_i)$. Output the public key and internal state

$$\mathsf{pk}_i = (\mathsf{pk}'_i, \pi_i),\ \mathsf{st}_i = (\mathsf{st}'_i, \widehat{\mathsf{tag}}_i).$$

– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{td}, (f, z))$: Parse $\mathsf{st}_i = (\mathsf{st}'_i, \widehat{\mathsf{tag}}_i)$. Sample

$$\mathsf{sk}'_i \leftarrow \mathsf{esRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}'_i, (a_i, \widehat{\mathsf{tag}}_i), \mathsf{td}_{\mathsf{esRABFE}}, (f, z)).$$

Output the secret key $\mathsf{sk}_i = (\mathsf{sk}'_i, \widehat{\mathsf{tag}}_i)$.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}^*_j, \mathsf{C}^*_j(x^*))\}_{j\in\mathcal{K}'})$: Here, x^* is the challenge message. We assume that in the challenge phase, the challenger knows the tags $\{\widehat{\mathsf{tag}}^*_j\}_{j\in\mathcal{K}'}$. For each $j \in \mathcal{K}'$, it computes $\mathsf{tag}^* = \mathsf{wPRF.Eval}(\mathsf{k}, a_j) \oplus \widehat{\mathsf{tag}}^*_j$ and sets $Q'_{\mathsf{tag}^*} = Q'_{\mathsf{tag}^*} \cup \{(\mathsf{C}^*_j, \mathsf{C}^*_j(x^*))\}$. Then, it simulates

$$\forall \mathsf{tag}^* \in [Q],\ \mathsf{ct}_{\mathsf{tag}^*} \leftarrow \mathsf{esRABFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_{\mathsf{k},\mathsf{tag}^*}, Q'_{\mathsf{tag}^*})$$

Output the challenge ciphertext $\mathsf{ct} = (\mathsf{k}, \mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$.

Game Sequence. We prove Theorem 10 via a sequence of games as follows:

– Game_0: Real Game.
– Game_1: Same as Game_0 except that for each query to oracle OGen it simulates π_i as in $\widetilde{\mathsf{Gen}}$.
– Game_2: Same as Game_1 except that it employs NIZK.ESetup, and in the challenge phase, it extracts the witness $w_j \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}, s_j, \pi^*_j)$ for each malicious slot j.
– Game_3: Same as Game_2 except that it generates $(\mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{pk}'_i, \mathsf{sk}'_i)$ via (ESetup, EGen, Equiv) of esRABFE.
– Game_4: Same as Game_3 except that it samples k as in $\widetilde{\mathsf{Setup}}$.
– Game_5: Same as Game_4 except that for each $\mathsf{tag}^* \in [Q]$, if $\mathsf{cnt}_{\mathsf{tag}^*} > 2\lambda$, it aborts and outputs $\perp$.
– Game_6: Same as Game_5 except that it completely replaces (Setup, Gen, Enc) with $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.

We see that in the final game Game_6, the challenge ciphertext ct^* and the corrupted secret keys $\{\mathsf{sk}_i\}_{i\in\mathcal{K}}$ can be simulated only using $\{(\mathsf{C}^*_i, \mathsf{C}^*_i(x^*))\}_{i\in\mathcal{K}}$. This is exactly the simulation world.

Lemma 11. *By the zero-knowledge property of* NIZK*, we claim* $\mathsf{Game}_0 \approx_c \mathsf{Game}_1$.

Proof. Since the main difference between Game_0 and Game_1 is that when $\mathcal{A}$ makes key generation queries, the challenger generates the proof π_i via either NIZK.Prv or NIZK.SPrv. Therefore, we can reduce the indistinguishability between Game_0 and Game_1 to the zero-knowledge property of the NIZK scheme. Since the reduction is standard, we omit details here. □

Lemma 12. *By the simulation-sound extractability of* NIZK*, we claim* $\mathsf{Game}_1 \approx_c \mathsf{Game}_2$.

Proof. Since the main difference between Game_1 and Game_2 is that the challenger samples the extractor information $\mathsf{ex}_{\mathsf{NIZK}}$ in the setup phase and then extracts all the witnesses from the proof π_j for all malicious slots j, we can reduce the indistinguishability between Game_1 and Game_2 to the simulation-sound extractability of the NIZK scheme. Since the reduction is standard, we omit details here. □

Lemma 13. *By the property of equivocal public keys of* esRABFE*, we claim* $\mathsf{Game}_2 \approx_c \mathsf{Game}_3$.

Proof. We see that given a tag $\widehat{\mathsf{tag}}_i \in [Q]$, the challenger can proceeds as follows:

– In Game_2, it runs

$$\mathsf{crs}_{\mathsf{esRABFE}} \leftarrow \mathsf{esRABFE.Setup}(1^\lambda, 1^L, 1^{2\lambda}, P, C),$$
$$(\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{esRABFE.Gen}(\mathsf{crs}_{\mathsf{esRABFE}}, i, (a_i, \widehat{\mathsf{tag}}_i))$$

– In Game_3, it instead runs

$$\begin{aligned}(\mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{ek}_{\mathsf{esRABFE}}) &\leftarrow \mathsf{esRABFE.ESetup}(1^\lambda, 1^L, 1^{2\lambda}, P, C),\\ (\mathsf{pk}'_i, \mathsf{st}'_i) &\leftarrow \mathsf{esRABFE.EGen}(\mathsf{crs}, i, \mathsf{ek}_{\mathsf{esRABFE}}),\\ \mathsf{sk}'_i &\leftarrow \mathsf{esRABFE.Equiv}(\mathsf{st}'_i, (a_i, \widehat{\mathsf{tag}}_i), \mathsf{ek}_{\mathsf{esRABFE}})\end{aligned}$$

By the property of equivocal public keys defined in Section 5, the components $(\mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{pk}'_i, \mathsf{sk}'_i)$ generated in both modes are indistinguishable from each other. This exactly claims $\mathsf{Game}_2 \approx_c \mathsf{Game}_3$. □

Lemma 14. *We claim* $\mathsf{Game}_3 \equiv \mathsf{Game}_4$.

Proof. In Game_4, it samples the evaluation key k in advance, where k is only related to the security parameter 1^λ. This does not change any distributions, so it naturally has $\mathsf{Game}_3 \equiv \mathsf{Game}_4$. □

Lemma 15. *By the pseudorandomness of* wPRF, *we claim* $\mathsf{Game}_4 \approx_c \mathsf{Game}_5$.

Proof. Recall that for each $\mathsf{tag}^* \in [Q]$, we let $\mathsf{cnt}_{\mathsf{tag}^*}$ be the number of tags such that $\mathsf{tag}^* = \mathsf{wPRF.Eval}(\mathsf{k}, a_j) \oplus \widehat{\mathsf{tag}}_j^*$ for any $j \in \mathcal{K}$, where $\widehat{\mathsf{tag}}_j^*$ is the tag associated with the registered public key pk_j^*. Now we further define $\mathsf{cnt}'_{\mathsf{tag}^*}$ and $\mathsf{cnt}''_{\mathsf{tag}^*}$, where $\mathsf{cnt}'_{\mathsf{tag}^*}$ and $\mathsf{cnt}''_{\mathsf{tag}^*}$ record the accumulated number of tags such that $\mathsf{tag}^* = \mathsf{wPRF.Eval}(\mathsf{k}, a_j) \oplus \widehat{\mathsf{tag}}_j^*$ for any $j \in \mathcal{K}'$ and any $j \in \mathcal{K} \setminus \mathcal{K}'$, respectively. Therefore, it must have $\mathsf{cnt}_{\mathsf{tag}^*} = \mathsf{cnt}'_{\mathsf{tag}^*} + \mathsf{cnt}''_{\mathsf{tag}^*}$ for each $\mathsf{tag}^* \in [Q]$.

At the first step, we claim the following argument:

$$\forall \mathsf{tag}^* \in [Q],\ \Pr[\mathsf{cnt}'_{\mathsf{tag}^*} > \lambda] \le \mathsf{negl}(\lambda). \tag{11}$$

If the adversary $\mathcal{A}$ can find a counterexample to the above argument, we can build an algorithm $\mathcal{B}$ against the pseudorandomness of wPRF. Concretely, in the setup phase, $\mathcal{B}$ invokes the challenger of wPRF and hence acquires all pairs of input elements and output tags $\{(a_j, \mathsf{tag}'_j)\}_{j\in[L]}$, where tag'_j is either generated from algorithm wPRF.Eval, or sampled by a random function. Here, $\mathcal{A}$ can only acquire $\{a_j\}_{j\in[L]}$ from crs. Then, in the challenge phase, $\mathcal{B}$ knows the set $\mathcal{K}'$ from $\mathcal{A}$. We know that if $\{\mathsf{tag}'_j\}_{j\in\mathcal{K}'}$ are randomly sampled, then the tags $\{\mathsf{tag}'_j \oplus \widehat{\mathsf{tag}}_j^*\}_{j\in\mathcal{K}'}$ are also randomly distributed. Thus, by Lemma 16, the argument in Eq. (11) must hold (where the variable X_u actually corresponds to $\mathsf{cnt}'_{\mathsf{tag}'_j \oplus \widehat{\mathsf{tag}}_j^*}$). Note that although the shift tags $\{\widehat{\mathsf{tag}}_j^*\}_{j\in[\mathcal{K}']}$ may be maliciously sampled by $\mathcal{A}$, they are independent of $\{\mathsf{tag}'_j\}_{j\in\mathcal{K}'}$.

Therefore, in the case that $\{\mathsf{tag}'_j\}_{j\in\mathcal{K}'}$ are generated from wPRF.Eval, if $\mathcal{B}$ can find a tag $\mathsf{tag}^* = \mathsf{tag}'_j \oplus \widehat{\mathsf{tag}}_j^*$ for any $j \in \mathcal{K}'$ such that $\mathsf{cnt}_{\mathsf{tag}^*} > \lambda$ with non-negligible probability, it can also break the pseudorandomness of wPRF with non-negligible probability. By contradiction, we say that the argument in Eq. (11) must hold even when $\{\mathsf{tag}'_j\}_{j\in\mathcal{K}'}$ are generated from algorithm wPRF.Eval.

At the second step, we claim

$$\forall \mathsf{tag}^* \in [Q],\ \Pr[\mathsf{cnt}''_{\mathsf{tag}^*} > \lambda] \le \mathsf{negl}(\lambda). \tag{12}$$

In Game_4, we see that the shift tags $\{\widehat{\mathsf{tag}}_j^*\}_{j\in\mathcal{K}\setminus\mathcal{K}'}$ are sampled randomly and independently in the view of $\mathcal{A}$. Similarly, by Lemma 16, the argument in Eq. (12) naturally holds (where the variable X_u now corresponds to $\mathsf{cnt}''_{\mathsf{wPRF.Eval}(\mathsf{k},j)\oplus\widehat{\mathsf{tag}}_j^*}$).

Above all, we naturally have

$$\forall \mathsf{tag}^* \in [Q],\ \Pr[\mathsf{cnt}_{\mathsf{tag}^*} > 2\lambda] < \Pr[\mathsf{cnt}'_{\mathsf{tag}^*} > \lambda] + \Pr[\mathsf{cnt}''_{\mathsf{tag}^*} > \lambda] \le \mathsf{negl}(\lambda).$$

Therfore, the lemma follows readily. □

Lemma 16 ([AV19]). *Consider random variables $X_{j,k}$ for $j \in [Q]$ and $k \in [Q]$as follows. For each $k \in [Q]$, $u_k \leftarrow [Q]$ is chosen and we set $X_{j,k} = 1$ if $j = u_k$; otherwise, $X_{j,k} = 0$. Let $X_u = \sum_{j=1}^{Q} X_{j,u}$, then we have*

$$\Pr[X_u > \lambda] \leq \mathsf{negl}(\lambda).$$

Lemma 17. *By the adaptive SIM security with selective policies of* esRABFE*, we claim* $\mathsf{Game}_5 \approx_c \mathsf{Game}_6$.

Proof. If the adversary $\mathcal{A}$ can distinguish these two games, we can build an algorithm $\mathcal{B}$ against the security of esRABFE. Initially, $\mathcal{B}$ receives the input $(1^L, 1^Q)$ from $\mathcal{A}$. Then it proceeds as follows:

- *Setup phase:* $\mathcal{B}$ samples $\mathsf{k} \leftarrow \mathsf{wPRF.Gen}(1^\lambda)$ and invokes the challenger of esRABFE by revealing the input $(1^L, 1^{2\lambda}, P, C)$ and policies $\mathsf{P}_{\mathsf{k},1}, \ldots, \mathsf{P}_{\mathsf{k},Q} \in P$. Here, since the policy $\mathsf{P}_{\mathsf{k},d}$ is defined over the evaluation key k and the index $d \in [Q]$, $\mathcal{B}$ can just reveal a single policy $\mathsf{P}^* = (\mathsf{k}, 1^Q)$ to the challenger. We can see that for each $d \in [Q]$, P_d can be efficiently derived from P^*, which is independent of Q. Therefore, we see that the runtime of the setup phase is also independent of Q.
  Next, $\mathcal{B}$ receives $\mathsf{crs}_{\mathsf{esRABFE}}$ from the challenger and then samples
  $$(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.EGen}(1^\lambda).$$
  and returns $\mathsf{crs} = (\mathsf{crs}_{\mathsf{esRABFE}}, \mathsf{crs}_{\mathsf{NIZK}})$ to $\mathcal{A}$.
- *Pre-challenge phase:*
  - $\mathsf{OGen}(i)$: $\mathcal{B}$ samples $\widehat{\mathsf{tag}}_i \leftarrow [Q]$ and queries the oracle esRABFE.OGen on input (i) to acquire the public key pk'. Then, it samples
    $$\pi_i \leftarrow \mathsf{NIZK.SPrv}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, s_i)$$
    where $s_i = (i, \widetilde{\mathsf{crs}}_{\mathsf{esRABFE}}, \mathsf{pk}')$. Return $\mathsf{pk} = (\mathsf{pk}', \pi)$ to $\mathcal{A}$. Update $\mathcal{D}_i[\mathsf{pk}] = \widehat{\mathsf{tag}}_i$.
  - $\mathsf{OCor}(i, \mathsf{pk})$: Parse $\mathsf{pk} = (\mathsf{pk}', \pi)$. $\mathcal{B}$ queries the oracle esRABFE.OCor on input $(i, \mathsf{pk}', (i, \widehat{\mathsf{tag}}_i))$ to acquire the secret key sk'. Return $\mathsf{sk} = (\mathsf{sk}', \mathcal{D}_i[\mathsf{pk}])$ to $\mathcal{A}$.
- *Challenge phase:* In this phase, $\mathcal{B}$ receives public keys $\{\mathsf{pk}_i^*\}_{i\in[L]}$, circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$, and a challenge message x^*, where $\mathsf{pk}_i^* = (\mathsf{pk}_i', \pi_i^*)$. For each malicious slot $j \in \mathcal{K}'$, $\mathcal{B}$ can extract the witness
  $$(\mathsf{sk}_j', \widehat{\mathsf{tag}}_j^*, r_j) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}, s_j, \pi_j^*)$$
  where the statement $s_j = (j, \widetilde{\mathsf{crs}}_{\mathsf{esRABFE}}, \mathsf{pk}_j')$ and r_j is the randomness used for generating pk_j'. Then, since other tags $\widehat{\mathsf{tag}}_i^*$ for honest/corrupted slots $i \in [L]$ has already been recorded in $\mathcal{D}_i[\mathsf{pk}_i^*]$, $\mathcal{B}$ can submit the challenge information $\{(\mathsf{pk}_i^*, (i, \widehat{\mathsf{tag}}_i^*), \mathsf{C}_i^*)\}_{i\in[L]}$ together with a list of the extracted randomness $\{r_j\}$ to the challenger of esRABFE.
  Next, for each $d \in [Q]$, it queries the oracle esRABFE.OEnc on input $(\mathsf{P}_{\mathsf{k},d}, x^*)$ to obtain the ciphertext ct_d^*. Finally, return the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_1^*, \ldots, \mathsf{ct}_Q^*)$ to $\mathcal{A}$.
- *Post-challenge phase:* In this phase, $\mathcal{A}$ can adaptively make corruption queries as follows:
  - $\mathsf{OCor}(i, \mathsf{pk})$: Parse $\mathsf{pk} = (\mathsf{pk}', \pi)$. $\mathcal{B}$ queries the oracle esRABFE.OCor on input $(i, \mathsf{pk}', (i, \mathcal{D}_i[\mathsf{pk}]))$ to acquire the secret key sk'. Return $\mathsf{sk} = (\mathsf{sk}', \mathcal{D}_i[\mathsf{pk}])$ to $\mathcal{A}$.

In Game_4, we have known $\mathsf{cnt}_d \leq 2\lambda$ for each $d \in [Q]$, meaning that the number of the corrupted/malicious slots j such that $\mathsf{P}_{\mathsf{k},d}(j, \widehat{\mathsf{tag}}_j^*) = 1$ must be not greater than the collusion bound 2λ. Therefore, the above reduction makes sense: in the real world of esRABFE, $\mathcal{B}$ exactly simulates Game_5; otherwise, it simulates Game_6. This proves the lemma readily. □

7 From Weakly Optimal Slotted Reg-FE to Dynamic Slotted Reg-FE

In this section, we compile weakly optimal sReg-FE into the dynamic bounded collusion sReg-FE, achieving fully adaptive SIM security.

7.1 Definition

A dynamic sReg-FE for circuit family $C : X \to Z$ is defined as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, C) \to \mathsf{crs}$. It takes as input the security parameter 1^λ, slot amount 1^L, and the description of circuit family C, then outputs a common reference string crs.
- $\mathsf{Gen}(\mathsf{crs}, i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. It takes as input the common reference string crs and a slot index $i \in [L]$, then outputs a public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
- $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) \to 0/1$. It takes as input the common reference string crs, a slot index i, and public key pk_i, outputs a bit indicating whether pk_i is valid.
- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$. It takes as input crs and a series of public keys and circuits $\{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]}$, then outputs the master public key mpk and helper keys $\{\mathsf{hsk}_j\}_{j\in[L]}$. This algorithm is deterministic.
- $\mathsf{Enc}(\mathsf{mpk}, x, 1^Q) \to \mathsf{ct}$. It takes as input the master public key mpk, a message $x \in X$, and the collusion bound 1^Q, then outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct}) \to z/\perp$. It takes as input a helper key hsk_{i^*}, a secret key sk_{i^*}, an attribute att_{i^*}, and a ciphertext ct, then outputs $z \in Z$ or an empty symbol $\perp$.

The above scheme is required to satisfy:

- **Compactness.** For all $\lambda, L \in \mathbb{N}$, and all C, it requires $|\mathsf{mpk}| = \mathsf{poly}(\lambda, C, \log L)$ and $|\mathsf{hsk}_j| = \mathsf{poly}(\lambda, C, \log L)$.
- **Completeness.** For all $\lambda, L \in \mathbb{N}$, all C, and all $i \in [L]$, we have

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, C) \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs}, i) \end{array}\right] = 1.$$

- **Correctness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all C, all $i^* \in [L]$, all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, C)$, all $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i^*)$, all $\{\mathsf{pk}_i\}_{i\in[L]\setminus\{i^*\}}$ such that $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$, for all messages $x \in X$ and circuits $\mathsf{C}_1, \ldots, \mathsf{C}_L \in C$, and computing $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$, we have

$$\Pr\left[\mathsf{Dec}\left(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{Enc}(\mathsf{mpk}, x, 1^Q)\right) = \mathsf{C}_{i^*}(x)\right] = 1.$$

- **(Non-)Adaptive SIM Security.** The security definition is the same as the (non-)adaptive SIM security of weakly optimal sReg-FE in Section 6.1, except that $\mathcal{A}$ specifies the collusion bound 1^Q in the challenge phase.

Static Collusion. We note that static bounded collusion sReg-FE can be defined in a similar way to dynamic bounded collusion sReg-FE, except that it must specify 1^Q in Setup and allow all system overheads to grow with Q.

7.2 Construction

Let $C : X \to Z$ be a circuit family, assuming a weakly optimal sReg-FE scheme $\mathsf{wsRFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, our construction works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, C)$: Let $\eta = \min(\lambda, \lceil \log L \rceil)$, then sample

$$\forall j \in [\eta],\ \mathsf{crs}_j \leftarrow \mathsf{wsRFE.Setup}(1^\lambda, 1^L, 2^j, C).$$

Output $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_\eta)$.

– $\mathsf{Gen}(\mathsf{crs}, i)$: Sample $(\mathsf{pk}_{i,j}, \mathsf{sk}_{i,j}) \leftarrow \mathsf{wsRFE.Gen}(\mathsf{crs}_j, i)$ for all $j \in [\eta]$. Output $\mathsf{pk}_i = (\{\mathsf{pk}_{i,j}\}_{j\in[\eta]})$ and $\mathsf{sk}_i = (\{\mathsf{sk}_{i,j}\}_{j\in[\eta]})$.

– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i)$: It checks whether $\mathsf{wsRFE.Ver}(\mathsf{crs}_j, i, \mathsf{pk}_{i,j}) \stackrel{?}{=} 1$ for all $j \in [\eta]$. If not, abort and output 0; otherwise, output 1.

– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: Compute

$$\forall j \in [\eta],\ (\mathsf{mpk}_j, \{\mathsf{hsk}_{k,j}\}_{k\in[L]}) \leftarrow \mathsf{wsRFE.Agg}(\mathsf{crs}_j, \{(\mathsf{pk}_{i,j}, \mathsf{C}_i)\}_{i\in[L]}).$$

Output $\mathsf{mpk} = \{\mathsf{mpk}_j\}_{j\in[\eta]}$ and $\mathsf{hsk}_k = \{\mathsf{hsk}_{k,j}\}_{j\in[\eta]}$.

– $\mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$: Output $\mathsf{ct} \leftarrow \mathsf{wsRFE.Enc}(\mathsf{mpk}_{\lceil \log Q \rceil}, x)$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct})$: Output $z \leftarrow \mathsf{wsRFE.Dec}(\mathsf{hsk}_{i^*,\lceil \log Q \rceil}, \mathsf{sk}_{i^*,\lceil \log Q \rceil}, \mathsf{ct})$.

Compactness, Completeness and Correctness. By the weakly optimal compactness of wsRFE, it must hold that $|\mathsf{mpk}| = \mathsf{poly}(\lambda, n, \log L)$ and $|\mathsf{hsk}_j| = \mathsf{poly}(\lambda, n, \log L)$. Besides, the completeness and correctness follow the underlying wsRFE scheme.

7.3 Proof

Theorem 11. *Assuming* wsRFE *achieves (non-)adaptive SIM security, then our construction achieves (non-)adaptive SIM security.*

Proof. Here, we define the simulator algorithms $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that work as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: Sample

$$\forall j \in [\eta],\ (\mathsf{crs}_j, \mathsf{td}_j) \leftarrow \mathsf{wsRFE.}\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 2^j, C).$$

Output $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_\eta)$ and $\mathsf{td} = (\mathsf{td}_1, \ldots, \mathsf{td}_\eta)$.

– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: Sample

$$\forall j \in [\eta],\ (\mathsf{pk}_{i,j}, \mathsf{st}_{i,j}) \leftarrow \mathsf{wsRFE.}\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_j, i, \mathsf{td}_j).$$

Output $\mathsf{pk}_i = (\{\mathsf{pk}_{i,j}\}_{j\in[\eta]})$ and $\mathsf{st}_i = (\{\mathsf{st}_{i,j}\}_{j\in[\eta]})$.

– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{td}, (f, z))$: Run

$$\forall j \in [\eta],\ \mathsf{sk}_{i,j} \leftarrow \mathsf{wsRFE.}\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,j}, \mathsf{td}_j, (f, z)).$$

Output $\mathsf{sk}_i = (\{\mathsf{sk}_{i,j}\}_{j\in[\eta]})$.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}}, 1^Q)$: Output

$$\mathsf{ct}^* \leftarrow \mathsf{wsRFE.}\widetilde{\mathsf{Enc}}(\mathsf{mpk}_{\lceil \log Q \rceil}, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}}).$$

Game Sequence. We prove Theorem 11 via a sequence of games as follows:

– Game_0: Real Game.

– $\mathsf{Game}_{1,\ell}(\ell \in [\eta])$: this game is identical to $\mathsf{Game}_{1,\ell-1}$ except that for all $j \le \ell$, it generates crs_j via $\mathsf{wsRFE.}\widetilde{\mathsf{Setup}}$, and for each query to oracles $(\mathsf{OGen}, \mathsf{OCor})$ it generates key pairs via $(\mathsf{wsRFE.}\widetilde{\mathsf{Gen}}_1, \mathsf{wsRFE.}\widetilde{\mathsf{Gen}}_2)$. In addition, if $\lceil \log Q \rceil > j$, it simulates the challenge ciphertext via $\mathsf{wsRFE.}\widetilde{\mathsf{Enc}}$.

We see that in the final game $\mathsf{Game}_{1,\eta}$, the challenge ciphertext ct^* and the corrupted secret keys $\{\mathsf{sk}_i\}_{i\in\mathcal{K}}$ can be simulated only using $\{(\mathsf{C}_i^*, \mathsf{C}_i^*(x^*))\}_{i\in\mathcal{K}}$. In addition, the collusion bound 1^Q is specified in the challenge phase. This is exactly the simulation world of dynamic sReg-FE.

Lemma 18. *By the security of* wsRFE, *we claim* $\mathsf{Game}_{1,\ell-1} \approx_c \mathsf{Game}_{1,\ell}$.

Proof. It is easy to see that we can complete the transition between $\mathsf{Game}_{1,\ell-1}$ and $\mathsf{Game}_{1,\ell}$ by relying on the (non-)adaptive SIM security of wsRFE. Since the reduction is standard, we omit details here. □

References

[ABDP15] Michel Abdalla, Florian Bourse, Angelo De Caro, and David Pointcheval. Simple functional encryption schemes for inner products. In Jonathan Katz, editor, *PKC 2015*, volume 9020 of *LNCS*, pages 733–751. Springer, Berlin, Heidelberg, March / April 2015. 1

[Agr17] Shweta Agrawal. Stronger security for reusable garbled circuits, general definitions and attacks. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 3–35. Springer, Cham, August 2017. 1

[AGVW13] Shweta Agrawal, Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption: New perspectives and lower bounds. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 500–518. Springer, Berlin, Heidelberg, August 2013. 2

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 308–326. Springer, Berlin, Heidelberg, August 2015. 2

[ALS16] Shweta Agrawal, Benoît Libert, and Damien Stehlé. Fully secure functional encryption for inner products, from standard assumptions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 333–362. Springer, Berlin, Heidelberg, August 2016. 1

[AMR25] Damiano Abram, Giulio Malavolta, and Lawrence Roy. Key-homomorphic computations for RAM: Fully succinct randomised encodings and more. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 236–268. Springer, Cham, August 2025. 26

[AMVY21] Shweta Agrawal, Monosij Maitra, Narasimha Sai Vempati, and Shota Yamada. Functional encryption for Turing machines with dynamic bounded collusion from LWE. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part IV*, volume 12828 of *LNCS*, pages 239–269, Virtual Event, August 2021. Springer, Cham. 1

[AR17] Shweta Agrawal and Alon Rosen. Functional encryption for bounded collusions, revisited. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part I*, volume 10677 of *LNCS*, pages 173–205. Springer, Cham, November 2017. 1

[AT24] Nuttapong Attrapadung and Junichi Tomida. A modular approach to registered ABE for unbounded predicates. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 280–316. Springer, Cham, August 2024. 2, 12, 59, 66

[AV19] Prabhanjan Ananth and Vinod Vaikuntanathan. Optimal bounded-collusion secure functional encryption. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 174–198. Springer, Cham, December 2019. 1, 7, 10, 17, 44

[AW17] Shashank Agrawal and David J. Wu. Functional encryption: Deterministic to randomized functions from simple assumptions. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part II*, volume 10211 of *LNCS*, pages 30–61. Springer, Cham, April / May 2017. 1

[BB04] Dan Boneh and Xavier Boyen. Secure identity based encryption without random oracles. In Matthew Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 443–459. Springer, Berlin, Heidelberg, August 2004. 1

[BCG+17] Zvika Brakerski, Nishanth Chandran, Vipul Goyal, Aayush Jain, Amit Sahai, and Gil Segev. Hierarchical functional encryption. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 67, pages 8:1–8:27. LIPIcs, January 2017. 5

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the Weil pairing. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 213–229. Springer, Berlin, Heidelberg, August 2001. 1

[BGW05] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 258–275. Springer, Berlin, Heidelberg, August 2005. 1

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In Ting Yu, George Danezis, and Virgil D. Gligor, editors, *ACM CCS 2012*, pages 784–796. ACM Press, October 2012. 14

[BLM+24] Pedro Branco, Russell W. F. Lai, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Ivy K. Y. Woo. Traitor tracing without trusted authority from registered functional encryption. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part III*, volume 15486 of *LNCS*, pages 33–66. Springer, Singapore, December 2024. 2, 3, 8

[BP14] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 353–370. Springer, Berlin, Heidelberg, August 2014. 11, 16

[BPR12] Abhishek Banerjee, Chris Peikert, and Alon Rosen. Pseudorandom functions and lattices. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 719–737. Springer, Berlin, Heidelberg, April 2012. 11, 16

[BSW11] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In Yuval Ishai, editor, *TCC 2011*, volume 6597 of *LNCS*, pages 253–273. Springer, Berlin, Heidelberg, March 2011. 1, 2

[BV15] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 171–190. IEEE Computer Society Press, October 2015. 2

[CGW15] Jie Chen, Romain Gay, and Hoeteck Wee. Improved dual system ABE in prime-order groups via predicate encodings. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 595–624. Springer, Berlin, Heidelberg, April 2015. 19

[Cha07] Melissa Chase. Multi-authority attribute based encryption. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 515–534. Springer, Berlin, Heidelberg, February 2007. 94

[CHW25a] Jeffrey Champion, Yao-Ching Hsieh, and David J. Wu. Registered ABE and adaptively-secure broadcast encryption from succinct LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 3–34. Springer, Cham, August 2025. 2, 57, 59, 66

[CHW25b] Jeffrey Champion, Yao-Ching Hsieh, and David J Wu. Registered abe and adaptively-secure broadcast encryption from succinct lwe. *CRYPTO*, 2025. 11, 26

[DN00] Ivan Damgård and Jesper Buus Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 432–450. Springer, Berlin, Heidelberg, August 2000. 9, 31

[DP23] Pratish Datta and Tapas Pal. Decentralized multi-authority attribute-based inner-product FE: Large universe and unbounded. Cryptology ePrint Archive, Report 2023/565, 2023. 91

[DPY24] Pratish Datta, Tapas Pal, and Shota Yamada. Registered FE beyond predicates: (attribute-based) linear functions and more. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 65–104. Springer, Singapore, December 2024. 2, 3, 8, 12

[DS15] Nico Döttling and Dominique Schröder. Efficient pseudorandom functions via on-the-fly adaptation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part I*, volume 9215 of *LNCS*, pages 329–350. Springer, Berlin, Heidelberg, August 2015. 11, 16

[EHK+13] Alex Escala, Gottfried Herold, Eike Kiltz, Carla Ràfols, and Jorge Villar. An algebraic framework for Diffie-Hellman assumptions. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 129–147. Springer, Berlin, Heidelberg, August 2013. 3, 13, 59

[FFM+23] Danilo Francati, Daniele Friolo, Monosij Maitra, Giulio Malavolta, Ahmadreza Rahimi, and Daniele Venturi. Registered (inner-product) functional encryption. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 98–133. Springer, Singapore, December 2023. 2, 3, 6, 8, 91

[FWW23] Cody Freitag, Brent Waters, and David J. Wu. How to use (plain) witness encryption: Registered ABE, flexible broadcast, and more. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part IV*, volume 14084 of *LNCS*, pages 498–531. Springer, Cham, August 2023. 11

[GGH+13] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai, and Brent Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *54th FOCS*, pages 40–49. IEEE Computer Society Press, October 2013. 2

[GGL24] Rachit Garg, Rishab Goyal, and George Lu. Dynamic collusion functional encryption and multi-authority attribute-based encryption. In Qiang Tang and Vanessa Teague, editors, *PKC 2024, Part II*, volume 14604 of *LNCS*, pages 69–104. Springer, Cham, April 2024. 94, 95

[GGLW22] Rachit Garg, Rishab Goyal, George Lu, and Brent Waters. Dynamic collusion bounded functional encryption from identity-based encryption. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 736–763. Springer, Cham, May / June 2022. 2, 9, 10, 31

[GHM+19] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, Ahmadreza Rahimi, and Sruthi Sekar. Registration-based encryption from standard assumptions. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 63–93. Springer, Cham, April 2019. 2

[GHMR18] Sanjam Garg, Mohammad Hajiabadi, Mohammad Mahmoody, and Ahmadreza Rahimi. Registration-based encryption: Removing private-key generator from IBE. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 689–718. Springer, Cham, November 2018. 2

[GKP+13] Shafi Goldwasser, Yael Tauman Kalai, Raluca A. Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *45th ACM STOC*, pages 555–564. ACM Press, June 2013. 1

[GLWW24] Rachit Garg, George Lu, Brent Waters, and David J. Wu. Reducing the CRS size in registered ABE systems. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 143–177. Springer, Cham, August 2024. 2, 12

[GVW12] Sergey Gorbunov, Vinod Vaikuntanathan, and Hoeteck Wee. Functional encryption with bounded collusions via multi-party computation. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 162–179. Springer, Berlin, Heidelberg, August 2012. 1, 3, 6, 9, 31

[GY24] Rishab Goyal and Saikumar Yadugiri. Multi-authority functional encryption with bounded collusions from standard assumptions. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part III*, volume 15366 of *LNCS*, pages 3–30. Springer, Cham, December 2024. 8, 11

[GY25] Rishab Goyal and Saikumar Yadugiri. Multi-authority encryption with malicious authorities. Cryptology ePrint Archive, Paper 2025/412, 2025. 5

[HLWW23] Susan Hohenberger, George Lu, Brent Waters, and David J. Wu. Registered attribute-based encryption. In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part III*, volume 14006 of *LNCS*, pages 511–542. Springer, Cham, April 2023. 2, 6, 8, 12, 57, 88, 90, 91, 92

[JLS21] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In Samir Khuller and Virginia Vassilevska Williams, editors, *53rd ACM STOC*, pages 60–73. ACM Press, June 2021. 2

[JLS22] Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from LPN over $\mathbb{F}_p$, DLIN, and PRGs in NC^0. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 670–699. Springer, Cham, May / June 2022. 2

[LLW21] Qiqi Lai, Feng-Hao Liu, and Zhedong Wang. New lattice two-stage sampling technique and its applications to functional encryption - stronger security and smaller ciphertexts. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part I*, volume 12696 of *LNCS*, pages 498–527. Springer, Cham, October 2021. 7

[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 Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 62–91. Springer, Berlin, Heidelberg, May / June 2010. 1

[LW11] Allison Lewko and Brent Waters. Decentralizing attribute-based encryption. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 568–588. Springer, 2011. 94, 95

[MP12] Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 700–718. Springer, Berlin, Heidelberg, April 2012. 13

[NR97] Moni Naor and Omer Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *38th FOCS*, pages 458–467. IEEE Computer Society Press, October 1997. 11, 16

[PST25] Tapas Pal, Robert Schädlich, and Erkan Tairi. Registered functional encryption for pseudorandom functionalities from lattices: Registered ABE for unbounded depth circuits and turing machines, and more. Cryptology ePrint Archive, Report 2025/967, 2025. 3

[Reg05] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 84–93. ACM Press, May 2005. 16

[RS60] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960. 9

[RVV24] Seyoon Ragavan, Neekon Vafa, and Vinod Vaikuntanathan. Indistinguishability obfuscation from bilinear maps and LPN variants. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024, Part IV*, volume 15367 of *LNCS*, pages 3–36. Springer, Cham, December 2024. 2

[SS10] Amit Sahai and Hakan Seyalioglu. Worry-free encryption: functional encryption with public keys. In Ehab Al-Shaer, Angelos D. Keromytis, and Vitaly Shmatikov, editors, *ACM CCS 2010*, pages 463–472. ACM Press, October 2010. 1

[SW05] Amit Sahai and Brent R. Waters. Fuzzy identity-based encryption. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 457–473. Springer, Berlin, Heidelberg, May 2005. 1

[Wee24] Hoeteck Wee. Circuit ABE with poly(depth, λ)-sized ciphertexts and keys from lattices. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part III*, volume 14922 of *LNCS*, pages 178–209. Springer, Cham, August 2024. 3, 13, 59

[WFL19] Zhedong Wang, Xiong Fan, and Feng-Hao Liu. FE for inner products and its application to decentralized ABE. In Dongdai Lin and Kazue Sako, editors, *PKC 2019, Part II*, volume 11443 of *LNCS*, pages 97–127. Springer, Cham, April 2019. 94

[WOG88] Avi Wigderson, MB Or, and S Goldwasser. Completeness theorems for noncryptographic fault-tolerant distributed computations. In *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, pages 1–10, 1988. 7

[WW25] Hoeteck Wee and David J. Wu. Unbounded distributed broadcast encryption and registered ABE from succinct LWE. In Yael Tauman Kalai and Seny F. Kamara, editors, *CRYPTO 2025, Part III*, volume 16002 of *LNCS*, pages 204–235. Springer, Cham, August 2025. 2, 26, 57, 59, 66

[WWW22] Brent Waters, Hoeteck Wee, and David J. Wu. Multi-authority ABE from lattices without random oracles. In Eike Kiltz and Vinod Vaikuntanathan, editors, *TCC 2022, Part I*, volume 13747 of *LNCS*, pages 651–679. Springer, Cham, November 2022. 94, 95

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*, pages 162–167. IEEE, 1986. 14

[ZCHZ24] Yijian Zhang, Jie Chen, Debiao He, and Yuqing Zhang. Bounded collusion-resistant registered functional encryption for circuits. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part I*, volume 15484 of *LNCS*, pages 32–64. Springer, Singapore, December 2024. 2, 3, 6, 30, 60, 80

[Zha25] Mark Zhandry. Optimal traitor tracing from pairings. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 305–335. Springer, 2025. 11, 16

[ZLZ+24] Ziqi Zhu, Jiangtao Li, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered functional encryptions from pairings. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 373–402. Springer, Cham, May 2024. 2, 3

[ZZC+25] Ziqi Zhu, Kai Zhang, Zhili Chen, Junqing Gong, and Haifeng Qian. Black-box registered ABE from lattices. Cryptology ePrint Archive, Report 2025/051, 2025. 11

[ZZGQ23] Ziqi Zhu, Kai Zhang, Junqing Gong, and Haifeng Qian. Registered ABE via predicate encodings. In Jian Guo and Ron Steinfeld, editors, *ASIACRYPT 2023, Part V*, volume 14442 of *LNCS*, pages 66–97. Springer, Singapore, December 2023. 8, 12, 19, 22, 59, 66

Appendices

A Proof of Theorem 8

In this section, we present the proof of Theorem 8. To prove it, we present a new construction of 1-bound equivocal sReg-ABFE, and then prove that it achieves adaptive SIM security (with selective policies).

Assuming an equivocal sReg-ABE scheme for NC^1 policies $\mathsf{esRABE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$ and a 1-bound equivocal sReg-ABFE scheme for $\mathsf{NC}^1 \circ \mathsf{P/Poly}$ with non-adaptive security $\mathsf{oesRABFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, our scheme works as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C)$: It proceeds as follows:
  - Invoke an instance of 1-bound equivocal sReg-ABFE with non-adaptive security, i.e.,

$$\mathsf{crs}_{\mathsf{oesRABFE}} \leftarrow \mathsf{oesRABFE.Setup}(1^\lambda, 1^L, 1^Q, P, C).$$

  - Invoke an instance of equivocal sReg-ABE, i.e.,

$$\mathsf{crs}_{\mathsf{esRABE}} \leftarrow \mathsf{esRABE.Setup}(1^\lambda, 1^L, P).$$

  Output the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{crs}_{\mathsf{esRABE}})$.
- $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$: It proceeds as follows:
  - Sample a public/secret key pair of 1-bound sReg-FE with non-adaptive security, i.e.,

$$(\mathsf{pk}'_i, \mathsf{sk}'_i) \leftarrow \mathsf{oesRABFE.Gen}(\mathsf{crs}_{\mathsf{oesRABFE}}, i, \mathsf{att}_i).$$

  - Let σ be the bit length of the ciphertext in the oesRABFE scheme. Then sample

$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ (\mathsf{pk}'_{i,w,b}, \mathsf{sk}'_{i,w,b}) \leftarrow \mathsf{esRABE.Gen}(\mathsf{crs}_{\mathsf{esRABE}}, i, \mathsf{att}_i).$$

  In addition, sample a random string $\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma$.

  Output the pubic key and secret key

$$\mathsf{pk}_i = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}}),\ \mathsf{sk}_i = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]}\}_{w\in[\sigma]}, \mathbf{b}_i).$$

- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: It proceeds as follows:
  - Compute the master public key and helper keys of oesRABFE:

$$(\mathsf{mpk}', \{\mathsf{hsk}'_j\}_{j\in[L]}) \leftarrow \mathsf{oesRABFE.Agg}(\mathsf{crs}_{\mathsf{oesRABFE}}, \{(\mathsf{pk}'_i, \mathsf{C}_i)\}_{i\in[L]}).$$

  - For each $w \in [\sigma]$ and $b \in \{0,1\}$, compute the master public keys and helper keys of esRABE:

$$(\mathsf{mpk}'_{w,b}, \{\mathsf{hsk}'_{j,w,b}\}_{j\in[L]}) \leftarrow \mathsf{esRABE.Agg}(\mathsf{crs}_{\mathsf{esRABE}}, \{\mathsf{pk}'_{i,w,b}\}_{i\in[L]}).$$

  Output the master public key and helper keys

$$\mathsf{mpk} = (\mathsf{mpk}', \{\mathsf{mpk}'_{w,b}\}_{w\in[n],b\in\{0,1\}}),\ \mathsf{hsk}_j = (\mathsf{hsk}'_j, \{\mathsf{hsk}'_{j,w,b}\}_{w\in[\sigma],b\in\{0,1\}}).$$

- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x)$: It proceeds as follows:
  - Compute a ciphertext of oesRABFE:

$$\mathsf{ct}' \leftarrow \mathsf{oesRABFE.Enc}(\mathsf{mpk}', x).$$

- Parse $\mathsf{ct}' = (\mathsf{ct}'[1], \ldots, \mathsf{ct}'[\sigma]) \in \{0,1\}^\sigma$. Then compute the following ciphertexts of esRABE:

$$\forall w \in [n] \text{ and } b \in \{0,1\},\ \mathsf{ct}'_{w,b} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}, \mathsf{ct}'[w]).$$

  Output the ciphertext $\mathsf{ct} = \{\mathsf{ct}'_{w,b}\}_{w\in[\sigma],b\in\{0,1\}}$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct})$: Parse $\mathsf{hsk}_{i^*} = (\mathsf{hsk}'_{i^*}, \{\mathsf{hsk}'_{i^*,w,b}\}_{w\in[\sigma],b\in\{0,1\}})$ and $\mathsf{sk}_{i^*} = (\mathsf{sk}'_{i^*}, \{\mathsf{sk}'_{i^*,w,b_{i^*}[w]}\}_{w\in[\sigma]}, \mathbf{b}_{i^*})$. It first recovers the full ciphertext of oesRABFE:

$$\forall w \in [\sigma],\ \mathsf{ct}'[w] \leftarrow \mathsf{esRABE.Dec}(\mathsf{hsk}'_{i^*,w,b_{i^*}[w]}, \mathsf{sk}'_{i^*,w,b_{i^*}[w]}, \mathsf{att}_{i^*}, \mathsf{ct}'_{w,b_{i^*}[w]}).$$

  Then, output the evaluation result $z \leftarrow \mathsf{oesRABFE.Dec}(\mathsf{hsk}'_{i^*}, \mathsf{sk}'_{i^*}, \mathsf{ct}')$.

Equivocal Public Key. We define the following algorithms:

– $\mathsf{ESetup}(1^\lambda, 1^L, 1^Q, P, C)$: Sample

$$(\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{ek}_{\mathsf{oesRABFE}}) \leftarrow \mathsf{oesRABFE.ESetup}(1^\lambda, 1^L, 1^Q, P, C),$$
$$(\mathsf{crs}_{\mathsf{esRABE}}, \mathsf{ek}_{\mathsf{esRABE}}) \leftarrow \mathsf{esRABE.ESetup}(1^\lambda, 1^L, P).$$

  Output the common reference string and equivocal key

$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{crs}_{\mathsf{esRABE}}),\ \mathsf{ek} = (\mathsf{ek}_{\mathsf{oesRABFE}}, \mathsf{ek}_{\mathsf{esRABE}}).$$

– $\mathsf{EGen}(\mathsf{crs}, i, \mathsf{ek})$: Sample $\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma$ and

$$(\mathsf{pk}'_i, \mathsf{st}'_i) \leftarrow \mathsf{oesRABFE.EGen}(\mathsf{crs}_{\mathsf{oesRABFE}}, i, \mathsf{ek}_{\mathsf{oesRABFE}}),$$
$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ (\mathsf{pk}'_{i,w,b}, \mathsf{st}'_{i,w,b}) \leftarrow \mathsf{esRABE.EGen}(\mathsf{crs}_{\mathsf{esRABE}}, i, \mathsf{ek}_{\mathsf{esRABE}}).$$

  Output the public key and internal state

$$\mathsf{pk}_i = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}}),\ \mathsf{st}_i = (\mathsf{st}'_i, \{\mathsf{st}'_{i,w,b_i[w]}\}_{w\in[\sigma]}, \mathbf{b}_i).$$

– $\mathsf{Equiv}(\mathsf{st}_i, \mathsf{att}_i, \mathsf{ek})$: Run

$$\mathsf{sk}'_i \leftarrow \mathsf{oesRABFE.Equiv}(\mathsf{st}'_i, \mathsf{att}_i, \mathsf{ek}_{\mathsf{oesRABFE}}),$$
$$\forall w \in [n],\ \mathsf{sk}'_{i,w,b_i[w]} \leftarrow \mathsf{esRABE.Equiv}(\mathsf{st}'_{i,w,b_i[w]}, \mathsf{att}_i, \mathsf{ek}_{\mathsf{esRABE}}).$$

  Output the secret key $\mathsf{sk}_i = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]}\}_{w\in[\sigma]}, \mathbf{b}_i)$.

We see that if the underlying schemes esRABE and oesRABFE satisfy the property of equivocal public keys, then so does the above construction.

Correctness and Compactness. The correctness follow the underlying schemes esRABE and oesRABFE. For compactness, by the compactness of oesRABFE and esRABE, we have

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, n, \log L) + 2\sigma \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, P, \log L),$$
$$|\mathsf{hsk}_j| = \mathsf{poly}(\lambda, n, \log L) + 2\sigma \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, P, \log L)$$

where n is the circuit size and $\sigma = \mathsf{poly}(\lambda, n, P, \log L)$. Thus, our construction meets the compactness requirement.

Theorem 12. *Assuming that* esRABE *achieves policy-selective IND security and* oesRABFE *achieves 1-bound non-adaptive SIM security with selective policies, then the above construction achieves 1-bound adaptive SIM security with selective policies as defined in Section 5.1.*

Proof. Note that in this setting, we need to additionally simulate multiple ciphertexts for Q_{Enc} encryption queries. We define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

- $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, P, C)$: This algorithm runs

$$(\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{td}_{\mathsf{oesRABFE}}) \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, P, C),$$
$$(\mathsf{crs}_{\mathsf{esRABE}}, \mathsf{ek}_{\mathsf{esRABE}}) \leftarrow \mathsf{esRABE}.\mathsf{ESetup}(1^\lambda, 1^L, P).$$

  Output the common reference $\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{crs}_{\mathsf{esRABE}})$ and trapdoor $\mathsf{td} = (\mathsf{td}_{\mathsf{oesRABFE}}, \mathsf{ek}_{\mathsf{esRABE}})$.
- $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: This algorithm samples

$$(\mathsf{pk}'_i, \mathsf{st}'_i) \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_{\mathsf{oesRABFE}}, i, \mathsf{td}_{\mathsf{oesRABFE}}),$$
$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ (\mathsf{pk}'_{i,w,b}, \mathsf{st}'_{i,w,b}) \leftarrow \mathsf{esRABE}.\mathsf{EGen}(\mathsf{crs}_{\mathsf{esRABE}}, i, \mathsf{ek}_{\mathsf{esRABE}}).$$

  Output the public key and internal state

$$\mathsf{pk}_i = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}}),\ \mathsf{st}_i = (\mathsf{st}'_i, \{\mathsf{st}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}}).$$

- $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{att}_i, \mathsf{td}, (f, z))$: It simulates

$$\mathsf{sk}'_i \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}'_i, \mathsf{att}_i, \mathsf{td}_{\mathsf{oesRABFE}}, (\perp, \perp))$$

  and for each $w \in [\sigma]$,

$$\mathsf{sk}'_{i,w,b_i[w]} \leftarrow \mathsf{esRABE}.\mathsf{Equiv}(\mathsf{st}'_{i,w,b_i[w]}, \mathsf{att}_i, \mathsf{ek}_{\mathsf{esRABE}}).$$

  where $\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma])$ is sampled as follows:
  - If $f = \perp$ and $z = \perp$, this means that $\mathcal{A}$ makes this corruption query in the pre-challenge phase, or the public key pk_i is not registered into the system, or it satisfies $\mathsf{P}_d(\mathsf{att}_i) = 0$ for all $d \in [Q_{\mathsf{Enc}}]$. Sample

$$\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma.$$

  - Otherwise, $\mathcal{A}$ makes this corruption query in the post-challenge phase, and we have $\mathsf{att}_i = \mathsf{att}^*_i$, $f = \mathsf{C}^*_i$ and $z = \mathsf{C}^*_i(x_d)$ such that $\mathsf{P}_d(\mathsf{att}^*_i) = 1$ for some $d \in [Q_{\mathsf{Enc}}]$. Here, it has known the master public key $\mathsf{mpk} = (\mathsf{mpk}', \{\mathsf{mpk}'_{w,b}\}_{w\in[n],b\in\{0,1\}})$ and the random vector $\mathbf{b}'_i$, so the challenger can simulate

$$\mathsf{ct}'_d \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}', \mathsf{P}_d, (\mathsf{C}^*_i, \mathsf{C}^*_i(x_d))).$$

  Output $\mathsf{sk}_i = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]}\}_{w\in[\sigma]}, \mathbf{b}_i)$, where $b_i[w] = b'_i[w] \oplus \mathsf{ct}'_d[w]$.
- $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_d, (f, z))$: In the challenge phase, the adversary $\mathcal{A}$ specifies L pairs of public keys and circuits $\{(\mathsf{pk}^*_i, \mathsf{C}^*_i)\}_{i\in[L]}$. With such information, the challenger can compute the master public key and helper keys

$$(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}^*_i, \mathsf{C}^*_i)\}_{i\in[L]}).$$

  where $\mathsf{mpk} = (\mathsf{mpk}', \{\mathsf{mpk}'_{w,b}\}_{w\in[n],b\in\{0,1\}})$. Next, to respond the d-th encryption query (P_d, x_d) made by $\mathcal{A}$, it considers the following two cases:
  - In the non-adaptive case, i.e., $f = \mathsf{C}^*_j$ and $z = \mathsf{C}^*_j(x_d)$ for some known corrupted/malicious slot $j \in [L]$, we can rely on the non-adaptive security of oesRABFE by simulating

$$\mathsf{ct}'_d \leftarrow \mathsf{oesRABFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}', \mathsf{P}_d, (\mathsf{C}^*_j, \mathsf{C}^*_j(x_d))).$$

    Then, it generates other ciphertexts of esRABE as in algorithm Enc:

$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ \mathsf{ct}'_{d,w,b} \leftarrow \mathsf{esRABE}.\mathsf{Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}_d, \mathsf{ct}'_d[w]).$$

- In the adaptive case, i.e., $f = \perp$ and $z = \perp$, it samples $\mathbf{b}'_i = (b'_i[1], \ldots, b'_i[\sigma]) \leftarrow \{0,1\}^\sigma$, assuming that $\mathsf{P}_d(\mathsf{att}^*_i) = 1$ and slot i will be corrupted in the post-challenge phase. Then simulate the ciphertexts

$$\forall w \in [\sigma],\ \mathsf{ct}'_{d,w,b'_i[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b'[w]}, \mathsf{P}_d, 0),$$
$$\mathsf{ct}'_{d,w,1-b'_i[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,1-b'[w]}, \mathsf{P}_d, 1).$$

  Finally, output the ciphertext $\mathsf{ct}_d = \{\mathsf{ct}'_{d,w,b}\}_{w\in[\sigma],b\in\{0,1\}}$ as the response to d-th encryption query.

Game Sequence. We prove Theorem 12 via a sequence of games as follows:

- Game_0: Real Game.
- Game_1: Same as Game_0 except that it changes the sampling time of $\mathbf{b}_i$ as in $(\widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.
- $\mathsf{Game}_{2,d}(d \in [Q_{\mathsf{Enc}}])$: Same as $\mathsf{Game}_{2,d-1}$ except that for d-th encryption query made by $\mathcal{A}$, in the adaptive case, it simulates $\mathbf{b}_i$ such that $\mathsf{P}_d(\mathsf{att}_i) = 1$ as in $\widetilde{\mathsf{Gen}}_2$ and ct_d as in $\widetilde{\mathsf{Enc}}$.
- Game_3: Same as $\mathsf{Game}_{2,Q_{\mathsf{Enc}}}$ except that in the adaptive case, it generates ct'_d when $\mathcal{A}$ queries to OCor.
- Game_4: Same as Game_3 except that it generates ct'_d for all $d \in [Q_{\mathsf{Enc}}]$ via the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ of oesRABFE.
- Game_5: Same as Game_4 except that it completely replaces $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc})$ with $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.

We see that in the final game Game_5, each ciphertext ct_d and the corrupted secret key sk_i can be simulated only using $(\mathsf{C}^*_i, \mathsf{C}^*_i(x_d))$. This is exactly the simulation world.

Lemma 19. *We claim that* $\mathsf{Game}_0 \equiv \mathsf{Game}_1$.

Proof. The only difference is the sampling time of the vector $\mathbf{b}_i$. Since $\mathbf{b}_i \leftarrow \{0,1\}^\sigma$ is uniformly sampled, the lemma follows readily. □

Lemma 20. *By the policy-selective IND security of* esRABE, *we claim* $\mathsf{Game}_{2,d-1} \approx_c \mathsf{Game}_{2,d}$.

Proof. We prove this lemma by introducing the following subgames:

- $\mathsf{Game}_{2,d-1,\ell}(\ell \in [\sigma])$: Same as $\mathsf{Game}_{2,d-1,\ell-1}$ except that for d-th encryption query made by $\mathcal{A}$, in the adaptive case, it simulates $(b_i[\ell], \mathsf{ct}'_{d,\ell,0}, \mathsf{ct}'_{d,\ell,1})$ as in $\widetilde{\mathsf{Gen}}_2$ and $\widetilde{\mathsf{Enc}}$, where $b_i[\ell]$ is the ℓ-th bit of the sampled vector $\mathbf{b}_i$ such that $\mathsf{P}_d(\mathsf{att}^*_i) = 1$.

We see that $\mathsf{Game}_{2,d-1} \equiv \mathsf{Game}_{2,d-1,0}$ and $\mathsf{Game}_{2,d-1,\sigma} \equiv \mathsf{Game}_{2,d}$. Therefore, it suffices to prove $\mathsf{Game}_{2,d-1,\ell-1} \approx_c \mathsf{Game}_{2,d-1,\ell}$ via the following lemma:

Lemma 21. *By the policy-selective IND security of* esRABE, $\mathsf{Game}_{2,d-1,\ell-1} \approx_c \mathsf{Game}_{2,d-1,\ell}$ *for each* $\ell \in [\sigma]$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{2,d-1,\ell-1}$ from $\mathsf{Game}_{2,d-1,\ell}$, we can build an algorithm $\mathcal{B}$ to break the IND security of esRABE. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, P, C)$ and a policy P^* from $\mathcal{A}$, where the policy P_d can be efficiently derived from P^* (independent of Q_{Enc}). Then, $\mathcal{B}$ proceeds as follows:

- *Setup phase:* It sends $(1^L, P)$ and the challenge policy P_d to the challenger of esRABE to obtain $\mathsf{crs}_{\mathsf{esRABE}}$. Then, it samples

$$(\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{ek}_{\mathsf{oesRABFE}}) \leftarrow \mathsf{oesRABFE.ESetup}(1^\lambda, 1^L, 1^Q, P, C).$$

  Return $\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{crs}_{\mathsf{esRABE}})$ to $\mathcal{A}$.
- *Pre-challenge phase:* Here, $\mathcal{A}$ can query the oracles below:

- $\mathsf{OGen}(i)$: For $w \in [\sigma]$ and $b \in \{0,1\}$, $\mathcal{B}$ queries the oracle esRABE.OGen on input i to obtain the public key $\mathsf{pk}'_{i,w,b}$. Then, it samples

$$(\mathsf{pk}'_i, \mathsf{st}'_i) \leftarrow \mathsf{oesRABFE.EGen}(\mathsf{crs}_{\mathsf{oesRABFE}}, i, \mathsf{ek}_{\mathsf{oesRABFE}}).$$

  Return $\mathsf{pk} = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}})$ to $\mathcal{A}$. Set $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}'_i$.
- $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: Sample

$$\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma.$$

  For $w \in [\sigma]$, $\mathcal{B}$ queries the oracle esRABE.OCor on input $(i, \mathsf{pk}'_{i,w,b_i[w]}, \mathsf{att})$ to obtain $\mathsf{sk}'_{i,w,b_i[w]}$. Then, by parsing $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}'_i$, it simulates

$$\mathsf{sk}'_i \leftarrow \mathsf{oesRABFE.Equiv}(\mathsf{st}'_i, \mathsf{att}, \mathsf{ek}_{\mathsf{oesRABFE}}).$$

  Return $\mathsf{sk} = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]}\}_{w\in[\sigma]}, \mathbf{b}_i)$ to $\mathcal{A}$.

– *Challenge phase:* $\mathcal{B}$ receives public keys $\{\mathsf{pk}^*_i\}_{i\in[L]}$, attributes $\{\mathsf{att}^*_i\}_{i\in[L]}$, and circuits $\{\mathsf{C}^*_i\}_{i\in[L]}$, where parse $\mathsf{pk}^*_i = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\})$. With such information, $\mathcal{B}$ can compute the challenger can compute the master public key $\mathsf{mpk} = (\mathsf{mpk}', \{\mathsf{mpk}'_{w,b}\}_{w\in[n],b\in\{0,1\}})$ as in algorithm Agg.

– *Post-challenge phase:*

- $\mathsf{OEnc}(\mathsf{P}_i, x_i)$: For $i < d$, $\mathcal{B}$ simulates the ciphertext ct_i as in $\widetilde{\mathsf{Enc}}$; for $i > d$, $\mathcal{B}$ generates the ciphertext ct_i as in Enc. For $i = d$, it generates

$$\mathsf{ct}'_d \leftarrow \mathsf{oesRABFE.Enc}(\mathsf{mpk}', \mathsf{P}_d, x_d).$$

  where we can parse $\mathsf{ct}'_d = (\mathsf{ct}'_d[1], \ldots, \mathsf{ct}'_d[\sigma]) \in \{0,1\}^\sigma$. Then consider the following two cases:
  * In the non-adaptive case, i.e., $\mathcal{B}$ knows a corrupted/malicous slot $j \in [L]$ such that $\mathsf{P}_d(\mathsf{att}^*_j) = 1$, then it generates other ciphertexts of esRABE as in algorithm Enc:

$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ \mathsf{ct}'_{d,w,b} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}_d, \mathsf{ct}'_d[w]).$$

  * In the adaptive case, it samples $\mathbf{b}'_i = (b'_i[1], \ldots, b'_i[\sigma]) \leftarrow \{0,1\}^\sigma$. For all $w < \ell$, it generates the ciphertexts of esRABE:

$$\mathsf{ct}'_{d,w,b'_i[w]\oplus\mathsf{ct}'_d[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,\mathsf{ct}'_d[w]}, \mathsf{P}_d, \mathsf{ct}'_d[w]),$$
$$\mathsf{ct}'_{d,w,1-b'_i[w]\oplus\mathsf{ct}'_d[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,1-\mathsf{ct}'_d[w]}, \mathsf{P}_d, 1-\mathsf{ct}'_d[w]).$$

  For $w = \ell$, $\mathcal{B}$ sends public keys $\{\mathsf{pk}'_{i,\ell,1-b'_i[\ell]\oplus\mathsf{ct}'_d[\ell]}\}_{i\in[L]}$, attributes $\{\mathsf{att}^*_i\}_{i\in[L]}$, and a pair of challenge messages $(1-\mathsf{ct}'_d[\ell], \mathsf{ct}'_d[\ell])$ to the challenger of esRABE; the challenger replies $\mathcal{B}$ with a ciphertext $\mathsf{ct}'_{d,\ell,1-b'_i[\ell]\oplus\mathsf{ct}'_d[\ell]}$. Then, $\mathcal{B}$ generates

$$\mathsf{ct}'_{d,\ell,b'_i[\ell]\oplus\mathsf{ct}'_d[\ell]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{\ell,\mathsf{ct}'[\ell]}, \mathsf{P}_d, \mathsf{ct}'_d[\ell]).$$

  For all $w > \ell$, it generates the ciphertexts of esRABE:

$$\mathsf{ct}'_{d,w,b'_i[w]\oplus\mathsf{ct}'_d[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,\mathsf{ct}'_d[w]}, \mathsf{P}_d, \mathsf{ct}'_d[w]),$$
$$\mathsf{ct}'_{d,w,1-b'_i[w]\oplus\mathsf{ct}'_d[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,1-\mathsf{ct}'_d[w]}, \mathsf{P}_d, \mathsf{ct}'_d[w]).$$

  Return the ciphertext $\mathsf{ct}_d = \{\mathsf{ct}'_{d,w,b}\}_{w\in[\sigma],b\in\{0,1\}}$ to $\mathcal{A}$.
- $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: If $\mathsf{pk} \neq \mathsf{pk}^*_i$ or $\mathsf{P}_j(\mathsf{att}) = 0$ for all $j \in [Q_{\mathsf{Enc}}]$, $\mathcal{B}$ acts as in the pre-challenge phase; otherwise, it means that $\mathsf{pk} \neq \mathsf{pk}^*_i$ and $\mathsf{att} = \mathsf{att}^*_i$ such that $\mathsf{P}_j(\mathsf{att}^*_i) = 1$ for some $j \in [Q_{\mathsf{Enc}}]$.

* If $j < d$, $\mathcal{B}$ can acquire $\mathbf{b}_i$ as in $\widetilde{\mathsf{Gen}}_2$.
* If $j = d$, it finds $\mathbf{b}'_i$ and then sets $b_i[w] = b'_i[w] \oplus \mathsf{ct}'_d[w]$ for $w \in [\sigma]$. Since $\mathbf{b}'_i$ is randomly sampled, $\mathbf{b}_i$ is randomly distributed in the view of $\mathcal{A}$.
* If $j > d$, it can acquire $\mathbf{b}_i$ as in Gen.

For each $w \in [\sigma]$, $\mathcal{B}$ queries the oracle esRABE.OCor on input $(i, \mathsf{pk}'_{i,w,b_i[w]}, \mathsf{att}^*_i)$ to obtain $\mathsf{sk}'_{i,w,b_i[w]}$. Then, by parsing $\mathcal{D}_i[\mathsf{pk}] = \mathsf{st}'_i$, it simulates

$$\mathsf{sk}'_i \leftarrow \mathsf{oesRABFE.Equiv}(\mathsf{st}'_i, \mathsf{att}^*_i, \mathsf{ek}_{\mathsf{oesRABFE}}).$$

Return $\mathsf{sk} = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]}\}_{w\in[\sigma]}, \mathbf{b}_i)$ to $\mathcal{A}$.

We see that if the challenge ciphertext of esRABE encrypts $\mathsf{ct}'_d[\ell]$, $\mathcal{B}$ exactly simulates $\mathsf{Game}_{2,d-1,\ell-1}$; otherwise, it simulates $\mathsf{Game}_{2,d-1,\ell}$. □

Lemma 22. *We claim* $\mathsf{Game}_{2,Q_{\mathsf{Enc}}} \equiv \mathsf{Game}_3$.

Proof. In $\mathsf{Game}_{2,Q_{\mathsf{Enc}}}$, we see that the ciphertexts $\mathsf{ct}'_{d,w,b}$ for $w \in [\sigma]$ exactly encrypt the bit $b \in \{0,1\}$ in the adaptive case. Therefore, the generation of $\mathsf{ct}'_{d,w,b}$ does not require the knowledge of the ciphertext ct' of oesRABFE. With this fact, we can delay the generation of ct' to the time when $\mathcal{A}$ makes the corruption query in the post-challenge phase. This exactly proves $\mathsf{Game}_{2,Q_{\mathsf{Enc}}} \equiv \mathsf{Game}_3$. □

Lemma 23. *By the 1-bound non-adaptive SIM security of* oesRABFE*, we claim that* $\mathsf{Game}_3 \approx_c \mathsf{Game}_4$.

Proof. If $\mathcal{A}$ can distinguish these two games, we can build an algorithm $\mathcal{B}$ to break the 1-bound non-adaptive SIM security of oesRABFE. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, P, C)$ and a policy P^* from $\mathcal{A}$. Then, $\mathcal{B}$ proceeds as follows:

– *Setup phase:* It sends $(1^L, 1^Q, P, C)$ and the policy P^* to the challenger of oesRABFE to obtain $\mathsf{crs}_{\mathsf{oesRABFE}}$. Then, it samples

$$(\mathsf{crs}_{\mathsf{esRABE}}, \mathsf{ek}_{\mathsf{esRABE}}) \leftarrow \mathsf{esRABE.ESetup}(1^\lambda, 1^L, P).$$

Return $\mathsf{crs} = (\mathsf{crs}_{\mathsf{oesRABFE}}, \mathsf{crs}_{\mathsf{esRABE}})$ to $\mathcal{A}$.

– *Pre-challenge phase:* Here, $\mathcal{A}$ can query the oracles below:

- $\mathsf{OGen}(i)$: $\mathcal{B}$ queries the oracle oesRABFE.OGen on input i to obtain the public key pk'_i. Then for each $w \in [\sigma]$ and $b \in \{0,1\}$, it samples

$$(\mathsf{pk}'_{i,w,b}, \mathsf{st}'_{i,w,b}) \leftarrow \mathsf{esRABE.EGen}(\mathsf{crs}_{\mathsf{esRABE}}, i, \mathsf{ek}_{\mathsf{esRABE}}).$$

Return the public key $\mathsf{pk} = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}})$. Set $\mathcal{D}_i[\mathsf{pk}] = \{\mathsf{st}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}}$.

- $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: Sample

$$\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma.$$

Then, $\mathcal{B}$ queries the oracle oesRABFE.OCor on input $(i, \mathsf{pk}'_i, \mathsf{att})$ to obtain sk'_i. Then, by parsing $\mathcal{D}_i[\mathsf{pk}] = \{\mathsf{st}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}}$, it simulates

$$\forall w \in [\sigma],\ \mathsf{sk}'_{i,w,b_i[w]} \leftarrow \mathsf{esRABE.Equiv}(\mathsf{st}'_{i,w,b}, \mathsf{att}, \mathsf{ek}_{\mathsf{esRABE}}).$$

Return $\mathsf{sk} = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]}\}_{w\in[\sigma]}, \mathbf{b}_i)$ to $\mathcal{A}$.

– *Challenge phase:* $\mathcal{B}$ receives public keys $\{\mathsf{pk}^*_i\}_{i\in[L]}$, attributes $\{\mathsf{att}^*_i\}_{i\in[L]}$, and circuits $\{\mathsf{C}^*_i\}_{i\in[L]}$, where parse $\mathsf{pk}^*_i = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\})$. Therefore, $\mathcal{B}$ can compute the master public key $\mathsf{mpk} = (\mathsf{mpk}', \{\mathsf{mpk}'_{w,b}\}_{w\in[n],b\in\{0,1\}})$ as in algorithm Agg. Next, $\mathcal{B}$ forwards challenge information $\{(\mathsf{pk}'_i, \mathsf{att}^*_i, \mathsf{C}^*_i)\}_{i\in[L]}$ to the challenger of oesRABFE.

– *Post-challenge phase:*

- $\mathsf{OEnc}(\mathsf{P}_d, x_d)$: It considers the following two cases:
  * In the non-adaptive case, i.e., $\mathcal{B}$ knows a corrupted/malicous slot $j \in [L]$ such that $\mathsf{P}_d(\mathsf{att}^*_j) = 1$, then it queries the oracle oesRABFE.OEnc on input (P_d, x_d) to obtain a ciphertext ct'_d. Then generate other ciphertexts of esRABE:

$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ \mathsf{ct}'_{d,w,b} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}_d, \mathsf{ct}'_d[w]).$$

  * In the adaptive case, it samples $\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma$ and generates the ciphertexts of esRABE:

$$\begin{aligned} \forall w \in [\sigma],\ \mathsf{ct}'_{d,w,b_i[w]} &\leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}_d, 0), \\ \mathsf{ct}'_{d,w,1-b_i[w]} &\leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}_d, 1). \end{aligned}$$

  Return the ciphertext $\mathsf{ct}_d = \{\mathsf{ct}'_{d,w,b}\}_{w\in[\sigma],b\in\{0,1\}}$ to $\mathcal{A}$.
- $\mathsf{OCor}(i, \mathsf{pk}, \mathsf{att})$: If $\mathsf{pk} \neq \mathsf{pk}^*_i$ or $\mathsf{P}_j(\mathsf{att}) = 0$ for all $j \in [Q_{\mathsf{Enc}}]$, $\mathcal{B}$ acts as in the pre-challenge phase; otherwise, it means that $\mathsf{pk} \neq \mathsf{pk}^*_i$ and $\mathsf{att} = \mathsf{att}^*_i$ such that $\mathsf{P}_d(\mathsf{att}^*_i) = 1$ for some $d \in [Q_{\mathsf{Enc}}]$.
  In the latter case, $\mathcal{B}$ queries the oracle oesRABFE.OCor on input $(i, \mathsf{pk}'_i, \mathsf{att}^*_i)$ to obtain sk'_i. Then, it queries the oracle oesRABFE.OEnc on input (P_d, x_d) to obtain a ciphertext ct'_d. By parsing $\mathcal{D}_i[\mathsf{pk}] = \{\mathsf{st}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}}$, it sets

$$\forall w \in [\sigma],\ \mathsf{sk}'_{i,w,b_i[w]} \leftarrow \mathsf{esRABE.Equiv}(\mathsf{st}'_{i,w,b_i[w]\oplus\mathsf{ct}'_d[w]}, \mathsf{att}^*_i, \mathsf{ek}_{\mathsf{esRABE}}).$$

  Return $\mathsf{sk} = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]\oplus\mathsf{ct}'_d[w]}\}_{w\in[\sigma]}, \mathbf{b}_i)$ to $\mathcal{A}$, where $\mathbf{b}_i$ has been sampled in the challenge phase.

We see that in the real world of oesRABFE, $\mathcal{B}$ exactly simulates Game_3; otherwise, it simulates Game_4. Therefore, the lemma follows readily. □

Lemma 24. *We claim* $\mathsf{Game}_4 \equiv \mathsf{Game}_5$.

Proof. In both non-adaptive and adaptive case, we see that Game_4 completely employs $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$, which is identical to Game_5. Therefore, the lemma follows readily. □

B Static Slotted Reg-FE with Non-Adaptive Security

In this section, we present a generic construction of static bounded collusion sReg-FE for circuits, achieving non-adaptive SIM security against $Q \geq 1$ malicious/corrupted users. In particular, we start with an sReg-ABE for simple policies and then leverage it to construct a 1-bound sReg-FE for circuits. We further upgrade 1-bound security into Q-bound security via a reusable dynamic MPC protocol.

B.1 Slotted Reg-ABE

We recap the notion of slotted Reg-ABE (sReg-ABE for short) [HLWW23,CHW25a,WW25], which serves as the core building block of all Reg-FE schemes proposed in this work. In particular, an sReg-ABE for policy family $P : A \to \{0,1\}$ works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, P) \to \mathsf{crs}$. It takes as input the security parameter 1^λ, the upper bound 1^L on the number of user slots, and the description of policy family P, then outputs a common reference string crs.

- $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. It takes as input the common reference string crs, a slot index $i \in [L]$, and an attribute $\mathsf{att}_i \in A$, then outputs a public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
- $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i, \mathsf{att}_i) \to 0/1$. It takes as input the common reference string crs, a slot index $i \in [L]$, a public key pk_i, and an attribute $\mathsf{att}_i \in A$, then outputs a bit indicating whether pk_i is valid.
- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{att}_i)\}_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$. It takes as input the common reference string crs and a series of public keys and attributes $(\mathsf{pk}_i, \mathsf{att}_i)$ for $i \in [L]$, then outputs the master public key mpk and a series of helper secret keys hsk_j for all $j \in [L]$. This algorithm is deterministic.
- $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, \mathsf{m}) \to \mathsf{ct}$. It takes as input the master public key mpk, a policy $\mathsf{P} \in P$, and a message $\mathsf{m} \in M$, then outputs a ciphertext ct.
- $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct}) \to \mathsf{m}/\perp$. It takes as input a helper key hsk_{i^*}, a secret key sk_{i^*}, an attribute $\mathsf{att}_{i^*} \in A$, and a ciphertext ct, then outputs a message $\mathsf{m} \in M$ or an empty symbol $\perp$.

The above scheme is required to satisfy:

- **Compactness.** For all $\lambda, L \in \mathbb{N}$, all P, and all $i \in [L]$, it holds that

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, P, \log L) \text{ and } |\mathsf{hsk}_i| = \mathsf{poly}(\lambda, P, \log L).$$

  On the other hand, the runtime of algorithms $(\mathsf{Setup}, \mathsf{Gen})$ is upper bounded by $\mathsf{poly}(\lambda, P)$ (ignoring the factor related to L).
- **Completeness.** For all $\lambda, L \in \mathbb{N}$, all P, all attributes $\mathsf{att} \in A$, and all slot indices $i \in [L]$, we have

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i, \mathsf{att}) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, P) \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}) \end{array}\right] = 1.$$

- **Correctness.** For all $\lambda, L \in \mathbb{N}$, all P, all $i^* \in [L]$ and all $\mathsf{att}_{i^*} \in A$, all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, P)$, all $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i^*, \mathsf{att}_{i^*})$, all $\{\mathsf{pk}_i\}_{i\in[L]\setminus\{i^*\}}$ such that $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$, all policies $\mathsf{P} \in P$ such that $\mathsf{P}(\mathsf{att}_{i^*}) = 1$ and all messages $\mathsf{m} \in M$, and computing $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{att}_i)\}_{i\in[L]})$, we have

$$\Pr\left[\mathsf{Dec}\left(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, \mathsf{m})\right) = \mathsf{m}\right] = 1.$$

- **Policy-Selective IND Security.** For all efficient adversaries $\mathcal{A}$ invoked with security parameter $\lambda \in \mathbb{N}$, the experiment is defined as follows:
  - *Setup phase:* Initially, $\mathcal{A}$ specifies the input $(1^L, P)$ and a challenge policy $\mathsf{P}^* \in P$. The challenger samples a common reference string $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, P)$. Return crs to $\mathcal{A}$.
  - *Pre-challenge phase:* The challenger initializes a dictionary $\{\mathcal{D}_i = \emptyset\}_{i\in[L]}$ and a set $C = \emptyset$. In this phase, $\mathcal{A}$ can freely make key generation/corruption queries to the following oracles:
    * $\mathsf{OGen}(i, \mathsf{att})$: The challenger runs $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att})$ and sets $\mathcal{D}_i[\mathsf{pk}] = (\mathsf{sk}, \mathsf{att})$. Return pk to $\mathcal{A}$.
    * $\mathsf{OCor}(i, \mathsf{pk})$: Parse $(\mathsf{sk}, \mathsf{att}) = \mathcal{D}_i[\mathsf{pk}]$. The challenger returns the secret key sk to $\mathcal{A}$ and updates $C = C \cup \{(i, \mathsf{pk}, \mathsf{att})\}$.
  - *Challenge phase:* In this phase, $\mathcal{A}$ specifies L pairs of public keys and attributes $\{(\mathsf{pk}_i^*, \mathsf{att}_i^*)\}_{i\in[L]}$ together with a pair of challenge messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$. The challenger computes

$$(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i^*, \mathsf{att}_i^*)\}_{i\in[L]}).$$

    Then the challenger checks: for each slot $i \in [L]$, it must fall in one of the following three cases:

$$\begin{aligned} &\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \notin C && && \text{(honest slot)} \\ &\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \in C \implies && \mathsf{P}^*(\mathsf{att}_i^*) = 0 && \text{(corrupted slot)} \\ &\mathcal{D}_i[\mathsf{pk}_i^*] = \perp && \implies \mathsf{P}^*(\mathsf{att}_i^*) = 0 && \text{(malicious slot)} \end{aligned}$$

It aborts if the above condition is not satisfied; otherwise, the challenger flips a random coin $b \in \{0,1\}$ and replies $\mathcal{A}$ with a challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}^*, \mathsf{m}_b^*)$.

- *Post-challenge phase:* $\mathcal{A}$ can make corruption queries as in the pre-challenge phase. For each query (i, pk_i^*) to OCor, it requires $\mathsf{P}^*(\mathsf{att}_i^*) = 0$.
- *Output phase:* $\mathcal{A}$ outputs a bit b' as the guess of b.

We say that a sReg-ABE scheme achieves policy-selective IND security if $|\Pr[b' = b] - \frac{1}{2}|$ is negligible in λ.

In this work, we consider sReg-ABE schemes for two types of policies: simple policy and NC^1 policy. Here, a simple policy accepts all attributes, i.e., $\mathsf{P}(\mathsf{att}) = 1$ for all $\mathsf{att} \in A$. An NC^1 policy can express an arbitrary circuit with shallow depth.

Theorem 13 (Existing Constructions of sReg-ABE [ZZGQ23,AT24,CHW25a,WW25]). *Under the standard k-Lin assumption [EHK+13] or the succinct LWE assumption [Wee24], there exist sReg-ABE schemes for simple policies with policy-selective IND security. Furthermore, under the same assumptions, there also exist sReg-ABE schemes for NC^1 policies with policy-selective IND security.*

B.2 From Slotted Reg-ABE to 1-Bound Slotted Reg-FE

For some circuit family $C : X \to Z$ of circuit size n, let $\mathsf{U}(\cdot,\cdot)$ be the universal circuit such that $\mathsf{U}(\mathsf{C}, x) = \mathsf{C}(x)$ for any circuit $\mathsf{C} \in C$ and input $x \in X$. Assuming a garbled circuit scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$ and an sReg-ABE scheme for simple policies $\mathsf{sRABE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, then our 1-bound sReg-FE for circuit family C works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$: It sets the collusion bound $Q = 1$ as default. Then, it proceeds as follows:
  - Define the simple policy family $P : \{0,1\} \to \{0,1\}$, where it just sets the attribute space as $\{0,1\}$. Run
$$\mathsf{crs}_{\mathsf{sRABE}} \leftarrow \mathsf{sRABE.Setup}(1^\lambda, 1^L, P).$$
  - For all $i \in [L]$, $w \in [n]$ and $b \in \{0,1\}$, it samples dummy public keys $\widehat{\mathsf{pk}}_{i,w,b} \leftarrow \mathsf{sRABE.Gen}(\mathsf{crs}_{w,b}, i, 1)$ (omit secret keys).

  Output the common reference string
$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{sRABE}}, \{\widehat{\mathsf{pk}}_{i,w,b}\}_{i\in[L],w\in[n],b\in\{0,1\}}).$$

– $\mathsf{Gen}(\mathsf{crs}, i)$: Fix $\mathsf{att}_i = 1$ and sample $(\mathsf{pk}_{i,w,b}, \mathsf{sk}_{i,w,b}) \leftarrow \mathsf{sRABE.Gen}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{att}_i)$ for all $w \in [n]$ and $b \in \{0,1\}$. Output
$$\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}},\ \mathsf{sk}_i = \{\mathsf{sk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}.$$

– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i)$: For all $w \in [n]$ and $b \in \{0,1\}$, it checks whether
$$\mathsf{sRABE.Ver}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{pk}_{i,w,b}, 1) \stackrel{?}{=} 1.$$
  If not, abort and output 0; otherwise, output 1.

– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: Parse each public key $\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$ and ciruict $\mathsf{C}_i = (\mathsf{C}_i[1], \ldots, \mathsf{C}_i[n]) \in \{0,1\}^n$, then it proceeds as follows:
  - For all $i \in [L]$, $w \in [n]$ and $b \in \{0,1\}$, set
$$\overline{\mathsf{pk}}_{i,w,b} := \begin{cases} \mathsf{pk}_{i,w,b}, & \text{when } \mathsf{C}_i[w] = b, \\ \widehat{\mathsf{pk}}_{i,w,b}, & \text{otherwise.} \end{cases} \tag{13}$$

- Run $(\mathsf{mpk}_{w,b}, \{\mathsf{hsk}_{j,w,b}\}_{j\in[L]}) \leftarrow \mathsf{sRABE.Agg}(\mathsf{crs}_{\mathsf{sRABE}}, \{(\overline{\mathsf{pk}}_{i,w,b}, 1)\}_{i\in[L]})$.

Output

$$\mathsf{mpk} = \{\mathsf{mpk}_{w,b}\}_{w\in[n],b\in\{0,1\}},\ \mathsf{hsk}_j = (\mathsf{C}_j, \{\mathsf{hsk}_{j,w,\mathsf{C}_j[w]}\}_{w\in[n]}).$$

– $\mathsf{Enc}(\mathsf{mpk}, x)$: Let $\mathsf{U}[x]$ be a universal circuit with x hard-wired. Run

$$(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{U}[x]).$$

Let P be a policy that simply outputs 1, then it runs

$$\forall w \in [n] \text{ and } b \in \{0,1\},\ \mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}, \mathsf{lab}_{w,b}).$$

Output the ciphertext $\mathsf{ct} = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct})$: For all $w \in [n]$, recover the labels

$$\mathsf{lab}_{w,\mathsf{C}_{i^*}[w]} \leftarrow \mathsf{sRABE.Dec}(\mathsf{hsk}_{i^*,w,\mathsf{C}_{i^*}[w]}, \mathsf{sk}_{i^*,w,\mathsf{C}_{i^*}[w]}, \mathsf{att}_{i^*}, \mathsf{ct}_{w,\mathsf{C}_{i^*}[w]}).$$

where $\mathsf{att}_{i^*} = 1$. Output $z \leftarrow \mathsf{GC.Eval}(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,\mathsf{C}_{i^*}[w]}\}_{w\in[n]})$.

Completeness and Compactness. The completeness follows the underlying sRABE. For compactness, by the compactness of sRABE, we have

$$|\mathsf{mpk}| = 2n \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, \log L),$$
$$|\mathsf{hsk}_j| = 2n \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, \log L),$$

where P can be some constant and n is the circuit size. Thus, our construction meets the compactness requirement.

Correctness. In algorithm Dec, we have $\mathsf{sk}_{i^*} = (\{\mathsf{sk}_{i^*,w,b}\}_{w\in[n],b\in\{0,1\}})$ and

$$\mathsf{ct} = \left(\tilde{\mathsf{U}}, \begin{pmatrix} \mathsf{sRABE.Enc}(\mathsf{mpk}_{1,0}, \mathsf{P}, \mathsf{lab}_{1,0}) & \cdots & \mathsf{sRABE.Enc}(\mathsf{mpk}_{n,0}, \mathsf{P}, \mathsf{lab}_{n,0}) \\ \mathsf{sRABE.Enc}(\mathsf{mpk}_{1,1}, \mathsf{P}, \mathsf{lab}_{1,1}) & \cdots & \mathsf{sRABE.Enc}(\mathsf{mpk}_{n,1}, \mathsf{P}, \mathsf{lab}_{n,1}) \end{pmatrix}\right).$$

By Eq. (13), we see that $\mathsf{sk}_{i^*,w,b}$ is the real secret key of $\overline{\mathsf{pk}}_{i^*,w,b}$ only when $\mathsf{C}_{i^*}[w] = b$. Note that the dummy public keys $\widetilde{\mathsf{pk}}_{i,w,b}$ are also the normal public keys of equivocal sReg-ABE, which are only employed to fill slot gaps and will not affect the decryption correctness. Thus, user i^* can use the correctness of sRABE to recover $\{\mathsf{lab}_{i^*,w,\mathsf{C}_{i^*}[w]}\}_{w\in[n]}$. Then, by the correctness of garbled circuits, we have $\mathsf{GC.Eval}(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,\mathsf{C}_{i^*}[w]}\}_{w\in[n]}) = \mathsf{U}(\mathsf{C}_{i^*}, x) = \mathsf{C}_{i^*}(x)$.

Theorem 14. *Assuming* GC *is secure and* sRABE *achieves policy-selective IND security, our construction achieves 1-bound non-adaptive SIM security.*

Proof. The proof is analogous to [ZCHZ24], so we just define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1\ \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: This algorithm is identical to Setup (set td $=\perp$).
– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: This algorithm is identical to Gen.
– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}, \mathsf{td}, (\perp, \perp))$: This algorithm directly outputs the secret key $\mathsf{sk} = \mathsf{st}$.
– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, (f, z))$: In the challenge phase, it considers two cases:
  - If there does not exist any corrupt/malicious slot, it must have $f = \perp$ and $z = \perp$, so the challenger generates $\mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}, \mathsf{m}_{w,b})$ for each $w \in [n]$ and $b \in \{0,1\}$, where P is a simple policy and $\mathsf{m}_{w,b} \leftarrow \{0,1\}^\lambda$.
  - Otherwise, we have $f = \mathsf{C}^*_j$ and $z = \mathsf{C}^*_j(x)$ for some corrupt/malicious slot j. Then, it samples

$$\forall w \in [n],\ \mathsf{lab}_{w,1-\mathsf{C}^*_j[w]} \leftarrow \{0,1\}^\lambda,$$
$$\left(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,\mathsf{C}^*_j[w]}\}_{w\in[n]}\right) \leftarrow \mathsf{GC.}\widetilde{\mathsf{Garble}}(1^\lambda, \mathsf{C}^*_j(x)).$$

  For each $w \in [n]$ and $b \in \{0,1\}$, run $\mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}, \mathsf{lab}_{w,b})$.

Output the challenge ciphertext $\mathsf{ct}^* = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$.

B.3 From 1-Bound Security to Q-Bound Security

For some circuit family $C : X \to Z$ of circuit size n, let $t = \Theta(Q^2\lambda), m = \Theta(t), v = \Theta(Q\lambda), p_1 = \Theta(m), p_2 = \Theta(v), N = \Theta(Q^2\lambda p_1)$ and $S = S(Q\lambda p_2)$, where p_1 and p_2 are primes. For any circuit $\mathsf{C} \in C$ and set $\Delta \subseteq [S]$ of size v, we define a circuit family G in which each circuit $\mathsf{G} \in G$ with input $\widehat{x}$ can compute

$$\mathsf{G}(\widehat{x}) = \mathsf{RDMPC.Local}((\mathsf{C}, \Delta), \widehat{x}) \tag{14}$$

where (C, Δ) is the constant hard-wired in G. Let $n' = \mathrm{poly}(\lambda, n, Q)$ denote the circuit size of G.

Assuming a reusable dynamic MPC scheme $\mathsf{RDMPC} = (\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode})$ and a 1-bound sReg-FE (with non-adaptive security) $\mathsf{osRFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, then our sReg-FE for circuit family C works as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$:
  - Sample
  $$\forall k \in [N],\ \mathsf{crs}_k \leftarrow \mathsf{osRFE.Setup}(1^\lambda, 1^L, 1, G).$$
  - Sample dummy public keys $\widehat{\mathsf{pk}}_{i,k} \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}_k, i)$ for each $i \in [L]$ and $k \in [N]$.
  - Let $\mathsf{Hash}_1 : \{0,1\}^* \to \mathbb{Z}_{p_1}^\lambda$ and $\mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{Z}_{p_2}^\lambda$ be two collision-resistant hash function families. Sample
  $$\mathsf{hk}_1 \leftarrow \mathsf{Hash}_1.\mathsf{Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash}_2.\mathsf{Gen}(1^\lambda).$$
  - Define two injective maps $\rho_1 : [Dt+1] \times \mathbb{Z}_{p_1} \to [N]$ and $\rho_2 : [Dt+1] \times \mathbb{Z}_{p_2} \to [S]$.

  Output $\mathsf{crs} = (\{\mathsf{crs}_k\}_{k\in[N]}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$.
- $\mathsf{Gen}(\mathsf{crs}, i)$: Sample
$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}_k, i).$$
  Output the public key $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ and secret key $\mathsf{sk}_i = \{\mathsf{sk}_{i,k}\}_{k\in[N]}$.
- $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i)$: It check whether
$$\forall k \in [N],\ \mathsf{osRFE.Ver}(\mathsf{crs}_k, i, \mathsf{pk}_{i,k}) \stackrel{?}{=} 1.$$
  If not, abort and output 0; otherwise, output 1.
- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: Parse $\mathsf{pk}_i = (\{\mathsf{pk}_{i,k}\}_{k\in[N]})$. It proceeds as follows:
  - For $i \in [L]$, it computes coefficients $(a_{i,0}, \ldots, a_{i,\lambda-1}) = \mathsf{Hash}_1.\mathsf{H}(\mathsf{hk}_1, i||\mathsf{pk}_i)$ and hence defines a polynomial function of degree $\lambda - 1$:
  $$f_i(x) = a_{i,\lambda-1}x^{\lambda-1} + \cdots + a_{i,1}x + a_{i,0}.$$
  For each $j \in [m]$, evaluate the point $f_i(j)$ and then set $\gamma_{i,j} = \rho_1(j, f_i(j))$. Set $\Gamma_i = \{\gamma_{i,j}\}_{j\in[m]}$.
  - For $i \in [L]$, it computes coefficients $(c_{i,0}, \ldots, c_{i,v-1}) = \mathsf{Hash}_2.\mathsf{H}(\mathsf{hk}_2, i||\mathsf{pk}_i)$ and hence defines a polynomial function of degree $\lambda - 1$:
  $$g_i(x) = c_{i,\lambda-1}x^{\lambda-1} + \cdots + c_{i,1}x + c_{i,0}.$$
  For each $j \in [v]$, evaluate the point $g_i(j)$ and then set $\delta_{i,j} = \rho_2(j, g_i(j))$. Set $\Delta_i = \{\delta_{i,j}\}_{j\in[v]}$.
  - For each $i \in [L]$ and $k \in [N]$, define circuits $\mathsf{G}_{i,k} = (\mathsf{G}_{i,k}[1], \ldots, \mathsf{G}_{i,k}[n']) \in \{0,1\}^{n'}$ as in Eq. (14), where $\mathsf{G}_{i,k}$ takes $\widehat{\mathsf{C}}_{i,k} = (\mathsf{C}_i, \Delta_i)$ as constant. Note that
  $$(\widehat{\mathsf{C}}_{i,1}, \ldots, \widehat{\mathsf{C}}_{i,N}) = \mathsf{RDMPC.CktEnc}(1^\lambda, 1^Q, \mathsf{C}_i, \Delta_i).$$
  - For all $i \in [L]$ and $k \in [N]$, set
  $$\overline{\mathsf{pk}}_{i,k} := \begin{cases} \mathsf{pk}_{i,k}, & \text{when } k \in \Gamma_i, \\ \widehat{\mathsf{pk}}_{i,k}, & \text{otherwise.} \end{cases}$$

- For all $k \in [N]$, run

$$(\mathsf{mpk}_k, \{\mathsf{hsk}_{j,k}\}_{j\in[L]}) = \mathsf{osRFE.Agg}(\mathsf{crs}_k, \{(\overline{\mathsf{pk}}_{i,k}, \mathsf{G}_i)\}_{i\in[L]}).$$

Output $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$ and $\mathsf{hsk}_j = (\Gamma_j, \{\mathsf{hsk}_{j,k}\}_{k\in\Gamma_j})$.

– $\mathsf{Enc}(\mathsf{mpk}, x)$: Parse $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$, then compute

$$(\widehat{x}_1, \ldots, \widehat{x}_N) \leftarrow \mathsf{RDMPC.InpEnc}(1^\lambda, 1^Q, x),$$
$$\forall k \in [N],\ \mathsf{ct}_k \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k).$$

Output $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct})$: Parse $\mathsf{hsk}_{i^*} = (\Gamma_{i^*}, \{\mathsf{hsk}_{i^*,k}\}_{k\in\Gamma_{i^*}})$, $\mathsf{sk}_{i^*} = \{\mathsf{sk}_{i^*,k}\}_{k\in[N]}$. Compute

$$\forall k \in \Gamma_{i^*},\ z_k \leftarrow \mathsf{osRFE.Dec}(\mathsf{hsk}_{i^*,k}, \mathsf{sk}_{i^*,k}, \mathsf{ct}_k).$$

Output $z = \mathsf{RDMPC.Decode}(\{z_k\}_{k\in\Gamma_{i^*}}, \Gamma_{i^*})$.

Completeness, Compactness and Correctness. The completeness follows the underlying osRFE scheme. By the compactness of osRFE, we have

$$|\mathsf{mpk}| = N \cdot \mathsf{poly}(\lambda, n', \log L) = \mathsf{poly}(\lambda, n, Q, \log L),$$
$$|\mathsf{hsk}_j| = N \cdot \mathsf{poly}(\lambda, n', \log L) = \mathsf{poly}(\lambda, n, Q, \log L),$$

where N depends on the collusion bound Q, and $n' = \mathsf{poly}(\lambda, n, Q)$. Thus, our construction satisfies the compactness requirement.

By the correctness of osRFE, the decryption algorithm first recovers the local encodings

$$\forall k \in \Gamma_{i^*},\ z_k = \widehat{y}_k = \mathsf{RDMPC.Local}((\mathsf{C}_{i^*}, \Delta_{i^*}), \widehat{x}_k).$$

By the correctness of RDMPC, we have $\mathsf{RDMPC.Decode}(\{\widehat{y}_k\}_{k\in\Gamma_{i^*}}, \Gamma_{i^*}) = \mathsf{C}_{i^*}(x)$.

Theorem 15. *Assuming* osRFE *achieves 1-bound non-adaptive SIM security, the above construction achieves Q-bound non-adaptive SIM security.*

Proof. Suppose the adversary $\mathcal{A}$ colludes with Q corrupted/malicious users whose slot indices are stored in the set $\mathcal{K} \subseteq [L]$, then define the following set

$$\mathcal{T} = \bigcup_{i\in\mathcal{K}, i\in\mathcal{K}\setminus\{j\}} (\Gamma_i^* \cap \Gamma_j^*)$$

where Γ_i^* is the pairwise set that is registered under slot i. With a challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_1^*, \ldots, \mathsf{ct}_N^*)$ and corrupted/malicious secret keys $\{\mathsf{sk}_j\}_{j\in\mathcal{K}}$, for each $k \in [N]$, the proof is organized as follows:

– If $k \notin \mathcal{T}$, it means there exists at most a slot $i \in \mathcal{K}$ such that $k \in \Gamma_i^*$ and $k \notin \Gamma_j^*$ for any $j \in \mathcal{K} \setminus \{i\}$. In particular, ct_k^* can only be decrypted using sk_i. Thus, it can rely on the 1-bound security of osRFE;
– Otherwise, it means that $\mathcal{A}$ holds more than one secret keys that correctly decrypt ct_k^*. In this way, 1-bound security would be compromised, and the security would in turn rely on the underlying RDMPC scheme.

Then, we define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: This algorithm is identical to Setup, except that it runs

$$\forall k \in [N],\ (\mathsf{crs}_k, \mathsf{td}_k) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, G),$$
$$\forall i \in [L] \text{ and } k \in [N],\ (\widehat{\mathsf{pk}}_{i,k}, \widehat{\mathsf{st}}_{i,k}) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i).$$

Output $\mathsf{crs} = (\{\mathsf{crs}_k\}_{k\in[N]}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$ and $\mathsf{td} = \{\mathsf{td}_k\}$.

– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: This algorithm samples

$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k).$$

Output $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ and $\mathsf{st}_i = \{\mathsf{st}_{i,k}\}_{k\in[N]}$.

– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{td}, (\perp, \perp))$: This algorithm samples

$$\forall k \in [N],\ \mathsf{sk}_{i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\perp, \perp)).$$

Output $\mathsf{sk}_i = \{\mathsf{sk}_{i,k}\}_{k\in[N]}$.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}^*_j, \mathsf{C}^*_j(x^*))\}_{j\in\mathcal{K}})$: In the non-adaptive setting, we can assume that the challenger has known sets $\{(\Gamma^*_j, \Delta^*_j)\}_{j\in\mathcal{K}}$, where $|\mathcal{K}| = Q$.

- It simulates the following input and local encodings

$$(\{\widehat{x}^*_k\}_{k\in\mathcal{T}}, \{\widehat{y}^*_{j,k}\}_{j\in\mathcal{K}, k\in\Gamma^*_j}) \leftarrow \mathsf{RDMPC}.\widetilde{\mathsf{InpEnc}}(1^{|x^*|}, \mathcal{T}, \{(\mathsf{C}^*_j, \mathsf{C}^*_j(x^*))\}_{j\in\mathcal{K}})$$

where it requires $\widehat{y}^*_{j,k} = \mathsf{RDMPC.Local}(\widehat{\mathsf{C}}^*_{j,k}, \widehat{x}^*_k)$ for all $k \in \Gamma^*_j \cap \mathcal{T}$.

- For all $k \in \mathcal{T}$, it generates

$$\mathsf{ct}^*_k \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}^*_k).$$

- For all $k \notin \mathcal{T}$, if there is a corrupted/malicious slot $i \in \mathcal{K}$ such that $k \in \Gamma_i$ and $k \notin \Gamma_j$ for any $j \in \mathcal{K} \setminus \{i\}$, the it simulates

$$\mathsf{ct}^*_k \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\mathsf{G}_{c_i}, \widehat{y}^*_{c_i,k})).$$

Otherwise, it simulates $\mathsf{ct}^*_k \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\perp, \perp))$.

Finally, output the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}^*_1, \ldots, \mathsf{ct}^*_N)$.

Game Sequence. We prove Theorem 15 via a sequence of games as follows:

– Game_0: Real Game.

– Game_1: Same as Game_0 except that if $\left|\bigcup_{i\in\mathcal{K}, j\in\mathcal{K}\setminus\{j\}} (\Gamma^*_i \cap \Gamma^*_j)\right| \geq t$, it aborts and outputs $\perp$.

– Game_2: Same as Game_1 except that for all $i \in \mathcal{K}$, if $\Delta^*_i \setminus \left(\bigcup_{j\in\mathcal{K}\setminus\{i\}} \Delta^*_j\right) = \emptyset$, it aborts and outputs $\perp$.

– $\mathsf{Game}_{3,\ell}(\ell \in [N])$: Same as $\mathsf{Game}_{3,\ell-1}$ except that for all $k \leq \ell$, it simulates $(\mathsf{crs}_k, \widehat{\mathsf{pk}}_{i,k})$ as in $\widetilde{\mathsf{Setup}}$, $(\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k})$ as in $(\widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2)$, and ct^*_k for $k \notin \mathcal{T}$ as in $\widetilde{\mathsf{Enc}}$.

– Game_4: Same as $\mathsf{Game}_{3,N}$ except that it completely replaces $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc})$ with $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.

We see that in the final game Game_4, the challenge ciphertext ct^* can be simulated only using $\{(\mathsf{C}^*_i, \mathsf{C}^*_i(x^*))\}_{i\in\mathcal{K}}$. This is exactly the simulation world.

Lemma 25. *By the collision resistance of* Hash_1, *we claim* $\mathsf{Game}_0 \approx_s \mathsf{Game}_1$.

Proof. The proof is analogous to Lemma 6, so we omit details here. □

Lemma 26. *By the collision resistance of* Hash_2, *we claim* $\mathsf{Game}_1 \approx_s \mathsf{Game}_2$.

Proof. The proof is analogous to Lemma 7, so we omit details here. □

Lemma 27. *By the 1-bound non-adaptive security of* osRFE, *we claim* $\mathsf{Game}_{3,\ell-1} \approx_c \mathsf{Game}_{3,\ell}$ *for each* $\ell \in [N]$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{3,\ell-1}$ from $\mathsf{Game}_{3,\ell}$, we can build an algorithm $\mathcal{B}$ to break the 1-bound non-adaptive SIM security of osRFE. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, C)$ from $\mathcal{A}$. Then, $\mathcal{B}$ proceeds as follows:

- *Setup phase:* It sends $(1^L, 1, G)$ to the challenger of osRFE to obtain crs_ℓ. It queries the oracle osRFE.OGen to obtain $\widehat{\mathsf{pk}}_{i,\ell}$ for all $i \in [L]$ and then proceeds as follows:
  - For $k < \ell$, it simulates $(\mathsf{crs}_k, \mathsf{td}_k) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, G)$ and dummy keys $(\widehat{\mathsf{pk}}_{i,k}, \widehat{\mathsf{st}}_{i,k}) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i\mathsf{td}_k)$ for all $i \in [L]$.
  - For $k > \ell$, it generates $\mathsf{crs}_k \leftarrow \mathsf{osRFE.Setup}(1^\lambda, 1^L, G)$ and dummy keys $(\widehat{\mathsf{pk}}_{i,k}, \widehat{\mathsf{sk}}_{i,k}) \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}_k, i)$ for all $i \in [L]$.

  Then, it selects hash keys $(\mathsf{hk}_1, \mathsf{hk}_2)$ and injective maps (ρ_1, ρ_2) as usual. Return the common reference string $\mathsf{crs} = (\{\mathsf{crs}_k\}_{k\in[N]}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$ to $\mathcal{A}$.
- *Pre-challenge phase:*
  - $\mathsf{OGen}(i)$: $\mathcal{B}$ queries the oracle osRFE.OGen to obtain $\mathsf{pk}_{i,\ell}$, then computes

$$\forall k < \ell,\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k),$$
$$\forall k > \ell,\ (\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}_k, i).$$

    Return $\mathsf{pk} = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ to $\mathcal{A}$.
  - $\mathsf{OCor}(i, \mathsf{pk})$: $\mathcal{B}$ queries the oracle osRFE.OCor to obtain $\mathsf{sk}_{i,\ell}$, then simulates

$$\forall k < \ell,\ \mathsf{sk}_{i,k} \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\perp, \perp)).$$

    Return $\mathsf{sk} = \{\mathsf{sk}_{i,k}\}_{k\in[N]}$ to $\mathcal{A}$.
- *Challenge phase:* It receives public keys $\{\mathsf{pk}_i^*\}_{i\in[L]}$, circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$ and a challenge message x^*. $\mathcal{B}$ can now learn all sets $\Gamma_1^*, \ldots, \Gamma_L^*, \Delta_1^*, \ldots, \Delta_L^*$ from these public keys. Besides, it acquires circuits $\{\mathsf{G}_i^*\}_{i\in[L]}$. Next, $\mathcal{B}$ runs

$$(\widehat{x}_1^*, \ldots, \widehat{x}_N^*) \leftarrow \mathsf{RDMPC.InpEnc}(1^\lambda, 1^Q, x^*).$$

  Then, $\mathcal{B}$ selects public keys $\{\overline{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}$ as in Eq. (13) and computes the master public key $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$ as in algorithm Agg. Since $\mathcal{B}$ has known the set of all corrupted/malicious slots $\mathcal{K}$, it can learn $\mathcal{T}$ and generate:
  - For $k < \ell$, if $k \in \mathcal{T}$, $\mathcal{B}$ directly computes $\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k^*)$; otherwise, if there is a slot i such as $k \in \Gamma_i^*$, it simulates

$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\mathsf{G}_i, \mathsf{RDMPC.Local}(\widehat{\mathsf{C}}_{i,k}^*, \widehat{x}_k^*)))$$

    otherwise, it simulates $\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\perp, \perp))$.
  - For $k = \ell$, if $k \in \mathcal{T}$, $\mathcal{B}$ computes $\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k^*)$; otherwise, it sends $\{(\overline{\mathsf{pk}}_{i,\ell}, \mathsf{G}_i^*)\}_{i\in[L]}$ along with a challenge message $\widehat{x}_\ell^*$ to the challenger of osRFE, hence obtaining ct_k^*.
  - For $k > \ell$, $\mathcal{B}$ directly computes $\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k^*)$.

If $\ell \in \mathcal{T}$, $\mathcal{B}$ will specify nothing in the challenge phase and hence $\mathsf{Game}_{3,\ell-1} \approx_c \mathsf{Game}_{3,\ell}$; otherwise, in the real world of osRFE, $\mathcal{B}$ simulates $\mathsf{Game}_{3,\ell-1}$; in the ideal world of osRFE, $\mathcal{B}$ simulates $\mathsf{Game}_{3,\ell}$. □

Lemma 28. *By the security of* RDMPC*, we claim* $\mathsf{Game}_{3,N} \approx_c \mathsf{Game}_4$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{3,N}$ from Game_4, we can build an algorithm $\mathcal{B}$ to break the security of RDMPC. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, C)$ from $\mathcal{A}$.

- *Setup phase:* $\mathcal{B}$ runs

$$\forall k \in [N],\ (\mathsf{crs}_k, \mathsf{td}_k) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, G)$$

  and sample dummy keys $\{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}$, hash keys $(\mathsf{hk}_1, \mathsf{hk}_2)$, and maps (ρ_1, ρ_2) as in $\widetilde{\mathsf{Setup}}$. Return

$$\mathsf{crs} = (\{\mathsf{crs}_k\}_{k\in[N]}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2).$$

– *Pre-challenge phase:*
  - $\mathsf{OGen}(i)$: $\mathcal{B}$ returns $\mathsf{pk} = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ to $\mathcal{A}$, where

$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k).$$

    Update $\mathcal{D}_i[\mathsf{pk}] = \{\mathsf{st}_{i,k}\}_{k\in[N]}$.
  - $\mathsf{OCor}(i, \mathsf{pk})$: $\mathcal{B}$ returns $\mathsf{sk} = \{\mathsf{sk}_{i,k}\}_{k\in[N]}$ to $\mathcal{A}$, where

$$\forall k \in [N],\ \mathsf{sk}_{i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\perp, \perp)).$$

– *Challenge phase:* $\mathcal{B}$ receives challenge public keys $\{\mathsf{pk}_i^*\}_{i\in[L]}$, circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$ and a challenge message x^*. $\mathcal{B}$ can now learn the sets $\Gamma_1^*, \ldots, \Gamma_L^*, \Delta_1^*, \ldots, \Delta_L^*, \mathcal{K}, \mathcal{T}$. Next, $\mathcal{B}$ invokes the challenger of RDMPC by revealing parameters (Q, N, m, t), corruption set $\mathcal{T}$, and $\{(\Gamma_j^*, \mathsf{C}_j, \Delta_j^*)\}_{j\in\mathcal{K}}$. The challenger replies $\mathcal{B}$ with the encodings $(\{\widehat{x}_k^*\}_{k\in\mathcal{T}}, \{\widehat{y}_{j,k}^*\}_{j\in\mathcal{K}, k\in\Gamma_j^*})$. Then, $\mathcal{B}$ computes the master public key $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$ as in algorithm Agg and generates:
  - For all $k \in \mathcal{T}$, $\mathcal{B}$ directly computes

$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k^*).$$

  - For all $k \notin \mathcal{T}$, if there exists a slot i such that $k \in \Gamma_i$, $\mathcal{B}$ simulates

$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\mathsf{G}_i, \widehat{y}_{i,k}^*))$$

    otherwise, it simulates $\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\perp, \perp))$.

We see that in the real world of RDMPC, $\mathcal{B}$ simulates $\mathsf{Game}_{2,N}$; in the ideal world of RDMPC, $\mathcal{B}$ simulates Game_4. Therefore, the lemma follows readily. □

C Static Slotted Reg-FE with Adaptive Security

In this section, we present a generic construction of static bounded collusion sReg-FE for circuits, achieving fully adaptive SIM security against $Q \geq 1$ malicious/corrupted users. For this goal, we now introduce a new property for registered primitives.

C.1 Rerandomized Public Key

Slotted Reg-ABE. For sReg-ABE defined in Appendix B.1, we introduce a new property named rerandomized public key. In particular, for all $\lambda, L \in \mathbb{N}$, all policy families $P : A \to \{0,1\}$, and all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, P)$, we define:

– $\mathsf{Rerand}(\mathsf{pk}_i, i; r) \to \widetilde{\mathsf{pk}}_i$. It takes as input a public key pk_i and a slot index $i \in [L]$, then outputs a rerandomized public key $\widetilde{\mathsf{pk}}_i$. Here, r is the randomness.

It requires that the following distributions are indistinguishable:

$$\{\mathsf{pk}_i, \mathsf{Rerand}(\mathsf{pk}_i, i)\} \approx_s \{\mathsf{pk}_i, \mathsf{pk}_i'\} \tag{15}$$

where $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$ and $(\mathsf{pk}_i', \mathsf{sk}_i') \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$.

On the other hand, we slightly adapt the experiment of policy-selective IND security. In particular, the adversary is allowed to register rerandomized public keys in the challenge phase:

– *Setup phase:* Same as Appendix B.1.

– *Pre-challenge phase:* Same as Appendix B.1.
– *Challenge phase:* In this phase, the adversary $\mathcal{A}$ specifies L pairs of public keys and randomness $\{(\mathsf{pk}_i^*, r_i)\}_{i\in[L]}$, attributes $\{\mathsf{att}_i^*\}_{i\in[L]}$, and a pair of challenge messages $(\mathsf{m}_0^*, \mathsf{m}_1^*)$. The challenger computes

$$\forall i \in [L],\ \widetilde{\mathsf{pk}}_i = \mathsf{Rerand}(\mathsf{pk}_i^*, i; r_i),$$
$$(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\widetilde{\mathsf{pk}}_i, \mathsf{att}_i^*)\}_{i\in[L]}).$$

For each slot $i \in [L]$, it must fall in one of the following three cases:

$$\begin{aligned}
&\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \notin C && \text{(honest slot)}\\
&\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*, \mathsf{att}_i^*) \in C \implies \mathsf{P}^*(\mathsf{att}_i^*) = 0 && \text{(corrupted slot)}\\
&\mathcal{D}_i[\mathsf{pk}_i^*] = \perp \implies \mathsf{P}^*(\mathsf{att}_i^*) = 0 && \text{(malicious slot)}
\end{aligned}$$

If pk_i^* is generated in an honest/corrupted/malicious way, then the rerandomized key pk_i' is also honest/corrupted/malicious. Next, the challenger flips a random coin $b \in \{0,1\}$ and replies $\mathcal{A}$ with a challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}^*, \mathsf{m}_b^*)$.
– *Post-challenge phase:* Same as Appendix B.1.
– *Output phase:* Same as Appendix B.1.

Remark. We observe that existing sReg-ABE schemes from pairings [ZZGQ23,AT24] or lattices [CHW25a,WW25] implicitly own the property of rerandomized public keys and satisfy the above adapted security.

For instance, in the pairing-based schemes, a public key is usually of the form $\mathsf{pk}_i = g^{u_i}$, where $\mathsf{sk}_i = u_i$, so we can rerandomize it by sampling a random scalar r_i and computing $\widetilde{\mathsf{pk}}_i = \mathsf{pk}_i \cdot g^{r_i}$. We see that $\widetilde{\mathsf{pk}}_i$ is statistically indistinguishable from a normally generated key. On the other hand, in the lattice-based sReg-ABE scheme of [WW25], we see that their public key is of the form $\mathsf{pk}_i = \mathbf{A}\mathbf{t}_i + \mathsf{Encode}(\mathsf{att}_i)$, where $\mathsf{sk}_i = \mathbf{t}_i$, so we can rerandomize it by sampling a random short vector $\mathbf{r}_i$ and computing $\widetilde{\mathsf{pk}}_i = \mathsf{pk}_i + \mathbf{A}\mathbf{r}_i$. By the well-known leftover hash lemma, we see that both pk_i and $\widetilde{\mathsf{pk}}_i$ are uniformly distributed, which exactly satisfies the argument in Eq. (15).

1-Bound Slotted Reg-FE. Similarly, for 1-bound sReg-FE, we also introduce the rerandomized public key. In particular, for all $\lambda, L \in \mathbb{N}$, all collusion bounds $Q \ll L$, all circuit families C, and all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$, we define:

– $\mathsf{Rerand}(\mathsf{pk}_i, i; r) \to \widetilde{\mathsf{pk}}_i$. It takes as input a public key pk_i and a slot index $i \in [L]$, then outputs a rerandomized public key $\widetilde{\mathsf{pk}}_i$. Here, r is the randomness used for this algorithm.

It requires that the following distributions are indistinguishable:

$$\{\mathsf{pk}_i, \mathsf{Rerand}(\mathsf{pk}_i, i)\} \approx_s \{\mathsf{pk}_i, \mathsf{pk}_i'\}$$

where $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs}, i)$ and $(\mathsf{pk}_i', \mathsf{sk}_i') \leftarrow \mathsf{Gen}(\mathsf{crs}, i)$.

On the other hand, we also slightly adjust the experiment of SIM security defined in Section 7.1. In particular, the adversary $\mathcal{A}$ is allowed to register rerandomized public keys in the challenge phase:

– *Setup phase:* Same as Section 7.1.
– *Pre-challenge phase:* Same as Section 7.1.
– *Challenge phase:* In this phase, $\mathcal{A}$ specifies L pairs of public keys and randomness $\{(\mathsf{pk}_i^*, r_i)\}_{i\in[L]}$, circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$, and a challenge message x^*. The challenger computes

$$\forall i \in [L],\ \widetilde{\mathsf{pk}}_i = \mathsf{Rerand}(\mathsf{pk}_i^*, i; r_i),$$
$$(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\widetilde{\mathsf{pk}}_i, \mathsf{C}_i^*)\}_{i\in[L]}).$$

For each slot $i \in [L]$, it must fall in one of the following three cases:

$$\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*) \notin C \qquad \text{(honest slot)}$$
$$\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \ \wedge (i, \mathsf{pk}_i^*) \in C \qquad \text{(corrupted slot)}$$
$$\mathcal{D}_i[\mathsf{pk}_i^*] = \perp \qquad \text{(malicious slot)}$$

If pk_i^* is generated in an honest/corrupted/malicious way, then the rerandomized key pk_i' is also honest/corrupted/malicious. Record all corrupted/malicious slot indices in the set $\mathcal{K}$, where $|\mathcal{K}| = q \leq Q$. Next, it returns the challenge ciphertext $\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^*)$ (real world) or $\mathsf{ct}^* \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}_i^*, \mathsf{C}_i^*(x^*))\}_{i\in\mathcal{K}})$ (ideal world) to $\mathcal{A}$.

– *Post-challenge phase:* Same as Section 7.1.
– *Output phase:* Same as Section 7.1.

Remark. If there exists a sReg-ABE scheme that owns the property of rerandomized public keys, our 1-bound sReg-FE scheme presented in Appendix B.2 also owns the property of rerandomized public keys and satisfies the above adapted security. This is because our scheme is essentially composed of sReg-ABE schemes.

C.2 From Slotted Reg-ABE to 1-Bound Slotted Reg-FE

For some circuit family $C : X \to Z$ of circuit size n, assuming an sReg-ABE scheme for simple policies $\mathsf{sRABE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Rerand}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$ and a 1-bound sReg-FE scheme with non-adaptive security $\mathsf{osRFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Rerand}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, our scheme for circuit family C works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$: It proceeds as follows:
  • Invoke an instance of 1-bound sReg-FE with non-adaptive security, i.e.,
  $$\mathsf{crs}_{\mathsf{osRFE}} \leftarrow \mathsf{osRFE.Setup}(1^\lambda, 1^L, 1^Q, C).$$
  • Define the simple policy family $P : \{0,1\} \to \{0,1\}$, where it just sets the attribute space as $\{0,1\}$. Run
  $$\mathsf{crs}_{\mathsf{sRABE}} \leftarrow \mathsf{sRABE.Setup}(1^\lambda, 1^L, P).$$
  Output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{osRFE}}, \mathsf{crs}_{\mathsf{sRABE}})$.
– $\mathsf{Gen}(\mathsf{crs}, i)$:
  • Sample a public/secret key pair of 1-bound sReg-FE with non-adaptive security, i.e.,
  $$(\mathsf{pk}_i', \mathsf{sk}_i') \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}_{\mathsf{osRFE}}, i).$$
  • Let σ be the bit length of the ciphertext in osRFE scheme. Fix $\mathsf{att}_i = 1$ and sample
  $$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ (\mathsf{pk}_{i,w,b}', \mathsf{sk}_{i,w,b}') \leftarrow \mathsf{sRABE.Gen}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{att}_i).$$
  In addition, sample a random string $\mathbf{b}_i = (b_i[1], \ldots, b_i[w]) \leftarrow \{0,1\}^\sigma$.
  Output the pubic key and secret key
  $$\mathsf{pk}_i = (\mathsf{pk}_i', \{\mathsf{pk}_{i,w,b}'\}_{w\in[\sigma],b\in\{0,1\}}),\ \mathsf{sk}_i = (\mathsf{sk}_i', \{\mathsf{sk}_{i,w,b_i[w]}'\}_{w\in[\sigma]}, \mathbf{b}_i).$$
– $\mathsf{Rerand}(\mathsf{pk}_i, i)$: Sample randomness $r_i = (r_i', \{r_{i,w,b}'\}_{w\in[\sigma],b\in\{0,1\}})$ and run
$$\widetilde{\mathsf{pk}}_i = \mathsf{osRFE.Rerand}(\mathsf{pk}_i', i; r_i'),$$
$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ \widetilde{\mathsf{pk}}_{i,w,b} = \mathsf{sRABE.Rerand}(\mathsf{pk}_{w,b}', i; r_{i,w,b}').$$
Output $\widetilde{\mathsf{pk}}_i = (\widetilde{\mathsf{pk}}_i, \{\widetilde{\mathsf{pk}}_{i,w,b}\}_{w\in[n],b\in\{0,1\}})$.

– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i)$: It checks whether

$$\mathsf{osRFE.Ver}(\mathsf{crs}_{\mathsf{osRFE}}, i, \mathsf{pk}'_i) \stackrel{?}{=} 1,$$
$$\forall w \in [\sigma], b \in \{0,1\},\ \mathsf{sRABE.Ver}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{pk}'_{i,w,b}, 1) \stackrel{?}{=} 1.$$

If not, abort and output 0; otherwise, output 1.

– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$:

- Compute the master public key and helper keys of osRFE:

$$(\mathsf{mpk}', \{\mathsf{hsk}'_j\}_{j\in[L]}) \leftarrow \mathsf{osRFE.Agg}(\mathsf{crs}_{\mathsf{osRFE}}, \{(\mathsf{pk}'_i, \mathsf{C}_i)\}_{i\in[L]}).$$

- For each $w \in [\sigma]$ and $b \in \{0,1\}$, compute the master public keys and helper keys of sRABE:

$$(\mathsf{mpk}'_{w,b}, \{\mathsf{hsk}'_{j,w,b}\}_{j\in[L]}) \leftarrow \mathsf{sRABE.Agg}(\mathsf{crs}_{\mathsf{sRABE}}, \{(\mathsf{pk}'_{i,w,b}, 1)\}_{i\in[L]}).$$

Output the master public key and helper keys

$$\mathsf{mpk} = (\mathsf{mpk}', \{\mathsf{mpk}'_{w,b}\}_{w\in[n],b\in\{0,1\}}),\ \mathsf{hsk}_j = (\mathsf{hsk}'_j, \{\mathsf{hsk}'_{j,w,b}\}_{w\in[\sigma],b\in\{0,1\}}).$$

– $\mathsf{Enc}(\mathsf{mpk}, x)$: It proceeds as follows:

- Compute a ciphertext of osRFE:

$$\mathsf{ct}' \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}', x).$$

- Parse $\mathsf{ct}' = (\mathsf{ct}'[1], \ldots, \mathsf{ct}'[\sigma]) \in \{0,1\}^\sigma$. Let P be a policy that simply outputs 1. For each $w \in [\sigma]$ and $b \in \{0,1\}$, compute the ciphertext of sRABE:

$$\mathsf{ct}'_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}, \mathsf{ct}'[w]).$$

Output the ciphertext $\mathsf{ct} = \{\mathsf{ct}'_{w,b}\}_{w\in[\sigma],b\in\{0,1\}}$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct})$: Parse $\mathsf{hsk}_{i^*} = (\mathsf{hsk}'_{i^*}, \{\mathsf{hsk}'_{i^*,w,b}\}_{w\in[\sigma],b\in\{0,1\}})$ and $\mathsf{sk}_{i^*} = (\mathsf{sk}'_{i^*}, \{\mathsf{sk}'_{i^*,w,b_{i^*}[w]}\}_{w\in[\sigma]}, \mathbf{b}_{i^*})$. It first recovers the full ciphertext of osRFE:

$$\forall w \in [\sigma],\ \mathsf{ct}'[w] \leftarrow \mathsf{sRABE.Dec}(\mathsf{hsk}'_{i^*,w,b_{i^*}[w]}, \mathsf{sk}'_{i^*,w,b_{i^*}[w]}, \mathsf{att}_{i^*}, \mathsf{ct}'_{w,b_{i^*}[w]}).$$

where $\mathsf{att}_{i^*} = 1$. Output $z \leftarrow \mathsf{osRFE.Dec}(\mathsf{hsk}'_{i^*}, \mathsf{sk}'_{i^*}, \mathsf{ct}')$.

Completeness, Correctness, and Compactness. The completeness and correctness follow the underlying schemes sRABE and osRFE. For compactness, by the compactness of osRFE and sRABE, we have

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, n, \log L) + 2\sigma \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, \log L),$$
$$|\mathsf{hsk}_j| = \mathsf{poly}(\lambda, n, \log L) + 2\sigma \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, \log L)$$

where P can be some constant, n is the circuit size, and $\sigma = \mathsf{poly}(\lambda, n, \log L)$. Thus, our construction meets the compactness requirement.

Theorem 16. *Assuming* sRABE *is IND secure and* osRFE *is 1-bound non-adaptive SIM secure, our construction achieves 1-bound adaptive SIM security as defined in Section C.1.*

Proof. The proof is a simpler case of Theorem 12. Here, we just define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: This algorithm runs

$$(\mathsf{crs}_{\mathsf{osRFE}}, \mathsf{td}) \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, C),$$
$$\mathsf{crs}_{\mathsf{sRABE}} \leftarrow \mathsf{sRABE.Setup}(1^\lambda, 1^L, P).$$

Output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{osRFE}}, \mathsf{crs}_{\mathsf{sRABE}})$ and td.

– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: This algorithm samples

$$(\mathsf{pk}'_i, \mathsf{st}'_i) \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_{\mathsf{osRFE}}, i, \mathsf{td}),$$
$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ (\mathsf{pk}'_{i,w,b}, \mathsf{sk}'_{i,w,b}) \leftarrow \mathsf{sRABE.Gen}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{att}_i).$$

Output $\mathsf{pk}_i = (\mathsf{pk}'_i, \{\mathsf{pk}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}})$ and $\mathsf{st}_i = (\mathsf{st}'_i, \{\mathsf{sk}'_{i,w,b}\}_{w\in[\sigma],b\in\{0,1\}})$.

– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{td}, (f, z))$: It simulates

$$\mathsf{sk}'_i \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}'_i, \mathsf{td}, (\perp, \perp)).$$

Then consider the following two cases:

- If $f = \perp$ and $z = \perp$, this means that $\mathcal{A}$ makes this corruption query in the pre-challenge phase (i.e., non-adaptive case) or the public key pk_i is not registered into the system. Sample

$$\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma.$$

- Otherwise, $\mathcal{A}$ makes this corruption query in the post-challenge phase (i.e., adaptive case), so we have $f = \mathsf{C}^*_j$ and $z = \mathsf{C}^*_j(x^*)$ for some corrupted slot $j \in [L]$, where x^* is the challenge message. Here, it has known the vector $\mathbf{b}_i$, the challenger can simulate

$$\mathsf{ct}' \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}', (\mathsf{C}^*_j, \mathsf{C}^*_j(x^*))).$$

Output $\mathsf{sk}_i = (\mathsf{sk}'_i, \{\mathsf{sk}'_{i,w,b_i[w]\oplus\mathsf{ct}'[w]}\}_{w\in[\sigma]}, \mathbf{b}_i)$.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, (f, z))$: Parse $\mathsf{mpk} = (\mathsf{mpk}', \{\mathsf{mpk}'_{w,b}\}_{w\in[n],b\in\{0,1\}})$. It considers the following two cases:

- If $f = \mathsf{C}^*_j$ and $z = \mathsf{C}^*_j(x^*)$ for some corrupted/malicious slot $j \in [L]$, we can rely on the non-adaptive security of osRFE by simulating

$$\mathsf{ct}' \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}', (\mathsf{C}^*_j, \mathsf{C}^*_j(x^*))).$$

Then, it generates other ciphertexts of sRABE as in algorithm Enc:

$$\forall w \in [\sigma] \text{ and } b \in \{0,1\},\ \mathsf{ct}'_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}'_{w,b}, \mathsf{P}, \mathsf{ct}'[w]).$$

- If $f = \perp$ and $z = \perp$, it samples $\mathbf{b}_i = (b_i[1], \ldots, b_i[\sigma]) \leftarrow \{0,1\}^\sigma$, assuming that i will be corrupted in the post-challenge phase. Then simulate the ciphertexts

$$\forall w \in [\sigma],\ \mathsf{ct}'_{w,b_i[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b}, 0),$$
$$\mathsf{ct}'_{w,1-b_i[w]} \leftarrow \mathsf{esRABE.Enc}(\mathsf{mpk}'_{w,b}, 1).$$

Finally, output the challenge ciphertext $\mathsf{ct}^* = \{\mathsf{ct}'_{w,b}\}_{w\in[\sigma],b\in\{0,1\}}$.

C.3 From 1-Bound Security to Q-Bound Security

For some circuit family $C : X \to Z$ of circuit size n, let $t = \Theta(Q^2\lambda), m = \Theta(t), v = \Theta(Q\lambda), p_1 = \Theta(m), p_2 = \Theta(v), N = \Theta(Q^2\lambda p_1)$ and $S = S(Q\lambda p_2)$, where p_1 and p_2 are primes. For any circuit $\mathsf{C} \in C$ and set $\Delta \subseteq [S]$ of size v, we define a circuit family G in which each circuit $\mathsf{G} \in G$ with input $\widehat{x}$ can compute

$$\mathsf{G}(\widehat{x}) = \mathsf{RDMPC.Local}((\mathsf{C}, \Delta), \widehat{x})$$

where (C, Δ) is the constant hard-wired in G. Let $n' = \mathsf{poly}(\lambda, n, Q)$ be the circuit size of G.

Assuming a reusable dynamic MPC scheme $\mathsf{RDMPC} = (\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode})$, a 1-bound sReg-FE scheme (with adaptive security) $\mathsf{osRFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Rerand}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, and a simulation-sound extractable non-interactive zero-knowledge argument $\mathsf{NIZK} = (\mathsf{Setup}, \mathsf{Prv}, \mathsf{Ver}, \mathsf{SPrv})$, then our Q-bound sReg-FE for circuit family C works as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$: It proceeds as follows:
  - Sample the common reference strings

$$\forall k \in [N],\ \mathsf{crs}_k \leftarrow \mathsf{osRFE.Setup}(1^\lambda, 1^L, G),$$
$$(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.Setup}(1^\lambda).$$

  - Sample dummy public keys $\widehat{\mathsf{pk}}_{i,k} \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}_k, i)$ for each $i \in [L]$ and $k \in [N]$ (omit secret keys).
  - Let $\mathsf{CHF} : \{0,1\}^* \to \mathbb{Z}_{p_1}^\lambda$ be a chameleon hash function family and $\mathsf{Hash} : \{0,1\}^* \to \mathbb{Z}_{p_2}^\lambda$ be a collision-resistant hash function family. Sample

$$(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) \leftarrow \mathsf{CHF.Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash.Gen}(1^\lambda).$$

  - Define two injective maps $\rho_1 : [Dt+1] \times \mathbb{Z}_{p_1} \to [N]$ and $\rho_2 : [Dt+1] \times \mathbb{Z}_{p_2} \to [S]$.

  Output $\mathsf{crs} = (\{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{crs}_{\mathsf{NIZK}}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$.
- $\mathsf{Gen}(\mathsf{crs}, i)$: It proceeds as follows:
  - Sample the randomness $r_i \leftarrow \{0,1\}^\lambda$ for the chameleon hash function.
  - For $i \in [L]$, it computes coefficients $(a_{i,0}, \ldots, a_{i,\lambda-1}) = \mathsf{CHF.H}(\mathsf{hk}_1, i; r_i)$ and hence defines a polynomial function of degree $\lambda - 1$:

$$f_i(x) = a_{i,\lambda-1}x^{\lambda-1} + \cdots + a_{i,1}x + a_{i,0}.$$

    For each $j \in [m]$, evaluate the point $f_i(j)$ and then set $\gamma_{i,j} = \rho_1(j, f_i(j))$. Set $\Gamma_i = \{\gamma_{i,j}\}_{j\in[m]}$.
  - Sample the following keys

$$\forall k \in \Gamma_i,\ (\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}_k, i),$$
$$\forall k \in [N] \setminus \Gamma_i,\ \mathsf{pk}_{i,k} \leftarrow \mathsf{osRFE.Rerand}(\widehat{\mathsf{pk}}_{i,k}, i; r_{i,k}).$$

    where $\{r_{i,k}\}_{k\in[N]\setminus\Gamma_i}$ are the sampled randomness.
  - Generate a proof for the above steps, i.e., run

$$\pi_i \leftarrow \mathsf{NIZK.Prv}(\mathsf{crs}_{\mathsf{NIZK}}, s_i, w_i)$$

    where we define the statement $s_i = (i, \{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{hk}_1, \rho_1, \{\mathsf{pk}_{i,k}\}_{k\in[N]})$ and the witness $w_i = (r_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i}, \{r_{i,k}\}_{k\in[N]\setminus\Gamma_i})$.

  Output the public key and secret key

$$\mathsf{pk}_i = (\{\mathsf{pk}_{i,k}\}_{k\in[N]}, \pi_i),\ \mathsf{sk}_i = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i}).$$

– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i)$: It check whether

$$\forall k \in [N],\ \mathsf{osRFE.Ver}(\mathsf{crs}_k, i, \mathsf{pk}_{i,k}) \stackrel{?}{=} 1,$$
$$\mathsf{NIZK.Ver}(\mathsf{crs}_{\mathsf{NIZK}}, s_i, \pi_i) \stackrel{?}{=} 1.$$

If not, abort and output 0; otherwise, output 1.

– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: Parse $\mathsf{pk}_i = (\{\mathsf{pk}_{i,k}\}_{k\in[N]})$, it proceeds as follows:

- For $i \in [L]$, it computes coefficients $(c_{i,0}, \ldots, c_{i,v-1}) = \mathsf{Hash.H}(\mathsf{hk}_2, i||\mathsf{pk}_i)$ and hence defines a polynomial function of degree $\lambda - 1$:
$$g_i(x) = c_{i,\lambda-1}x^{\lambda-1} + \cdots + c_{i,1}x + c_{i,0}.$$
For each $j \in [v]$, evaluate the point $g_i(j)$ and then set $\delta_{i,j} = \rho_2(j, g_i(j))$. Set $\Delta_i = \{\delta_{i,j}\}_{j\in[v]}$.
- For each $i \in [L]$ and $k \in [N]$, define circuits $\mathsf{G}_{i,k} = (\mathsf{G}_{i,k}[1], \ldots, \mathsf{G}_{i,k}[n']) \in \{0,1\}^{n'}$, where $\mathsf{G}_{i,k}$ takes $\widehat{\mathsf{C}}_{i,k} = (\mathsf{C}_i, \Delta_i)$ as constant. Note that
$$(\widehat{\mathsf{C}}_{i,1}, \ldots, \widehat{\mathsf{C}}_{i,N}) = \mathsf{RDMPC.CktEnc}(1^\lambda, 1^Q, \mathsf{C}_i, \Delta_i).$$
- For all $k \in [N]$, run
$$(\mathsf{mpk}_k, \{\mathsf{hsk}_{j,k}\}_{j\in[L]}) = \mathsf{osRFE.Agg}(\mathsf{crs}_k, \{(\mathsf{pk}_{i,k}, \mathsf{G}_i)\}_{i\in[L]}).$$

Output $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$ and $\mathsf{hsk}_j = \{\mathsf{hsk}_{j,k}\}_{k\in[N]}$.

– $\mathsf{Enc}(\mathsf{mpk}, x)$: Parse $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$, then compute

$$(\widehat{x}_1, \ldots, \widehat{x}_N) \leftarrow \mathsf{RDMPC.InpEnc}(1^\lambda, 1^Q, x),$$
$$\forall k \in [N],\ \mathsf{ct}_k \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k).$$

Output the ciphertext $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct})$: Parse $\mathsf{hsk}_{i^*} = \{\mathsf{hsk}_{i^*,k}\}_{k\in[N]}$, $\mathsf{sk}_{i^*} = (\Gamma_{i^*}, \{\mathsf{sk}_{i^*,k}\}_{k\in\Gamma_{i^*}})$. Compute

$$\forall k \in \Gamma_{i^*},\ \widehat{y}_k \leftarrow \mathsf{osRFE.Dec}(\mathsf{hsk}_{i^*,k}, \mathsf{sk}_{i^*,k}, \mathsf{ct}_k).$$

Output $z = \mathsf{RDMPC.Decode}(\{\widehat{y}_k\}_{k\in\Gamma_{i^*}}, \Gamma_{i^*})$.

Completeness, Compactness and Correctness. The completeness follows the underlying osRFE scheme. By the compactness of osRFE, we have

$$|\mathsf{mpk}| = N \cdot \mathsf{poly}(\lambda, n', \log L) = \mathsf{poly}(\lambda, n, Q, \log L),$$
$$|\mathsf{hsk}_j| = N \cdot \mathsf{poly}(\lambda, n', \log L) = \mathsf{poly}(\lambda, n, Q, \log L),$$

where N depends on the collusion bound Q, and $n' = \mathsf{poly}(\lambda, n, Q)$. Thus, our construction satisfies the compactness requirement.

By the correctness of osRFE, the decryption algorithm first recover the local encodings

$$\forall k \in \Gamma_{i^*},\ \widehat{y}_k = \mathsf{RDMPC.Local}((\mathsf{C}_{i^*}, \Delta_{i^*}), \widehat{x}_k).$$

By the correctness of RDMPC, we have $\mathsf{RDMPC.Decode}(\{\widehat{y}_k\}_{k\in\Gamma_{i^*}}, \Gamma_{i^*}) = \mathsf{C}_{i^*}(x)$.

Theorem 17. *Assuming* osRFE *achieves 1-bound adaptive SIM security, the above construction achieves Q-bound adaptive SIM security.*

Proof. Let $\mathcal{K}'$ and $\mathcal{K}$ be the sets of the corrupted/malicious slot indices stored in the challenge phase and the final output phase, respectively. Define the following set

$$\mathcal{T} = \bigcup_{i\in\mathcal{K}, j\in\mathcal{K}\setminus\{j\}} (\Gamma_i^* \cap \Gamma_j^*).$$

where Γ_i^* is the pairwise set that is registered under slot i. With a challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_1^*, \ldots, \mathsf{ct}_N^*)$ and corrupted/malicious secret keys $\{\mathsf{sk}_j\}_{j\in\mathcal{K}}$, for each $k \in [N]$, the proof is organized as follows:

- If $k \notin \mathcal{T}$, it means there exists at most a slot $i \in \mathcal{K}$ such that $k \in \Gamma_i^*$ and $k \notin \Gamma_j^*$ for any $j \in \mathcal{K} \setminus \{i\}$. In particular, ct_k^* can only be decrypted using sk_i. Thus, it can rely on the 1-bound security of osRFE;
- Otherwise, it means that $\mathcal{A}$ holds more than one secret keys that correctly decrypt ct_k^*. In this way, 1-bound security would be compromised, and the security would in turn rely on the underlying RDMPC scheme.

Then, we define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

- $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: This algorithm samples

$$\begin{aligned}
\forall k \in [N],\ (\mathsf{crs}_k, \mathsf{td}_k) &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, G),\\
\forall i \in [L] \text{ and } k \in [N],\ \widehat{\mathsf{pk}}_{i,k} &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k),\\
(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) &\leftarrow \mathsf{CHF.Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash.Gen}(1^\lambda),\\
(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}) &\leftarrow \mathsf{NIZK.EGen}(1^\lambda).
\end{aligned}$$

  Then, it defines injective maps (ρ_1, ρ_2) as in algorithm Setup. Output

$$\begin{aligned}
\mathsf{crs} &= (\{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{crs}_{\mathsf{NIZK}}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2),\\
\mathsf{td} &= (\{\mathsf{td}_k\}_{k\in[N]}, \mathsf{td}_{\mathsf{CHF}}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}).
\end{aligned}$$

- $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: This algorithm samples

$$\begin{aligned}
\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k),\\
\pi_i &\leftarrow \mathsf{NIZK.SPrv}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, s_i),
\end{aligned}$$

  where we define the statement $s_i = (i, \{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{hk}_1, \rho_1, \{\mathsf{pk}_{i,k}\}_{k\in[N]})$. Output $\mathsf{pk}_i = (\{\mathsf{pk}_{i,k}\}_{k\in[N]}, \pi_i)$ and $\mathsf{st}_i = \{\mathsf{st}_{i,k}\}_{k\in[N]}$.
- $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{td}, (f, z))$: Parse $\mathsf{st}_i = \{\mathsf{st}_{i,k}\}_{k\in[N]}$. This algorithm considers the following two cases:
  - If $f = \perp$ and $z = \perp$, this means that $\mathcal{A}$ makes this corruption query in the pre-challenge phase, or the corresponding public key pk_i is not registered. It samples the set Γ_i as in algorithm Gen and computes

$$\forall k \in \Gamma_i,\ \mathsf{sk}_{i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\perp, \perp)).$$

    Output $\mathsf{sk}_i = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i})$.
  - Otherwise, $\mathcal{A}$ makes this corruption query in the post-challenge phase, and we have $f = \mathsf{C}_i^*$ and $z = \mathsf{C}_i^*(x^*)$ for the corrupted slot $i \in [L]$, where x^* is the challenge message. Here, we assume that the challenger has acquired sets $(\Gamma_i^*, \Delta_i^*, \mathcal{T})$ during the challenger phase. Therefore, it can simulate

$$\begin{aligned}
\{\widehat{y}_{i,k}^*\}_{k\in\Gamma_i^*} &\leftarrow \mathsf{RDMPC}.\widetilde{\mathsf{Local}}\left(i, \mathsf{C}_i^*, \mathsf{C}_i^*(x^*)\right),\\
\forall k \in \Gamma_i^* \cap \mathcal{T},\ \mathsf{sk}_{i,k} &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\perp, \perp)),\\
\forall k \in \Gamma_i^* \setminus \mathcal{T},\ \mathsf{sk}_{i,k} &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\mathsf{G}_i^*, \widehat{y}_{i,k}^*)).
\end{aligned}$$

    where G_i^* is defined over C_i^* and Δ_i^*. Output $\mathsf{sk}_i = (\Gamma_i^*, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i^*})$.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}'})$: Here, we require $|\mathcal{K}'| = q \le Q$. We assume that in the challenge phase, the challenger has known $\{\Gamma_j^*\}_{j\in\mathcal{K}}$, $\Delta_1^*, \ldots, \Delta_L^*$ (note that $\mathcal{K}\setminus\mathcal{K}'$ is still unknown). Therefore, the challenger can also acquire the set $\mathcal{T}$. Then, it proceeds as follows:
  - It simulates the following input and local encodings
$$(\{\widehat{x}_k^*\}_{k\in\mathcal{T}}, \{\widehat{y}_{j,k}^*\}_{j\in\mathcal{K}', k\in\Gamma_j^*}) \leftarrow \mathsf{RDMPC}.\widetilde{\mathsf{InpEnc}}(1^{|x^*|}, \mathcal{T}, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}'})$$
where $\widehat{y}_{j,k}^* = \mathsf{RDMPC}.\mathsf{Local}(\widehat{\mathsf{C}}_{j,k}^*, \widehat{x}_k^*)$ for all $k \in \Gamma_j^* \cap \mathcal{T}$.
  - For all $k \in \mathcal{T}$, it generates
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\mathsf{Enc}(\mathsf{mpk}_k, \widehat{x}_k^*).$$
  - For all $k \notin \mathcal{T}$, if there is a malicious/corrupted slot $i \in \mathcal{K}'$ such that $k \in \Gamma_i$, we can simulate
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\mathsf{G}_i^*, \widehat{y}_{i,k}^*)).$$
where the circuit G_i^* is defined over C_i^* and Δ_i^*; otherwise, we simulate
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\bot, \bot)).$$

  Finally, output the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_1^*, \ldots, \mathsf{ct}_N^*)$.

Game Sequence. We prove Theorem 17 via a sequence of games as follows:

– Game_0: Real Game.
– Game_1: Same as Game_0 except that for each key generation query to the oracle OGen it simulates π_i as in $\widetilde{\mathsf{Gen}}_1$.
– Game_2: Same as Game_1 except that for each key generation query to the oracle OGen, it replaces all rerandomized keys with normally generated keys, i.e., it returns the public key $\mathsf{pk}_i = (\{\mathsf{pk}_{i,k}\}_{i\in[N]}, \pi_i)$ to $\mathcal{A}$, where $\mathsf{pk}_{i,k} \leftarrow \mathsf{osRFE}.\mathsf{Gen}(\mathsf{crs}_k, i)$ for all $k \in [N]$.
– Game_3: Same as Game_2 except that it generates $\mathsf{crs}_{\mathsf{NIZK}}$ as in $\widetilde{\mathsf{Setup}}$, and for each $i \in \mathcal{K}'$, it extracts the witness $w_i \leftarrow \mathsf{NIZK}.\mathsf{Ext}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}, s_i, \pi_i)$.
– Game_4: Same as Game_3 except that in the challenge phase, for each $i = q+1, \ldots, Q$, it samples a new set Γ_i via the chameleon hash function CHF.
– Game_5: Same as Game_4 except that for each $c_i \in \mathcal{K}\setminus\mathcal{K}'$, where $i = q+1, \ldots, Q$, it set $\Gamma_{c_i}^* = \Gamma_i$.
– Game_6: Same as Game_5 except that if $\left|\bigcup_{i\in\mathcal{K}, j\in\mathcal{K}\setminus\{j\}}(\Gamma_i^* \cap \Gamma_j^*)\right| \ge t$, it aborts and outputs $\bot$.
– Game_7: Same as Game_6 except that for all $i \in \mathcal{K}$, if $\Delta_i^* \setminus \left(\bigcup_{j\in\mathcal{K}\setminus\{i\}} \Delta_j^*\right) = \emptyset$, it aborts and outputs $\bot$.
– $\mathsf{Game}_{8,\ell}(\ell \in [N])$: Same as $\mathsf{Game}_{*,\ell-1}$ except that for all $k \le \ell$, it simulates $(\mathsf{crs}_k, \widehat{\mathsf{pk}}_{i,k})$ as in $\widetilde{\mathsf{Setup}}$, $(\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k})$ as in $(\widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2)$, and ct_k^* for $k \notin \mathcal{T}$ as in $\widetilde{\mathsf{Enc}}$.
– Game_9: Same as $\mathsf{Game}_{8,N}$ except that it completely replaces $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc})$ with $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.

We see that in the final game Game_9, the challenger ciphertext ct^* and the corrupted secret keys $\{\mathsf{sk}_i\}_{i\in\mathcal{K}}$ can be simulated only using $\{(\mathsf{C}_i^*, \mathsf{C}_i^*(x^*))\}_{i\in\mathcal{K}}$. This is exactly the simulation world.

Lemma 29. *By the zero-knowledge property of* NIZK*, we claim* $\mathsf{Game}_0 \approx_c \mathsf{Game}_1$.

Proof. Since the main difference between Game_0 and Game_1 is that when $\mathcal{A}$ makes key generation queries, the challenger generates the proof π_i via either NIZK.Prv or NIZK.SPrv. Therefore, we can reduce the indistinguishability between Game_0 and Game_1 to the zero-knowledge property of the NIZK scheme. Since the reduction is standard, we omit details here. □

Lemma 30. *By the property of rerandomized public keys as defined in Section C.1, we claim* $\mathsf{Game}_1 \approx_s \mathsf{Game}_2$.

Proof. The only difference between Game_1 and Game_2 is that the challenger all rerandomized keys with normally generated keys when responding to the key generation query from $\mathcal{A}$. Since the rerandomized public key is statistically indistinguishable from the normally generated public key, we naturally have $\mathsf{Game}_1 \approx_s \mathsf{Game}_2$. □

Lemma 31. *By the simulation-sound extractability of* NIZK*, we claim* $\mathsf{Game}_2 \approx_c \mathsf{Game}_3$.

Proof. Since the main difference between Game_2 and Game_3 is that the challenger generates the extraction information ex and then uses ex to extract all the witnesses from the proof π_i for all corrupted/malicious slots $i \in \mathcal{K}'$, we can reduce the indistinguishability between Game_2 and Game_3 to the simulation-sound extractability of the NIZK scheme. Since the reduction is standard, we omit details here. □

Lemma 32. *We claim that* $\mathsf{Game}_3 \equiv \mathsf{Game}_4$.

Proof. The additional sampling operation does not affect any distributions. Therefore, the lemma follows readily. □

Lemma 33. *By the chameleon property of* CHF*, we claim* $\mathsf{Game}_4 \approx_s \mathsf{Game}_5$.

Proof. The proof is analogous to Lemma 5, so we omit details here. □

Lemma 34. *By the collision resistance of chameleon hash function, we claim* $\mathsf{Game}_5 \approx_s \mathsf{Game}_6$.

Proof. The proof is analogous to Lemma 6, so we omit details here. □

Lemma 35. *By the collision resistance of hash function, we claim* $\mathsf{Game}_6 \approx_s \mathsf{Game}_7$.

Proof. The proof is analogous to Lemma 7, so we omit details here. □

Lemma 36. *By the 1-bound adaptive SIM security of* osRFE*, we claim that* $\mathsf{Game}_{8,\ell-1} \approx_c \mathsf{Game}_{8,\ell}$ *for each* $\ell \in [N]$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{8,\ell-1}$ from $\mathsf{Game}_{8,\ell}$, we can build an algorithm $\mathcal{B}$ to break the 1-bound adaptive SIM security of osRFE. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, C)$ from $\mathcal{A}$. Then, $\mathcal{B}$ proceeds as follows:

- *Setup phase:* It sends $(1^L, 1, G)$ to the challenger of osRFE to obtain crs_ℓ. Then, it proceeds as follows:
  - For $k < \ell$, it simulates $(\mathsf{crs}_k, \mathsf{td}_k) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, G)$ and
  $$\forall i \in [L],\ (\widehat{\mathsf{pk}}_{i,k}, \widehat{\mathsf{st}}_{i,k}) \leftarrow \mathsf{osRFE.}\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k).$$
  - For $k = \ell$, it queries the oracle osRFE.OGen to obtain $\widehat{\mathsf{pk}}_{i,\ell}$ for all $i \in [L]$.
  - For $k > \ell$, it generates $\mathsf{crs}_k \leftarrow \mathsf{osRFE.Setup}(1^\lambda, 1^L, G)$ and
  $$\forall i \in [L],\ (\widehat{\mathsf{pk}}_{i,k}, \widehat{\mathsf{sk}}_{i,k}) \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}, i).$$

  Then, it samples
  $$(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) \leftarrow \mathsf{CHF.Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash.Gen}(1^\lambda),$$
  $$(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}) \leftarrow \mathsf{NIZK.EGen}(1^\lambda).$$
  and defines injective maps (ρ_1, ρ_2). Return $\mathcal{A}$ with
  $$\mathsf{crs} = (\{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{crs}_{\mathsf{NIZK}}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2).$$

– *Pre-challenge phase:* Here, $\mathcal{A}$ can query the oracles below:
  - $\mathsf{OGen}(i)$: $\mathcal{B}$ proceeds as follows:
    * For $k < \ell$, it simulates $(\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k)$.
    * For $k = \ell$, it queries the oracle osRFE.OGen to obtain $\mathsf{pk}_{i,\ell}$.
    * For $k > \ell$, it computes $(\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{osRFE.Gen}(\mathsf{crs}, i)$.
    * It simulates
    $$\pi_i \leftarrow \mathsf{NIZK.SPrv}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, s_i)$$
    where $s_i = (i, \{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{hk}_1, \rho_1, \{\mathsf{pk}_{i,k}\}_{k\in[N]})$.

    Return $\mathsf{pk} = (\{\mathsf{pk}_{i,k}\}_{k\in[N]}, \pi_i)$ to $\mathcal{A}$.
  - $\mathsf{OCor}(i, \mathsf{pk})$:
    * $\mathcal{B}$ samples a set Γ_i as in algorithm Gen.
    * For $k \in \Gamma_i$ and $k < \ell$, simulate $\mathsf{sk}_{i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\perp, \perp))$.
    * For $k \in \Gamma_i$ and $k = \ell$, it queries the oracle osRFE.OCor to obtain the secret key $\mathsf{sk}_{i,\ell}$.

    Return $\mathsf{sk} = (\Gamma_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i})$ to $\mathcal{A}$.

– *Challenge phase:* It receives challenge public keys $\{\mathsf{pk}_i^*\}_{i\in[L]}$, circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$ and a challenge message x^*, where $\mathsf{pk}_i^* = (\{\mathsf{pk}_{i,k}^*\}_{k\in[N]}, \pi_i^*)$. For each corrupted/malicious slot $i \in \mathcal{K}'$, $\mathcal{B}$ can extract the witness

$$(r_i, \{\mathsf{sk}_{i,k}\}_{k\in\Gamma_i}, \{r_{i,k}\}_{k\in[N]\setminus\Gamma_i}) \leftarrow \mathsf{NIZK.Ext}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}, s_i, \pi_i^*)$$

Therefore, $\mathcal{B}$ can acquire the set Γ_i^* based on the randomness r_i. As in Game_7, $\mathcal{B}$ can now acquire the sets $\{\Gamma_j^*\}_{j\in\mathcal{K}}, \Delta_1^*, \ldots, \Delta_L^*$ and hence compute the set $\mathcal{T}$. Next, $\mathcal{B}$ computes the master public key $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$ as in algorithm Agg and derives the input encodings

$$(\widehat{x}_1^*, \ldots, \widehat{x}_N^*) \leftarrow \mathsf{RDMPC.InpEnc}(1^\lambda, 1^Q, x^*).$$

$\mathcal{B}$ replies $\mathcal{A}$ with the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_1^*, \ldots, \mathsf{ct}_N^*)$ which is generated as follows:

- For $k < \ell$, if $k \in \mathcal{T}$, $\mathcal{B}$ directly computes
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k^*)$$
otherwise, if there exists a slot $i \in \mathcal{K}'$ such that $k \in \Gamma_i^*$, it simulates
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\mathsf{G}_i^*, \mathsf{G}_i^*(\widehat{x}_k^*)))$$
where the circuit G_i^* is defined over C_i^* and Δ_i^*; otherwise, it simulates
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\perp, \perp)).$$
- For $k = \ell$, if $k \in \mathcal{T}$, $\mathcal{B}$ computes
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k^*).$$
Otherwise, for each $i \in [L]$, if the randomness $r_{i,\ell}$ is unassigned, it sets $r_{i,\ell} = \perp$ (meaning that the rerandomized key is exactly the input key). Next, $\mathcal{B}$ sends public keys and randomness $\{(\mathsf{pk}_{i,\ell}^*, r_{i,\ell})\}_{i\in[L]}$, circuits $\{\mathsf{G}_i^*\}_{i\in[L]}$, and a challenge message $\widehat{x}_\ell^*$ to the challenger of osRFE, hence obtaining a ciphertext ct_k^*.
- For $k > \ell$, $\mathcal{B}$ directly computes
$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}_k^*).$$

– *Post-challenge phase:* Here, we only consider the case when $\mathcal{A}$ make corruption queries on registered keys:

- $\mathsf{OCor}(c_i, \mathsf{pk}^*_{c_i})$: As in Game_7, $\mathcal{B}$ can exploit the trapdoor $\mathsf{td}_{\mathsf{CHF}}$ to acquire the randomness r_{c_i} that is used to generate the set $\Gamma^*_{c_i}$. Then, for each $k \in \Gamma^*_{c_i}$, it queries the oracle osRFE.OCor to obtain the secret key $\mathsf{sk}_{c_i,k}$. Note that these queries are admissible, as there is at most a single key $\mathsf{pk}^*_{c_i,\ell}$ that is registered into osRFE, while other keys $\{\mathsf{pk}^*_{c_i,k}\}_{k\neq\ell}$ are not registered. Return $\mathsf{sk} = (\Gamma^*_{c_i}, \{\mathsf{sk}_{c_i,k}\}_{k\in\Gamma^*_{c_i}})$ to $\mathcal{A}$.

(Note that if $\mathcal{A}$ makes corruption queries on non-registered keys, $\mathcal{B}$ can respond as in the pre-challenge phase.)

We see that if $\ell \in \mathcal{T}$, $\mathcal{B}$ will reveal nothing in the challenge phase and hence $\mathsf{Game}_{8,\ell-1} \approx_c \mathsf{Game}_{8,\ell}$; otherwise, in the real world of osRFE, $\mathcal{B}$ just simulates $\mathsf{Game}_{8,\ell-1}$; in the ideal world of osRFE, $\mathcal{B}$ simulates $\mathsf{Game}_{8,\ell}$. Therefore, the lemma follows readily. □

Lemma 37. *By the security of* RDMPC, *we claim* $\mathsf{Game}_{8,N} \approx_c \mathsf{Game}_9$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{9,N}$ from Game_{10}, we can build an algorithm $\mathcal{B}$ to break the security of RDMPC. Initially, $\mathcal{B}$ receives $(1^L, 1^Q, C)$ from $\mathcal{A}$. Then, $\mathcal{B}$ proceeds as follows:

– *Setup phase:* As in $\mathsf{Game}_{9,N}$, $\mathcal{B}$ samples

$$\begin{aligned}
\forall k \in [N],\ (\mathsf{crs}_k, \mathsf{td}_k) &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, G),\\
\forall i \in [L] \text{ and } k \in [N],\ \widehat{\mathsf{pk}}_{i,k} &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i, \mathsf{td}_k),\\
(\mathsf{hk}_1, \mathsf{td}_{\mathsf{CHF}}) &\leftarrow \mathsf{CHF.Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash.Gen}(1^\lambda),\\
(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, \mathsf{ex}_{\mathsf{NIZK}}) &\leftarrow \mathsf{NIZK.EGen}(1^\lambda).
\end{aligned}$$

Then, it defines injective maps (ρ_1, ρ_2) as in algorithm Setup. It replies $\mathcal{A}$ with the common reference string

$$\mathsf{crs} = (\{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{crs}_{\mathsf{NIZK}}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2).$$

– *Pre-challenge phase:* Here, $\mathcal{A}$ can query the oracles below:

- $\mathsf{OGen}(i)$: $\mathcal{B}$ samples

$$\begin{aligned}
\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) &\leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i),\\
\pi_i &\leftarrow \mathsf{NIZK.SPrv}(\mathsf{crs}_{\mathsf{NIZK}}, \mathsf{td}_{\mathsf{NIZK}}, s_i),
\end{aligned}$$

  where we define the statement $s_i = (i, \{\mathsf{crs}_k\}_{k\in[N]}, \mathsf{hk}_1, \rho_1, \{\mathsf{pk}_{i,k}\}_{k\in[N]})$. Return $\mathsf{pk} = (\{\mathsf{pk}_{i,k}\}_{k\in[N]}, \pi_i)$ to $\mathcal{A}$ and update $\mathcal{D}_i[\mathsf{pk}] = \{\mathsf{st}_{i,k}\}_{k\in[N]}$.
- $\mathsf{OCor}(i, \mathsf{pk})$: $\mathcal{B}$ samples the set Γ_i as in algorithm Gen and computes

$$\forall k \in \Gamma_i,\ \mathsf{sk}_{i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{td}_k, (\bot, \bot)).$$

  where $\{\mathsf{st}_{i,k}\}_{k\in[N]}$ is stored in $\mathcal{D}_i[\mathsf{pk}]$. Return $\mathsf{sk} = \{\mathsf{sk}_{i,k}\}_{k\in[N]}$ to $\mathcal{A}$.

– *Challenge phase:* $\mathcal{B}$ receives challenge public keys $\{\mathsf{pk}^*_i\}_{i\in[L]}$, circuits $\{\mathsf{C}^*_i\}_{i\in[L]}$ and a challenge message x^*. As in $\mathsf{Game}_{9,N}$, $\mathcal{B}$ can acquire the sets $\{\Gamma^*_j\}_{j\in\mathcal{K}}, \Delta^*_1, \ldots, \Delta^*_L$. Therefore, it can also learn the set $\mathcal{T}$.
Next, $\mathcal{B}$ invokes the challenger of RDMPC by specifying parameters (Q, N, m, t), corruption set $\mathcal{T}$, challenge message x^*, sets $\{\Gamma^*_j\}_{j\in\mathcal{K}}$, and $\{(\mathsf{C}_j, \Delta^*_j)\}_{j\in\mathcal{K}'}$, where $\mathcal{K}' \subseteq \mathcal{K}$ stores the corrupted/malicious slots that are only known in the pre-challenge phase. The challenger replies $\mathcal{B}$ with the following encodings

$$\{\widehat{x}^*_k\}_{k\in\mathcal{T}} \text{ and } \{\widehat{y}^*_{c_j,k}\}_{j\in[q],k\in\Gamma^*_{c_j}}.$$

Then, $\mathcal{B}$ computes the master public key $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$ as in algorithm Agg. It generates the ciphertexts:

- For all $k \in \mathcal{T}$, $\mathcal{B}$ directly computes

$$\mathsf{ct}^*_k \leftarrow \mathsf{osRFE.Enc}(\mathsf{mpk}_k, \widehat{x}^*_k).$$

- For all $k \notin \mathcal{T}$, if there exists a slot $i \in \mathcal{K}'$ such that $k \in \Gamma_i$, $\mathcal{B}$ simulates

$$\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\mathsf{G}_i^*, \widehat{y}_{i,k}^*)).$$

  Otherwise, it simulates $\mathsf{ct}_k^* \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, (\bot, \bot))$.

– *Post-challenge phase:*
- $\mathsf{OCor}(c_i, \mathsf{pk}_{c_i})$: Parse $\mathcal{D}_{c_i}[\mathsf{pk}_{c_i}] = \{\mathsf{st}_{c_i,k}\}_{k\in[N]}$, then it considers the following two cases:
  * If pk_{c_i} is not registered into the system, it samples the set Γ_{c_i} as in algorithm Gen and computes

$$\forall k \in \Gamma_{c_i},\ \mathsf{sk}_{c_i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{c_i,k}, \mathsf{td}_k, (\bot, \bot)).$$

    Return $\mathsf{sk} = (\Gamma_{c_i}, \{\mathsf{sk}_{c_i,k}\}_{k\in\Gamma_{c_i}})$ to $\mathcal{A}$.
  * Otherwise, As in Game_7, $\mathcal{B}$ can exploit the trapdoor $\mathsf{td}_{\mathsf{CHF}}$ to acquire the randomness r_{c_i} that is used to generate the set $\Gamma_{c_i}^*$. Here, $\mathcal{B}$ makes a circuit query $(\mathsf{C}_{c_i}^*, \Delta_{c_i}^*)$ to the oracle RDMPC.OCor. Then, $\mathcal{B}$ receives the encodings $\{\widehat{y}_{c_i,k}^*\}_{k\in\Gamma_{c_i}^*}$ from the challenger. It can simulate

$$\forall k \in \Gamma_{c_i}^* \cap \mathcal{T},\ \mathsf{sk}_{c_i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{c_i,k}, \mathsf{td}_k, (\bot, \bot)),$$
$$\forall k \in \Gamma_{c_i}^* \setminus \mathcal{T},\ \mathsf{sk}_{c_i,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{c_i,k}, \mathsf{td}_k, (\mathsf{G}_{c_i}^*, \widehat{y}_{c_i,k}^*)).$$

    where $\mathsf{G}_{c_i}^*$ is defined over $\mathsf{C}_{c_i}^*$ and $\Delta_{c_i}^*$. Return $\mathsf{sk} = (\Gamma_{c_i}^*, \{\mathsf{sk}_{c_i,k}\}_{k\in\Gamma_{c_i}^*})$ to $\mathcal{A}$.

We see that in the real world of RDMPC, $\mathcal{B}$ simulates $\mathsf{Game}_{8,N}$; in the ideal world of RDMPC, $\mathcal{B}$ simulates Game_9. Therefore, the lemma follows readily. □

D Dynamic Slotted Reg-FE with Non-Adaptive Security

In this section, we present a generic construction of dynamic bounded collusion sReg-FE for P/Poly, achieving non-adaptive SIM security. Concretely, we first exploit sReg-ABE for NC^1 to construct 1-bound sReg-ABFE for $\mathsf{NC}^1 \circ$ P/Poly, then upgrade 1-bound security into Q-bound security. After that, we can construct weakly optimal sReg-FE for P/Poly with non-adaptive security from Q-bound sReg-ABFE for $\mathsf{NC}^1 \circ$ P/Poly. Lastly, dynamic bounded collusion sReg-FE with non-adaptive security can be generically derived from weakly optimal sReg-FE.

D.1 Slotted Reg-ABFE

A slotted Reg-ABFE (sReg-ABFE for short) for policy family $P : A \to \{0,1\}$ and circuit family $C : X \to Z$ is defined as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C) \to \mathsf{crs}$. It takes as input the security parameter 1^λ, slot amount 1^L, collusion bound 1^Q, policy decryption P, and circuit description C, then outputs a common reference string crs.
– $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i) \to (\mathsf{pk}_i, \mathsf{sk}_i)$. It takes as input the common reference string crs, a slot index $i \in [L]$ and an attribute $\mathsf{att}_i \in A$, then outputs a public/secret key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$.
– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i, \mathsf{att}_i) \to 0/1$. It takes as input the common reference string crs, a slot index $i \in [L]$, a public key pk_i, and an attribute $\mathsf{att}_i \in A$, then outputs a bit indicating whether pk_i is valid.
– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{att}_i, \mathsf{C}_i)\}_{i\in[L]}) \to (\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]})$. It takes as input crs and a series of public keys, attributes and circuits $(\mathsf{pk}_i, \mathsf{att}_i, \mathsf{C}_i)$ for $i \in [L]$, then outputs the master public key mpk and a series of helper keys hsk_j for all $j \in [L]$. This algorithm is deterministic.
– $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x) \to \mathsf{ct}$. It takes as input the master public key mpk, a policy $\mathsf{P} \in P$, and a message $x \in X$, then outputs a ciphertext ct.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct}) \to z/\perp$. It takes as input a helper key hsk_{i^*}, a secret key sk_{i^*}, an attribute $\mathsf{att}_{i^*} \in A$ and a ciphertext ct, then outputs $z \in Z$ or an empty symbol $\perp$.

The above scheme is required to satisfy the following properties:

– **Compactness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all P and all C, and all $i \in [L]$, it requires

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, P, C, Q, \log L) \text{ and } |\mathsf{hsk}_i| = \mathsf{poly}(\lambda, P, C, Q, \log L).$$

– **Completeness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all policy families P, all circuit families C, all $i \in [L]$, and all $\mathsf{att}_i \in A$, we have

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i, \mathsf{att}_i) = 1 \;\middle|\; \begin{array}{l} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C) \\ (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i) \end{array}\right] = 1.$$

– **Correctness.** For all $\lambda, L \in \mathbb{N}$, all $Q \ll L$, all P and all C, all $\mathsf{C}_1, \ldots, \mathsf{C}_L \in C$ and $\mathsf{att}_1, \ldots, \mathsf{att}_L$, all $i^* \in [L]$, all $\mathsf{att}_{i^*} \in A$, all $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C)$, all $(\mathsf{pk}_{i^*}, \mathsf{sk}_{i^*}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i^*, \mathsf{att}_{i^*})$, all public keys $\{\mathsf{pk}_i\}_{i\in[L]\setminus\{i^*\}}$ such that $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i, \mathsf{att}_i) = 1$, all $\mathsf{P} \in P$ such that $\mathsf{P}(\mathsf{att}_{i^*}) = 1$, and all $x \in X$, and computing $(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) \leftarrow \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{att}_i, \mathsf{C}_i)\}_{i\in[L]})$, we have

$$\Pr\left[\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x)) = \mathsf{C}_{i^*}(x)\right] = 1.$$

– **Policy-Selective SIM Security.** For all $\lambda \in \mathbb{N}$ and all efficient adversaries $\mathcal{A}$, there exists a simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ such that it is computationally hard to distinguish the following two worlds:

- *Setup phase:* Initially, $\mathcal{A}$ specifies the input $(1^L, 1^Q, P, C)$ and multiple policies $\mathsf{P}_1, \ldots, \mathsf{P}_{Q_{\mathsf{Enc}}} \in P$, where Q_{Enc} denotes the total number of encryption queries. In the real world (left hand) or the ideal world (right hand), the challenger samples

$$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C) \;\text{ or }\; (\mathsf{crs}, \mathsf{td}) \leftarrow \widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, P, C).$$

  Return crs to $\mathcal{A}$. Here, td is the trapdoor of crs.
- *Pre-challenge phase:* The challenger initializes a dictionary $\{\mathcal{D}_i = \emptyset\}_{i\in[L]}$ and a set $C = \emptyset$. In this phase, $\mathcal{A}$ can freely make key generation/corruption queries to the following oracles:
  * $\mathsf{OGen}(i, \mathsf{att})$: In the real world (left hand) or the ideal world (right hand), the challenger runs

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}) \;\text{ or }\; (\mathsf{pk}, \mathsf{st}) \leftarrow \widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{att}, \mathsf{td}).$$

    Then, the challenger sets $\mathcal{D}_i[\mathsf{pk}] = (\mathsf{sk}, \mathsf{att})$ or $\mathcal{D}_i[\mathsf{pk}] = (\mathsf{st}, \mathsf{att})$, where st denotes the internal state of pk. Return pk to $\mathcal{A}$.
  * $\mathsf{OCor}(i, \mathsf{pk})$: The challenger returns the secret key sk to $\mathcal{A}$, where

$$(\mathsf{sk}, \mathsf{att}) = \mathcal{D}_i[\mathsf{pk}] \;\text{ or }\; \mathsf{sk} \leftarrow \widetilde{\mathsf{Gen}}_2(\mathcal{D}_i[\mathsf{pk}], \mathsf{td}, (\perp, \perp)).$$

    Then, the challenger updates $C = C \cup \{(i, \mathsf{pk}, \mathsf{att})\}$.
- *Challenge phase:* In this phase, $\mathcal{A}$ specifies L tuples of public keys, attributes and circuits $\{(\mathsf{pk}_i^*, \mathsf{att}_i^*, \mathsf{C}_i^*)\}_{i\in[L]}$. The challenger computes

$$(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) = \mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i^*, \mathsf{att}_i^*, \mathsf{C}_i^*)\}_{i\in[L]}).$$

  For each slot $i \in [L]$, it must fall in one of the following three cases:

$$\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \;\wedge\; (i, \mathsf{pk}_i^*) \notin C \qquad \text{(honest slot)}$$

$$\mathcal{D}_i[\mathsf{pk}_i^*] \neq \perp \;\wedge\; (i, \mathsf{pk}_i^*) \in C \qquad \text{(corrupted slot)}$$

$$\mathcal{D}_i[\mathsf{pk}_i^*] = \perp \qquad \text{(malicious slot)}$$

  Here, we allow the number of corrupted/malicious slots to exceed Q.

- *Post-challenge phase:* In this phase, $\mathcal{A}$ can make encryption/corruption queries as follows:
  * $\mathsf{OEnc}(\mathsf{P}_d, x_d)$: We assume that this is the d-th encryption query made by $\mathcal{A}$, where $d = 1, \ldots, Q_{\mathsf{Enc}}$. Record the corrupted/malicious slot indices in the set $\mathcal{K}_d$, where it requires $|\mathcal{K}_d| \leq Q$ and $\mathsf{P}_d(\mathsf{att}_j^*) = 1$ for each $j \in \mathcal{K}_d$. Next, in the real world (left hand) or the ideal world (right hand), it replies $\mathcal{A}$ with a ciphertext ct_d by sampling either

$$\mathsf{ct}_d \leftarrow \mathsf{Enc}(\mathsf{mpk}, \mathsf{P}_d, x_d) \ \text{ or } \ \mathsf{ct}_d \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_d, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x))\}_{j \in \mathcal{K}_d}).$$

  * $\mathsf{OCor}(i, \mathsf{pk})$: Same as in the pre-challenge phase, except that it requires either $\mathsf{P}_d(\mathsf{att}) = 0$ for all $d \in [Q_{\mathsf{Enc}}]$ where att can be found in $\mathcal{D}_i[\mathsf{pk}]$, or pk is not registered into the system.
- *Output phase:* $\mathcal{A}$ outputs a bit to indicate the real/ideal world.

In addition, for the above experiment, we require

(i) One can express the policies $\mathsf{P}_1, \ldots, \mathsf{P}_{Q_{\mathsf{Enc}}}$ as a new policy P^* that is of size independent of Q_{Enc}. Given any index $d \in [Q_{\mathsf{Enc}}]$, P_d can be efficiently derived from P^*, which is also independent of Q_{Enc}.

(ii) If $\mathcal{A}$ does not make any query to the oracle OEnc, it is still hard to distinguish the above two worlds, even when $\mathcal{A}$ colludes with more than Q malicious/corrupted users.

D.2 From Slotted Reg-ABE to 1-Bound Slotted Reg-ABFE

For some circuit family $C : X \to Z$ of circuit size n, let $\mathsf{U}(\cdot, \cdot)$ be the universal circuit such that $\mathsf{U}(\mathsf{C}, x) = \mathsf{C}(x)$ for any circuit $\mathsf{C} \in C$ and input $x \in X$. Let $P : A \to \{0,1\}$ be a policy family that can be expressed in the class of NC^1 circuits. Assuming a garbled circuit scheme $\mathsf{GC} = (\mathsf{Garble}, \mathsf{Eval})$, and an sReg-ABE scheme for NC^1 policies $\mathsf{sRABE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, our 1-bound sReg-ABFE works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C)$: It sets the collusion bound $Q = 1$ as default. Then, it proceeds as follows:
  - Sample

$$\mathsf{crs}_{\mathsf{sRABE}} \leftarrow \mathsf{sRABE.Setup}(1^\lambda, 1^L, P).$$

  - For all $i \in [L]$, $w \in [n]$ and $b \in \{0,1\}$, it samples dummy public keys $\widehat{\mathsf{pk}}_{i,w,b} \leftarrow \mathsf{sRABE.Gen}(\mathsf{crs}_{w,b}, i)$ (omit secret keys).

  Output the common reference string

$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{sRABE}}, \{\widehat{\mathsf{pk}}_{i,w,b}\}_{i \in [L], w \in [n], b \in \{0,1\}}).$$

– $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$: Sample

$$\forall w \in [n] \text{ and } b \in \{0,1\}, \ (\mathsf{pk}_{i,w,b}, \mathsf{sk}_{i,w,b}) \leftarrow \mathsf{sRABE.Gen}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{att}_i).$$

  Output the public key and secret key

$$\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w \in [n], b \in \{0,1\}}, \ \mathsf{sk}_i = \{\mathsf{sk}_{i,w,b}\}_{w \in [n], b \in \{0,1\}}.$$

– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i, \mathsf{att}_i)$: For all $w \in [n]$ and $b \in \{0,1\}$, it checks whether

$$\mathsf{sRABE.Ver}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{pk}_{i,w,b}, \mathsf{att}_i) \stackrel{?}{=} 1.$$

  If not, abort and output 0; otherwise, output 1.

– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{att}_i, \mathsf{C}_i)\}_{i \in [L]})$: Parse public key $\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w \in [n], b \in \{0,1\}}$ and circuit $\mathsf{C}_i = (\mathsf{C}_i[1], \ldots, \mathsf{C}_i[n]) \in \{0,1\}^n$, then it proceeds as follows:

- For all $i \in [L]$, $w \in [n]$ and $b \in \{0,1\}$, set

$$\overline{\mathsf{pk}}_{i,w,b} := \begin{cases} \mathsf{pk}_{i,w,b}, & \text{when } \mathsf{C}_i[w] = b, \\ \widehat{\mathsf{pk}}_{i,w,b}, & \text{otherwise.} \end{cases}$$

- Run $(\mathsf{mpk}_{w,b}, \{\mathsf{hsk}_{j,w,b}\}_{j\in[L]}) \leftarrow \mathsf{sRABE.Agg}(\mathsf{crs}_{\mathsf{sRABE}}, \{(\overline{\mathsf{pk}}_{i,w,b}, \mathsf{att}_i)\}_{i\in[L]})$.

Output

$$\mathsf{mpk} = \{\mathsf{mpk}_{w,b}\}_{w\in[n],b\in\{0,1\}},\ \mathsf{hsk}_j = (\mathsf{C}_j, \{\mathsf{hsk}_{j,w,\mathsf{C}_j[w]}\}_{w\in[n]}).$$

– $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x)$: Let $\mathsf{U}[x]$ be a universal circuit with x hard-wired. Run

$$(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{U}[x]).$$

For all $w \in [n], b \in \{0,1\}$, it computes

$$\mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}, \mathsf{lab}_{w,b}).$$

Output the ciphertext $\mathsf{ct} = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct})$: Parse $\mathsf{hsk}_{i^*} = (\mathsf{C}_{i^*}, \{\mathsf{hsk}_{i^*,w,\mathsf{C}_j[w]}\}_{w\in[n]})$ and $\mathsf{sk}_i = \{\mathsf{sk}_{i^*,w,b}\}_{w\in[n],b\in\{0,1\}}$. If $\mathsf{P}(\mathsf{att}_{i^*}) = 1$, it can recover the labels

$$\forall w \in [n],\ \mathsf{lab}_{w,\mathsf{C}_{i^*}[w]} \leftarrow \mathsf{sRABE.Dec}(\mathsf{hsk}_{i^*,w,\mathsf{C}_{i^*}[w]}, \mathsf{sk}_{i^*,w,\mathsf{C}_{i^*}[w]}, \mathsf{ct}_{w,\mathsf{C}_{i^*}[w]}).$$

Output the evaluation result $z \leftarrow \mathsf{GC.Eval}(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,b_w}\}_{w\in[n]})$.

Completeness and Compactness. The completeness follows the underlying scheme sRABE and we omit details. For compactness, by the compactness of sRABE, we have

$$|\mathsf{mpk}| = 2n \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, P \log L),$$
$$|\mathsf{hsk}_j| = 2n \cdot \mathsf{poly}(\lambda, P, \log L) = \mathsf{poly}(\lambda, n, P, \log L)$$

where n is the circuit size. Thus, our construction meets the compactness requirement.
Correctness. In algorithm Dec, we have $\mathsf{sk}_{i^*} = (\{\mathsf{sk}_{i^*,w,b}\}_{w\in[n],b\in\{0,1\}})$ and

$$\mathsf{ct} = \left(\tilde{\mathsf{U}}, \begin{pmatrix} \mathsf{sRABE.Enc}(\mathsf{mpk}_{1,0}, \mathsf{P}, \mathsf{lab}_{1,0}) & \cdots & \mathsf{sRABE.Enc}(\mathsf{mpk}_{n,0}, \mathsf{P}, \mathsf{lab}_{n,0}) \\ \mathsf{sRABE.Enc}(\mathsf{mpk}_{1,1}, \mathsf{P}, \mathsf{lab}_{1,1}) & \cdots & \mathsf{sRABE.Enc}(\mathsf{mpk}_{n,1}, \mathsf{P}, \mathsf{lab}_{n,1}) \end{pmatrix}\right).$$

Here, each $\mathsf{mpk}_{w,b}$ is generated from $\{\overline{\mathsf{pk}}_{i,w,b}\}_{i\in[L]}$. By Eq. (13), we see that $\mathsf{sk}_{i^*,w,b}$ is the real secret key of $\overline{\mathsf{pk}}_{i^*,w,b}$ only when $\mathsf{C}_{i^*}[w] = b$. Thus, the correctness of sRABE ensures that $\{\mathsf{lab}_{i^*,w,\mathsf{C}_{i^*}[w]}\}_{w\in[n]}$ can be properly recovered. Then, by the correctness of garbled circuits, it must hold that $z = \mathsf{U}(\mathsf{C}_{i^*}, x) = \mathsf{C}_{i^*}(x)$. Therefore, the correctness follows readily.

Theorem 18. *Assuming* GC *is secure and* sRABE *achieves policy-selective IND security, then our construction achieves 1-bound policy-selective SIM security.*

Proof. The proof is analogous to [ZCHZ24]. We define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, P, C)$: This algorithm is identical to Setup (setting $\mathsf{td} = \perp$).
– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{att}_i, \mathsf{td})$: This algorithm is identical to Gen. Namely, it samples

$$\forall w \in [n] \text{ and } b \in \{0,1\},\ (\mathsf{pk}_{i,w,b}, \mathsf{sk}_{i,w,b}) \leftarrow \mathsf{sRABE.Gen}(\mathsf{crs}_{\mathsf{sRABE}}, i, \mathsf{att}_i).$$

Output $\mathsf{pk}_i = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$ and $\mathsf{st}_i = \{\mathsf{sk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$.

– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}, \mathsf{att}, \mathsf{td}, (\bot, \bot))$: This algorithm outputs the secret key $\mathsf{sk} = \mathsf{st}$.
– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_d, (f, z))$: Parse $\mathsf{mpk} = \{\mathsf{mpk}_{w,b}\}_{w\in[n],b\in\{0,1\}}$. For the d-th encryption query on the policy P_d and the message x_d, where $d \in [Q_{\mathsf{Enc}}]$, it considers two cases:
  - If there does not exist any corrupt/malicious slot, it must have $f = \bot$ and $z = \bot$, so the challenger generates
  $$\forall w \in [n] \text{ and } b \in \{0,1\},\ \mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{m}_{w,b})$$
  where $\mathsf{m}_{w,b} \leftarrow \{0,1\}^\lambda$.
  - Otherwise, we have $f = \mathsf{C}_i^*$ and $z = \mathsf{C}_i^*(x_d))$ for some corrupt/malicious slot i such that $\mathsf{P}_d(\mathsf{att}_i^*) = 1$. Then, it samples
  $$\forall w \in [n],\ \mathsf{lab}_{w,1-\mathsf{C}_i^*[w]} \leftarrow \{0,1\}^\lambda,$$
  $$\left(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,\mathsf{C}_i^*[w]}\}_{w\in[n]}\right) \leftarrow \mathsf{GC.}\widetilde{\mathsf{Garble}}(1^\lambda, \mathsf{C}_i^*(x_d)).$$
  For each $w \in [n]$ and $b \in \{0,1\}$, run $\mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{lab}_{w,b})$.

  Output the ciphertext $\mathsf{ct}_d = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$.

Game Sequence. We prove Theorem 18 via a sequence of games as follows:

– Game_0: Real Game.
– $\mathsf{Game}_{1,d}(d \in [Q_{\mathsf{Enc}}])$: Same as $\mathsf{Game}_{1,d-1}$ except that to respond the d-th encryption query to the oracle OEnc, it employs $\widetilde{\mathsf{Enc}}$, instead of Enc.
– Game_2: Same as $\mathsf{Game}_{1,Q_{\mathsf{Enc}}}$ except that it completely replaces (Setup, Gen, Enc) with $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$.

We see that in the final game Game_2, each ciphertext ct_d can be simulated only using $(\mathsf{C}_i^*, \mathsf{C}_i^*(x_d))$ for some corrupted slot i. This is exactly the simulation world.

Since $\mathsf{Game}_{1,d} \equiv \mathsf{Game}_2$, it suffices to prove the indistinguishability between $\mathsf{Game}_{1,d-1}$ and $\mathsf{Game}_{1,d}$ via the following lemma.

Lemma 38. *By the security of* GC *and* sRABE, *we claim* $\mathsf{Game}_{1,d-1} \approx_c \mathsf{Game}_d$.

Proof. We define the following sub-games to prove the lemma:

– $\mathsf{Game}_{1,d-1,\ell}(\ell \in [n])$: This game is identical to $\mathsf{Game}_{1,d-1,\ell-1}$ except that to respond the d-th encryption query, it changes the encrypted labels. Concretely, it defines a universal circuit $\mathsf{U}[x_d]$ with the queried message x_d hard-wired and samples
$$(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{U}[x_d]).$$
Let $c_d \in [L]$ be the unique corrupted/malicious slot such that $\mathsf{P}_d(\mathsf{att}_{c_d}^*) = 1$,
  - For $w \le \ell$, set
  $$\mathsf{m}_{w,\mathsf{C}_{c_d}^*[w]} = \mathsf{lab}_{w,\mathsf{C}_{c_d}^*[w]},\ \mathsf{m}_{w,1-\mathsf{C}_{c_d}^*[w]} \leftarrow \{0,1\}^\lambda.$$
  - For $w > \ell$ and $b \in \{0,1\}$, set $\mathsf{m}_{w,b} = \mathsf{lab}_{w,b}$.

  Next, it computes
  $$\forall w \in [n] \text{ and } b \in \{0,1\},\ \mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{m}_{w,b}).$$
  Output the ciphertext $\mathsf{ct}_d = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$.
– $\mathsf{Game}_{1,d-1,n+1}$: This game is identical to $\mathsf{Game}_{1,d-1,n}$ except that when generating the challenge ciphertext, it employs $\mathsf{GC.}\widetilde{\mathsf{Garble}}$, instead of GC.Garble.

We can see that $\mathsf{Game}_{1,d-1} \equiv \mathsf{Game}_{1,d-1,0}$ and $\mathsf{Game}_{1,d-1,n+1} \equiv \mathsf{Game}_{1,d}$. Therefore, it suffices to prove $\mathsf{Game}_{1,d-1,\ell-1} \approx_c \mathsf{Game}_{1,d-1,\ell}$ for each $\ell \in [n+1]$.

Lemma 39. *By the policy-selective IND security of* sRABE, $\mathsf{Game}_{1,d-1,\ell-1} \approx_c \mathsf{Game}_{1,d-1,\ell}$ *for each* $\ell \in [n]$.

Proof. If $\mathcal{A}$ can distinguish $\mathsf{Game}_{1,d-1,\ell-1}$ from $\mathsf{Game}_{1,d-1,\ell}$, we can build an algorithm $\mathcal{B}$ to break the policy-selective IND security of sRABE. Initially, $\mathcal{B}$ receives the input $(1^L, P, C)$ and a policy P^* from $\mathcal{A}$, where P^* can be used to efficiently derive a policy P_d for the d-th encryption query. Then, $\mathcal{B}$ proceeds as follows:

- *Setup phase:* It sends the input $(1^L, P)$ and a challenge policy P_d to the challenger of sRABE to obtain $\mathsf{crs}_{\mathsf{sRABE}}$. Then, it makes key generation queries to the oracle sRABE.OGen to obtain dummy keys $\{\widehat{\mathsf{pk}}_{i,w,b}\}_{i\in[L],w\in[n],b\in\{0,1\}}$. Return the following common reference string to $\mathcal{A}$:

$$\mathsf{crs} = (\mathsf{crs}_{\mathsf{sRABE}}, \{\widehat{\mathsf{pk}}_{i,w,b}\}_{i\in[L],w\in[n],b\in\{0,1\}}).$$

- *Pre-challenge phase:* Here, $\mathcal{A}$ can query the oracles below:
  - $\mathsf{OGen}(i, \mathsf{att})$: For each $w \in [n]$ and $b \in \{0,1\}$, $\mathcal{B}$ queries the oracle sRABE.OGen on input (i, att) to obtain the public key $\mathsf{pk}_{i,k,b}$. Return $\mathsf{pk} = \{\mathsf{pk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$ to $\mathcal{A}$.
  - $\mathsf{OCor}(i, \mathsf{pk})$: For each $w \in [n]$, $\mathcal{B}$ queries the oracle sRABE.OCor on input $(i, \mathsf{pk}_{i,w,b})$ to obtain the secret key $\mathsf{sk}_{i,w,b}$. Return the secret key $\mathsf{sk} = \{\mathsf{sk}_{i,w,b}\}_{w\in[n],b\in\{0,1\}}$ to $\mathcal{A}$.
- *Challenge phase:* It receives public keys $\{\mathsf{pk}_i^*\}_{i\in[L]}$, attributes $\{\mathsf{att}_i^*\}_{i\in[L]}$, and circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$. With such information, $\mathcal{B}$ can compute the master public key $\mathsf{mpk} = \{\mathsf{mpk}_{w,b}\}_{w\in[n],b\in\{0,1\}}$ as in algorithm Agg.
- *Post-challenge phase:* In this phase, $\mathcal{A}$ can query the corruption/encryption oracles as follows:
  - $\mathsf{OEnc}(\mathsf{P}_j, x_j)$: For $j < d$, $\mathcal{B}$ generates the ciphertext ct_j as in algorithm $\widetilde{\mathsf{Enc}}$; For $j > d$, $\mathcal{B}$ generates the ciphertext ct_j as in algorithm Enc. Next, we just consider the case when $j = d$. Concretely, $\mathcal{B}$ defines a universal circuit $\mathsf{U}[x_d]$ with x_d hard-wired and samples

$$(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,b}\}_{w\in[n],b\in\{0,1\}}) \leftarrow \mathsf{GC.Garble}(1^\lambda, \mathsf{U}[x_d]).$$

    Let $c_d \in [L]$ be the unique corrupted/malicious slot such that $\mathsf{P}_d(\mathsf{att}_{c_d}^*) = 1$, then it proceeds as follows:
    * For $w < \ell$, set

$$\mathsf{m}_{w,\mathsf{C}_{c_d}^*[w]} = \mathsf{lab}_{w,\mathsf{C}_{c_d}^*[w]},\ \mathsf{m}_{w,1-\mathsf{C}_{c_d}^*[w]} \leftarrow \{0,1\}^\lambda,$$
$$\forall b \in \{0,1\},\ \mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{m}_{w,b})$$

    * For $w = \ell$, $\mathcal{B}$ submits the public keys $\{\mathsf{pk}^*_{i,\ell,1-\mathsf{C}_{c_d}^*[\ell]}\}_{i\in[L]}$, attributes $\{\mathsf{att}_i^*\}_{i\in[L]}$, and a pair of challenge messages $(\mathsf{lab}_{\ell,1-\mathsf{C}_{c_d}^*[\ell]}, \mathsf{m})$, where $\mathsf{m} \leftarrow \{0,1\}^\lambda$, to the challenger of sRABE; $\mathcal{B}$ receives the ciphertext $\mathsf{ct}_{\ell,1-\mathsf{C}_{c_d}^*[\ell]}$. Then, $\mathcal{B}$ generates the other ciphertext

$$\mathsf{ct}_{\ell,\mathsf{C}_{c_d}^*[\ell]} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{lab}_{\ell,\mathsf{C}_{c_d}^*[\ell]}).$$

    * For $w > \ell$ and $b \in \{0,1\}$, set $\mathsf{m}_{w,b} = \mathsf{lab}_{w,b}$ and generate

$$\mathsf{ct}_{w,b} \leftarrow \mathsf{sRABE.Enc}(\mathsf{mpk}_{w,b}, \mathsf{P}_d, \mathsf{lab}_{w,b})$$

    Return the ciphertext $\mathsf{ct}_d = (\tilde{\mathsf{U}}, \{\mathsf{ct}_{w,b}\}_{w\in[n],b\in\{0,1\}})$ to $\mathcal{A}$.
  - $\mathsf{OCor}(i, \mathsf{pk})$: $\mathcal{B}$ acts as in the pre-challenge phase. Note that it requires either $\mathsf{P}_d(\mathsf{att}) = 0$ for all $d \in [Q_{\mathsf{Enc}}]$ where att can be found in $\mathcal{D}_i[\mathsf{pk}]$, or pk is not registered into the system. These corruption queries are admissible for the challenger of sRABE.

We see that if the challenge ciphertext of sRABE encrypts m, $\mathcal{B}$ will simulates $\mathsf{Game}_{1,d-1,\ell}$; otherwise, it simulates $\mathsf{Game}_{1,d-1,\ell-1}$. Therefore, the lemma follows readily. □

Lemma 40. *By the security of* GC*, we claim* $\mathsf{Game}_{1,d-1,n} \approx_c \mathsf{Game}_{1,d-1,n+1}$.

Proof. In $\mathsf{Game}_{1,d-1,n+1}$, the generation of ct_d now only require the knowledge of a garbled circuit $\tilde{\mathsf{U}}$ and half labels $\{\mathsf{lab}_{w,\mathsf{C}^*_{c_d}[w]}\}_{w\in[n]}$. Therefore, we can rely on the simulator $\mathsf{GC}.\widetilde{\mathsf{Garble}}$ to simulate $(\tilde{\mathsf{U}}, \{\mathsf{lab}_{w,\mathsf{C}^*_{c_d}[w]}\}_{w\in[n]})$ and hence generate ct_d. This exactly claims $\mathsf{Game}_{1,d-1,n} \approx_c \mathsf{Game}_{1,d-1,n+1}$. □

D.3 From 1-Bound Security to Q-Bound Security

For some circuit family $C : X \to Z$ of circuit size n, let $t = \Theta(Q^2\lambda), m = \Theta(t), v = \Theta(Q\lambda), p_1 = \Theta(m), p_2 = \Theta(v), N = \Theta(Q^2\lambda p_1)$ and $S = S(Q\lambda p_2)$, where p_1 and p_2 are primes. For any circuit $\mathsf{C} \in C$ and set $\Delta \subseteq [S]$ of size v, we define a circuit family G in which each circuit $\mathsf{G} \in G$ with input $\widehat{x}$ can compute

$$\mathsf{G}(\widehat{x}) = \mathsf{RDMPC.Local}((\mathsf{C}, \Delta), \widehat{x})$$

where (C, Δ) is the constant hard-wired in G. Let $n' = \mathsf{poly}(\lambda, n, Q)$ be the circuit size of G and $\mathsf{P} : \{0,1\}^\lambda \leftarrow \{0,1\}$ be an NC^1 circuit family.

Assuming a reusable dynamic MPC scheme $\mathsf{RDMPC} = (\mathsf{CktEnc}, \mathsf{InpEnc}, \mathsf{Local}, \mathsf{Decode})$, and a 1-bound sReg-ABFE scheme $\mathsf{osRABFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, then our Q-bound sReg-ABFE scheme works as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, P, C)$:
  - Sample a common reference string
  $$\mathsf{crs}_{\mathsf{osRABFE}} \leftarrow \mathsf{osRABFE.Setup}(1^\lambda, 1^L, G).$$
  - Fix an attribute $\mathsf{att} \in A$ and sample dummy public keys
  $$\forall i \in [L] \text{ and } k \in [N],\ \widehat{\mathsf{pk}}_{i,k} \leftarrow \mathsf{osRABFE.Gen}(\mathsf{crs}_{\mathsf{osRABFE}}, i, \mathsf{att}).$$
  - Let $\mathsf{Hash}_1 : \{0,1\}^* \to \mathbb{Z}^\lambda_{p_1}$ and $\mathsf{Hash}_2 : \{0,1\}^* \to \mathbb{Z}^\lambda_{p_2}$ be two collision-resistant hash function families. Sample
  $$\mathsf{hk}_1 \leftarrow \mathsf{Hash}_1.\mathsf{Gen}(1^\lambda),\ \mathsf{hk}_2 \leftarrow \mathsf{Hash}_2.\mathsf{Gen}(1^\lambda).$$
  - Define two injective maps $\rho_1 : [Dt+1] \times \mathbb{Z}_{p_1} \to [N]$ and $\rho_2 : [Dt+1] \times \mathbb{Z}_{p_2} \to [S]$.

  Output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{osRABFE}}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$.
- $\mathsf{Gen}(\mathsf{crs}, i, \mathsf{att}_i)$: Sample
$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{sk}_{i,k}) \leftarrow \mathsf{osRABFE.Gen}(\mathsf{crs}_{\mathsf{osRABFE}}, i, \mathsf{att}_i).$$
  Output $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ and $\mathsf{sk}_i = \{\mathsf{sk}_{i,k}\}_{k\in[N]}$.
- $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i, \mathsf{att}_i)$: It check whether
$$\forall k \in [N],\ \mathsf{osRABFE.Ver}(\mathsf{crs}_{\mathsf{osRABFE}}, i, \mathsf{pk}_{i,k}, \mathsf{att}_i) \stackrel{?}{=} 1.$$
  If not, abort and output 0; otherwise, output 1.
- $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{att}_i, \mathsf{C}_i)\}_{i\in[L]})$: Parse $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$, then it proceeds as follows:
  - For $i \in [L]$, it computes coefficients $(a_{i,0}, \ldots, a_{i,\lambda-1}) = \mathsf{Hash}_1.\mathsf{H}(\mathsf{hk}_1, i||\mathsf{pk}_i)$ and hence defines a polynomial function of degree $\lambda - 1$:
  $$f_i(x) = a_{i,\lambda-1}x^{\lambda-1} + \cdots + a_{i,1}x + a_{i,0}.$$
  For each $j \in [m]$, evaluate the point $f_i(j)$ and then set $\gamma_{i,j} = \rho_1(j, f_i(j))$. Set $\Gamma_i = \{\gamma_{i,j}\}_{j\in[m]}$.

- For $i \in [L]$, it computes coefficients $(c_{i,0}, \ldots, c_{i,v-1}) = \mathsf{Hash}_2.\mathsf{H}(\mathsf{hk}_2, i||\mathsf{pk}_i)$ and hence defines a polynomial function of degree $\lambda - 1$:
$$g_i(x) = c_{i,\lambda-1}x^{\lambda-1} + \cdots + c_{i,1}x + c_{i,0}.$$
For each $j \in [v]$, evaluate the point $g_i(j)$ and then set $\delta_{i,j} = \rho_2(j, g_i(j))$. Set $\Delta_i = \{\delta_{i,j}\}_{j\in[v]}$.
- For each $i \in [L]$ and $k \in [N]$, define circuits $\mathsf{G}_{i,k} = (\mathsf{G}_{i,k}[1], \ldots, \mathsf{G}_{i,k}[n']) \in \{0,1\}^{n'}$, where $\mathsf{G}_{i,k}$ takes $\widehat{\mathsf{C}}_{i,k} = (\mathsf{C}_i, \Delta_i)$ as constant. Note that
$$(\widehat{\mathsf{C}}_{i,1}, \ldots, \widehat{\mathsf{C}}_{i,N}) = \mathsf{RDMPC.CktEnc}(1^\lambda, 1^Q, \mathsf{C}_i, \Delta_i).$$
- For all $i \in [L]$ and $k \in [N]$, set
$$\overline{\mathsf{pk}}_{i,k} := \begin{cases} \mathsf{pk}_{i,k}, & \text{when } k \in \Gamma_i, \\ \widehat{\mathsf{pk}}_{i,k}, & \text{otherwise.} \end{cases}$$
- For all $k \in [N]$, run
$$(\mathsf{mpk}_k, \{\mathsf{hsk}_{j,k}\}_{j\in[L]}) = \mathsf{osRABFE.Agg}(\mathsf{crs}_{\mathsf{osRABFE}}, \{(\overline{\mathsf{pk}}_{i,k}, \mathsf{att}_i, \mathsf{G}_i)\}_{i\in[L]}).$$

Output $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$ and $\mathsf{hsk}_j = (\Gamma_j, \{\mathsf{hsk}_{j,k}\}_{k\in\Gamma_j})$.

– $\mathsf{Enc}(\mathsf{mpk}, \mathsf{P}, x)$: Parse $\mathsf{mpk} = \{\mathsf{mpk}_k\}_{k\in[N]}$, then compute
$$(\widehat{x}_1, \ldots, \widehat{x}_N) \leftarrow \mathsf{RDMPC.InpEnc}(1^\lambda, 1^Q, x),$$
$$\forall k \in [N],\ \mathsf{ct}_k \leftarrow \mathsf{osRABFE.Enc}(\mathsf{mpk}_k, \mathsf{P}, \widehat{x}_k).$$

Output $\mathsf{ct} = (\mathsf{ct}_1, \ldots, \mathsf{ct}_N)$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct})$: Parse $\mathsf{hsk}_{i^*} = (\Gamma_{i^*}, \{\mathsf{hsk}_{i^*,k}\}_{k\in\Gamma_{i^*}})$ and $\mathsf{sk}_{i^*} = \{\mathsf{sk}_{i^*,k}\}_{k\in[N]}$. Compute
$$\forall k \in \Gamma_{i^*},\ \widehat{y}_k \leftarrow \mathsf{osRABFE.Dec}(\mathsf{hsk}_{i^*,k}, \mathsf{sk}_{i^*,k}, \mathsf{att}_{i^*}, \mathsf{ct}_k).$$

Output $z = \mathsf{RDMPC.Decode}(\{\widehat{y}_k\}_{k\in\Gamma_{i^*}}, \Gamma_{i^*})$.

Completeness, Compactness and Correctness. For completeness, it follows the underlying osRABFE scheme. For compactness, by the compactness of osRABFE, we have
$$|\mathsf{mpk}| = N \cdot \mathsf{poly}(\lambda, n', P, \log L) = \mathsf{poly}(\lambda, n, P, Q, \log L),$$
$$|\mathsf{hsk}_j| = N \cdot \mathsf{poly}(\lambda, n', P, \log L) = \mathsf{poly}(\lambda, n, P, Q, \log L),$$
where N depends on the collusion bound Q, and $n' = \mathsf{poly}(\lambda, n, Q)$. Thus, our construction satisfies the compactness requirement.

On the other hand, the correctness of the above construction follows the correctness of the underlying osRABFE scheme and RDMPC protocol.

Theorem 19. *Assuming* osRABFE *achieves policy-selective SIM security, then the above construction achieves Q-bound policy-selective SIM security.*

Proof. The proof is analogous to Theorem 15. Here, we only define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: This algorithm is identical to Setup, except that it runs
$$(\mathsf{crs}_{\mathsf{osRABFE}}, \mathsf{td}) \leftarrow \mathsf{osRABFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, G).$$

Output $\mathsf{crs} = (\mathsf{crs}_{\mathsf{osRABFE}}, \{\widehat{\mathsf{pk}}_{i,k}\}_{i\in[L],k\in[N]}, \mathsf{hk}_1, \mathsf{hk}_2, \rho_1, \rho_2)$ and td.

– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{att}_i, \mathsf{td})$: This algorithm samples

$$\forall k \in [N],\ (\mathsf{pk}_{i,k}, \mathsf{st}_{i,k}) \leftarrow \mathsf{osRABFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_{\mathsf{osRABFE}}, i, \mathsf{att}_i, \mathsf{td}).$$

Output $\mathsf{pk}_i = \{\mathsf{pk}_{i,k}\}_{k\in[N]}$ and $\mathsf{st}_i = \{\mathsf{st}_{i,k}\}_{k\in[N]}$.

– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{att}_i, \mathsf{td}, (\bot, \bot))$: This algorithm samples

$$\forall k \in [N],\ \mathsf{sk}_{i,k} \leftarrow \mathsf{osRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_{i,k}, \mathsf{att}_i, \mathsf{td}, (\bot, \bot)).$$

Output $\mathsf{sk}_i = \{\mathsf{sk}_{i,k}\}_{k\in[N]}$.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_d, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x_d))\}_{i\in\mathcal{K}_d})$: For d-th encryption query on the policy P_d and the message x_d, the set $\mathcal{K}_d$ records all corrupted/malicious slot indices j such that $\mathsf{P}_d(\mathsf{att}_j^*) = 1$. Since we consider non-adaptive security, it can assume that $|\mathcal{K}_d| = Q$ and the challenger knows the sets $\{\Gamma_j^*\}_{j\in\mathcal{K}_d}, \Delta_1^*, \ldots, \Delta_L^*$. Therefore, it can define the set

$$\mathcal{T}_d = \bigcup_{i\in\mathcal{K}_d, j\in\mathcal{K}_d\setminus\{i\}} (\Gamma_i^* \cap \Gamma_j^*).$$

Then, it proceeds as follows:

- It simulates the following input and local encodings

$$(\{\widehat{x}_{d,k}\}_{k\in\mathcal{T}_d}, \{\widehat{y}_{j,k}\}_{j\in\mathcal{K}_d, k\in\Gamma_j^*}) \leftarrow \mathsf{RDMPC}.\widetilde{\mathsf{InpEnc}}(1^{|x_d|}, \mathcal{T}_d, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x_d))\}_{i\in\mathcal{K}_d})$$

  where $\widehat{y}_{j,k}^* = \mathsf{RDMPC.Local}(\widehat{\mathsf{C}}_{j,k}^*, \widehat{x}_{d,k})$ for all $k \in \Gamma_j^* \cap \mathcal{T}_d$.
- For all $k \in \mathcal{T}_d$, it generates

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{osRABFE.Enc}(\mathsf{mpk}_k, \mathsf{P}_d, \widehat{x}_{d,k}).$$

- For all $k \notin \mathcal{T}_d$, if there is a slot $i \in \mathcal{K}_d$ such that $k \in \Gamma_i^*$ and $k \notin \Gamma_j^*$ for any $j \in \mathcal{K}_d \setminus \{i\}$, we can simulate

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, \mathsf{P}_d, (\mathsf{G}_i^*, \widehat{y}_{i,k}^*)).$$

  Otherwise, we simulate

$$\mathsf{ct}_{d,k} \leftarrow \mathsf{osRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, \mathsf{P}_d, (\bot, \bot)).$$

Finally, output the ciphertext $\mathsf{ct}_d = (\mathsf{ct}_{d,1}, \ldots, \mathsf{ct}_{d,N})$.

D.4 From Q-Bound Slotted Reg-ABFE to Weakly Optimal Slotted Reg-FE

Assuming a weak pseudorandom function family $\mathsf{wPRF} : \{0,1\}^\lambda \times \{0,1\}^\lambda \to [Q]$ that is computable in NC^1 circuits, we define a policy family $P : [L] \leftarrow \{0,1\}$. In particular, the circuit $\mathsf{P} \in P$ with input $\mathsf{att} \in \{0,1\}^\lambda$ works as follows:

$$\mathsf{P}(\mathsf{att}) = 1 \iff \mathsf{wPRF.Eval}(\mathsf{k}, \mathsf{att}) = \mathsf{tag}. \tag{16}$$

where $\mathsf{k} \in \{0,1\}^\lambda$ is the key of the weak pseudorandom function, and $\mathsf{tag} \in [Q]$ is a tag, both of which are hardwired into P. Apparently, P is computable in the class of NC^1 circuits, and the size of P is bounded by $\mathsf{poly}(\lambda, \log Q, \log L)$.

Assume a Q-bound sReg-ABFE scheme $\mathsf{sRABFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, then our weakly optimal sReg-FE for circuits works as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, 1^Q, C)$: Sample $\mathsf{att}_i \leftarrow \{0,1\}^\lambda$ for each $i \in [L]$ and

$$\mathsf{crs}_{\mathsf{sRABFE}} \leftarrow \mathsf{sRABFE.Setup}(1^\lambda, 1^L, 1^\lambda, P, C).$$

Output the common reference string $\mathsf{crs} = (\mathsf{crs}_{\mathsf{sRABFE}}, \{\mathsf{att}_i\}_{i\in[L]})$.

– $\mathsf{Gen}(\mathsf{crs}, i)$: Sample

$$(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{sRABFE.Gen}(\mathsf{crs}_{\mathsf{sRABFE}}, i, \mathsf{att}_i)$$

Output the public key pk_i and secret key sk_i.

– $\mathsf{Ver}(\mathsf{crs}, i, \mathsf{pk}_i)$: It checks whether

$$\mathsf{sRABFE.Ver}(\mathsf{crs}_{\mathsf{sRABFE}}, i, (\mathsf{pk}_i, \mathsf{att}_i)) \stackrel{?}{=} 1.$$

If not, output 0; otherwise, output 1.

– $\mathsf{Agg}(\mathsf{crs}, \{(\mathsf{pk}_i, \mathsf{C}_i)\}_{i\in[L]})$: Run

$$(\mathsf{mpk}, \{\mathsf{hsk}_j\}_{j\in[L]}) \leftarrow \mathsf{sRABFE.Agg}(\mathsf{crs}_{\mathsf{sRABFE}}, \{(\mathsf{pk}_i, \mathsf{att}_i, \mathsf{C}_i)\}_{i\in[L]}).$$

Output the master public key mpk and helper keys hsk_j for $j \in [L]$.

– $\mathsf{Enc}(\mathsf{mpk}, x)$: Sample

$$\mathsf{k} \leftarrow \mathsf{wPRF.Gen}(1^\lambda),$$
$$\forall \mathsf{tag} \in [Q],\ \mathsf{ct}_{\mathsf{tag}} \leftarrow \mathsf{sRABFE.Enc}(\mathsf{mpk}, \mathsf{P}_{\mathsf{k},\mathsf{tag}}, x)$$

where the policy $\mathsf{P}_{\mathsf{k},\mathsf{tag}}$ is as defined in Eq. (16) which takes $(\mathsf{k}, \mathsf{tag})$ as constants. Output the ciphertext $\mathsf{ct} = (\mathsf{k}, \mathsf{ct}_1, \ldots, \mathsf{ct}_Q)$.

– $\mathsf{Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{ct})$: Run

$$\mathsf{tag}^* = \mathsf{wPRF.Eval}(\mathsf{k}, \mathsf{att}_i^*),\ z \leftarrow \mathsf{sRABFE.Dec}(\mathsf{hsk}_{i^*}, \mathsf{sk}_{i^*}, \mathsf{att}_{i^*}, \mathsf{ct}_{\mathsf{tag}^*}).$$

Output the evaluation result z.

Completeness, Correctness and Compactness. The completeness and correctness follow the underlying sRABFE scheme. For compactness, recall that we set the collusion bound as 1^λ and the policy $\mathsf{P} \in P$ is of size $\mathsf{poly}(\lambda, \log Q, \log L)$, so we have

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, C, \log Q, \log L),\ |\mathsf{hsk}_j| = \mathsf{poly}(\lambda, C, \log Q, \log L).$$

Thus, our construction satisfies the weakly optimal compactness.

Theorem 20. *Assuming* sRABFE *achieves Q-bound policy-selective SIM security and* wPRF *achieves the pseudorandomness, then the above construction achieves Q-bound non-adaptive SIM security.*

Proof. Let $\mathcal{K}$ be the set of all corrupted/malicious slot indices, and for all $\mathsf{tag}^* \in [Q]$, let $\mathsf{cnt}_{\mathsf{tag}^*}$ be the accumulated number of corrupted/malicious slots such that $\mathsf{tag}^* = \mathsf{wPRF.Eval}(\mathsf{k}, j)$ for all $j \in \mathcal{K}$. Here, we define the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ that works as follows:

– $\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^Q, C)$: Sample $\mathsf{att}_i \leftarrow \{0,1\}^\lambda$ for each $i \in [L]$ and

$$\mathsf{k} \leftarrow \mathsf{wPRF.Gen}(1^\lambda),$$
$$(\mathsf{crs}_{\mathsf{sRABFE}}, \mathsf{td}_{\mathsf{sRABFE}}) \leftarrow \mathsf{sRABFE.}\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, 1^\lambda, P, C).$$

Output the common reference $\mathsf{crs} = (\mathsf{crs}_{\mathsf{sRABFE}}, \{\mathsf{att}_i\}_{i\in[L]})$ and trapdoor $\mathsf{td} = \mathsf{td}_{\mathsf{sRABFE}}$.

– $\widetilde{\mathsf{Gen}}_1(\mathsf{crs}, i, \mathsf{td})$: Sample

$$(\mathsf{pk}_i, \mathsf{st}_i) \leftarrow \mathsf{sRABFE.}\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_{\mathsf{sRABFE}}, i, \mathsf{att}_i, \mathsf{td}).$$

Output the public key pk_i and internal state st_i.

– $\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{td}, (\perp, \perp))$: Let $\mathsf{att}_i = i$, then sample

$$\mathsf{sk}_i \leftarrow \mathsf{sRABFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_i, \mathsf{att}_i, \mathsf{td}, (\perp, \perp)).$$

Output the secret key sk_i.

– $\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}})$: In the non-adaptive setting, we assume that in the challenge phase, the challenger knows all corrupted/malicious slots stored in $\mathcal{K}$, where $|\mathcal{K}| = Q$. For each $\mathsf{tag} \in [Q]$, if $\mathsf{wPRF.Eval}(\mathsf{k}, j) = \mathsf{tag}$ for any $j \in \mathcal{K}$, it sets $\mathcal{K}_{\mathsf{tag}} = \mathcal{K}_{\mathsf{tag}} \cup \{j\}$. Then, it computes

$$\forall \mathsf{tag} \in [Q],\ \mathsf{ct}_{\mathsf{tag}}^* \leftarrow \mathsf{sRABFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}, \mathsf{P}_{\mathsf{k},\mathsf{tag}}, \{(\mathsf{C}_j^*, \mathsf{C}_j^*(x^*))\}_{j\in\mathcal{K}_{\mathsf{tag}}})$$

Output the challenge ciphertext $\mathsf{ct}^* = (\mathsf{k}, \mathsf{ct}_1^*, \ldots, \mathsf{ct}_Q^*)$.

Game Sequence. We prove Theorem 20 via a sequence of games as follows:

– Game_0: Real Game.
– Game_1: Same as Game_2 except that it samples k as in $\widetilde{\mathsf{Setup}}$.
– Game_2: Same as Game_1 except that for all $\mathsf{tag}^* \in [Q]$, if $\mathsf{cnt}_{\mathsf{tag}^*} > \lambda$, it aborts and outputs $\perp$.
– Game_3: Same as Game_2 except that it completely replaces $(\mathsf{Setup}, \mathsf{Gen}, \mathsf{Enc})$ with $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}, \widetilde{\mathsf{Enc}})$.

We see that in the final game Game_3, the challenge ciphertext ct^* can be simulated only using $\{(\mathsf{C}_i^*, \mathsf{C}_i^*(x^*))\}_{i\in\mathcal{K}}$. This is exactly the simulation world.

Lemma 41. *We claim* $\mathsf{Game}_0 \equiv \mathsf{Game}_1$.

Proof. In Game_1, it samples the evaluation key k in the setup phase, where $\mathsf{k} \leftarrow \mathsf{wPRF.Gen}(1^\lambda)$ is only related to the security parameter 1^λ. Therefore, this difference does not change any distributions, so it naturally has $\mathsf{Game}_0 \equiv \mathsf{Game}_1$. □

Lemma 42. *By the pseudorandomness of* wPRF, *we claim* $\mathsf{Game}_1 \approx_c \mathsf{Game}_2$.

Proof. If the adversary $\mathcal{A}$ can distinguish these two games, we can build an algorithm $\mathcal{B}$ against the pseudorandomness of wPRF.

in the setup phase, $\mathcal{B}$ invokes the challenger of wPRF and hence acquires all pairs of input elements and output tags $\{(\mathsf{att}_j, \mathsf{tag}'_j)\}_{j\in[L]}$, where tag'_j is either generated from algorithm wPRF.Eval, or sampled by a random function. Here, $\mathcal{A}$ can only acquire $\{\mathsf{att}_j\}_{j\in[L]}$ from crs. Then, in the challenge phase, $\mathcal{B}$ knows the set $\mathcal{K}$. By Lemma 16 (where the variable X_u actually corresponds to cnt_u), we know that if $\{\mathsf{tag}_j^*\}_{j\in\mathcal{K}}$ are randomly sampled, it must have

$$\forall j \in \mathcal{K},\ \Pr[\mathsf{cnt}_{\mathsf{tag}_j^*} > \lambda] \le \mathsf{negl}(\lambda) \tag{17}$$

Therefore, in the case that the above tags are generated from wPRF.Eval, if $\mathcal{B}$ can find a corrupted/malicious slot j such that $\mathsf{cnt}_{\mathsf{tag}_j^*} > \lambda$ with non-negligible probability, it can also break the pseudorandomness of wPRF with non-negligible probability.

By contradiction, we say that the argument in Eq. (17) must hold even when the tags are generated from algorithm wPRF.Eval, which exactly corresponds to Game_2. Therefore, the lemma follows readily. □

Lemma 43. *By the policy-selective SIM security of* sRABFE, *we claim* $\mathsf{Game}_2 \approx_c \mathsf{Game}_3$.

Proof. If the adversary $\mathcal{A}$ can distinguish these two games, we can build an algorithm $\mathcal{B}$ against the policy-selective SIM security of sRABFE. Initially, $\mathcal{B}$ receives the input $(1^L, 1^Q)$ from $\mathcal{A}$. Then it proceeds as follows:

– *Setup phase:* $\mathcal{B}$ samples $\mathsf{k} \leftarrow \mathsf{wPRF.Gen}(1^\lambda)$ and invokes the challenger of sRABFE by revealing the input $(1^L, 1^\lambda, P, C)$ and the policies $(\mathsf{P}_{\mathsf{k},1}, \ldots, \mathsf{P}_{\mathsf{k},Q})$. Here, since the policy $\mathsf{P}_{\mathsf{k},d}$ is defined over the evaluation key k and the index $d \in [Q]$, $\mathcal{B}$ can just reveal a single policy $\mathsf{P}^* = (\mathsf{k}, 1^Q)$ to the challenger, instead of Q policies. We can see that for each $d \in [Q]$, P_d can be efficiently derived from P^*, which is independent of Q. In this way, the runtime of the setup phase would be independent of Q.
After revealing the above information, $\mathcal{B}$ receives $\mathsf{crs}_{\mathsf{sRABFE}}$ from the challenger and returns $\mathsf{crs} = \mathsf{crs}_{\mathsf{sRABFE}}$ to $\mathcal{A}$.
– *Pre-challenge phase:* Here, $\mathcal{A}$ can query two oracles as below:
  - $\mathsf{OGen}(i)$: $\mathcal{B}$ queries the oracle sRABFE.OGen on input (i, i) to acquire the public key pk. Return pk to $\mathcal{A}$.
  - $\mathsf{OCor}(i, \mathsf{pk})$: $\mathcal{B}$ queries the oracle sRABFE.OCor on input (i, pk) to acquire the secret key sk. Return sk to $\mathcal{A}$.
– *Challenge phase:* In this phase, $\mathcal{B}$ receives public keys $\{\mathsf{pk}_i^*\}_{i\in[L]}$, circuits $\{\mathsf{C}_i^*\}_{i\in[L]}$, and a challenge message x^*. Then, it submits $\{(\mathsf{pk}_i^*, i, \mathsf{C}_i^*)\}_{i\in[L]}$ to the challenger of sRABFE.
Next, for each $d \in [Q]$, it queries the oracle osRABFE.OEnc on input (P_d, x^*) to obtain the ciphertext ct_d^*. Finally, return the challenge ciphertext $\mathsf{ct}^* = (\mathsf{ct}_1^*, \ldots, \mathsf{ct}_Q^*)$ to $\mathcal{A}$.

In Game_1, we have known $\mathsf{cnt}_d \leq \lambda$ for each $d \in [Q]$, meaning that the number of the corrupted/malicious slots j such that $\mathsf{P}_{\mathsf{k},d}(j) = 1$ must be not greater than the collusion bound λ. Therefore, the above reduction makes sense: in the real world of sRABFE, $\mathcal{B}$ exactly simulates Game_2; otherwise, it simulates Game_3. This proves the lemma readily. □

D.5 From Weakly Optimal Slotted Reg-FE to Dynamic Slotted Reg-FE

Using the same method presented in Section 7, we can compile weakly optimal slotted Reg-FE in Appendix D.4 into a dynamic bounded collusion slotted Reg-FE for general circuits, achieving non-adaptive SIM security against $Q \geq 1$ corrupted/malicious users. Since the entire process is the same as Section 7, we omit more details.

E From Slotted Reg-FE to Reg-FE

In this section, we present the formal definition of dynamic/static bounded collusion Reg-FE. Then we present a generic construction based on dynamic bounded collusion slotted Reg-FE, which follows the "power-of-two" method [HLWW23]. Note that static bounded collusion Reg-FE can also be obtained via the same method. Besides, since static bounded collusion Reg-FE is a weaker variant of dynamic bounded collusion Reg-FE, we will mainly focus on the latter in this section.

E.1 Definition

A dynamic bounded-collusion Reg-FE for circuit family $C : X \to Z$ consists of six efficient algorithms as follows:

– $\mathsf{Setup}(1^\lambda, 1^L, C) \to \mathsf{crs}$. It takes as input the security parameter 1^λ, the maximum number of users 1^L and circuit family C, outputs a common reference string crs.
– $\mathsf{Gen}(\mathsf{crs}, \mathsf{aux}) \to (\mathsf{pk}, \mathsf{sk})$. It takes as input crs and state aux, outputs key pair $(\mathsf{pk}, \mathsf{sk})$.
– $\mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C}) \to (\mathsf{mpk}, \mathsf{aux}')$. It takes as input crs, aux, pk along with $\mathsf{C} \in C$, outputs master public key mpk and updated state aux′.
– $\mathsf{Upd}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}) \to \mathsf{hsk}$. It takes as input crs, aux and pk, outputs a helper key hsk.
– $\mathsf{Enc}(\mathsf{mpk}, x, 1^Q) \to \mathsf{ct}$. It takes as input mpk, $x \in X$ and collusion bound 1^Q, outputs a ciphertext ct.

– $\mathsf{Dec}(\mathsf{hsk}, \mathsf{sk}, \mathsf{ct}) \to z/\perp/\texttt{getupd}$. It takes as input hsk, sk and ct, outputs a value $z \in Z$, or an empty symbol $\perp$ to indicate a decryption failure, or a symbol getupd to indicate the need of an updated helper key.

The above scheme is required to satisfy:

– **Correctness.** For all stateful adversary $\mathcal{A}$, the following advantage function is negligible in λ:

$$\Pr[b = 1 \mid \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, C); b = 0; \mathcal{A}^{\mathsf{ORegNT}(\cdot,\cdot),\mathsf{ORegT}(\cdot),\mathsf{OEnc}(\cdot,\cdot,\cdot),\mathsf{ODec}(\cdot)}(\mathsf{crs})]$$

where the oracles work as follows with initial setting $\mathsf{aux} = \perp$, $\mathcal{E} = \emptyset$, $\mathcal{R} = \emptyset$ and $t = \perp$:

- $\mathsf{ORegNT}(\mathsf{pk}, \mathsf{C})$: run $(\mathsf{mpk}, \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C})$, update $\mathsf{aux} = \mathsf{aux}'$, append $(\mathsf{mpk}, \mathsf{aux})$ to $\mathcal{R}$ and return $(|\mathcal{R}|, \mathsf{mpk}, \mathsf{aux})$;
- $\mathsf{ORegT}(\mathsf{C}^*)$: run $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{aux})$ and $(\mathsf{mpk}, \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{C}^*)$, update $\mathsf{aux} = \mathsf{aux}'$, compute $\mathsf{hsk}^* \leftarrow \mathsf{Upd}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$, append $(\mathsf{mpk}, \mathsf{aux})$ to $\mathcal{R}$. Return $(t = |\mathcal{R}|, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk}^*, \mathsf{sk}^*, \mathsf{hsk}^*)$;
- $\mathsf{OEnc}(i, x, 1^Q)$: let $\mathcal{R}[i] = (\mathsf{mpk}, \cdot)$ and run $\mathsf{ct} \leftarrow \mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$, append (x, ct) to $\mathcal{E}$. Return $(|\mathcal{E}|, \mathsf{ct})$;
- $\mathsf{ODec}(j)$: let $\mathcal{E}[j] = (x_j, \mathsf{ct}_j)$, run $z_j \leftarrow \mathsf{Dec}(\mathsf{hsk}^*, \mathsf{sk}^*, \mathsf{ct}_j)$. If $z_j = \texttt{getupd}$, run $\mathsf{hsk}^* \leftarrow \mathsf{Upd}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}^*)$ and recompute $z_j \leftarrow \mathsf{Dec}(\mathsf{hsk}^*, \mathsf{sk}^*, \mathsf{ct}_j)$. Set $b = 1$ when $z_j \neq \mathsf{C}^*(x_j)$.

with the following restrictions:

- there is at least one query to ORegT;
- for query (i, x) to OEnc, it holds that $t \geq i, \mathcal{R}[i] \neq \perp$;
- for query (j) to ODec, it holds that $\mathcal{E}[j] \neq \perp$.

– **Compactness and Update Efficiency.** For all $\lambda, L \in \mathbb{N}$, and all C, it holds that

$$|\mathsf{mpk}| = \mathsf{poly}(\lambda, C, \log L) \text{ and } |\mathsf{hsk}| = \mathsf{poly}(\lambda, C, \log L).$$

Furthermore, the number of invocations of Upd in ODec is at most $O(\log |\mathcal{R}|)$ and each invocation costs $\mathsf{poly}(\log |\mathcal{R}|)$ time.

– **(Non-)Adaptive SIM Security.** For all efficient adversaries $\mathcal{A}$ invoked with security parameter $\lambda \in \mathbb{N}$, there exists a simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ such that it is computationally hard for $\mathcal{A}$ to distinguish the following two worlds:

- *Setup phase:* Initially, $\mathcal{A}$ specifies the input $(1^L, C)$. In the real world (left hand) or the ideal world (right hand), the challenger samples

$$\mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, 1^L, C) \text{ or } (\mathsf{crs}, \mathsf{td}) \leftarrow \widetilde{\mathsf{Setup}}(1^\lambda, 1^L, C).$$

Return crs to $\mathcal{A}$. Here, td is the trapdoor of crs.

- *Pre-challenge phase:* The challenger initially sets $\mathsf{mpk} = \perp$, $\mathsf{aux} = \perp$, $\mathcal{R} = \emptyset, C = \emptyset$ and $\mathcal{D}$ being a dictionary with $\mathcal{D}[\mathsf{pk}] = \emptyset$ for all possible pk. Then, $\mathcal{A}$ can make key generation/corruption queries to the following oracles:
  * $\mathsf{ORegCK}(\mathsf{pk}, \mathsf{C})$: The challenger runs

$$(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C})$$

Then, it updates $\mathsf{mpk} = \mathsf{mpk}'$, $\mathsf{aux} = \mathsf{aux}'$, $\mathcal{D}[\mathsf{pk}] = \mathcal{D}[\mathsf{pk}] \cup \{\mathsf{C}\}$, and appends pk to C. Return $(|\mathcal{R}|, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk})$.

  * $\mathsf{ORegHK}(\mathsf{C})$: In the real world (left hand) or the ideal world (right hand), the challenger runs

$$(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Gen}(\mathsf{crs}, \mathsf{aux}) \text{ or } (\mathsf{pk}, \mathsf{st}) \leftarrow \widetilde{\mathsf{Gen}}_1(\mathsf{crs}, \mathsf{aux}, \mathsf{td}).$$

Run $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C})$, update $\mathsf{mpk} = \mathsf{mpk}', \mathsf{aux} = \mathsf{aux}', \mathcal{D}[\mathsf{pk}] = \mathcal{D}[\mathsf{pk}] \cup \{\mathsf{C}\}$, and append $(\mathsf{pk}, \mathsf{sk})$ or $(\mathsf{pk}, \mathsf{st})$ to $\mathcal{R}$. Return $(|\mathcal{R}|, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk})$.

* $\mathsf{OCorHK}(i)$: let $\mathcal{R}[i] = (\mathsf{pk}, \mathsf{st})$ and $\mathsf{C} = \mathcal{D}[\mathsf{pk}]$, append pk to C. In the real world (left hand) or the ideal world (right hand), the challenger returns the secret key sk to $\mathcal{A}$, where

$$\mathsf{sk} = \mathsf{st} \;\text{ or }\; \mathsf{sk} \leftarrow \widetilde{\mathsf{Gen}}_2(\mathsf{st}, \mathsf{td}, (\perp, \perp)).$$

- *Challenge phase:* In this phase, $\mathcal{A}$ specifies a collusion bound 1^Q and a challenge message x^*. For the i-th registered user, where $i \in [L]$, it must be registered into the system by querying one of the oracles ORegCK (i.e., malicious user), ORegHK (i.e., honest user), and OCorHK (i.e., corrupted user). Let $C[i] = \mathsf{pk}_i$ and $\mathcal{D}[\mathsf{pk}_i] = \mathsf{C}_i$ for all $i \in |C|$, in the real world (left hand) or the ideal world (right hand), it replies $\mathcal{A}$ with a challenge ciphertext ct^* by either sampling

$$\mathsf{ct}^* \leftarrow \mathsf{Enc}(\mathsf{mpk}, x^*) \;\text{ or }\; \mathsf{ct}^* \leftarrow \widetilde{\mathsf{Enc}}(\mathsf{mpk}, \{(\mathsf{C}_i, \mathsf{C}_i(x^*))\}_{i \in |C|}).$$

- *Post-challenge phase:* $\mathcal{A}$ continues to make corruption queries:
  * $\mathsf{OCorHK}(i)$: If $|\mathcal{K}| = Q$, $\mathcal{A}$ is not allowed to corrupt any registered key in this phase. Otherwise, let $\mathcal{R}[i] = (\mathsf{pk}, \mathsf{st})$ and $\mathsf{C} = \mathcal{D}[\mathsf{pk}]$, append pk to C. in the real world (left hand) or the ideal world (right hand), it computes

$$\mathsf{sk} = \mathsf{st} \;\text{ or }\; \mathsf{sk} \leftarrow \widetilde{\mathsf{Gen}}_2(\mathsf{st}, \mathsf{td}, (\mathsf{C}, \mathsf{C}(x^*))).$$

    Return the secret key sk to $\mathcal{A}$.
- *Output phase:* $\mathcal{A}$ outputs a bit to indicate the real/ideal world.

The above definition aims at adaptive security. The only difference between non-adaptive security and adaptive security is that, in the non-adaptive case, $\mathcal{A}$ is not allowed to make any corruption queries in the post-challenge phase.

Static Collusion. We note that static bounded collusion Reg-FE can be defined in a similar way to dynamic bounded collusion Reg-FE, except that it must specify the collusion bound 1^Q in Setup and allow all system overheads to grow with Q.

E.2 Construction

We exploit the "power-of-two" approach from [HLWW23] to generically convert dynamic sReg-FE into dynamic Reg-FE. Assume a full-fledged Reg-FE mostly supports $L = 2^\ell$ users, this approach needs $\ell + 1$ copies of sReg-FE with $1, 2, 4, \ldots, 2^\ell$ slots. And the public state $\mathsf{aux} = (\mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$ consists of the following terms:

- $\mathcal{D}_1[k, i] = (\mathsf{pk}, y)$: where $k \in [0, \ell]$ and $i \in [2^k]$. This dictionary assigns a user's (pk, y) to the slot i of the 2^k-slotted Reg-FE scheme.
- $\mathcal{D}_2[k, n] = \mathsf{hsk}$: where $k \in [0, \ell]$ and $n \in [L]$. This dictionary assigns a hsk of sReg-FE to the 2^k-slotted Reg-FE scheme and the user index n.
- $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$ denotes the current master public key. Where $\{\mathsf{mpk}_k\}_{k \in [0,\ell]}$ denote the master public keys of $\ell + 1$ copies of sReg-FE, and ctr denotes the number of currently registered users. When no registered user, we initially set $\mathsf{mpk} = (0, \perp, \ldots, \perp)$.

When no registered user, we initially set $\mathsf{aux} = (\emptyset, \emptyset, \perp)$. Assume a dynamic sReg-FE for circuits $\mathsf{sRFE} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Ver}, \mathsf{Agg}, \mathsf{Enc}, \mathsf{Dec})$, a full-fledged dynamic Reg-FE for circuits can be constructed as follows:

- $\mathsf{Setup}(1^\lambda, 1^L, C)$: Let $\ell = \log L$. It runs $\mathsf{crs}_k \leftarrow \mathsf{sRFE.Setup}(1^\lambda, 1^{2^k}, C)$ for all $k \in [0, \ell]$. Output

$$\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell).$$

– $\mathsf{Gen}(\mathsf{crs}, \mathsf{aux})$: Fetch $\mathsf{crs} = \{\mathsf{crs}_k\}_{k\in[0,\ell]}$ and $\mathsf{aux} = (\mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$, where $\mathsf{mpk} = \big(\mathsf{ctr}, \{\mathsf{mpk}_k\}_{k\in[0,\ell]}\big)$. For all $k \in [0, \ell]$, compute

$$i_k = (\mathsf{ctr} \bmod 2^k) + 1$$

and run $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{sRFE.Gen}(\mathsf{crs}_k, i_k)$. Set $\mathsf{ctr}' = \mathsf{ctr}$ and output

$$\mathsf{pk} = (\mathsf{ctr}', \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell) \quad \text{and} \quad \mathsf{sk} = (\mathsf{ctr}', \mathsf{sk}_0, \ldots, \mathsf{sk}_\ell).$$

– $\mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C})$: Fetch $\mathsf{crs} = \{\mathsf{crs}_k\}_{k\in[0,\ell]}$, $\mathsf{aux} = (\mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$, and $\mathsf{pk} = \big(\mathsf{ctr}', \{\mathsf{pk}_k\}_{k\in[0.\ell]}\big)$, where $\mathsf{mpk} = (\mathsf{ctr}, \{\mathsf{mpk}_k\}_{k\in[0,\ell]})$. For all $k \in [0, \ell]$, do the following operates:

- Compute $i_k = (\mathsf{ctr} \bmod 2^k) + 1$;
- Check if $\mathsf{sRFE.Ver}(\mathsf{crs}_k, i_k, \mathsf{pk}_k) = 1$ and $\mathsf{ctr}' = \mathsf{ctr}$. If the check passes, set $\mathsf{ctr} = \mathsf{ctr} + 1$, if the check fails,the algorithm halts and output $(\mathsf{mpk}, \mathsf{aux})$;
- Update $\mathcal{D}_1[k, i_k] = (\mathsf{pk}, \mathsf{C})$;
- If $i_k = 2^k$, compute

$$(\mathsf{mpk}'_k, \{\mathsf{hsk}_{k,j}\}_{j\in[2^k]}) \leftarrow \mathsf{sRFE.Agg}(\mathsf{crs}_k, \{\mathcal{D}_1[k, i]\}_{i\in[2^k]}).$$

Update $\mathsf{mpk}_k = \mathsf{mpk}'_k$, and for all $j \in [2^k]$, update $\mathcal{D}_2[k, \mathsf{ctr} - 2^k + j] = \mathsf{hsk}_{k,j}$.

Update the master public key $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$ and public state $\mathsf{aux} = (\mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$, output $(\mathsf{mpk}, \mathsf{aux})$.

– $\mathsf{Upd}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk})$: Fetch $\mathsf{crs} = \{\mathsf{crs}_k\}_{k\in[0,\ell]}$, $\mathsf{aux} = (\mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$, and $\mathsf{pk} = \big(\mathsf{ctr}', \{\mathsf{pk}_k\}_{k\in[0,\ell]}\big)$, where $\mathsf{mpk} = (\mathsf{ctr}, \{\mathsf{mpk}_k\}_{k\in[0,\ell]})$. Output

$$\mathsf{hsk} = \begin{cases} (\underbrace{\mathcal{D}_2[0, \mathsf{ctr}' + 1]}_{\mathsf{hsk}_0}, \ldots, \underbrace{\mathcal{D}_2[\ell, \mathsf{ctr}' + 1]}_{\mathsf{hsk}_\ell}) & \text{if } \mathsf{ctr}' < \mathsf{ctr} \\ \bot & \text{otherwise} \end{cases} \tag{18}$$

– $\mathsf{Enc}(\mathsf{mpk}, x, 1^Q)$: Fetch $\mathsf{mpk} = (\mathsf{ctr}, \{\mathsf{mpk}_k\}_{k\in[0,\ell]})$. For all $k \in [0, \ell]$, compute:

$$\mathsf{ct}_k = \begin{cases} \mathsf{sRFE.Enc}(\mathsf{mpk}_k, x, 1^{\min(Q, 2^k)}) & \text{if } \mathsf{mpk}_k \neq \bot \\ \bot & \text{otherwise} \end{cases} \tag{19}$$

Output $\mathsf{ct} = (\mathsf{ctr}, \mathsf{ct}_0, \ldots, \mathsf{ct}_\ell)$.

– $\mathsf{Dec}(\mathsf{hsk}, \mathsf{sk}, \mathsf{ct})$: Fetch $\mathsf{sk} = \big(\mathsf{ctr}', \{\mathsf{sk}_k\}_{k\in[0,\ell]}\big)$, $\mathsf{hsk} = \{\mathsf{hsk}_k\}_{k\in[0,\ell]}$ and $\mathsf{ct} = (\mathsf{ctr}, \{\mathsf{ct}_k\}_{k\in[0,\ell]})$. Proceed as follows:

- If $\mathsf{ctr}' \geq \mathsf{ctr}$: output $\bot$;
- Otherwise, compute $\mathsf{ctr} = (a_\ell, \ldots, a_0)_2$ and $\mathsf{ctr}' = (b_\ell, \ldots, b_0)_2$. We denote k_d as the maximum $k \in [0, \ell]$ such that $a_k \neq b_k$. if $\mathsf{mpk}_{k_d} \neq \bot$ and $\mathsf{hsk}_{k_d} = \bot$, output $\mathtt{getupd}$;
- Otherwise, output $\mathsf{sRFE.Dec}(\mathsf{hsk}_{k_d}, \mathsf{sk}_{k_d}, \mathsf{ct})$.

Analysis. We would demonstrate the correctness, compactness, efficiency and security of the above construction via a series of theorems.

Theorem 21 (Correctness). *Assuming* sRFE *is complete and perfectly correct, then our construction is perfectly correct.*

Proof. Similar to [HLWW23,FFM+23,DP23], we omit details here. □

Theorem 22 (Compactness). *Assuming* sRFE *is compact, then our construction is compact.*

Proof. Observe that $|\mathsf{mpk}| = |\mathsf{ctr}| + \sum_{i\in[0,\ell]} |\mathsf{mpk}_i|$ and $|\mathsf{hsk}| = \sum_{i\in[0,\ell]} |\mathsf{hsk}_i|$, where ctr is a ℓ-bit number. According to the compactness of sRFE, we have $|\mathsf{mpk}_i| = \mathsf{poly}(\lambda, C, \log L)$ and $|\mathsf{hsk}_i| = \mathsf{poly}(\lambda, C, \log L)$ for all $i \in [0, \ell]$. Then it holds that $|\mathsf{mpk}| = \mathsf{poly}(\lambda, C, \log i)$ and $|\mathsf{hsk}| = \mathsf{poly}(\lambda, C, \log |\mathcal{R}|)$. □

Theorem 23 (Update Efficiency). *Assuming* sRFE *is compact, then our construction meets update efficiency.*

Proof. Observe that the number of invocations of Upd is at most $\ell + 1 = O(\log |\mathcal{R}|)$ and Upd is only invoked when one of $\{\mathsf{hsk}_k\}_{k\in[0,\ell]}$ is $\perp$. Thus, the number of invocations of Upd in ODec is at most $O(\log |\mathcal{R}|)$.

On the other hand, $|\mathsf{hsk}_k| = \mathsf{poly}(\lambda, C, \log |\mathcal{R}|)$ for $k \in [0, \ell]$ according to the compactness of sRFE. Since aux maintains a dictionary $\mathcal{D}_2$ mapping each index slot index k to its set of helper decryption keys, each invocation of Upd runs in $\mathsf{poly}(\log |\mathcal{R}|)$ time (in RAM model). □

E.3 Proof

Theorem 24. *Assuming* sRFE *satisfies (non)-adaptive SIM security, then our construction satisfies (non)-adaptive SIM security.*

Proof. The proof of non-adaptive SIM security is a simpler case of the proof of adaptive security, so we focus on adaptive SIM security in the following. Analogous to [HLWW23], suppose that there exists an adversary $\mathcal{A}$ who breaks the SIM security of sRFE with non-negligible advantage, we start by defining a sequence of hybrid experiments, each parameterized by an index $v \in [0, \ell]$:

- Game_v : Same as real experiment, except that it employs the simulator $(\widetilde{\mathsf{Setup}}, \widetilde{\mathsf{Gen}}_1, \widetilde{\mathsf{Gen}}_2, \widetilde{\mathsf{Enc}})$ for the first $k \le v$ underlying sRFE instances. Here, $\mathcal{K}$ is the set of all corrupted/malicious slot information of Reg-FE (in the pre-challenge phase). Concretely, this game proceeds as follows:

  Setup. In this phase, $\mathcal{A}$ proceeds as follows:
  - $\mathcal{A}$ chooses $(1^L, C)$ and sends them to the challenger.
  - The challenger computes $(\mathsf{crs}_k, \mathsf{td}_k) \leftarrow \mathsf{sRFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, C)$ for all $k \le v$ and $\mathsf{crs}_k \leftarrow \mathsf{sRFE}.\mathsf{Setup}(1^\lambda, 1^L, C)$ for all $k > v$.
  - Then it initializes a counter $t = 0$, master public key $\mathsf{mpk} = \perp$ and auxiliary input $\mathsf{aux} = (\mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$ where $\mathcal{D}_1 = \emptyset, \mathcal{D}_2 = \emptyset$ and $\mathsf{mpk} = (0, \perp, \ldots, \perp)$; Set $\mathcal{R} = \emptyset$, $C = \emptyset$, $\mathcal{S} = \emptyset$ and a dictionary $\mathcal{D}$ with $\mathcal{D}[\mathsf{pk}] = \emptyset$ for all possible pk.
  - Finally, it sends $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$ to $\mathcal{A}$.

  Pre-Challenge. In this phase, $\mathcal{A}$ can make queries as follows:
  - $\mathsf{ORegCK}(\mathsf{pk}, \mathsf{C})$: $\mathcal{A}$ specifies a public key pk and a cricuit C. The challenger runs

    $$(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C}),$$

    updates $\mathsf{mpk} = \mathsf{mpk}'$, $\mathsf{aux} = \mathsf{aux}'$, $\mathcal{D}[\mathsf{pk}] = \mathcal{D}[\mathsf{pk}] \cup \{\mathsf{C}\}$, appends pk to C, and returns $(|\mathcal{R}|, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk})$.
  - $\mathsf{ORegHK}(\mathsf{C})$: $\mathcal{A}$ specifies a circuit C. The challenger sets $t \leftarrow t + 1$ and samples $\mathsf{pk} = (\mathsf{ctr}', \{\mathsf{pk}_k\}_{k\in[0,\ell]})$ and $\mathsf{sk} = (\mathsf{ctr}', \{\mathsf{sk}_k\}_{k\in[0,\ell]})$, where

    $$\forall\, k \le v,\ (\mathsf{pk}_k, \mathsf{st}_k) \leftarrow \mathsf{sRFE}.\widetilde{\mathsf{Gen}}_1(\mathsf{crs}_k, i_k, \mathsf{td}_k),$$
    $$\forall\, k > v,\ (\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{sRFE}.\mathsf{Gen}(\mathsf{crs}_k, i_k).$$

    Then run $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C})$. It updates $\mathsf{mpk} = \mathsf{mpk}'$, $\mathsf{aux} = \mathsf{aux}'$, $\mathcal{D}[\mathsf{pk}] = \mathcal{D}[\mathsf{pk}] \cup \{\mathsf{C}\}$, append $(\mathsf{pk}, \mathsf{sk})$ or $(\mathsf{pk}, \mathsf{st})$ to $\mathcal{R}$ and return $(t, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk})$.

- $\mathsf{OCorHK}(i)$: $\mathcal{A}$ specifies an index $i \in [t]$. let $\mathcal{R}[i] = (\mathsf{pk}, \mathsf{st})$ and $\mathsf{C} = \mathcal{D}[\mathsf{pk}]$, append pk to C and return $\mathsf{sk} = \{\mathsf{sk}_k\}_{k\in[0,\ell]}$ to $\mathcal{A}$, where

$$\forall\, k \le v,\ \mathsf{sk}_k = \mathsf{sRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_k, \mathsf{td}_k, (\bot, \bot)),$$
$$\forall\, k > v,\ \mathsf{sk}_k = \mathsf{st}_k.$$

Challenge. In the challenge phase, $\mathcal{A}$ submits the challenge message x^* and collusion bound 1^Q. The challenger proceeds as follows:

- Let $\mathsf{aux} = (\mathsf{ctr}, \mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$ where $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$.
- For each $k \in [0, \ell]$, if $\mathsf{mpk}_k = \bot$, then set $\mathsf{ctr}_k = \bot$; otherwise, if $k \le v$, compute $\mathsf{ct}_k^* \leftarrow \mathsf{sRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, Q)$, and if $k > v$, compute $\mathsf{ct}_k^* \leftarrow \mathsf{sRFE.Enc}(\mathsf{mpk}_k, x^*, 1^Q)$. Note that the set Q records the pairs of corrupted/malicious circuits along with evaluations on x^*.
- The challenger replies to $\mathcal{A}$ with $\mathsf{ct}^* = (\mathsf{ctr}, \mathsf{ct}_0^*, \ldots, \mathsf{ct}_\ell^*)$.

Post-Challenge. In this phase, $\mathcal{A}$ can make queries as follows:

- $\mathsf{OCorHK}(i)$: $\mathcal{A}$ specifies an index $i \in [t]$. let $\mathcal{R}[i] = (\mathsf{pk}, \mathsf{st})$ and $\mathsf{C} = \mathcal{D}[\mathsf{pk}]$, append pk to C and return $\mathsf{sk} = \{\mathsf{sk}_k\}_{k\in[0,\ell]}$ to $\mathcal{A}$, where

$$\forall\, k \le v,\ \mathsf{sk}_k = \mathsf{sRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_k, \mathsf{td}_k, (\mathsf{C}, \mathsf{C}(x^*))),$$
$$\forall\, k > v,\ \mathsf{sk}_k = \mathsf{st}_k.$$

Lemma 44. *By the adaptive SIM security of* sRFE*, we claim* $\mathsf{Game}_{v-1} \approx_c \mathsf{Game}_v$.

Proof. The algorithm $\mathcal{B}$ will simulated Game_{v-1} or Game_v as follows:

Setup. In this phase, $\mathcal{B}$ proceeds as follows:

- When $\mathcal{B}$ receives L from $\mathcal{A}$, it invokes the challenger of v-th underlying sRFE construction, then obtains crs_v;
- $\mathcal{B}$ computes $(\mathsf{crs}_k, \mathsf{td}_k) \leftarrow \mathsf{sRFE}.\widetilde{\mathsf{Setup}}(1^\lambda, 1^L, C)$ for all $k < v$ and $\mathsf{crs}_k \leftarrow \mathsf{sRFE.Setup}(1^\lambda, 1^L, C)$ for all $k > v$;
- Then it initializes a counter $t = 0$, master public key $\mathsf{mpk} = \bot$ and auxiliary input $\mathsf{aux} = (\mathcal{D}_1, \mathcal{D}_2, \mathsf{mpk})$ where $\mathcal{D}_1 = \emptyset, \mathcal{D}_2 = \emptyset$ and $\mathsf{mpk} = (0, \bot, \ldots, \bot)$; Set a dictionary $\mathcal{D}$ with $\mathcal{D}[\mathsf{pk}] = \emptyset$ for all possible pk. In addition, $\mathcal{B}$ maintains a dictionary $\mathcal{D}_s$ to track the secret keys associated with each ORegHK query, and initializes two ordered list $\mathcal{S}_1 = \bot, \mathcal{S}_2 = (\bot, \ldots, \bot)$ to track the public keys and circuits and aggregate them as a part of mpk_v.
- Finally, it sends $\mathsf{crs} = (\mathsf{crs}_0, \ldots, \mathsf{crs}_\ell)$ to $\mathcal{A}$.

Pre-Challenge. In the query phase, $\mathcal{B}$ simulates queries made by $\mathcal{A}$ as follows:

- $\mathsf{ORegCK}(\mathsf{pk}, \mathsf{C})$: When $\mathcal{A}$ specifies a public key pk and a cricuit C, let ctr be the current counter in aux. For each $k \in [0, \ell]$, it computes $i_k = (\mathsf{ctr} \bmod 2^k) + 1$. $\mathcal{B}$ runs $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C})$, update $\mathsf{mpk} = \mathsf{mpk}', \mathsf{aux} = \mathsf{aux}', \mathcal{D}[\mathsf{pk}] = \mathcal{D}[\mathsf{pk}] \cup \{\mathsf{C}\}$, append pk to C and return $(|\mathcal{R}|, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk})$;
- $\mathsf{ORegHK}(\mathsf{C})$: When $\mathcal{A}$ specifies a circuit C, let ctr be the current counter in aux. For each $k \in [0, \ell]$, compute $i_k = (\mathsf{ctr} \bmod 2^k) + 1$. Then,
  - $\mathcal{B}$ generates $(\mathsf{pk}_k, \mathsf{st}_k) \leftarrow \mathsf{sRFE}.\widetilde{\mathsf{Gen}}(\mathsf{crs}_k, i_k, \mathsf{td}_k)$ for $k < v$.
  - $\mathcal{B}$ makes a query i_v to $\mathsf{OGen}(\cdot)$, then obtains a public key pk_v.
  - $\mathcal{B}$ generates $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{sRFE.Gen}(\mathsf{crs}_k, i_k)$ for $k > v$.

  It sets $\mathsf{pk} = (\mathsf{ctr}, \mathsf{pk}_0, \ldots, \mathsf{pk}_\ell)$. The challenger sets $t \leftarrow t + 1$ and $\mathcal{D}_s[t] = (\mathsf{ctr}, \{\mathsf{st}_k\}_{k\in[0,v-1]}, \{\mathsf{sk}_k\}_{k\in[v+1,\ell]})$. Next, $\mathcal{B}$ runs $(\mathsf{mpk}', \mathsf{aux}') \leftarrow \mathsf{Reg}(\mathsf{crs}, \mathsf{aux}, \mathsf{pk}, \mathsf{C})$ and updates $\mathsf{mpk} = \mathsf{mpk}', \mathsf{aux} = \mathsf{aux}', \mathcal{D}[\mathsf{pk}] = \mathcal{D}[\mathsf{pk}] \cup \{\mathsf{C}\}$. Append $(\mathsf{pk}, \mathcal{D}_s[t])$ to $\mathcal{R}$ and return $(t, \mathsf{mpk}, \mathsf{aux}, \mathsf{pk})$. In addition, $\mathcal{B}$ updates $\mathcal{S}_2[i_v] = (t, \mathsf{C})$. Moreover, if $i_v = 2^v$, $\mathcal{B}$ sets $\mathcal{S}_1 = \mathcal{S}_2$.

– $\mathsf{OCorHK}(i)$: When $\mathcal{A}$ specifies an index $i \in [t]$, $\mathcal{B}$ looks up $\mathcal{D}_s[i] = (\mathsf{ctr}, \{\mathsf{st}_k\}_{k\in[0,\nu-1]}, \{\mathsf{sk}_k\}_{k\in[\nu+1,\ell]})$. Then, it computes

$$\forall\, k \le \nu,\ \mathsf{sk}_k = \mathsf{sRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_k, \mathsf{td}_k, (\bot, \bot)).$$

Let $\mathsf{sk} = (\mathsf{ctr}, \{\mathsf{st}_k\}_{k\in[0,\ell]\setminus\{\nu\}})$ and $\mathcal{R}[i] = (\mathsf{pk}, \mathsf{sk})$, then it makes a corruption query (i, pk_ν) to $\mathsf{OCor}(\cdot,\cdot)$ and obtains sk_ν. Next, append pk to C, set $Q = Q \cup \{(\mathsf{C}, \mathsf{C}(x^*))\}$ and return $\mathsf{sk} = (\mathsf{ctr}, \mathsf{sk}_0, \ldots, \mathsf{sk}_\ell)$.

Challenge. In the challenge phase, $\mathcal{A}$ submits the challenge message x^* and collusion bound 1^Q. Let $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_0, \ldots, \mathsf{mpk}_\ell)$ be the current master public key. For each $k \in [0, \ell]$, $\mathcal{B}$ proceeds as follows:

– If $\mathsf{mpk}_k = \bot$, then set $\mathsf{ct}_k^* = \bot$;
– If $\mathsf{mpk}_k \neq \bot$ and $k < \nu$, then compute $\mathsf{ct}_k^* \leftarrow \mathsf{sRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_k, Q)$;
– If $\mathsf{mpk}_k \neq \bot$ and $k = \nu$, $\mathcal{B}$ makes a challenge query to obtain ct_ν^*;
– If $\mathsf{mpk}_k \neq \bot$ and $k > \nu$, then compute $\mathsf{ct}_k^* \leftarrow \mathsf{sRFE}.\mathsf{Enc}(\mathsf{mpk}_k, x^*, 1^Q)$;
– The challenger replies to $\mathcal{A}$ with $\mathsf{ct}^* = (\mathsf{ctr}, \mathsf{ct}_0^*, \ldots, \mathsf{ct}_\ell^*)$.

Post-Challenge. In this phase, $\mathcal{A}$ can make queries as follows:

– $\mathsf{OCorHK}(i)$: When $\mathcal{A}$ specifies an index $i \in [t]$, $\mathcal{B}$ looks up $\mathcal{D}_s[i] = (\mathsf{ctr}, \{\mathsf{st}_k\}_{k\in[0,\nu-1]}, \{\mathsf{sk}_k\}_{k\in[\nu+1,\ell]})$ and $\mathsf{C} = \mathcal{D}[\mathsf{pk}]$. Then, it computes

$$\forall\, k \le \nu,\ \mathsf{sk}_k = \mathsf{sRFE}.\widetilde{\mathsf{Gen}}_2(\mathsf{st}_k, \mathsf{td}_k, (\mathsf{C}, \mathsf{C}(x^*))).$$

Let $\mathsf{sk} = (\mathsf{ctr}, \{\mathsf{st}_k\}_{k\in[0,\ell]\setminus\{\nu\}})$ and $\mathcal{R}[i] = (\mathsf{pk}, \mathsf{sk})$, then it makes a corruption query (i, pk_ν) to $\mathsf{OCor}(\cdot,\cdot)$ and obtains sk_ν. Next, append pk to C return $\mathsf{sk} = (\mathsf{ctr}, \mathsf{sk}_0, \ldots, \mathsf{sk}_\ell)$.

In above experiment, $\mathcal{B}$ perfectly simulates an execution of dynamic bounded Reg-FE against $\mathcal{A}$. Note that if $2^\nu < Q$, it means that all secret keys in z^ν-slotted Reg-FE can be corrupted, which will not affect the simulation of ct_ν^*, since other corrupt secret keys are not registered in z^ν-slotted Reg-FE, which means that adversary still owns at most 2^ν valid keys and then the simulation security of z^ν-slotted Reg-FE still hold. Thus, in the case where $\mathsf{ct}_\nu^* \leftarrow \mathsf{sRFE}.\mathsf{Enc}(\mathsf{mpk}_\nu, x^*, 1^Q)$, $\mathcal{B}$ simulates $\mathsf{Game}_{\nu-1}$, while in the case where $\mathsf{ct}_\nu^* \leftarrow \mathsf{sRFE}.\widetilde{\mathsf{Enc}}(\mathsf{mpk}_\nu, Q)$, $\mathcal{B}$ simulates Game_ν. □

F Static Bounded Collusion MA-ABE

In multi-authority ABE [Cha07] (MA-ABE), it decentralizes the trust of a single central authority to a number of independent (sub-)authorities. Here, each authority independently creates a pair of master keys for encryption and decryption key delegation. To generate a user decryption key, the user holding a global unique identifier GID needs to collect decryption keys from multiple authorities. In this case, each authority only handles partial secrets and holds different attributes. The Q-bound IND security states: encryption of x_0 is indistinguishable from encryption x_1 for an adversary who simultaneously obtains Q secret keys from users and controls a proportion of the authorities holding unauthorized attributes. In the non-adaptive setting, the adversary will not corrupt users or authorities after seeing the challenge ciphertext.

As a byproduct, our derandomization technique directly improves upon existing MA-ABE. See Table 2 for a comprehensive comparison. Since the emergence of the multi-authority notion, building MA-ABE in the standard model has been an interesting subject for a long time. However, prior works in the standard model require specific assumptions [LW11,WFL19,WWW22]. Now the state-of-the-art MA-ABE proposed by Garg et al. [GGL24] just relies on minimal assumptions such as public-key encryption (PKE), but it is proved in the random oracle model. Garg et al. also remarked that their construction would only achieve weakly selective security when removing. In this

Table 2. Comparison of MA-ABE schemes

	Policy	Security	Collusion	Standard	Assumption
[LW11]	NC^1	wSEL-IND	Static	✓	SD
[WWW22]	DNF	wSEL-IND	Full	✓	evLWE
[GGL24]	mP/Poly	AD-IND	Static	×	PKE
	mP/Poly	AD-IND	Dynamic	×	IBE
§F	mP/Poly	NA-IND	Static	✓	PKE

* In the column of "Security", "wSEL" stands for weakly selective security, which can also be called static corruption security in prior work. The column "Standard" indicates whether it is proved in the standard model. In the column of "Assumption", "SD" stands for subgroup decision assumptions over composite-order bilinear groups.

work, we can apply our technique to derive a new bounded-collusion MA-ABE for monotone circuits solely based on PKE, achieving non-adaptive security in the standard model.

We give a sketch of our new construction of static bounded collusion MA-ABE for monotone circuits (i.e., mP/Poly), which relies on the minimal assumption of PKE and is non-adaptive secure in the standard model. We start with Garg et al.'s bounded MA-FE scheme, which just relies on PKE but necessitates the random oracle. Informally, it works as follows: at the beginning, a global authority initializes a common reference string crs and then publishes it to n sub-authorities. Then the authority responsible for any attribute u creates its own master public/secret key pair $(\mathsf{mpk}_u, \mathsf{msk}_u)$ using crs. Here it publishes mpk_u and keeps msk_u in secret. A system user, holding a unique global identity GID and an attribute set S, can launch a secret key request (GID, u) to the corresponding authority if $u \in S$. The authority turns to use msk_i to delegate a (partial) secret key $\mathsf{sk}_{\mathsf{GID},u}$ as return. After collecting all $\{\mathsf{sk}_{\mathsf{GID},u}\}_{u\in S}$, the user can decrypt the ciphertext encrypted by a message x, a (monotone) circuit C and master public keys $\{\mathsf{mpk}_i\}_{i\in[n]}$. Here, the IND security requires that ct_0 associated with message x_0 is computationally indistinguishable from ct_1 associated with message x_1, if (i) there are at most Q corrupted secret keys associated with different GID, and (ii) there is no combination of secret keys and corrupted authorities that are authorized to decrypt the challenge ciphertext.

Technically, to construct bounded collusion MA-ABE, Garg et al. [GGL24] follow a similar strategy as in Section 2.1: first construct a 1-bound MA-ABE from PKE and then upgrade it into Q-bound security by adding redundancy. Thus, they have to answer a key question: *how to let all authorities individually generate the same sets* $(\Gamma_{\mathsf{GID}}, \Delta_{\mathsf{GID}})$, *subject to small pairwise intersections and cover freeness, for a user with unique global identity* GID. Their solution is to employ a random oracle to generate all these sets according to GID.

Fortunately, our derandomization technique presented in Section 2.2 provides an effective solution, immediately leading to a more standard scheme. Roughly speaking, we can use error-correcting codes to publicly derandomize the generation of these sets. In more detail, each authority can take GID as hash input and derive the same $(\Gamma_{\mathsf{GID}}, \Delta_{\mathsf{GID}})$ from hash outputs, and hence distribute proper partial secret keys to users, without the need of a random oracle. As a result, we obtain the first bounded collusion MA-ABE for mP/Poly solely based on PKE, achieving non-adaptive IND security in the standard model. Since the modification is minor, we omit more details.