Registered (Inner-Product) Functional Encryption

Danilo Francati[1], Daniele Friolo[2], Monosij Maitra[3,4**], Giulio Malavolta[3], Ahmadreza Rahimi[3], and Daniele Venturi[2]

[1] Aarhus University
dfrancati@cs.au.dk
[2] Sapienza University of Rome
friolo@di.uniroma1.it
venturi@di.uniroma1.it
[3] Ruhr-Universität Bochum
monosij.maitra@rub.de
[4] Max-Planck Institute for Security and Privacy
giulio.malavolta@hotmail.it
ahmadrezar@pm.me

Abstract. Registered encryption (Garg *et al.*, TCC'18) is an emerging paradigm that tackles the key-escrow problem associated with identity-based encryption by replacing the private-key generator with a much weaker entity known as the key curator. The key curator holds no secret information, and is responsible to: (i) update the master public key whenever a new user registers its own public key to the system; (ii) provide helper decryption keys to the users already in the system, in order to make them still able to decrypt. For practical purposes, tasks (i) and (ii) need to be efficient, in the sense that the size of the public parameters, of the master public key, and of the helper decryption keys, as well as the running time for key generation and user registration, and the number of updates, must be small. In this paper, we generalize the notion of registered encryption to the setting of functional encryption (FE). Our contributions are twofold: On the one hand, we show that registered FE exists assuming indistinguishability obfuscation and somewhere statistically binding hash functions. On the other hand, we show an efficient construction of registered FE for the special case of inner-product predicates, over asymmetric bilinear groups of prime order, with provable security in the generic group model.

Keywords: Registered encryption, functional encryption, inner-product predicate encryption.

** Research conducted partially at Technische Universität Darmstadt, Germany.

Table of Contents

1 Introduction

Functional encryption (FE) [SW05, O'N10, BSW11] enriches standard public-key encryption with fine-grained access control over encrypted data. This added feature is made possible by using a so-called master secret key msk that can be used (by an authority) in order to generate decryption keys sk_f associated with functions f, in such a way that decrypting any ciphertext c, corresponding to a plaintext m, reveals $f(m)$ and nothing more. Recent years have seen a flourish of works exploring FE constructions in various settings and from different assumptions [GGH+13, SW14, GGHZ16, GKP+13, BGG+14, ABSV15, GLSW15, Lin16, LV16, AS17, Lin17, AJL+19, Agr19, JLMS19, JLS21, BDGM20, WW21, GP21, BDGM22, GJLS21, JLS22], and its applications to building powerful cryptographic tools such as reusable garbled circuits [GKP+13], adaptive garbling [HJO+16], multi-party non-interactive key exchange [GPSZ17], universal samplers [GPSZ17], verifiable random functions [GHKW17, Bit20], and indistinguishability obfuscation (iO) [BV15, AJ15] (which, in turn, implies a plethora of other cryptographic primitives [SW14]).

An important limitation of FE is the well-known *key escrow* problem: The authority holding the master secret key (sometimes referred to as the private key generator – PKG) can generate secret keys for any function, which in turn allows the PKG to arbitrarily decrypt messages that are intended for specific recipients. Thus, we must assume the PKG is fully trusted, which severely restricts the applicability of FE in many scenarios.

Registered Encryption. A recent line of research proposes to tackle the key-escrow problem in the much simpler case of identity-based encryption[5] (IBE) [Sha84]. This led to the notion of *registered* IBE (RIBE) [GHMR18], where the main idea is to replace the PKG with a much weaker entity called the key curator (KC), whose role is to register the public keys of the users (without possessing any secret key). In particular, in a RIBE scheme there is an initial setup phase in which a common reference string (CRS) is sampled. Hence, the CRS is given to the KC which publishes an (initially empty) master public key. Each user now can sample its own public and secret key, and can register its identity and the chosen public key to the KC; the KC is required to generate a new master public key, which somehow includes the new registered public key, and which will permit encrypting messages to any of the registered users. Moreover, since the master public key is updated over time, the KC is responsible for providing any decrypting party with a so-called helper decryption key, i.e., auxiliary information connecting its public key with the updated master public key.

Recently, the notion of RIBE has been extended to the setting of attribute-based encryption (ABE) [HLWW22], where users can encrypt with respect to policies, and where decryptors can recover the message if their attributes satisfy the policy embedded in the ciphertext. However, in this state of affairs, there is an outstanding question that remains:

Can we construct registered FE?

1.1 Our Contributions

In this work, we make significant progress towards answering the above question. Our results are summarized in Table 1 (in Section 3 on page 11), where we also make a comparison with the other registered encryption schemes in the literature. On the theoretical side, we show that registered FE (RFE) exists in Obfustopia:

Theorem 1 (Informal). *Let λ be the security parameter. Assuming somewhere statistically binding hash functions [HW15, OPWW15] and iO [BGI+12], there is a (non black-box) construction of RFE supporting arbitrary functions and an arbitrary number of users, satisfying the following properties:*

– *The sizes of the common reference string, master public key, and helper decryption keys are $\mathsf{poly}(\lambda)$.*
– *The running time of key generation and registration are $\mathsf{poly}(\lambda)$ and $L \cdot \mathsf{poly}(\lambda)$, respectively, where L stands for the* current *number of registered users.*

[5] IBE can be seen as a special case of FE for equality predicates f_y such that $f_y(x, m) = m$ if and only if $y = x$ (and $\perp$ otherwise). Here, x and y have the role of the parties' identities (which do not need to be secret), and m is the encrypted message.

– *Each registered user receives at most $O(\log L)$ updates from the KC over the entire lifetime of the system, where L is the* current *number of registered users.*

Moreover, the running time of both encryption and decryption is $\mathsf{poly}(\lambda)$.

We highlight that the above construction achieves the same efficiency properties of the (iO-based) RABE construction of Hohenberger *et al.* [HLWW22].

We further investigate practical constructions of RFE, from different assumptions. To this end, we provide a direct construction for registered inner-product predicate encryption (RIPE) based on asymmetric pairings. In particular, our RIPE supports the class of functions $\mathcal{F} = \{f_{\mathbf{x}}(\cdot,\cdot)\}_{\mathbf{x}\in\mathbb{Z}_q^{n+}}$ defined as follows:

$$f_{\mathbf{x}}(m, \mathbf{y}) = \begin{cases} m & \text{if } \langle \mathbf{x}, \mathbf{y} \rangle = 0 \\ \bot & \text{otherwise} \end{cases} \tag{1}$$

where $\mathbf{x}$ and $\mathbf{y}$ are n-size vectors over $\mathbb{Z}_q^{n^+} = \mathbb{Z}_q^n \setminus \{\mathbf{0}^n\}$, and q is a prime. This construction is proven secure in the generic group model over prime-order pairing groups [BCFG17, BFF$^+$19]. We stress that our construction supports a *large universe*, in contrast to [HLWW22], and satisfies the strong notion of *two-sided* security, where no information about the vector $\mathbf{y}$ is learned (besides the orthogonality predicate) even if the decryption is successful, which is akin to what is achieved in [KSW08]. Further, our RIPE scheme can support constant degree polynomial evaluations, disjunctions, conjunctions, and evaluating CNF and DNF formulas, following the transformations described in [KSW08].

Theorem 2 (Informal). *Let λ be the security parameter, let n denote the length of the supported vectors, and let L be a bound on the maximum number of users. In the generic bilinear group model, there is a (black-box) construction of RIPE supporting a large universe and up to L users, satisfying the following properties:*

– *The size of the common reference string is $n \cdot L^2 \cdot \mathsf{poly}(\lambda, \log L)$.*
– *The size of the master public key and of helper decryption keys is $n \cdot \mathsf{poly}(\lambda, \log L)$.*
– *The running time of key generation and registration is $L \cdot \mathsf{poly}(\lambda, \log L)$ and $n \cdot L^2 \cdot \mathsf{poly}(\lambda, \log L)$, respectively.*
– *Each registered user receives at most $O(\log L)$ updates from the KC over the entire lifetime of the system.*

Moreover, the running time of both encryption and decryption is $n \cdot \mathsf{poly}(\lambda, \log L)$.

Somewhat interestingly, our proof strategy and construction template are substantially different from the typical inner-product predicate encryption constructions in the literature (e.g., [KSW08]). Roughly speaking, traditional proof strategies work by "programming" the function output (for the challenge ciphertext) in the key given by the adversary, and then arguing that this new key is indistinguishable from the original distribution. In the registered setting, the adversary can sample its own key, so the reduction has no control over it and cannot modify its distribution. Hence, we view our RIPE scheme as the main technical contribution of this work.

1.2 Open Problems

We view our work as an initial first step in the world of registered FE, however many open problems remain. For example, a natural question is whether registered FE can be constructed generically from any FE scheme. Another interesting direction is to design schemes for specialized function classes from weaker assumptions. Finally, a technical open problem is to prove our RIPE scheme (or some modification thereof) from standard assumptions in bilinear groups.

2 Technical Overview

In the following we describe the notion of registered FE and highlight the properties of interest. Then, we describe the high-level intuition behind our schemes, to convey a sense about the techniques that we develop in this work.

Syntax and Definition of RFE. We start by describing the high-level goals of RFE, and how we chose to formalize its security and efficiency properties. The main property of RFE is that it allows users to generate their keys (associated to a function) without the need of a trusted authority, which is replaced with a KC which does not hold any secret. The KC is simply responsible of administrating a data structure which contains the public keys (together with the corresponding functions) of registered users. Roughly, the RFE work flow goes as follows: On initialization, a common reference string crs is honestly generated by running $\mathsf{Setup}(1^\lambda, |\mathcal{F}|)$ (where $\mathcal{F}$ is the space of the functions supported by the scheme).[6] Then, the crs is given to the KC that will also initialize its state $\alpha = \bot$ (i.e., the data structure) and the initial master public key $\mathsf{mpk} = \bot$. After the initialization is completed, users can register their own (pk, f) pairs. Specifically, a user generates its keys $(\mathsf{pk}, \mathsf{sk}) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{crs}, \alpha)$ and then submits a registration request (pk, f) to the KC, where f is the function the user wishes to associate to its public key (note that sk is never released to the KC). Then, the KC updates its state α and the current master public key mpk by setting $\alpha = \alpha'$ and $\mathsf{mpk} = \mathsf{mpk}'$ where (mpk', α') are outputted by the (deterministic) registration algorithm $\mathsf{RegPK}(\mathsf{crs}, \alpha, \mathsf{pk}, f)$. Intuitively, the new master public key will permits to encrypt messages which can be later decrypted by the users registered at the time of mpk's generation. A sender can encrypt a message m by executing $c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{mpk}, m)$. During decryption, a registered user uses sk to decrypt a ciphertext. However, since mpk is periodically updated (after each registration), users need to get helper decryption keys hsk by issuing an update request to the KC that, in turn, will return $\mathsf{hsk} = \mathsf{Update}(\mathsf{crs}, \alpha, \mathsf{pk})$. Helper decryption keys hsk can be seen as the information needed to make a (previously registered) user's secret key sk valid with respect to a new mpk. Once hsk is obtained, the ciphertext can be decrypted by running $m = \mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, c)$. In terms of efficiency, for a system with L registered users, we wish to achieve the following optimal properties:

(1) *Compact public parameters:* The size of the public parameters (i.e., common reference string) must be small, e.g. $\mathsf{poly}(\lambda, \log L)$ where $\lambda \in \mathbb{N}$ is the security parameter.
(2) *Efficient key generation:* Users can generate public and secret keys efficiently, e.g. the running time for key generation is $\mathsf{poly}(\lambda, \log L)$.
(3) *Efficient registration:* The registration (executed by the KC) of new public keys (together with a corresponding function) is efficient, e.g. it takes at most $\mathsf{poly}(\lambda, \log L)$ time.
(4) *Compact master public keys:* The size of each master public key (updated by the KC after each registration) is small, e.g., $\mathsf{poly}(\lambda, \log L)$.
(5) *Compact helper decryption keys:* The size of helper decryption keys must be small, e.g., $\mathsf{poly}(\lambda, \log L)$.
(6) *Small number of updates:* Each registered user receives new updates (i.e., new helper decryption keys) at most $O(\log L)$ number of times over the lifetime of the system.

A registered encryption scheme (including RFE) can either support an unbounded or a bounded number of users. In particular:

- If a registered encryption scheme supports an unbounder number of users, then the setup is independent from the number of users. Hence, the parameter L of the above efficiency requirements (1)–(6) refers to the *current* number of registered users.
- On the other hand, if a registered encryption scheme supports a only a bounded number of users, then the setup depends on such an (a-priori known) bound L. In turn, the parameter L of requirements (1)–(6) refers to such bound (note that this differs from the unbounded case).

[6] Although the common reference string is generated by a trusted setup, the important difference is that there is no long-term secret that needs to be stored throughout the lifetime of the system. Furthermore, in some cases, the setup algorithm could be "transparent", and therefore computable using just a hash function.

Security of RFE is analogous to that of RIBE [GHMR18] and RABE [HLWW22]. In particular, an adversary A, that controls a subset of k registered users (i.e., A knows $\mathsf{sk}_1, \ldots, \mathsf{sk}_k$ corresponding to the k registered tuples $(\mathsf{pk}_1, f_1), \ldots, (\mathsf{pk}_k, f_k)$), cannot distinguish between $\mathsf{Enc}(\mathsf{mpk}, m_0)$ and $\mathsf{Enc}(\mathsf{mpk}, m_1)$, as long as $f_i(m_0) = f_i(m_1)$ for $i \in [k]$. Moreover, the above indistinguishability notion must hold even in case of an adversary A that registers malformed public keys. For more details, we refer the reader to Section 5.1.

Slotted RFE. Following the approach of Hohenberger *et al.* [HLWW22], it is going to be convenient to define the intermediate notion of *slotted* RFE, as a stepping stone towards fully-fledged RFE. Compared to RFE, the main difference is that (i) there is a single update, referred to as *aggregation*, and (ii) users are assigned to "slots", and the master public key is only defined once all slots are filled. In more details, initialization and key generation work as in standard RFE: The scheme is initialized by executing $\mathsf{crs} \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|)$ where L is the number of slots (i.e., the maximum number of keys that can be aggregated), and users can generate their keys (for a particular slot $i \in [L]$) by running $(\mathsf{pk}, \mathsf{sk}) \leftarrow\$ \mathsf{KGen}(\mathsf{crs}, i)$. After these steps, L users can submit an aggregation request (to the KC) in order to compute a common master public key mpk. Specifically, on input $((\mathsf{pk}_i, f_i))_{i \in [L]}$, the KC deterministically computes a master public key mpk and L helper decryption keys $(\mathsf{hsk}_i)_{i \in [L]}$ (for each user) by executing $(\mathsf{mpk}, (\mathsf{hsk}_i)_{i \in [L]}) = \mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i, f_i))_{i \in [L]})$. The former must receive (at the same time) all the L keys during aggregation. On the other hand, the latter is able to handle registration requests, each issued in different moments. Finally, encryption and decryption works as usual, i.e., $c \leftarrow\$ \mathsf{Enc}(\mathsf{mpk}, m)$ and $m = \mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, c)$.

Similarly to RFE, security of slotted RFE says that, once aggregation has been executed $(\mathsf{mpk}, (\mathsf{hsk}_i)_{i \in [L]}) = \mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i, f_i))_{i \in [L]})$, $\mathsf{Enc}(\mathsf{mpk}, m_0)$ and $\mathsf{Enc}(\mathsf{mpk}, m_1)$ are computationally indistinguishable, as long as for every corrupted slot j (i.e., secret key sk_j is known by the adversary) we have $f_j(m_0) = f_j(m_1)$ where $(\mathsf{mpk}, (\mathsf{hsk}_i)_{i \in [L]}) = \mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i, f_i))_{i \in [L]})$. More details are given in Section 5.2.

It was shown in Hohenberger *et al.* [HLWW22] that slotted RABE implies standard RABE via a generic transformation, and the same holds (up to minor syntactical modifications) for slotted RFE. Loosely speaking, this transformation uses a "power-of-two" approach, where users are assigned to different slotted schemes with increasing capacities, and they are moved forward as new users join the system. This transformation yields a fully-fledged RFE that supports $O(\log L)$ number of updates (requirement (6)) and incurs multiplicative $O(\log L)$ overhead on the size of the public parameters (requirement (1)), key-generation time (requirement (2)), ciphertext size, encryption time, and master public key and helper decryption keys size (requirements (4) and (5)), compared to that of the underlying slotted RFE scheme.

On the other hand, the registration time (requirement (3)) is dominated by $O(t_{\mathsf{Aggr}} + L \cdot t_{\mathsf{hsk}})$ where t_{Aggr} and t_{hsk} are, respectively, the running time and the size of the helper decryption keys of the aggregation algorithm of slotted RFE.

For completeness, we present the transformation in Appendix B. However, for the sake of this overview, we will ignore this aspect, and focus on the goal of constructing slotted RFE.

2.1 (Bounded Users) Slotted RIPE from Pairings

We start with an overview of our scheme for inner-product predicates. This is a special case of functional encryption, where vectors $\mathbf{x} \in \mathbb{Z}_q^{n^+} (= \mathbb{Z}_q^n \setminus \{\mathbf{0}^n\})$ denote functions $f_{\mathbf{x}}$ (associated to keys), and messages consist of a tuple $(m, \mathbf{y})$. The function $f_{\mathbf{x}}$ can be recast as:

$$f_{\mathbf{x}}(m, \mathbf{y}) = \begin{cases} m & \text{if } \langle \mathbf{x}, \mathbf{y} \rangle = 0 \\ \perp & \text{otherwise} \end{cases}$$

where we denote the length of vectors by $n = n(\lambda)$, and assume the attribute space to be $\mathcal{U} = \mathbb{Z}_q^{n^+}$ (i.e., domain of vectors). Our scheme follows the blueprint of [HLWW22]. However, achieving security in the setting of functional encryption requires us to introduce important modifications to their scheme, which we highlight after the overview of our construction below. Furthermore, the security analysis is completely different.

Single-Slot Scheme. We begin by discussing a simplified construction where $L = 1$ (i.e., there is a single slot). A description of each algorithm in the scheme follows below.

– **Generating the CRS:** Let us first describe how the common reference string crs is generated. The CRS can be partitioned into three different parts, a general part, a slot-specific part, and a key-specific part. We will describe how each part is generated individually.
  - *General part*: First, we generate an asymmetric pairing group of prime order q, denoted as $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, g_1, g_2, e)$. Then, we sample $\alpha, \beta, \gamma \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $h = g_1^\beta$, $Z = e(g_1, g_2)^\alpha$. (We will need γ for the multi-slot scheme, which we describe later.)
  - *Slot-specific part*: We associate each slot with a set of group elements, for this case we sample $t \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $A = g_2^t$ and $B = g_2^\alpha A^\beta = g_2^{\alpha+\beta t}$.
  - *Key-specific part*: We also associate a group element to each component of the key vector, plus the secret key. To do this, for each $w \in [n+1]$, we sample $u_w \leftarrow \mathbb{Z}_q$ and set $U_w = g_1^{u_w}$.

  In the end, we set the CRS to be:

$$\mathsf{crs} = \left(\mathcal{G}, Z, h, A, B, \{U_w\}_{w\in[n+1]}\right).$$

– **Generating keys:** To compute a new pair of public/secret keys, we sample a non-zero secret key $\mathsf{sk} \leftarrow\!\!\$\ \mathbb{Z}_q$ and set $\mathsf{pk} = U_{n+1}^{-\mathsf{sk}}$. Note that we are conceptually treating the secret key as one more element of the predicate vector, which is a major structural difference with respect to [HLWW22].

– **Key Aggregation:** Since we only have one slot, given the values pk and crs, and a predicate vector (or key) $\mathbf{x} = (x_1, \ldots, x_n)$, we set the master public key to be

$$\mathsf{mpk} = \left(\mathcal{G}, h, Z, \{U_w\}_{w\in[n+1]}, \mathsf{pk} \cdot \prod_{w=1}^{n} U_w^{-x_w}\right).$$

– **Encryption:** In order to encrypt a message $m \in \mathbb{G}_\mathsf{T}$ with respect to a non-zero attribute vector $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{Z}_q^{n^+}$, and the master public key mpk, we create a ciphertext that has two components, a message-embedding component, and a key-slot-embedding component.
  - *Message embedding:* We sample $s \leftarrow\!\!\$\ \mathbb{Z}_q \setminus \{0\}$, and set $C_1 = m \cdot Z^s$, and $C_2 = g_1^s$.
  - *Key-slot embedding:* First, we sample $r, z \leftarrow\!\!\$\ \mathbb{Z}_q \setminus \{0\}$. Then, we set

$$C_{3,w} = h^{y_w \cdot r + s} \cdot U_w^{-z} \ (\forall w \in [n]), \quad C_{3,n+1} = h^s \cdot U_{n+1}^{-z}, \quad \text{and}$$

$$C_{3,n+2} = h^s \cdot \mathsf{pk}^{-z} \prod_{w=1}^{n} U_w^{z \cdot x_w}.$$

  The final ciphertext will be $(C_1, C_2, \{C_{3,w}\}_{w\in[n+1]})$.

– **Decryption:** Before describing the actual decryption process, let us check the intuition behind each element of the ciphertext. The first component $C_1 = m \cdot Z^s$ is just a masking of the message with a random power of Z from the CRS. Consider B from the CRS, and the ciphertext components C_1 and C_2, and observe the following:

$$\frac{C_1}{e(C_2, B)} = \frac{m \cdot e\left(g_1, g_2\right)^{\alpha \cdot s}}{e\left(g_1, g_2\right)^{\alpha \cdot s} \cdot e\left(g_1, g_2\right)^{s\beta t}} = \frac{m}{e\left(h^s, A\right)}.$$

  Thus, in order to recover the message, it suffices to recompute $e(h^s, A)$. Observe that h^s is already present in some form in the $C_{3,*}$ components. We can partition $C_{3,*}$ ciphertexts into three different groups, and see how h^s appears in each group.

  1. For all $w \in [n]$, we have $C_{3,w} = h^s \cdot h^{y_w \cdot r} \cdot U_w^{-z}$. In this case, there are extra terms $y_w \cdot r$ as well as U_w present in the ciphertext. However, since $\mathbf{x}$ and $\mathbf{y}$ are orthogonal (otherwise decryption fails), we

can eliminate these extra terms by raising each $C_{3,w}$ to the power of x_w for $w \in [n]$ and compute their product. Thus, we will have:

$$\prod_{w=1}^{n} C_{3,w}^{x_w} = \prod_{w=1}^{n} h^{x_w \cdot s} \cdot h^{x_w \cdot y_w \cdot r} \cdot \prod_{w=1}^{n} U_w^{-z \cdot x_w}$$
$$= h^{s \cdot \sum_{w=1}^{n} x_w} \cdot \underbrace{h^{r \cdot \sum_{w=1}^{n} x_w \cdot y_w}}_{=1} \cdot \prod_{w=1}^{n} U_w^{-z \cdot x_w}.$$

Therefore, we are left with two terms $h^{s \cdot \sum_{w=1}^{n} x_w}$ and $\prod_{w=1}^{n} U_w^{-z \cdot x_w}$.

2. For $w = n+1$, we have $C_{3,n+1} = h^s \cdot U_{n+1}^{-z}$, where the term h^s is masked with U_{n+1}^{-z}.
3. For $w = n+2$, we have $C_{3,n+2} = h^s \cdot \mathsf{pk}^{-z} \prod_{w=1}^{n} U_w^{z \cdot x_w} = h^s \cdot U_{n+1}^{z \cdot \mathsf{sk}} \cdot \prod_{w=1}^{n} U_w^{z \cdot x_w}$.

Multiplying together the remaining components we obtain:

$$C_{3,n+2} \cdot C_{3,n+1}^{\mathsf{sk}} \cdot \prod_{w=1}^{n} C_{3,w}^{x_w} = h^s \cdot h^{s \cdot \mathsf{sk}} \cdot h^{s \cdot \sum_{w=1}^{n} x_w} = h^{s \cdot \left(1 + \mathsf{sk} + \sum_{w=1}^{n} x_w\right)}.$$

The decryptor can now raise $h^{s \cdot \left(1 + \mathsf{sk} + \sum_{w=1}^{n} x_w\right)}$ to the power of $(1 + \mathsf{sk} + \sum_{w=1}^{n} x_w)^{-1}$ to get h^s. Once h^s is obtained, it can be paired with A from the CRS to decrypt the message.

Multi-Slot Scheme. To gain an intuition on how our construction handles multiple slots, we describe a toy example where $L = 2$, i.e., we are in the two-slot setting. Notice that one trivial generalization is to individually generate public keys as before, and concatenate them into the master public key. However, this approach will not work, since we want the size of the master public key to be independent of the number of slots. Instead, we expand the slot-specific components in the CRS to A_1, B_1 (for slot 1) and A_2, B_2 (for slot 2), which are generated in the same way as A, B in the one-slot setting, but using independent random elements $t_1, t_2 \leftarrow\!\!\$\ \mathbb{Z}_q$ in generating A_1, A_2. We will also need to link the slots to the keys so that we can use the slot in the key-generation algorithm. For this, instead of generating only one set of $\{U_w\}_{w \in [n]}$, we generate them with respect to both slots

$$\{U_{w,1} = g_1^{u_{w,1}}\}_{w \in [n+1]} \qquad \text{and} \qquad \{U_{w,2} = g_1^{u_{w,2}}\}_{w \in [n+1]}$$

where the elements $\{u_{w,i}\}_{i \in \{1,2\}}$ are chosen independently and uniformly at random. Accordingly, in the key generation we can set

$$\mathsf{pk}_1 = U_{n+1,1}^{-\mathsf{sk}_1} \qquad \text{and} \qquad \mathsf{pk}_2 = U_{n+1,2}^{-\mathsf{sk}_2}$$

and we aggregate the keys as

$$\{\widehat{U}_w = U_{w,1} \cdot U_{w,2}\}_{w \in [n+1]} \qquad \text{and} \qquad \widehat{U}_{n+2} = \mathsf{pk}_1 \cdot \mathsf{pk}_2 \cdot \prod_{w=1}^{n} U_{w,1}^{-x_{w,1}} \prod_{w=1}^{n} U_{w,2}^{-x_{w,2}}$$

where $\mathbf{x}_1$ and $\mathbf{x}_2$ are the chosen keys. One can encrypt using the new $\widehat{U}$ values instead of U, however, once we try to decrypt and expand the corresponding equations, we realize that many terms will not cancel out as before. For example, if a message is encrypted for slot 1, during decryption we will have,

$$\prod_{w \in [n]} C_{3,w}^{x_{w,1}} = \prod_{w \in [n]} h^{(y_w \cdot r + s) \cdot x_{w,1}} \cdot \prod_{w=1}^{n} U_{w,1}^{-z \cdot x_{w,1}} \cdot \prod_{w=1}^{n} U_{w,2}^{-z \cdot x_{w,1}}$$
$$C_{3,n+1}^{\mathsf{sk}_1} = h^{s \cdot \mathsf{sk}_1} \cdot U_{n+1,1}^{-z \cdot \mathsf{sk}_1} \cdot U_{n+1,2}^{-z \cdot \mathsf{sk}_1}$$
$$C_{3,n+2} = h^s \cdot U_{n+1,1}^{z \cdot \mathsf{sk}_1} \cdot U_{n+1,2}^{z \cdot \mathsf{sk}_2} \cdot \prod_{w=1}^{n} U_{w,1}^{z \cdot x_{w,1}} \prod_{w=1}^{n} U_{w,2}^{z \cdot x_{w,2}}$$

where the terms in blue can be canceled out using a similar multiplication trick as before. However, the terms $U_{n+1,2}^{-z\cdot\mathsf{sk}_1}$, $U_{n+1,2}^{z\cdot\mathsf{sk}_2}$, $\prod_{w\in[n]} U_{w,2}^{-z\cdot x_{w,1}}$ and $\prod_{w=1}^{n} U_{w,2}^{z\cdot x_{w,2}}$ *cannot* be canceled as they do not appear anywhere else, and furthermore we assume the decryptor only knows sk_1, but not sk_2. However, we can circumvent this issue by introducing some "cross-terms" into the CRS, and use them in the aggregation to compute helper secret keys that enable the decryptor (holding sk_1 and $\mathbf{x}_1$) to cancel such terms. We create these terms such that they include both slot-specific and key-specific parts. Intuitively, they bind each slot to other slots and keys together. For slots $i, j \in [2]$ where $i \neq j$ and key indices $w \in [n+1]$, we define these terms as:

$$W_{i,j,w} = A_i^{u_{j,w}}.$$

We add $\{W_{i,j,w}\}_{i\neq j\in[2],w\in[n+1]}$ to the CRS as:

$$\mathsf{crs} = \left(\mathcal{G}, Z, h, \{A_i, B_i\}_{i\in[2]}, \left\{\{U_{w,i}\}, \{W_{i,j,w}\}_{i\neq j}\right\}_{i,j\in[2],w\in[n+1]}\right).$$

In addition, we will let the user publish $\{W_{j,i,n+1}^{\mathsf{sk}_i}\}_{i\in\{1,2\},j\neq i}$ in their respective public keys, to enable the other users to cancel out the desired cross terms, and publish in the ciphertext an additional element $C_4 = g_1^z$, to be paired with the W's in order to compute the correct terms.

Unfortunately, as described, the construction is correct but *insecure*. At a high level, the problem is that the adversary can pair C_4 with the wrong elements, recovering some extra information about the encrypted message. For this reason, instead of putting g_1^z directly in the ciphertext, we introduce an extra component $\Gamma = g_1^\gamma, \gamma \leftarrow_{\$} \mathbb{Z}_q$ in the CRS, and set $C_4 = \Gamma^z$. The only other modification that we must apply is the generation of the CRS itself, where for slots $i, j \in \{1, 2\}$ with $i \neq j$, and key indices $w \in [n+1]$, we define:

$$W_{i,j,w} = A_i^{u_{j,w}/\gamma}.$$

The rest of the construction remains the same. See Section 6 for more details.

Proof Sketch. We prove the above slotted RIPE scheme secure in the generic bilinear group model (GGM). Recall that in the GGM, the adversary is supplied with handles to the corresponding group elements from the scheme. Further, it can also learn handles to arbitrary linear combinations of existing and new elements (in the same group $\mathbb{G}_\mathsf{t}, \mathsf{t} \in \{1, 2, \mathsf{T}\}$) via the group oracles it is provided with. Additionally, since we are in the bilinear setting, the adversary also gets access to the pairing oracle that lets it learn handles referring to the product of any two terms from the source groups $\mathbb{G}_1$ and $\mathbb{G}_2$. However, the only crucial information it can actually learn in this whole interaction is via the zero-tests that work again only in $\mathbb{G}_\mathsf{T}$.

Our formal multi-slot RIPE scheme in Section 6 introduces several variables with different combinations of indices. To argue indistinguishability in a convenient way between subsequent hybrids in the proof, we first switch from the GGM to the symbolic group model (SGM) via the Schwarz-Zippel lemma. In particular, the SGM allows us to represent all the terms, that the adversary can learn in the security game, as multivariate polynomials (in respective groups) from a ring of variables. The heart of the proof relies on arguing properties of the *coefficients* of these polynomials that correspond to *successful* zero-tests, which aids in proving indistinguishability directly. More specifically, these claims set in where we switch the challenge attribute from $\mathbf{y}_0$ to $\mathbf{y}_1$ in the ciphertext components $C_{3,w}$ for all $w \in [n+2]$ and helps in arguing the following:

1. Coefficients of such polynomials formed by pairing terms $C_{3,w} \in \mathbb{G}_1$ with *any* element in $\mathbb{G}_2$, except $A_i, i \in [2]$, must be *all zero.*
2. Such a coefficient vector must be *orthogonal* to $\mathbf{y}_b$ for $b \in \{0, 1\}$, and in particular, either be a *constant multiple* of the vector $\widetilde{\mathbf{x}}_i = (\mathbf{x}_i, \mathsf{sk}_i), i \in [2]$ or be *all zero.*

The claim in Item 1 follows from observing that the monomials formed symbolically (in the exponent) when pairing $C_{3,w}$ with *anything* in $\mathbb{G}_2$ (except A_1 or A_2) are all linearly independent and do not cancel out. Item 2 follows from two observations. The first one is that the randomness r (appearing as an independent symbolic term, but only in the components $C_{3,w}$'s) can only cancel out in zero-tests when the coefficients are orthogonal to $\mathbf{y}_b$. The second one follows additionally from linear independence of some specific symbolic terms and observing further that the vector of first $n+1$ coefficients can be expressed as a constant multiple of $\widetilde{\mathbf{x}}_i$. Overall, these claims ensure that the only non-trivial adversarial queries can be for vectors lying in the span of both *registered and valid* predicates. The rest of the proof follows from the admissibility of the adversary, and by reusing these claims. We refer to Section 6 for more details.

Comparison with the slotted RABE of [HLWW22]. Although our construction of slotted RIPE from prime-order pairings and the construction of slotted RABE from composite-order pairings achieve totally different functionalities and security notions, there are similarities between the two constructions. For instance, in both constructions the message-embedding mechanism works in the same way, which is by masking the message with the randomness in the term $e(h^s, A_i)$. On the other hand, the way slots and attributes are "glued" together is fundamentally different: In [HLWW22], the ciphertext has two specific components, an attribute-specific component and a slot-specific one, where one party can decrypt a message if it manages to succeed to decrypt the slot-specific component and the attribute-specific component simultaneously. On the other hand, in our construction, the slots and the attribute part are intertwined in the same ciphertext component. Furthermore, we conceptually treat the secret key as "one more dimension" in the attribute vector, whereas the construction in [HLWW22] uses a separate machinery that takes care of the key component.

2.2 (Unbounded Users) Slotted RFE from iO

As a feasibility result, we show that RFE for arbitrary functions can be built from indistinguishability obfuscation (iO) [BGI+12] and (succinct) somewhere statistically binding hash functions (SSB) [HW15, OPWW15]. The technique is analogous to that of Hohenberger *et al.* [HLWW22], and consists in constructing an (sufficiently efficient) iO-based slotted RFE scheme and then applying the "power-of-two" transformation. We give a brief overview of the iO-based slotted RFE construction below.

The CRS is simply set to be the SSB hash key hk, and users' keys are generated through a PRG G (with large enough stretch), i.e., $(\mathsf{pk}, \mathsf{sk}) = (\mathsf{G}(s), s)$ where s is a random seed. To aggregate $((\mathsf{pk}_i, f_i))_{i\in[L]}$ into a single master public key, the KC computes a Merkle tree hash $h = \mathsf{Hash}(\mathsf{hk}, ((\mathsf{pk}_i, f_i))_{i\in[L]})$ and sets $\mathsf{mpk} = (\mathsf{hk}, h)$. Moreover, the helper decryption key hsk_i (of the i-th slot) is essentially the SBB opening π_i for the i-th (hashed) block (pk_i, f_i). Now, a ciphertext c (encrypting a message m) is simply the obfuscation of a circuit $C_{h,m}$ that, on input $(i, \mathsf{pk}_i, f_i, \pi_i, \mathsf{sk}_i)$, returns $f_i(m)$ if the following two conditions are satisfied: (i) π_i is a valid opening for the i-th block (pk_i, f_i) (i.e., (pk_i, f_i) has been aggregated and associated to the i-th slot), and (ii) $(\mathsf{pk}_i, \mathsf{sk}_i)$ is a valid key pair (i.e., $\mathsf{G}(\mathsf{sk}_i) = \mathsf{pk}_i$). Hence, on decryption, it is sufficient to use the secret key sk_i and the helper decryption key $\mathsf{hsk}_i = \pi_i$ to evaluate the obfuscated circuit (i.e., the ciphertext) on input $(i, \mathsf{pk}_i, f_i, \pi_i, \mathsf{sk}_i)$.

The construction supports any polynomial-time computable function. As for efficiency, the sizes of crs, mpk, and hsk (of our slotted iO-based RFE) are proportional to the sizes of SSB's hash keys, output, and openings. Hence, by combining a succinct SSB hash function (i.e., hash keys, outputs, and openings sizes are poly-logarithmic in the number of blocks L) and the "power-of-two" transformation, we obtain an RFE scheme satisfying requirements (1)–(2) and (4)–(6). Lastly, because of the above poly-logarithmic dependency, we can set the final RFE scheme with $L = 2^\lambda$ in order to handle an arbitrary number of users. Regarding the registration (requirement (3)), its running time remains linear in the *current/effective* number of registered users (e.g., polynomial) at the time of registration. We provide more details in Section 7 and Appendix B.

2.3 On Function-Privacy in (Slotted) RFE

By definition, RFE allows users to sample their own keys and functions. Thus, the notion of function-privacy, that is typically considered in the setting of (secret-key) FE [SSW09, BS15], does not make much sense from this perspective. However, one can still define function-privacy here with respect to any other registered or unregistered party. In more detail, in the general setting of RFE, a user choosing its own keys and functions may want to hide its function from any party including the KC. Capturing this requires a mild change in the

Reference	Type	CRS size	Keygen runtime	Registration key runtime	Master public key size	Helper dec. key size	# Updates	Unbounded users	BB	Assumptions
[GHMR18]	IBE	$O(1)$	$O(1)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$O(\log L)$	✓	✗	iO + SSB
[GHMR18]	IBE	$O(1)$	$O(1)$	$O(L)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$O(\log L)$	✓	✗	CDH/LWE
[GHM+19]	Anon. IBE	$O(1)$	$O(1)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$O(\log L)$	✓	✗	CDH/LWE
[GV20]	IBE	$O(1)$	$O(1)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$\mathsf{poly}(\log L)$	$O(\log L)$	✓	✗	CDH/LWE
[CES21]	IBE	$O(1)$	$O(1)$	$\mathsf{poly}(\log L)$	$O(\sqrt{L})$	$\mathsf{poly}(\log L)$	$O(\log L)$	✓	✗	CDH/LWE
[GKMR22]	IBE $O(1)$-size ciphertexts	$O(\sqrt{L})$	$O(\sqrt{L})$	$O(\sqrt{L})$	$O(\sqrt{L})$	$O(\sqrt{L})$	$O(\sqrt{L})$	✗	✓	Pairings of Prime Order
[GKMR22]	IBE $O(\log \sqrt{L})$-size ciphertexts	$O(\sqrt{L})$	$O(\sqrt{L})$	$O(\sqrt{L} \log \sqrt{L})$	$O(\sqrt{L} \log \sqrt{L})$	$O(\log \sqrt{L})$	$O(\log \sqrt{L})$	✗	✓	Pairings of Prime Order
[HLWW22]	ABE small attribute space $\mathcal{U}$ LSSS policies	$L^2 \cdot \mathsf{poly}(\lvert\mathcal{U}\rvert, \log L)$	$L \cdot \mathsf{poly}(\lvert\mathcal{U}\rvert, \log L)$	$L \cdot \mathsf{poly}(\lvert\mathcal{U}\rvert, \log L)$	$\lvert\mathcal{U}\rvert \cdot \mathsf{poly}(\log L)$	$\lvert\mathcal{U}\rvert \cdot \mathsf{poly}(\log L)$	$O(\log L)$	✗	✓	Pairings of Composite Order
[HLWW22]	ABE large attribute space $\mathcal{U}$ arbitrary policies	$O(1)$	$O(1)$	$O(L)$	$O(1)$	$O(1)$	$O(\log L)$	✓	✗	iO + SSB
Ours §6	Inner-Product PE large function space $\mathcal{F}$ n-size vectors	$n \cdot L^2 \cdot \mathsf{poly}(\log L)$	$L \cdot \mathsf{poly}(\log L)$	$n \cdot L^2 \cdot \mathsf{poly}(\log L)$	$n \cdot \mathsf{poly}(\log L)$	$n \cdot \mathsf{poly}(\log L)$	$O(\log L)$	✗	✓	Pairings of Prime Order + GGM
Ours §7	FE large function space $\mathcal{F}$ arbitrary functions	$O(1)$	$O(1)$	$O(L)$	$O(1)$	$O(1)$	$O(\log L)$	✓	✗	iO + SSB

Table 1. Comparing known constructions of registered encryption schemes in terms of efficiency and assumptions. We only consider worst-case running time complexity. For constructions supporting an unbounded (resp. bounded) number of users, the value L stands for the *current* number of registered users (resp. the maximum number of supported users). To simplify the table, we omitted the term λ, e.g. $O(1)$ stands for $\mathsf{poly}(\lambda)$, $O(k)$ stands for $k \cdot \mathsf{poly}(\lambda)$, $\mathsf{poly}(\log k)$ stands for $\mathsf{poly}(\lambda, \log k)$, etc. GGM stands for generic (bilinear) group model, and BB stands for black-box, i.e. the construction makes only black-box use of cryptography. The value $\mathcal{U}$ of [HLWW22] stands for the attribute space supported by the corresponding scheme. Similarly, the value $\mathcal{F}$ stands for the function space supported by our schemes in §5.1 and §6 (we recall that each function $f \in \mathcal{F}$ of our RIPE construction corresponds to an n-size vector from $\mathbb{Z}_q^{n^+}$).

RFE syntax, where the function can be input to the key generation algorithm instead of aggregation and also require that the generated user key-pair is tied to this function. On the other hand, the KC gets access only to the users' public keys to aggregate them and generate the master public key with helper decryption keys.[7] The security definition would need to change accordingly. In particular, it would now additionally require each public key to computationally hide the function tied to it. Further, the aggregated master public key should also computationally hide all information about the functions embedded in each of its public keys.[8]

All our schemes can be modified to satisfy this syntax. For example, our slotted RIPE from pairings can be easily adapted to this notion since the *extended* key $\widetilde{\mathbf{x}}_i = (\mathbf{x}_i, \mathsf{sk}_i, 1)$ is embedded in the public-key pk_i for slot $i \in [2]$ as $\mathsf{pk}_i = \prod_{w=1}^{n+1} U_{w,i}^{-\widetilde{x}_{w,i}}$. This holds similarly for the cross-terms as well. Using a NIZK, the users can prove that they always choose a non-zero vector as its predicate. It is also easy to verify the same for our slotted RFE from iO. However, for simplicity, we avoid formalizing this in our definitions and schemes. Our formal constructions from Sections 6 and 7 are thus in the traditional RFE setting (i.e., without function-privacy). Building more efficient function-private RFE for specific functions is left as a future work.

3 Related Work

In Table 1, we compare our schemes with the other registered encryption schemes in the literature. The original paper [GHMR18] showed how to construct RIBE, achieving requirements (1)–(6), from iO [BGI+01] and SSB hash functions [HW15, OPWW15]; this was later improved by Garg *et al.* [GHM+19], who showed

[7] In such a setting users can try to register arbitrary functions of their choice which would allow them to learn arbitrary information about encrypted messages. To prevent this, one can restrict the function class during the setup meaningfully (e.g., excluding trivial functions like identity). Any user wanting to register its public key would thus need to prove the validity of its chosen function w.r.t this class of functions.

[8] Since aggregation is deterministic and is publicly computable, its output would also computationally hide all information about the functions embedded in each of its public keys.

how to obtain RIBE (with the same level of efficiency) from standard assumptions (e.g., CDH and LWE) even for the stronger primitive of *anonymous* IBE (where the identity y remains private). Subsequent work in the context of RIBE focused on adding verifiability [GV20], proving lower bounds on the number of required decryption updates [MQR22], and improving on practical efficiency. Cong *et al.* [CES21] propose a RIBE scheme achieving a constant efficiency speed-up w.r.t. [GV20] of 15% on average in terms of computation and communication (ciphertext size) by the encryptor, and a 30% speed-up in computation by the decryptor with the same efficiency requirements. By relaxing requirement (4) to $O(\sqrt{L})$ they reduce the ciphertext size (communication cost) by 57.5% on average. Glaeser *et al.* [GKMR22] propose a *black-box construction* of RIBE, but they relax the efficiency requirements (1)–(6) to be $O(\sqrt{L})$. Moreover, they show that it is possible to improve requirements (5) and (6) to $O(\log \sqrt{L})$ at the price of increasing by a multiplicative factor of $O(\log \sqrt{L})$ the size of the ciphertexts and of the master public key (requirement (4)), and the registration running time (requirement (3)).

Very recently, the concept of RIBE has been extended to the more general setting of attribute-based encryption (ABE) [GPSW06]. In particular, Hohenberger *et al.* [HLWW22] showed how to construct (in a black-box fashion) a registered ABE (RABE) scheme that supports an a priori *bounded* number of users, and policies that can be described by a linear secret sharing scheme, from assumptions on composite-order pairing groups. However, their scheme does not fully meet requirements (1)–(3), in that the size of the public parameters and the running time for both key generation and registration are $L^2 \cdot \mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$ and $L \cdot \mathsf{poly}(\lambda, |\mathcal{U}|, \log L)$ where L is the maximum number of supported users, and $|\mathcal{U}|$ is the size of the attribute space.[9]

Note that the dependency on $|\mathcal{U}|$ affects their RABE construction, allowing the latter to only support a small attribute space (e.g., $|\mathcal{U}| \in \mathsf{poly}(\lambda)$). Notably, our (paring-based) RIPE construction does not suffer from this limitation since the parameters of our scheme depend only on the length $n = n(\lambda)$ of the vectors (see Table 1); hence, our RIPE scheme supports a function space $\mathcal{F}$ of exponential size (recall that, in the case of inner-product PE, each function $f \in \mathcal{F}$ corresponds to a particular vector). Also, as a feasibility result, [HLWW22] proposed a RABE construction (based on iO and SBB hash functions) supporting an *unbounded* number of users, arbitrary policies, and satisfying all the efficiency requirements except for the registration running time (requirement (3)), which is still $L \cdot \mathsf{poly}(\lambda)$ where L is the *current* number of registered users at the time of execution.

In [GV20], the authors further introduce a RABE extension to more general access structures. In particular, they propose a universal definition of registration-based encryption in which the algorithms take as an additional input the description of an FE scheme (although no construction is presented). Such algorithms compile the standard algorithm behavior of the FE scheme into a (verifiable) registration-based one. However, our tailored notion for the functional encryption setting is more natural and follows directly from the RABE definition.

Finally, we also mention a related work on dynamic decentralized FE [CDSG+20], where the keys are also sampled by the users and there is no trusted authority. However, an important difference from the registered setting, is that in decentralized FE there is no requirement on the size of the master public key, which can be as large as the number of registered users. This is a major challenge (and arguably the defining feature) of all registered encryption schemes. We also mention that the work of [CDSG+20], achieves a slightly different inner-product functionality, where the output is always the inner product $\langle \mathbf{x}, \mathbf{y} \rangle$, whereas our scheme is in the more challenging predicate settings (with two-sided security).

4 Preliminaries

Notation. We write $[n] = \{1, 2, \ldots, n\}$ and $[0, n] = \{0\} \cup [n]$. Capital bold-face letters (such as $\mathbf{X}$) are used to denote random variables, small bold-face letters (such as $\mathbf{x}$) to denote vectors, small letters (such as x)

[9] Despite the sizes of both master public keys and helper decryption keys (requirements (4) and (5)) are polylogarithmic in L, they are still linear in the size of the attribute space $|\mathcal{U}|$, i.e., $|\mathcal{U}| \cdot \mathsf{poly}(\lambda, \log L)$. See Table 1 for more details.

to denote concrete values, calligraphic letters (such as $\mathcal{X}$) to denote sets, serif letters (such as A) to denote algorithms. All of our algorithms are modeled as (possibly interactive) Turing machines.

For a string $x \in \{0,1\}^*$, we let $|x|$ be its length; if $\mathcal{X}$ is a set or a list, $|\mathcal{X}|$ represents the cardinality of $\mathcal{X}$. When x is chosen uniformly in $\mathcal{X}$, we write $x \leftarrow\!\$\, \mathcal{X}$. If A is an algorithm, we write $y \leftarrow\!\$\, \mathsf{A}(x)$ to denote a run of A on input x and output y; if A is randomized, y is a random variable and $\mathsf{A}(x;r)$ denotes a run of A on input x and (uniform) randomness r. An algorithm A is *probabilistic polynomial-time* (PPT) if A is randomized and for any input $x, r \in \{0,1\}^*$ the computation of $\mathsf{A}(x;r)$ terminates in a polynomial number of steps (in the input size). We write $C(x) = y$ to denote the evaluation of the circuit C on input x and output y. For any integer $k \in \mathbb{N}$, we denote $\mathbb{Z}_q^{k^+} = \mathbb{Z}_q^k \setminus \{\mathbf{0}^k\}$ as the set of all non-zero k-size vectors over $\mathbb{Z}_q$, and $\mathbb{Z}_q^+ = \mathbb{Z}_q \setminus \{0\}$.

Negligible functions. Throughout the paper, we denote by $\lambda \in \mathbb{N}$ the security parameter and we implicitly assume that every algorithm takes as input the security parameter. A function $\nu(\cdot)$ is called negligible in the security parameter $\lambda \in \mathbb{N}$ if it vanishes faster than the inverse of any polynomial in λ, i.e. $\nu(\lambda) \in O(1/p(\lambda))$ for all positive polynomials $p(\lambda)$. We sometimes write $\mathsf{negl}(\lambda)$ (resp. $\mathsf{poly}(\lambda)$) to denote an unspecified negligible function (resp. polynomial function) in the security parameter.

Indistinguishability Obfuscation. Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ be an ensemble of circuits. An indistinguishability obfuscator (iO) [BGI+12] is a PPT algorithm Obf that, on input the security parameter 1^λ and a circuit $C \in \mathcal{C}_\lambda$, it outputs an obfuscation $\mathsf{Obf}(1^\lambda, C)$ of C. An iO obfuscator Obf must (i) preserve the functionality of the original circuit C (correctness), and (ii) produce obfuscations of "small" size (polynomial slowdown), i.e., polynomial in the size $|C|$ of the original circuit C. As for security, iO guarantees that, for every pair of functionally-equivalent circuits $C_0, C_1 \in \mathcal{C}_\lambda$ (i.e., $\forall x \in \{0,1\}^*$, $C_0(x) = C_1(x)$), the obfuscations $\mathsf{Obf}(1^\lambda, C_0)$ and $\mathsf{Obf}(1^\lambda, C_1)$ are computational indistinguishable.

We recall the formal definition below.

Definition 1 (Indistinguishability obfuscation). *Let $\mathcal{C} = \{\mathcal{C}_\lambda\}_{\lambda \in \mathbb{N}}$ an ensemble of circuits. A PPT algorithm Obf is an iO obfuscator if the following conditions hold:*

Correctness. *$\forall \lambda \in \mathbb{N}$, $\forall C \in \mathcal{C}_\lambda$, $\forall x \in \{0,1\}^*$, we have $C'(x) = C(x)$ where $C' \leftarrow\!\$\, \mathsf{Obf}(1^\lambda, C)$.*

Polynomial slowdown. *There exists a polynomial $p(\cdot)$ such that for every $C \in \mathcal{C}_\lambda$, we have $|\mathsf{Obf}(1^\lambda, C)| \leq p(|C|)$.*

Indistinguishability. *For every pair of functionally-equivalent circuits $C_0, C_1 \in \mathcal{C}_\lambda$, for every PPT adversary D, we have that*

$$\left|\mathbb{P}\left[\mathsf{D}(1^\lambda, \mathsf{Obf}(1^\lambda, C_0)) = 1\right] - \mathbb{P}\left[\mathsf{D}(1^\lambda, \mathsf{Obf}(1^\lambda, C_1)) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

Somewhere Statistically Binding Hash Functions. A somewhere statistically binding (SSB) hash function [HW15, OPWW15] (supporting local openings) with block length $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, output length $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$, and opening length $\ell_{\mathsf{open}} = \ell_{\mathsf{open}}(\lambda)$, is composed of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i)$**:** On input the security parameter 1^λ, a block size $1^{\ell_{\mathsf{blk}}}$, a message length $N \leq 2^\lambda$, and an index $i \in [N]$, the randomized setup algorithm outputs a key hk. Here, we assume that N and i are encoded in binary, i.e., the size of both N and $i \in [N]$ are bounded by $O(\log(N))$.

$\mathsf{Hash}(\mathsf{hk}, (x_i)_{i \in [N]})$**:** On input a key hk and an input $(x_i)_{i \in [N]}$ (where $x_i \in \{0,1\}^{\ell_{\mathsf{blk}}}$), the deterministic hash algorithm outputs an hash $h \in \{0,1\}^{\ell_{\mathsf{out}}}$.

$\mathsf{Open}(\mathsf{hk}, (x_i)_{i \in [N]}, i)$**:** On input a key hk, an input $(x_i)_{i \in [N]}$ (where $x_i \in \{0,1\}^{\ell_{\mathsf{blk}}}$), and an index $i \in [N]$, the deterministic open algorithm outputs an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$.

$\mathsf{Verify}(\mathsf{hk}, h, i, x_i, \pi_i)$**:** On input a key hk, an hash $h \in \{0,1\}^{\ell_{\mathsf{out}}}$, an index $i \in [N]$, an input $x_i \in \{0,1\}^{\ell_{\mathsf{blk}}}$, and an opening $\pi_i \in \{0,1\}^{\ell_{\mathsf{open}}}$, the deterministic algorithm outputs a decision bit $b \in \{0,1\}$.

Correctness of SSB says that honest openings always verify. As for security, SSB guarantees *index hiding* and *somewhere statistically binding*. The former guarantees that an adversary cannot distinguish whether hk is generated (on setup) under an index $i_0 \in [N]$ or index $i_1 \in [N]$. On the other hand, the latter guarantees that, whenever hk is generated w.r.t. an index $i \in [N]$ (i.e., $\mathsf{dk} \leftarrow_\$ \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i)$), the i-th slot is statistically binding, i.e., it does not exists $h \in \{0,1\}^{\ell_{\mathsf{out}}}$ and $(x, \pi), (x', \pi') \in \{0,1\}^{\ell_{\mathsf{blk}}} \times \{0,1\}^{\ell_{\mathsf{open}}}$ such that $x \neq x'$ and $\mathsf{Verify}(\mathsf{hk}, h, i, x, \pi) = \mathsf{Verify}(\mathsf{hk}, h, i, x', \pi') = 1$.

Definition 2 (Correctness). *A SSB scheme $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ is correct if, $\forall \lambda \in \mathbb{N}$, $\forall \ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, $\forall$ integers $N \leq 2^\lambda$, $\forall i^*, i \in [N]$, and $\forall (x_i)_{i \in [N]} \in \{0,1\}^{\ell_{\mathsf{blk}} \cdot N}$, we have:*

$$\mathbb{P}\left[\mathsf{Verify}(\mathsf{hk}, h, i, x_i, \pi_i) = 1 \middle| \begin{array}{c} \mathsf{hk} \leftarrow_\$ \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i^*), \\ h = \mathsf{Hash}(\mathsf{hk}, (x_i)_{i \in [N]}), \\ \pi_i = \mathsf{Open}(\mathsf{hk}, (x_i)_{i \in [N]}, i) \end{array}\right] = 1.$$

Definition 3 (Index Hiding). *A SSB scheme $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ satisfies index hiding if for every PPT adversary D, for every $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, for every integer $N \in \mathbb{N}$, for every indexes $i_0, i_1 \in [N]$, we have:*

$$\left| \mathbb{P}\left[\mathsf{D}(1^\lambda, \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i_0)) = 1\right] - \mathbb{P}\left[\mathsf{D}(1^\lambda, \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i_1)) = 1\right] \right| \leq \mathsf{negl}(\lambda).$$

Definition 4 (Somewhere Statistically Binding). *A SSB scheme $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ is somewhere statistically binding if, for every $\ell_{\mathsf{blk}} = \ell_{\mathsf{blk}}(\lambda)$, for every $N \leq 2^\lambda$, for every $i \in [N]$, we have:*

$$\mathbb{P}\left[\begin{array}{c} \nexists (h, x, x', \pi, \pi') \in \{0,1\}^{\ell_{\mathsf{out}} + 2\ell_{\mathsf{blk}} + 2\ell_{\mathsf{open}}} \\ s.t.\ x \neq x' \ and \\ \mathsf{Verify}(\mathsf{hk}, h, i, x, \pi) = 1 \ and \\ \mathsf{Verify}(\mathsf{hk}, h, i, x', \pi') = 1 \end{array} \middle| \mathsf{hk} \leftarrow_\$ \mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i) \right] \geq 1 - \mathsf{negl}(\lambda).$$

In addition to the above properties, we focus on succinct and efficient SSB schemes which can be built from different assumptions such as DDH, ϕ-Hiding, DCR, and LWE [HW15, OPWW15].

Definition 5 (Succinctness and efficiency of SBB). *A SSB scheme $\Pi_{\mathsf{SSB}} = (\mathsf{Setup}, \mathsf{Hash}, \mathsf{Open}, \mathsf{Verify})$ is succinct and efficient if*

Succinctness. *The output length ℓ_{out}, the opening length ℓ_{open}, and the size of hk (output by $\mathsf{Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, N, i)$ are bounded by $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$.*

Efficiency. *The running times of Setup, Hash, and Open are bounded by $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$, $N \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$, and $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log N)$, respectively.*

Pseudorandom Generators. Let $\ell_{\mathsf{in}} = \ell_{\mathsf{in}}(\lambda)$, $\ell_{\mathsf{out}} = \ell_{\mathsf{out}}(\lambda)$, and $\mathsf{G} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ be two polynomials (in the security parameters) such that $\ell_{\mathsf{in}}(\lambda) < \ell_{\mathsf{out}}(\lambda)$ and an efficiently computable function G, respectively. We say that G is a pseudorandom generator (PRG) if $\mathsf{G}(s)$ and $y \leftarrow_\$ \{0,1\}^{\ell_{\mathsf{out}}}$ are computationally indistinguishable whenever $s \leftarrow_\$ \{0,1\}^{\ell_{\mathsf{in}}}$.

Definition 6 (Pseudorandomness). *A PRG $\mathsf{G} : \{0,1\}^{\ell_{\mathsf{in}}} \to \{0,1\}^{\ell_{\mathsf{out}}}$ is secure if for every PPT adversary D we have that*

$$\left| \mathbb{P}\left[\mathsf{D}(1^\lambda, \mathsf{G}(s)) = 1\right] - \mathbb{P}\left[\mathsf{D}(1^\lambda, y) = 1\right] \right| \leq \mathsf{negl}(\lambda),$$

where $s \leftarrow_\$ \{0,1\}^{\ell_{\mathsf{in}}}$ and $y \leftarrow_\$ \{0,1\}^{\ell_{\mathsf{out}}}$.

5 (Slotted) Registered Functional Encryption

We provide the formal definition of RFE in Section 5.1. In Section 5.2, we formalize the slotted flavor of RFE (that is the main building block to construct RFE). In Appendix B, we give the transformation to convert slotted RFE into RFE. The transformation leverages the same technique of Hohenberger *et al.* [HLWW22]. Our slotted RFE schemes for inner-product predicates and general functions are presented in Sections 6 and 7 respectively.

5.1 Registered Functional Encryption

We focus on RFE supporting a function space $\mathcal{F}$ of exponential size. An RFE scheme with message space $\mathcal{M}$ and function space $\mathcal{F} = \{f_i : \mathcal{M} \to \mathcal{Y}\}$ is composed of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda, |\mathcal{F}|)$**:** On input the security parameter 1^λ, the size parameter $|\mathcal{F}|$ (in binary) of the function space $\mathcal{F}$, the randomized setup algorithm outputs a common reference string crs.

$\mathsf{KGen}(\mathsf{crs}, \alpha)$**:** On input the common reference string crs and a (possibly empty) state α, the randomized key-generation algorithm outputs a public key pk and a secret key sk.

$\mathsf{RegPK}(\mathsf{crs}, \alpha, \mathsf{pk}, f)$**:** On input the common reference string crs, a (possibly empty) state α, a public key pk, and a function $f \in \mathcal{F}$, the deterministic registration algorithm outputs a master public key mpk and a new state α'.

$\mathsf{Enc}(\mathsf{mpk}, m)$**:** On input the master public key mpk and a message $m \in \mathcal{M}$, the randomized encryption algorithm outputs a ciphertext c.

$\mathsf{Update}(\mathsf{crs}, \alpha, \mathsf{pk})$**:** On input the common reference string crs, a state α, and a public key pk, the deterministic update algorithm outputs an helper decryption key hsk.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, c)$**:** On input a secret key sk, an helper decryption key hsk, and a ciphertext c, the deterministic decryption algorithm outputs a message $m \in \mathcal{Y} \cup \{\bot, \mathsf{getUpdate}\}$.

Correctness, compactness, and efficiency. An RFE scheme must be *correct*, i.e., an honest user, which has registered its public key under a function $f \in \mathcal{F}$, will be able to decrypt all future ciphertexts, obtaining $f(m)$. In addition, RFE must satisfy some efficiency requirements defined over the following aspects: (i) size of crs, (ii) KGen's running time, (iii) RegPK's running time, (iv) size of mpk, (v) size of hsk, and (vi) maximum number of updates that each user needs to receive during the lifetime of the system.[10] Optimally, each of these requirements should be bounded by $\mathsf{poly}(\lambda, \log L)$, where L represents the number of users currently registered in the system. Moreover, all the above properties (i.e., correctness and efficiency requirements) must hold even in the presence of an adversary that register arbitrary (e.g., malformed) public keys.

Definition 7 ((Perfect) Correctness of RFE). *We say an RFE scheme* $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RegPK}, \mathsf{Enc}, \mathsf{Update}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *is correct (resp. perfectly correct) if for every unbounded adversary* A *making at most a polynomial number of queries, we have:*

$$\mathbb{P}\Big[\mathbf{Game}^{\mathsf{corr\text{-}rfe}}_{\Pi_{\mathsf{RFE}},\mathsf{A}}(\lambda) = 1\Big] \le \mathsf{negl}(\lambda) \qquad \Big(\textit{resp. } \mathbb{P}\Big[\mathbf{Game}^{\mathsf{corr\text{-}rfe}}_{\Pi_{\mathsf{RFE}},\mathsf{A}}(\lambda) = 1\Big] = 0\Big),$$

where experiment $\mathbf{Game}^{\mathsf{corr\text{-}rfe}}_{\Pi_{\mathsf{RFE}},\mathsf{A}}(\lambda)$ *is defined as follows:*

- **Setup phase:** *The challenger computes* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, |\mathcal{F}|)$ *and initializes both the state* $\alpha = \bot$ *and the initial master public key* $\mathsf{mpk}_0 = \bot$*. Also, the challenger initializes three counters* $\mathsf{ctr}_{\mathsf{reg}} = 0$, $\mathsf{ctr}_{\mathsf{enc}} = 0$, $\mathsf{ctr}^*_{\mathsf{reg}} = \bot$ *to keep track of the number of registration queries, the number of encryption queries, and the index of the target key, respectively. Also, it sets* $\mathsf{out} = 0$ *(this variable defines the output of the experiment). Finally, the challenger sends* crs *to the adversary* A*.*
- **Query phase:** *The adversary* A *can submit the following queries to the challenger:*
  - **Register non-target key query:** A *sends a public key* pk *and a function* $f \in \mathcal{F}$ *to the challenger which proceeds as follows:*
    - *It increments* $\mathsf{ctr}_{\mathsf{reg}} = \mathsf{ctr}_{\mathsf{reg}} + 1$ *and computes* $(\mathsf{mpk}_{\mathsf{ctr}_{\mathsf{reg}}}, \alpha') = \mathsf{RegPK}(\mathsf{crs}, \alpha, \mathsf{pk}, f)$*.*
    - *Finally, it updates* $\alpha = \alpha'$ *and sends* $(\mathsf{ctr}_{\mathsf{reg}}, \mathsf{mpk}_{\mathsf{ctr}_{\mathsf{reg}}}, \alpha)$ *to* A*.*
  - **Register target key query:** A *sends a target function* $f^* \in \mathcal{F}$ *to the challenger. If* $\mathsf{ctr}^*_{\mathsf{reg}} \neq \bot$ *(i.e., the adversary has already submitted a target key query), the challenger returns* $\bot$*. Otherwise, it proceeds as follows:*
    - *It increments* $\mathsf{ctr}_{\mathsf{reg}} = \mathsf{ctr}_{\mathsf{reg}} + 1$*, and computes* $(\mathsf{pk}^*, \mathsf{sk}^*) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{crs}, \alpha)$ *and* $(\mathsf{mpk}_{\mathsf{ctr}_{\mathsf{reg}}}, \alpha') = \mathsf{RegPK}(\mathsf{crs}, \alpha, \mathsf{pk}^*, f^*)$*.*

[10] Following previous work, we measure the running times of algorithms in the RAM model of computation. In such a model, the running time of an algorithm can be sublinear in the size of its inputs.

– *It updates* $\alpha = \alpha'$, $\mathsf{ctr}^*_{\mathsf{reg}} = \mathsf{ctr}_{\mathsf{reg}}$ *and computes* $\mathsf{hsk}^* = \mathsf{Update}(\mathsf{crs}, \alpha, \mathsf{pk}^*)$.
– *Finally, the challenger sends* $(\mathsf{ctr}_{\mathsf{reg}}, \mathsf{mpk}_{\mathsf{ctr}_{\mathsf{reg}}}, \alpha, \mathsf{pk}^*, \mathsf{hsk}^*, \mathsf{sk}^*)$ *to* A.

– **Encryption query:** *The adversary* A *chooses an index* i *of a public key such that* $\mathsf{ctr}^*_{\mathsf{reg}} \leq i \leq \mathsf{ctr}_{\mathsf{reg}}$, *and a message* $m_{\mathsf{ctr}_{\mathsf{enc}}} \in \mathcal{M}$. *If* $\mathsf{ctr}^*_{\mathsf{reg}} = \perp$, *the challenger returns* $\perp$. *Otherwise, the challenger sets* $\mathsf{ctr}_{\mathsf{enc}} = \mathsf{ctr}_{\mathsf{enc}} + 1$ *and computes* $c_{\mathsf{ctr}_{\mathsf{enc}}} \leftarrow\!\$\ \mathsf{Enc}(\mathsf{mpk}_i, m_{\mathsf{ctr}_{\mathsf{enc}}})$. *Finally, it returns* $(\mathsf{ctr}_{\mathsf{enc}}, c_{\mathsf{ctr}_{\mathsf{enc}}})$ *to* A.

– **Decryption query:** *The adversary* A *selects an index* $j \in [\mathsf{ctr}_{\mathsf{reg}}]$. *The challenger computes* $y_j = \mathsf{Dec}(\mathsf{sk}^*, \mathsf{hsk}^*, c_j)$. *If* $y_j = \mathsf{getUpdate}$, *it updates the helper decryption key* $\mathsf{hsk}^* = \mathsf{Update}(\mathsf{crs}, \alpha, \mathsf{pk}^*)$ *and recompute* $y_j = \mathsf{Dec}(\mathsf{sk}^*, \mathsf{hsk}^*, c_j)$. *If* $y_j \neq f^*(m_j)$, *the challenger sets* $\mathsf{out} = 1$ *(i.e., the adversary manages to break correctness).*

- **End phase:** *After the adversary* A *has finished making queries, the challenger returns* out *as the output of the experiment.*

Definition 8 (Compactness and efficiency of RFE). *We say an RFE scheme* $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RegPK}, \mathsf{Enc}, \mathsf{Update}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *is* $(t_{\mathsf{crs}}, t_{\mathsf{mpk}}, t_{\mathsf{hsk}})$*-compact and* $(t_{\mathsf{KGen}}, t_{\mathsf{RegPK}}, t_{\mathsf{num}}, t_{\mathsf{Update}})$*-efficient if for every unbounded adversary* A *making at most a polynomial number of queries, the following conditions hold in each step of the execution of experiment* $\mathbf{Game}^{\mathsf{corr\text{-}rfe}}_{\Pi_{\mathsf{RFE}},\mathsf{A}}(\lambda)$:

$(t_{\mathsf{crs}}, t_{\mathsf{mpk}}, t_{\mathsf{hsk}})$**-compactness.**
- t_{crs}**-compact** crs**:** *The size of the common reference string is bounded by* t_{crs}.
- t_{mpk}**-compact** mpk**:** *The size of the each master public key is bounded by* t_{mpk}.
- t_{hsk}**-compact** hsk**:** *The size of the each helper decryption key is bounded by* t_{hsk}.

$(t_{\mathsf{KGen}}, t_{\mathsf{RegPK}}, t_{\mathsf{num}}, t_{\mathsf{Update}})$**-efficiency.**
- t_{KGen}**-efficient** KGen**:** *The key-generation (worst-case) running time is bounded by* t_{KGen}.
- t_{RegPK}**-efficient** RegPK**:** *The registration (worst-case) running time is bounded by* t_{RegPK}.
- $(t_{\mathsf{num}}, t_{\mathsf{Update}})$**-efficient** Update**:** *The challenger executes* Update *at most (worst-case)* t_{num} *times and each invocation runs in time (worst-case)* t_{Update}.

The running times of the above algorithms are in the RAM model of computation.

Security. Security of RFE is intuitive: An adversary cannot distinguish between $\mathsf{Enc}(\mathsf{mpk}, m_0)$ and $\mathsf{Enc}(\mathsf{mpk}, m_1)$ if it holds secret keys, registered to functions $f_1, \ldots, f_n$, such that $f_i(m_0) = f_1(m_1)$ for $i \in [n]$. This is formalized by an experiment in which the adversary can register honest users (whose secret keys are kept secret) or register corrupted users (whose public keys can be arbitrarily and maliciously chosen by the adversary).

Definition 9 (Security of RFE). *An RFE scheme* $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RegPK}, \mathsf{Enc}, \mathsf{Update}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *is secure if for every PPT valid adversary* A, *we have:*

$$\left| \mathbb{P}\Big[\mathbf{Game}^{\mathsf{rfe}}_{\Pi_{\mathsf{RFE}},\mathsf{A}}(\lambda, 0) = 1\Big] - \mathbb{P}\Big[\mathbf{Game}^{\mathsf{rfe}}_{\Pi_{\mathsf{RFE}},\mathsf{A}}(\lambda, 1) = 1\Big] \right| \leq \mathsf{negl}(\lambda),$$

where the experiment $\mathbf{Game}^{\mathsf{rfe}}_{\Pi_{\mathsf{RFE}},\mathsf{A}}(\lambda, b)$ *is defined as follows:*

- **Setup phase:** *The challenger computes* $\mathsf{crs} \leftarrow\!\$\ \mathsf{Setup}(1^\lambda, |\mathcal{F}|)$ *and initializes both the state* $\alpha = \perp$ *and the master public key* $\mathsf{mpk} = \perp$. *Also, it initializes a counter* $\mathsf{ctr} = 0$ *(for the number of honest registration queries submitted by the adversary), a set of corrupted public keys* $\mathcal{C} = \emptyset$, *and a dictionary* $\mathsf{D} = \emptyset$ *(storing the mapping between registered public keys and their corresponding functions). Finally, the challenger sends* crs *to the adversary* A.
- **Query phase:** *The adversary* A *can submit the following queries:*

– **Register corrupted key query:** A *sends a public key* pk *and a function* $f \in \mathcal{F}$ *to the challenger which proceeds as follows:*
– *It computes* $(\mathsf{mpk}', \alpha') = \mathsf{RegPK}(\mathsf{crs}, \alpha, \mathsf{pk}, f)$.
– *It updates* $\alpha = \alpha'$, $\mathsf{mpk} = \mathsf{mpk}'$, $\mathcal{C} = \mathcal{C} \cup \{\mathsf{pk}\}$, *and* $\mathsf{D}[\mathsf{pk}] = \mathsf{D}[\mathsf{pk}] \cup \{f\}$.
– *Finally, it returns* (α, mpk) *to* A.

– **Register honest key query:** A *sends a target function* $f \in \mathcal{F}$ *which proceeds as follows:*

– *It sets* $\mathsf{ctr} = \mathsf{ctr} + 1$ *and computes* $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{crs}, \alpha)$.
– *It registers the key* $(\mathsf{pk}_{\mathsf{ctr}}, f)$ *by executing* $(\mathsf{mpk}', \alpha') = \mathsf{RegPK}(\mathsf{crs}, \alpha, \mathsf{pk}_{\mathsf{ctr}}, f)$.
– *It updates* $\alpha = \alpha'$, $\mathsf{mpk} = \mathsf{mpk}'$, *and* $\mathsf{D}[\mathsf{pk}_{\mathsf{ctr}}] = \mathsf{D}[\mathsf{pk}_{\mathsf{ctr}}] \cup \{f\}$.
– *Finally, it returns* $(\mathsf{ctr}, \alpha, \mathsf{mpk}, \mathsf{pk}_{\mathsf{ctr}})$ *to* A.

– **Corrupt honest key:** A *selects an index* $i \in [\mathsf{ctr}]$. *The challenger updates* $\mathcal{C} = \mathcal{C} \cup \{\mathsf{pk}_i\}$ *and returns* sk_i *to* A *where* $(\mathsf{pk}_i, \mathsf{sk}_i)$ *is the i-th public and secret key generated during the i-th honest registration query.*

- **Challenge phase:** A *chooses two messages* (m_0^*, m_1^*). *The challenger returns* $c^* \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{mpk}, m_b^*)$.
- **Output phase:** A *returns a bit* b' *which is also the output of the experiment.*

An adversary A *is considered valid if* $f(m_0^*) = f(m_1^*)$ *for every* $f \in \{f \in D[\mathsf{pk}] | \mathsf{pk} \in \mathcal{C}\}$ *(i.e., for every function, whose registered secret key is known by the adversary, we have the same output).*

Remark 1 (Exponential function spaces). In this paper, we focus on (slotted) RFE (see Section 5.2 for the definition of slotted RFE) supporting a class of functions of exponential size. In particular, our iO-based construction is able to handle any function of bounded size (e.g., $\mathsf{poly}(\lambda)$) whereas our pairing-based RIPE construction supports vectors from $\mathbf{x} \in \mathbb{Z}_q^{n^+}$ where q is a λ-bit prime, i.e., $\mathbf{x}$ is an n-length vector whose elements comes from $\mathbb{Z}_q^+$. We highlight that this differs from the pairing-based RABE construction of [HLWW22], that only supports small (i.e., $\mathsf{poly}(\lambda)$) attribute universe.

Remark 2 (Bounded vs. Unbounded number of users). The setup algorithm of RFE does not take as input a bound on the maximum number of registered users, i.e., the crs will allow the KC to handle any number of users. Our iO-based construction achieves this notion. Through the paper, we also consider the notion of *bounded* RFE in which there is a bound on the number of registered users (this will apply to our pairing-based RIPE construction). In the case of bounded RFE, we abuse notation and denote by L the a-priori bounded number of users (recall that, in the case of unbounded RFE, L instead denotes the current number of registered users). Here, the setup algorithm takes as input L in unary (i.e., $\mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|)$). Analogously, during the execution of the security experiment (Definition 9) for bounded RFE, the adversary can specify the bound 1^L and, in turn, submit at most L registration queries (the challenger will reply with $\perp$ after L queries are submitted).

Remark 3 (On the security of RFE without post-challenge queries). Definition 9 does not allow the adversary to submit queries after the challenge phase. For the case of RABE, Hohenberger *et al.* [HLWW22] showed that security without post-challenge queries implies security with post-challenge queries. Intuitively, this is because the deterministic RegPK and Update algorithms are publicly computable (they do not require knowledge of any secret) and their behavior can be simulated by the adversary. The exact same result holds for RFE. This follows by using the same technique of [HLWW22, Remark 4.5 and Lemma 4.10] except that the validity of the RABE adversary (i.e., $f(x) = 0$ where f is the policy and x are the attributes) is replaced with the validity of the RFE adversary (i.e., $f(m_0) = f(m_1)$). An identical argument applies to slotted RFE (Definition 13). We refer the reader to [HLWW22, Remark 4.5 and Lemma 4.10] for more details.

5.2 Slotted Registered Functional Encryption

We now formalize the notion of slotted RFE (for function spaces $\mathcal{F}$ of exponential size). Formally, a slotted RFE with message space $\mathcal{M}$ and function space $\mathcal{F} = \{f_i : \mathcal{M} \to \mathcal{Y}\}$ is composed of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|)$**:** On input the security parameter 1^λ, the slot parameter 1^L, and the size $|\mathcal{F}|$ (in binary) of the function space $\mathcal{F}$, the randomized setup algorithm outputs a common reference string crs.

$\mathsf{KGen}(\mathsf{crs}, i)$**:** On input the common reference string crs and a slot index $i \in [L]$, the randomized key generation algorithm outputs a public key pk_i and a secret key sk_i.

$\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$**:** On input the common reference string crs, a slot index $i \in [L]$, and a public key pk_i, the deterministic validation algorithm outputs a decision bit $b \in \{0, 1\}$.

$\mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i, f_i))_{i\in[L]})$**:** On input the common reference string crs and L pairs $(\mathsf{pk}_1, f_1), \ldots, (\mathsf{pk}_L, f_L)$ each composed of a public key pk_i and its corresponding function $f_i \in \mathcal{F}$, the deterministic aggregation algorithm outputs a master public key mpk and L helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$.

$\mathsf{Enc}(\mathsf{mpk}, m)$**:** On input the master public key mpk and a message $m \in \mathcal{M}$, the randomized encryption algorithm outputs a ciphertext c.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, c)$**:** On input a secret key sk, an helper decryption key hsk, and a ciphertext c, the deterministic decryption algorithm outputs a message $m \in \mathcal{Y} \cup \{\bot\}$.

Completeness, correctness, and efficiency. Completeness of slotted RFE says that honestly generated public keys for a slot index $i \in [L]$ are valid with respect to the same slot i, i.e., $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$. Similarly, correctness says that honest ciphertexts correctly decrypt under honestly generated and aggregated keys. Finally, we extend the efficiency requirements of RFE (Definition 8) to the slotted setting.

Definition 10 (Completeness of slotted RFE). *A slotted RFE scheme* $\Pi_{\mathsf{sRFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *is complete if,* $\forall \lambda \in \mathbb{N}$, $\forall L \in \mathbb{N}$, *and* $\forall i \in [L]$, *we have:*

$$\mathbb{P}\Big[\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1 \Big| \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|), (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{crs}, i)\Big] = 1$$

Definition 11 (Perfect correctness of slotted RFE). *A slotted RFE scheme* $\Pi_{\mathsf{sRFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *is perfectly correct if,* $\forall \lambda \in \mathbb{N}$, $\forall L \in \mathbb{N}$, $\forall i \in [L]$, $\forall \mathsf{crs}$ *output by* $\mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|)$, $\forall (\mathsf{pk}_i, \mathsf{sk}_i)$ *output by* $\mathsf{KGen}(\mathsf{crs}, i)$, $\forall$ *collection of public key* $\{\mathsf{pk}_j\}_{j\in[L]\setminus\{i\}}$ *such that* $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$, $\forall m \in \mathcal{M}$, $\forall f_1, \ldots, f_L \in \mathcal{F}$, *we have:*

$$\mathbb{P}\left[\mathsf{Dec}(\mathsf{sk}_i, \mathsf{hsk}_i, c) = f_i(m) \,\middle|\, \begin{array}{c}(\mathsf{msk}, (\mathsf{hsk}_i)_{i\in[L]}) = \mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_j, f_j))_{j\in[L]}),\\ c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{mpk}, m)\end{array}\right] = 1$$

Definition 12 (Compactness and efficiency of slotted RFE). *We say an RFE scheme* $\Pi_{\mathsf{sRFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *is* $(t_{\mathsf{crs}}, t_{\mathsf{mpk}}, t_{\mathsf{hsk}})$*-compact and* $(t_{\mathsf{KGen}}, t_{\mathsf{IsValid}}, t_{\mathsf{Aggr}})$*-efficient if the following conditions hold:*

$(t_{\mathsf{crs}}, t_{\mathsf{mpk}}, t_{\mathsf{hsk}})$**-compactness.** *This is identical to that of Definition 8.*
$(t_{\mathsf{KGen}}, t_{\mathsf{IsValid}}, t_{\mathsf{Aggr}})$**-efficiency.**

- t_{KGen}**-efficient** KGen**:** *This is identical to that of Definition 8.*
- t_{IsValid}**-efficient** $\mathsf{IsValid}$**:** *The validation (worst-case) running time is bounded by* t_{IsValid}.
- t_{Aggr}**-efficient** Aggr**:** *The aggregation (worst-case) running time is bounded by* t_{Aggr}.

The (worst-case) running times of the above algorithms are measured in the RAM model of computation.

Security. A secure slotted RFE guarantees that $\mathsf{Enc}(\mathsf{mpk}, m_0^*)$ and $\mathsf{Enc}(\mathsf{mpk}, m_1^*)$ are computationally indistinguishable where $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i^*, f_i^*))_{i\in[L]})$ and $(\mathsf{pk}_1^*, f_1^*), \ldots, (\mathsf{pk}_L^*, f_L^*)$ are chosen by the adversary. This must hold even if the adversary knows some of the secret keys, each corresponding to one (or more) of the aggregated public keys. Naturally, to rule out trivial attacks, we require the adversary to be valid, i.e., for each sk_i^* known by the adversary (corresponding to i-th aggregated pk_i^*), we have $f_i^*(m_0^*) = f_i^*(m_1^*)$ where f_i^* is the function associated to pk_i^*.

Definition 13 (Security of slotted RFE). *A slotted RFE scheme* $\Pi_{\mathsf{sRFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *is secure if for every PPT valid adversary* A, *we have:*

$$\left|\mathbb{P}\Big[\mathbf{Game}^{\mathsf{slot\text{-}rfe}}_{\Pi_{\mathsf{sRFE}},\mathsf{A}}(\lambda, 0) = 1\Big] - \mathbb{P}\Big[\mathbf{Game}^{\mathsf{slot\text{-}rfe}}_{\Pi_{\mathsf{sRFE}},\mathsf{A}}(\lambda, 1) = 1\Big]\right| \le \mathsf{negl}(\lambda),$$

where the experiment $\mathbf{Game}^{\mathsf{slot\text{-}rfe}}_{\Pi_{\mathsf{sRFE}},\mathsf{A}}(\lambda, b)$ *is defined in the following way:*

- **Setup phase:** *The adversary* A *sends a slot parameter* 1^L *to the challenger. The challenger initializes a counter* $\mathsf{ctr} = 0$*, a dictionary* $\mathsf{D} = \emptyset$*, and a set of corrupted slot indexes* $\mathcal{C} = \emptyset$*. Finally, it sends* crs *to* A *where* $\mathsf{crs} \leftarrow\$ \mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|)$.
- **Query phase:** *The adversary* A *can submit queries to the following oracles:*
  - **Honest key-generation query:** A *sends* $i \in [L]$*. The challenger computes* $\mathsf{ctr} = \mathsf{ctr}+1$, $(\mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}}) \leftarrow\$ \mathsf{KGen}(\mathsf{crs}, i)$*, and sets* $\mathsf{D}[\mathsf{ctr}] = (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$*. Finally, it returns* $(\mathsf{ctr}, \mathsf{pk}_{\mathsf{ctr}})$ *to* A.
  - **Corruption query:** A *sends* $j \in [\mathsf{ctr}]$*. The challenger returns* sk' *where* $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[j]$*. Let* $\mathcal{Q}_{\mathsf{Corr}}$ *be the set of corruption queries submitted by the adversary.*
- **Challenge phase:** A *sends the challenge* $((c_i^*, f_i^*, \mathsf{pk}_i^*)_{i\in[L]}, m_0^*, m_1^*)$ *where* $c_i^* \in [\mathsf{ctr}] \cup \{\perp\}$.[11] *Then, for every* $i \in [L]$*, it proceeds as follows:*
  - *If* $c_i^* \in [\mathsf{ctr}]$*, the challenger retrieves* $(i', \mathsf{pk}', \mathsf{sk}') = \mathsf{D}[c_i^*]$*. If* $i' = i$*, it sets* $\mathsf{pk}_i = \mathsf{pk}'$*. In addition, if* $c_i^* \in \mathcal{Q}_{\mathsf{Corr}}$*, the challenger updates* $\mathcal{C} = \mathcal{C} \cup \{i\}$*. Otherwise, if* $i' \neq i$*, the challenger aborts.*
  - *If* $c_i^* = \perp$*, the challenger checks the validity of* pk_i^**. If* $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*) = 0$*, it aborts; otherwise (i.e.,* $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*) = 1$*), the challenger sets* $\mathsf{pk}_i = \mathsf{pk}_i^*$ *and updates* $\mathcal{C} = \mathcal{C} \cup \{i\}$.

  Finally, the challenger sends $c^* \leftarrow\$ \mathsf{Enc}(\mathsf{mpk}, m_b^*)$ *to the adversary where* $(\mathsf{mpk}, \mathsf{hsk}_1, \ldots, \mathsf{hsk}_L) = \mathsf{Aggr}(\mathsf{crs}, (\mathsf{pk}_1, f_1^*), \ldots, (\mathsf{pk}_L, f_L^*))$.[12]
- **Output phase:** *The adversary* A *outputs* $b' \in \{0, 1\}$ *which is also the output of the experiment.*

An adversary A *is considered valid if* $f_i^*(m_0^*) = f_i^*(m_1^*)$ *for every* $i \in \mathcal{C}$.

As discussed in Remark 3, security without post-challenge queries implies security with post-challenge queries. This is because Aggr is deterministic and does not require any secret. Hence, an adversary can simulate the post-challenge queries itself.

6 Slotted Registered IPE from Prime-Order Pairings

In this section, we provide our slotted RIPE scheme built on prime-order bilinear groups. We assume the message space $\mathcal{M} = \mathbb{G}_\mathsf{T}$ and the attribute universe $\mathcal{U} = \mathbb{Z}_q^{n^+}$ for some $n = n(\lambda)$ that is an a-priori fixed polynomial in the security parameter λ. Attributes are thus represented as vectors $\mathbf{y} = (y_1, \ldots, y_n) \in \mathcal{U}$ and vectors $\mathbf{x} \in \mathcal{U}$ represent predicates $f_\mathbf{x}$ that outputs 1 when $\langle \mathbf{x}, \mathbf{y} \rangle = 0$. Note that it is essentially an RFE scheme with a message space $\mathcal{M}' = \mathcal{M} \times \mathcal{U}$ and function space (of exponential size) $\mathcal{F} = \{f_\mathbf{x} : \mathcal{M}' \to \mathcal{M}\}$, where $f_\mathbf{x}$ is defined as follows:

$$f_\mathbf{x}(m, \mathbf{y}) = \begin{cases} m & \text{if } \langle \mathbf{x}, \mathbf{y} \rangle = 0 \\ \perp & \text{otherwise} \end{cases} \tag{2}$$

for $\mathbf{x}, \mathbf{y} \in \mathbb{Z}_q^{n^+}$ and $m \in \mathbb{G}_\mathsf{T}$. Our slotted RIPE supports an a-priori fixed number of slots $L = L(\lambda)$, i.e., the scheme supports a bounded number of slots. For completeness, we provide some notations on bilinear groups and state the slotted RIPE definitions in Appendix A. Below, we describe our formal scheme.

Construction 1 *The slotted RIPE scheme* $\Pi_{\mathsf{sRIPE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M} = \mathbb{G}_\mathsf{T}$ *and attribute space* $\mathcal{U} = \mathbb{Z}_q^{n^+}$ *is as follows:*

$\mathsf{Setup}(1^\lambda, 1^n, 1^L)$**:** *On input the security parameter* λ*, the attribute size* n *and the number of slots* L*, compute* $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, g_1, g_2, e) \leftarrow\$ \mathsf{GroupGen}(1^\lambda)$ *and generate the common reference string as follows.*

1. *Sample* $\alpha, \beta, \gamma \leftarrow\$ \mathbb{Z}_q^+$ *and set* $h = g_1^\beta, Z = e(g_1, g_2)^\alpha, \Gamma = g_1^\gamma, n' = n + 1$.
2. *For each index* $i \in [0, L]$*, do the following:*
   (a) *for each* $w \in [n']$*, sample* $u_{w,i} \leftarrow\$ \mathbb{Z}_q$ *and set* $U_{w,i} = g_1^{u_{w,i}}$.

[11] If $c_i^* \neq \perp$, then pk_i refers to a public key generated by the challenger. On the other hand, if $c_i^* = \perp$, then pk_i is an arbitrary public key chosen by A.

[12] Note that the challenger does not send the master public key mpk and the helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$ to the adversary A since Aggr is deterministic, i.e., A can compute both mpk and $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$ by itself.

(b) for a slot index $i > 0$, sample $t_i \leftarrow\$ \mathbb{Z}_q$ and set $A_i = g_2^{t_i}, B_i = g_2^{\alpha} \cdot A_i^{\beta}$.

(c) for a slot index $i > 0$, $\forall w \in [n'], j \in [0, L] \setminus \{i\}$, set $W_{i,j,w} = A_i^{u_{w,j}/\gamma}$.

3. *Sample $\widetilde{\mathbf{x}}_0 = (\widetilde{x}_{1,0}, \ldots, \widetilde{x}_{n,0}, \widetilde{r}_0) \leftarrow\$ \mathbb{Z}_q^{n'^+}$. Set $\mathsf{sk}_0 = \widetilde{\mathbf{x}}_0$ and*

$$T_0 = \left(\prod_{w=1}^{n} U_{w,0}^{-\widetilde{x}_{w,0}} \right) \cdot U_{n',0}^{-\widetilde{r}_0} \quad , \quad \widetilde{W}_{i,0} = \left(\prod_{w=1}^{n} W_{i,0,w}^{\widetilde{x}_{w,0}} \right) \cdot W_{i,0,n'}^{\widetilde{r}_0}, \quad \forall i \in [L].$$

Also, set $\mathsf{pk}_0 = \left(T_0, \left\{ \widetilde{W}_{i,0} \right\}_{i \in [L]} \right)$.

Finally, output the common reference string

$$\mathsf{crs} = (\mathcal{G}, Z, h, \Gamma, \{A_i, B_i\}_{i \in [L]}, \{\{U_{w,i}\}_{i \in [0,L]}, \{W_{i,j,w}\}_{i \in [L], j \in [0,L] \setminus \{i\}}\}_{w \in [n']}, \mathsf{pk}_0)$$

Remark: Note that in steps 2.(a), 2.(c) and 3 above, we generate additional elements $\{U_{w,0}\}_{w \in [n']}$, $\{W_{i,0,w}\}_{i \in [L], w \in [n']}$ and pk_0 respectively to introduce a dummy slot "0" in the scheme. However, pre-registering a dummy but honestly generated key pk_0 in this slot does not impact the security definition in any way. This is because the associated secret key sk_0 corresponding to the random vector $\widetilde{\mathbf{x}}_0$ is thrown away once the one-time setup is executed and thus, is never released in the clear. We emphasize that this modification is done only for a simpler analysis of the security proof in the GGM. In particular, it is easy to see that the scheme still works without adding the dummy key pk_0, i.e., without the additional changes made (in order to add $\widetilde{\mathbf{x}}_0$) in the Setup, KGen and Aggr algorithms.

$\mathsf{KGen}(\mathsf{crs}, i)$: *On input the common reference string crs and a slot index $i \in [L]$, do the following.*

1. *Parse the common reference string*

$$\mathsf{crs} = \Big(\mathcal{G}, Z, h, \Gamma, \{A_i, B_i\}_{i \in [L]}, \left\{ \{U_{w,i}\}_{i \in [0,L]}, \{W_{i,j,w}\}_{i \in [L], j \in [0,L] \setminus \{i\}} \right\}_{w \in [n']}, \mathsf{pk}_0 \Big).$$

2. *Sample $\widetilde{r}_i \leftarrow\$ \mathbb{Z}_q^+$ and pick elements $U_{n',i}$ and $\{W_{j,i,n'}\}_{j \in [L] \setminus \{i\}}$ from crs.*
3. *Compute $T_i = U_{n',i}^{-\widetilde{r}_i}$ and $\widetilde{W}_{j,i} = W_{j,i,n'}^{\widetilde{r}_i}, \forall j \in [L] \setminus \{i\}$.*
4. *Output $\mathsf{pk}_i = \left(T_i, \{\widetilde{W}_{j,i}\}_{j \in [L] \setminus \{i\}} \right)$ and $\mathsf{sk}_i = \widetilde{r}_i$.*

$\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$: *On input the common reference string crs, a slot index $i \in [L]$ and a purported public key $\mathsf{pk}_i = \left(T_i, \{\widetilde{W}_{j,i}\}_{j \in [L] \setminus \{i\}} \right)$, the key-validation algorithm first affirms that each of the components in pk_i is a valid group element, namely: $\left(T_i \overset{?}{\in} \mathbb{G}_1 \setminus \{1_{\mathbb{G}_1}\} \quad \wedge \quad \widetilde{W}_{j,i} \overset{?}{\in} G_2 \setminus \{1_{\mathbb{G}_2}\}, \quad \forall j \in [L] \setminus \{i\} \right)$ where $1_{\mathbb{G}_\mathsf{t}}$ denotes the identity in $\mathbb{G}_\mathsf{t}$ for $\mathsf{t} \in [2]$. If the checks pass, it picks the elements $U_{n',i}$ and $\{W_{j,i,n'}\}_{j \in [L] \setminus \{i\}}$ from crs and checks further that*

$$e\left(T_i^{-1}, W_{j,i,n'}\right) \overset{?}{=} e\left(U_{n',i}, \widetilde{W}_{j,i}\right), \forall j \in [L] \setminus \{i\}.$$

If all checks pass, it outputs 1. Else, it outputs 0.

$\mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i, \mathbf{x}_i))_{i \in [L]})$: *On input the common reference string crs and a set of L public keys $\mathsf{pk}_i = \left(T_i, \{\widetilde{W}_{j,i}\}_{j \in [L] \setminus \{i\}} \right)$ together with vectors $\mathbf{x}_i = (x_{1,i}, \ldots, x_{n,i}) \in \mathbb{Z}_q^{n^+}$ (representing predicates $f_{\mathbf{x}_i}$), compute the following.*

1. *Parse the common reference string*

$$\mathsf{crs} = \Big(\mathcal{G}, Z, h, \Gamma, \{A_i, B_i\}_{i \in [L]}, \left\{ \{U_{w,i}\}_{i \in [0,L]}, \{W_{i,j,w}\}_{i \in [L], j \in [0,L] \setminus \{i\}} \right\}_{w \in [n']}, \mathsf{pk}_0 \Big).$$

2. *Fuse the predicate vector* $\mathbf{x}_i$ *into* pk_i *by updating each of its components as*

$$T_i = \left(\prod_{w=1}^{n} U_{w,i}^{-x_{w,i}}\right) \cdot T_i \quad , \quad \widetilde{W}_{j,i} = \left(\prod_{w=1}^{n} W_{j,i,w}^{x_{w,i}}\right) \cdot \widetilde{W}_{j,i}, \forall j \in [L] \setminus \{i\}$$

and set $\mathsf{pk}_i = \left(T_i, \left\{\widetilde{W}_{j,i}\right\}_{j\in[L]\setminus\{i\}}\right)$. *Further, parse* pk_0 *as follows:*

$$\mathsf{pk}_0 = \left(T_0, \left\{\widetilde{W}_{j,0}\right\}_{j\in[0,L]\setminus\{0\}}\right).$$

3. *For each* $w \in [n']$, *compute* $\widehat{U}_w = \prod_{i\in[0,L]} U_{w,i}$ *and* $\widehat{U}_{n'+1} = \prod_{i\in[0,L]} T_i$.
4. *Compute the cross-terms as follows. For each slot index* $i \in [L]$:
   (a) *for each* $w \in [n']$, *compute* $\widehat{W}_{w,i} = \prod_{j\in[0,L]\setminus\{i\}} W_{i,j,w}$.
   (b) *compute* $\widehat{W}_{n'+1,i} = \left(\prod_{j\in[0,L]\setminus\{i\}} \widetilde{W}_{i,j}\right)^{-1}$.
5. *Output the master public key and the slot-specific helper secret keys as* $\mathsf{mpk} = \left(\mathcal{G}, h, Z, \Gamma, \left\{\widehat{U}_w\right\}_{w\in[n'+1]}\right)$, *and*

$$\mathsf{hsk}_i = \left(\mathcal{G}, i, \mathbf{x}_i, A_i, B_i, \left\{\widehat{W}_{w,i}\right\}_{w\in[n'+1]}\right), \forall i \in [L].$$

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{y}, m)$: *On input the master public key* mpk, *a vector* $\mathbf{y} = (y_1, \ldots, y_n) \in \mathbb{Z}_q^{n^+}$ *(as an attribute) and a message* $m \in \mathbb{G}_\mathsf{T}$, *the ciphertext is computed as:*

1. *Parse* $\mathsf{mpk} = \left(\mathcal{G}, h, Z, \Gamma, \left\{\widehat{U}_w\right\}_{w\in[n'+1]}\right)$.
2. *Set* $\widetilde{\mathbf{y}} = (\mathbf{y}, 0, 0) \in \mathbb{Z}_q^{n'+1}$ *and sample* $s, r, z \leftarrow\!\!\$\, \mathbb{Z}_q^+$. *Also, parse* $\widetilde{\mathbf{y}} = (\widetilde{y}_1, \ldots, \widetilde{y}_{n'+1})$.
3. *Message embedding: set* $C_1 = m \cdot Z^s$ *and* $C_2 = g_1^s$.
4. *Attribute and Slot embedding: for each* $w \in [n'+1]$, *set* $C_{3,w} = h^{\widetilde{y}_w \cdot r + s} \cdot \widehat{U}_w^{-z}$. *Set* $C_4 = \Gamma^z$.
5. *Output the ciphertext* $c = (C_1, C_2, \{C_{3,w}\}_{w\in[n'+1]}, C_4)$.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, c)$: *Parse the input secret key* sk, *helper secret key* hsk *and ciphertext* c *as* $\mathsf{sk} = \widetilde{r}_i$, *and*

$$\mathsf{hsk} = \left(\mathcal{G}, i, \mathbf{x}_i, A_i, B_i, \left\{\widehat{W}_{w,i}\right\}_{w\in[n'+1]}\right), c = \left(C_1, C_2, \{C_{3,w}\}_{w\in[n'+1]}, C_4\right),$$

for some $i \in [L]$. *Let* $\widetilde{\mathbf{x}}_i = (\widetilde{x}_{1,i}, \ldots, \widetilde{x}_{n'+1,i}) = (\mathbf{x}_i, \widetilde{r}_i, 1) \in \mathbb{Z}_q^{n'+1}, X_i = \sum_{w=1}^{n'+1} \widetilde{x}_{w,i} \in \mathbb{Z}_q$. *Compute and output the following:*

$$\frac{C_1}{e(C_2, B_i)} \cdot \left[\prod_{w=1}^{n'+1} \left\{ e\left(C_{3,w}^{\widetilde{x}_{w,i}}, A_i\right) \cdot e\left(C_4, \widehat{W}_{w,i}^{\widetilde{x}_{w,i}}\right)\right\}\right]^{X_i^{-1}}.$$

Remark 4 (On Extended Functionalities). The above scheme can support constant degree polynomial evaluations, disjunctions, conjunctions, and evaluating CNF and DNF formulas following the transformations described in [KSW08].

Theorem 3 (Compactness and Efficiency of Construction 1). *The slotted RIPE scheme* Π_{sRIPE} *with message space* $\mathcal{M} = \mathbb{G}_\mathsf{T}$ *and attribute space* $\mathcal{U} = \mathbb{Z}_q^{n^+}$ *from Construction 1 satisfies the following properties:*

- $(n \cdot L^2 \cdot \mathsf{poly}(\lambda), n \cdot \mathsf{poly}(\lambda), n \cdot \mathsf{poly}(\lambda) + O(\log L))$*-compact, and*
- $(L \cdot \mathsf{poly}(\lambda), L \cdot \mathsf{poly}(\lambda), n \cdot L^2 \cdot \mathsf{poly}(\lambda))$*-efficient (Definition 12).*

Proof. Recall $n' = n + 1$. We demonstrate each property individually.

- $(n \cdot L^2 \cdot \mathsf{poly}(\lambda))$**-compact** crs**:** The common reference string crs consists the following elements. The description of group $\mathcal{G}$, which is of size $\mathsf{poly}(\lambda)$, the group elements Z, h, Γ group $\mathbb{G}_1$, where their sizes are also in $\mathsf{poly}(\lambda)$. The set of $\mathbb{G}_2$ elements $\{A_i, B_i\}_{i\in[L]}$, which is of size $L \cdot \mathsf{poly}(\lambda)$, the set $\{U_{w,i}\}_{i\in[0,L],w\in[n']}$ of $\mathbb{G}_1$ elements which is of size $n \cdot L \cdot \mathsf{poly}(\lambda)$, pk_0 and its helper secret keys which consists of $L+1$ group elements, of total size $L \cdot \mathsf{poly}(\lambda)$, and finally the largest part of the crs will be the set

$$\{W_{i,j,w}\}_{i\in[L],j\in[0,L]\setminus\{i\},w\in[n']} \text{ (of } \mathbb{G}_2 \text{ elements)}$$

  whose size will be $n \cdot L^2 \cdot \mathsf{poly}(\lambda)$. Hence, we have $|\mathsf{crs}| = n \cdot L^2 \cdot \mathsf{poly}(\lambda)$.
- $(n \cdot \mathsf{poly}(\lambda))$**-compact** mpk**:** The master public key mpk consists of elements $\mathcal{G}, h, z, \Gamma$ of size $\mathsf{poly}(\lambda)$, and $\left\{\widehat{U}_w\right\}_{w\in[n'+1]}$ of size $n \cdot \mathsf{poly}(\lambda)$.
- $(n \cdot \mathsf{poly}(\lambda) + O(\log L))$**-compact** hsk**:** Each helper decryption key hsk_i, consists of elements $\mathcal{G}, i, B_i, A_i, \mathbf{x}_i$, $\left\{\widehat{W}_{w,i}\right\}_{w\in[n'+1]}$ where $\mathcal{G}, B_i, A_i$ are of size $\mathsf{poly}(\lambda)$, i is of size $O(\log L)$, and $\mathbf{x}_i$ and $\left\{\widehat{W}_{w,i}\right\}_{w\in[n'+1]}$ of size $n \cdot \mathsf{poly}(\lambda)$.
- $(O(L) \cdot \mathsf{poly}(\lambda))$**-efficient** KGen**:** Note that we do not need to parse the full crs here. For a particular $i \in [L]$, we only pick the elements $U_{n',i}$ and $W_{j,i,n'}$ for all $j \in [L] \setminus \{i\}$, which can be done in total $O(L)$ time. The key generation algorithm has to perform $L-1$ exponentiations on the cross terms to create the $\widetilde{W}$ part of the public key, so this operation only takes linear time in L. The rest of the operations can be performed in constant time.
- $(L \cdot \mathsf{poly}(\lambda))$**-efficient** $\mathsf{IsValid}$**:** Note that we do not need to parse the full crs here. For a particular $i \in [L]$, we only pick the elements $U_{n',i}$ and $W_{j,i,n'}$ for all $j \in [L] \setminus \{i\}$, which can be done in total $O(L)$ time. Further, the validation algorithm simply computes and checks $L-1$ pairings. Assuming each pairing takes $\mathsf{poly}(\lambda)$ time, the total running time becomes $L \cdot \mathsf{poly}(\lambda)$.
- $(n \cdot L^2 \cdot \mathsf{poly}(\lambda))$**-efficient** Aggr**:** We analyze the running time of the Aggr algorithm by analyzing each step individually.
  - Step 1: The aggregation parses the crs which takes $n \cdot L^2 \cdot \mathsf{poly}(\lambda)$ time.
  - Step 2: In order to compute T_i, we need to compute $\prod_{w=1}^{n} U_{w,i}^{-x_{w,i}}$ for each user. This part takes $n \cdot L \cdot \mathsf{poly}(\lambda)$ time. Using the same logic, computing $\widetilde{W}_{j,i}$ for all users, also takes $n \cdot L^2 \cdot \mathsf{poly}(\lambda)$ time.
  - Step 3: This step takes $n \cdot L \cdot \mathsf{poly}(\lambda)$ time.
  - Step 4: Since we need to compute $\widehat{W}_{w,i}$, for each $w \in [n']$ and each index $i \in [L]$, this step takes $n \cdot L^2 \cdot \mathsf{poly}(\lambda)$.

  Above, steps 2, 4 dominate the runtime for Aggr which is $n \cdot L^2 \cdot \mathsf{poly}(\lambda)$.

Theorem 4 (Completeness of Construction 1). *The slotted RIPE scheme Π_{sRIPE} with message space $\mathcal{M} = \mathbb{G}_\mathsf{T}$ and attribute space $\mathcal{U} = \mathbb{Z}_q^{n^+}$ from Construction 1 is complete.*

Proof. Fix the security parameter λ, $n = n(\lambda)$, and $L = L(\lambda)$. Let $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^n, 1^L)$. Take any index $i \in [L]$ and let $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{crs}, i)$. Recall $\mathsf{pk}_i = \left(T_i, \{\widetilde{W}_{j,i}\}_{j\in[L]\setminus\{i\}}\right)$ and $\mathsf{sk}_i = \widetilde{r}_i \in \mathbb{Z}_q^+$, where

$$T_i = U_{n',i}^{-\widetilde{r}_i} \quad \text{and} \quad \widetilde{W}_{j,i} = W_{j,i,n'}^{\widetilde{r}_i}, \forall j \in [L] \setminus \{i\}$$

and $U_{n',i}$ and $\{W_{j,i,n'}\}_{j\in[L]\setminus\{i\}}$ are elements from the crs. The theorem follows by observing that $\forall j \in [L]\setminus\{i\}$ we have

$$e\left(T_i^{-1}, W_{j,i,n'}\right) = e\left(U_{n',i}^{\widetilde{r}_i}, W_{j,i,n'}\right) = e\left(U_{n',i}, W_{j,i,n'}^{\widetilde{r}_i}\right) = e\left(U_{n',i}, \widetilde{W}_{j,i}\right).$$

Theorem 5 (Perfect Correctness of Construction 1). *The slotted RIPE scheme Π_{sRIPE} with message space $\mathcal{M} = \mathbb{G}_\mathsf{T}$ and attribute space $\mathcal{U} = \mathbb{Z}_q^{n^+}$ from Construction 1 is perfectly correct.*

Proof. Fix some λ, attribute size $n = n(\lambda)$, a slot count $L = L(\lambda)$ and an index $i \in [L]$. Let $\mathsf{crs} \leftarrow_\$ \mathsf{Setup}(1^\lambda, 1^n, 1^L)$ and $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow_\$ \mathsf{KGen}(\mathsf{crs}, i)$ be defined as in the scheme from Construction 1. Take any set of public keys $\{\mathsf{pk}_j\}_{j\in[L]\setminus\{i\}}$, where $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$. Therefore, we have

$$\mathsf{pk}_j = \left(T_j, \left\{\widetilde{W}_{\ell,j}\right\}_{\ell\in[L]\setminus\{j\}}\right), \forall j \in [L] \setminus \{i\} \quad , \quad \mathsf{sk}_j = \widetilde{r}_j \text{ for some } \widetilde{r}_j \in \mathbb{Z}_q^+.$$

For each $j \in [L]$, let $\mathbf{x}_j \in \mathbb{Z}_q^{n^+}$ be the predicate vector associated to pk_j and let $\widetilde{\mathbf{x}}_j = (\mathbf{x}_j, \widetilde{r}_j, 1)$. Further, let mpk and hsk_i be as computed by $\mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_j, \mathbf{x}_j))_{j\in[L]})$. Now, note that in the Dec algorithm, the computation associated to the message components yield

$$\frac{C_1}{e(C_2, B_i)} = \frac{m \cdot Z^s}{e\left(g_1^s, g_2^\alpha \cdot A_i^\beta\right)} = \frac{m \cdot e(g_1, g_2)^{\alpha\cdot s}}{e(g_1, g_2)^{\alpha\cdot s} \cdot e(g_1, g_2)^{s\beta t_i}} = \frac{m}{e(g_1, g_2)^{s\beta t_i}} \tag{3}$$

Now observe that for any vector $\mathbf{x}_i \in \mathbb{Z}_q^{n^+}$ for some $i \in [L]$ and an attribute $\mathbf{y} \in \mathbb{Z}_q^{n^+}$ with $\langle \mathbf{x}_i, \mathbf{y}\rangle = 0$, it also holds that $\langle \widetilde{\mathbf{x}}_i, \widetilde{\mathbf{y}}\rangle = \langle \mathbf{x}_i, \mathbf{y}\rangle + \langle \widetilde{r}_i, 0\rangle + 1 \cdot 0 = 0$. For brevity, we set up the notations $g_\mathsf{T} = e(g_1, g_2)$ and the discrete logarithm as $\mathsf{DL}(K) = k$, where $K = g_\mathsf{t}^k$ for any $k \in \mathbb{Z}_q$ (i.e., irrespective of any group type $\mathsf{t} \in \{1, 2, \mathsf{T}\}$) for the rest of the proof. To ensure correctness with the rest of decryption above, it is thus enough to show that

$$\prod_{w=1}^{n'+1} \left\{e\left(C_{3,w}^{\widetilde{x}_{w,i}}, A_i\right) \cdot e\left(C_4, \widehat{W}_{w,i}^{\widetilde{x}_{w,i}}\right)\right\} = g_\mathsf{T}^{s\beta t_i X_i} \tag{4}$$

so that Dec yields the message $m \in \mathbb{G}_\mathsf{T}$. We will analyze individual pairing products in the above form separately. Before that we note a few things about the public keys *after they are fused with the predicate vectors* during Aggr. For any $i \in [L], j \in [0, L]$, we have

$$T_j = \left(\prod_{w\in[n]} U_{w,j}^{-x_{w,j}}\right) \cdot U_{n',j}^{-\widetilde{r}_j} = \prod_{w\in[n']} g_1^{-u_{w,j}\widetilde{x}_{w,j}} = g_1^{-\sum_{w\in[n']} u_{w,j}\widetilde{x}_{w,j}}$$
$$\implies \mathsf{DL}(T_j) = -\sum_{w\in[n']} u_{w,j}\widetilde{x}_{w,j},$$
$$\text{and thus, } \widetilde{W}_{i,j} = \left(\prod_{w\in[n]} W_{i,j,w}^{x_{w,j}}\right) \cdot W_{i,j,n'}^{\widetilde{r}_j} = \prod_{w\in[n']} \left(A_i^{u_{w,j}/\gamma}\right)^{\widetilde{x}_{w,j}}$$
$$= A_i^{\frac{1}{\gamma}\cdot\sum_{w\in[n']} u_{w,j}\widetilde{x}_{w,j}} = A_i^{-\mathsf{DL}(T_j)/\gamma},$$

where we redefined $\widetilde{x}_{n',0} = \widetilde{r}_0$. Further, for any $w \in [n']$ and $i \in [L]$, we have:

$$\widehat{W}_{w,i}^{\widetilde{x}_{w,i}} = \prod_{j\in[0,L]\setminus\{i\}} W_{i,j,w}^{\widetilde{x}_{w,i}} = \prod_{j\in[0,L]\setminus\{i\}} \left(A_i^{u_{w,j}/\gamma}\right)^{\widetilde{x}_{w,i}} = A_i^{\left(\widetilde{x}_{w,i}\cdot\sum_{j\in[0,L]\setminus\{i\}} u_{w,j}\right)/\gamma} \tag{5}$$

Defining the first pairing product as $\theta_1 = \prod_{w=1}^{n'+1} e\left(C_{3,w}^{\widetilde{x}_{w,i}}, A_i\right)$, we have:

$$\theta_1 = \prod_{w=1}^{n'+1} e\left(\left(h^{\widetilde{y}_w\cdot r+s} \cdot \widehat{U}_w^{-z}\right)^{\widetilde{x}_{w,i}}, A_i\right)$$
$$= \prod_{w=1}^{n'+1} \left\{e\left(h^{r\cdot\widetilde{x}_{w,i}\widetilde{y}_w}, A_i\right) \cdot e\left(h^{s\cdot\widetilde{x}_{w,i}}, A_i\right) \cdot e\left(\widehat{U}_w^{-z\widetilde{x}_{w,i}}, A_i\right)\right\}$$

$$\begin{aligned}
\Rightarrow \theta_1 &= e\left(h^{r\cdot\sum_{w=1}^{n'+1}\widetilde{x}_{w,i}\widetilde{y}_w}, A_i\right)\cdot e\left(g_1^{s\beta\sum_{w=1}^{n'+1}\widetilde{x}_{w,i}}, A_i\right)\cdot\prod_{w=1}^{n'+1} e\left(\widehat{U}_w^{-z\widetilde{x}_{w,i}}, A_i\right)\\
&= e\left(h^0, A_i\right)\cdot e\left(g_1^{s\beta X_i}, g_2^{t_i}\right)\cdot\prod_{w=1}^{n'+1} e\left(\widehat{U}_w^{-z\widetilde{x}_{w,i}}, A_i\right)\\
&= g_{\mathsf{T}}^{s\beta t_i X_i}\ \times\ \theta_{11}\ \times\ \theta_{12},
\end{aligned}$$

$$\text{where }\ \theta_{11} = \prod_{w=1}^{n'} e\left(\widehat{U}_w^{-z\widetilde{x}_{w,i}}, A_i\right)\ \text{ and } \theta_{12} = e\left(\widehat{U}_{n'+1}^{-z}, A_i\right)\ (\because \widetilde{x}_{n'+1,i} = 1)$$

$$\begin{aligned}
\therefore \theta_{11} &= \prod_{w\in[n']} e\left(\prod_{j=0}^{L} U_{w,j}^{-z\widetilde{x}_{w,i}}, A_i\right) = \prod_{w\in[n']} e\left(\left(g_1^{\sum_{j=0}^{L} u_{w,j}}\right)^{-z\widetilde{x}_{w,i}}, g_2^{t_i}\right)\\
&= \prod_{w\in[n']} g_{\mathsf{T}}^{-zt_i\widetilde{x}_{w,i}\sum_{j=0}^{L} u_{w,j}} = \prod_{w\in[n']} g_{\mathsf{T}}^{zt_i(-\widetilde{x}_{w,i}u_{w,i})}\cdot\prod_{w\in[n']} g_{\mathsf{T}}^{-zt_i\widetilde{x}_{w,i}\sum_{j\in[0,L]\setminus\{i\}} u_{w,j}}\\
&= g_{\mathsf{T}}^{zt_i\mathsf{DL}(T_i)}\cdot\zeta_1,\ \text{ where } \zeta_1 = \prod_{w\in[n']} g_{\mathsf{T}}^{-zt_i\widetilde{x}_{w,i}\sum_{j\in[0,L]\setminus\{i\}} u_{w,j}}\ \text{ and}
\end{aligned}$$

$$\begin{aligned}
\theta_{12} &= e\left(\widehat{U}_{n'+1}^{-z}, A_i\right) = e\left(\prod_{j=0}^{L} T_j^{-1}, A_i^z\right) = \prod_{j=0}^{L} e\left(T_j^{-1}, A_i^z\right) = \prod_{j=0}^{L} e\left(\prod_{w=1}^{n'} U_{w,j}^{\widetilde{x}_{w,j}}, A_i^z\right)\\
&= \prod_{j=0}^{L} e\left(g_1^{\sum_{w\in[n']} u_{w,j}\widetilde{x}_{w,j}}, A_i^z\right) = \prod_{j=0}^{L} e\left(g_1^{-\mathsf{DL}(T_j)}, g_2^{zt_i}\right) = \prod_{j=0}^{L} g_{\mathsf{T}}^{-zt_i\mathsf{DL}(T_j)}\\
&= g_{\mathsf{T}}^{-zt_i\mathsf{DL}(T_i)}\cdot\zeta_2,\ \text{ where } \zeta_2 = g_{\mathsf{T}}^{-zt_i\sum_{j\in[0,L]\setminus\{i\}}\mathsf{DL}(T_j)}.
\end{aligned}$$

$$\therefore\ \text{We have } \theta_1 = g_{\mathsf{T}}^{s\beta t_i X_i}\ \times\left(\cancel{g_{\mathsf{T}}^{zt_i\mathsf{DL}(T_i)}}\cdot\zeta_1\right)\ \times\ \left(\cancel{g_{\mathsf{T}}^{-zt_i\mathsf{DL}(T_i)}}\cdot\zeta_2\right)\ \Rightarrow\ \theta_1 = g_{\mathsf{T}}^{s\beta t_i X_i}\ \times\ \zeta_1\ \times\ \zeta_2$$

Defining the second pairing product as $\theta_2 = \prod_{w=1}^{n'+1} e\left(C_4, \widehat{W}_{w,i}^{\widetilde{x}_{w,i}}\right)$, we have:

$$\begin{aligned}
\theta_2 &= \left\{\prod_{w\in[n']} e\left(g_1^{z\gamma}, \widehat{W}_{w,i}^{\widetilde{x}_{w,i}}\right)\right\}\cdot e\left(g_1^{z\gamma}, \widehat{W}_{n'+1,i}\right)\ (\because \widetilde{x}_{n'+1,i} = 1 \text{ and } C_4 = \Gamma^z = g^{z\gamma})\\
&= \left\{\prod_{w\in[n']} e\left(g_1^{z\gamma}, A_i^{\left(\widetilde{x}_{w,i}\cdot\sum_{j\in[0,L]\setminus\{i\}} u_{w,j}\right)/\gamma}\right)\right\}\cdot e\left(g_1^{z\gamma}, \left(\prod_{j\in[0,L]\setminus\{i\}}\widetilde{W}_{i,j}\right)^{-1}\right)\\
&= \left\{\prod_{w\in[n']} e\left(g_1^{z\gamma}, g_2^{\left(t_i\widetilde{x}_{w,i}\cdot\sum_{j\in[0,L]\setminus\{i\}} u_{w,j}\right)/\gamma}\right)\right\}\cdot\prod_{j\in[0,L]\setminus\{i\}} e\left(g_1^{z\gamma}, \left(A_i^{-\mathsf{DL}(T_j)/\gamma}\right)^{-1}\right)\\
&= \prod_{w\in[n']} g_{\mathsf{T}}^{zt_i\widetilde{x}_{w,i}\cdot\sum_{j\in[0,L]\setminus\{i\}} u_{w,j}}\cdot\prod_{j\in[0,L]\setminus\{i\}} e\left(g_1^{z\gamma}, g_2^{t_i\mathsf{DL}(T_j)/\gamma}\right)
\end{aligned}$$

$$\Rightarrow \theta_2 = \zeta_1^{-1}\cdot\prod_{j\in[0,L]\setminus\{i\}} g_{\mathsf{T}}^{zt_i\mathsf{DL}(T_j)} = \zeta_1^{-1}\cdot g_{\mathsf{T}}^{zt_i\sum_{j\in[0,L]\setminus\{i\}}\mathsf{DL}(T_j)} = \zeta_1^{-1}\ \times\ \zeta_2^{-1}$$

This completes the proof since

$$\prod_{w=1}^{n'+1}\left\{e\left(C_{3,w}^{\widetilde{x}_{w,i}}, A_i\right)\cdot e\left(C_4, \widehat{W}_{w,i}^{\widetilde{x}_{w,i}}\right)\right\} = \theta_1\cdot\theta_2 = g_{\mathsf{T}}^{s\beta t_i X_i}\times\zeta_1\times\zeta_2\times\zeta_1^{-1}\times\zeta_2^{-1} = g_{\mathsf{T}}^{s\beta t_i X_i}.$$

□

Theorem 6 (Security of Construction 1). *The slotted RIPE scheme Π_{sRIPE} with message space $\mathcal{M} = \mathbb{G}_\mathsf{T}$ and attribute space $\mathcal{U} = \mathbb{Z}_q^{n^+}$ from Construction 1 is secure in the* GGM.

Proof. Below, we show that our slotted RIPE scheme Π_{sRIPE} (Construction 1) is secure in the generic group model. We start with some notations and definitions for generic and symbolic group models.

Generic Bilinear Group Model. Our definitions for generic bilinear group model is adapted from [BCFG17, AY20]. Let $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, g_1, g_2, e)$ be a bilinear group setting, $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_\mathsf{T}$ be lists of group elements in $\mathbb{G}_1, \mathbb{G}_2$, and $\mathbb{G}_\mathsf{T}$ respectively. Let $\mathcal{D}$ be a distribution over $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_\mathsf{T}$. The generic group model for a bilinear group setting $\mathcal{G}$ and a distribution $\mathcal{D}$ is described in Figure 1. In this model, the challenger first initializes the lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_\mathsf{T}$ by sampling the group elements according to $\mathcal{D}$, and the adversary receives handles for the elements in the lists. For $\mathsf{t} \in \{1, 2, \mathsf{T}\}$, $\mathcal{L}_\mathsf{t}[h]$ denotes the h-th element in the list $\mathcal{L}_\mathsf{t}$. The handle to this element is simply the pair (t, h). An adversary A running in the generic bilinear group model can apply group operations and the bilinear map e to the elements in the lists. To do this, A has to call the appropriate oracle specifying handles for the input elements. A also gets access to the internal state variables of the challenger via handles, and we assume that the equality queries are "free", in the sense that they do not count when measuring the computational complexity A. For $\mathsf{t} \in \{1, 2, \mathsf{T}\}$, the challenger computes the result of a query, say $\delta \in \mathbb{G}_\mathsf{t}$, and stores it in the corresponding list as $\mathcal{L}_\mathsf{t}[\mathsf{pos}] = \delta$ where pos is its next *empty* position in $\mathcal{L}_\mathsf{t}$, and returns to A its (newly created) handle $(\mathsf{t}, \mathsf{pos})$. Handles are not unique (i.e., the same group element may appear more than once in a list under different handles). As in [AY20], the equality test oracle in [BCFG17] is replaced with the zero-test oracle $\mathsf{Zt}_\mathsf{T}(\cdot)$ that, on input a handle (t, h), returns 1 if $\mathcal{L}_\mathsf{t}[h] = 0$ and 0 otherwise only for the case $\mathsf{t} = \mathsf{T}$.

State: Lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_\mathsf{T}$ over $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ respectively.

Initializations: Lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_\mathsf{T}$ sampled according to distribution $\mathcal{D}$.

Oracles: The oracles provide black-box access to the group operations, the bilinear map, and zero-tests.

- $\forall \mathsf{t} \in \{1, 2, \mathsf{T}\}$: $\mathsf{Add}_\mathsf{t}(h_1, h_2)$ appends $\mathcal{L}_\mathsf{t}[h_1] + \mathcal{L}_\mathsf{t}[h_2]$ to $\mathcal{L}_\mathsf{t}$ and returns its handle $(\mathsf{t}, |\mathcal{L}_\mathsf{t}|)$.
- $\forall \mathsf{t} \in \{1, 2, \mathsf{T}\}$: $\mathsf{Neg}_\mathsf{t}(h)$ appends $-\mathcal{L}_\mathsf{t}[h]$ and returns its handle $(\mathsf{t}, |\mathcal{L}_\mathsf{t}|)$.
- $\mathsf{Map}_e(h_1, h_2)$ appends $e(\mathcal{L}_1[h_1], \mathcal{L}_2[h_2])$ and returns its handle $(\mathsf{T}, |\mathcal{L}_\mathsf{T}|)$.
- $\mathsf{Zt}_\mathsf{T}(h)$ returns 1 if $\mathcal{L}_\mathsf{T}[h] = 0$ and 0 otherwise.

All oracles return $\perp$ when given invalid indices.

Fig. 1: GGM for bilinear group setting $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, g_1, g_2, e)$ and distribution $\mathcal{D}$.

Symbolic Group Model. The symbolic group model (SGM) for a bilinear group setting $\mathcal{G}$ and a distribution $\mathcal{D}$ gives to the adversary the same interface as the corresponding generic group model (GGM), except that internally the challenger stores lists of elements from the ring $\mathbb{Z}_q[\mathtt{x}_1, \ldots, \mathtt{x}_k]$ instead of lists of group elements, where $\mathtt{x}_1, \ldots, \mathtt{x}_k$ for $k \in \mathbb{N}$ are indeterminates. The oracles $\mathsf{Add}_\mathsf{t}(\cdot,\cdot), \mathsf{Neg}_\mathsf{t}(\cdot), \mathsf{Map}_e(\cdot,\cdot), \mathsf{Zt}_\mathsf{T}(\cdot)$ computes addition, negation, multiplication, and zero tests in the ring. For our proof, we will work in the ring $\mathbb{Z}_q[\mathtt{x}_1, \ldots, \mathtt{x}_k, 1/\mathtt{x}_i]$ for some $i \in [k]$. Note that any element $f \in \mathbb{Z}_q[\mathtt{x}_1, \ldots, \mathtt{x}_k, 1/\mathtt{x}_i]$ can be represented as

$$f(\mathtt{x}_1, \ldots, \mathtt{x}_n) = \sum_{\mathbf{c} \in \mathbb{Z}^k} \eta_\mathbf{c} \prod_{i=1}^{k} \mathtt{x}_i^{c_i} \text{ with } \mathbf{c} = (c_1, \ldots, c_k) \in \mathbb{Z}^k$$

using $\{\eta_\mathbf{c} \in \mathbb{Z}_q\}_{\mathbf{c} \in \mathbb{Z}^k}$, where $\eta_\mathbf{c} = 0$ for all but finite $\mathbf{c} \in \mathbb{Z}^k$. Note that this expression is unique.

In the remaining part of this section, we prove the security (Theorem 6) of our slotted RIPE scheme $\Pi_{\mathsf{sRIPE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ from Construction 1.

Proof. At a high level, the proof shows a sequence of hybrids each of which is an interaction between a challenger and a PPT adversary A. In the first (resp., last) hybrid, the challenger encrypts a pair $(\mathbf{y}_b, m_b)$ corresponding to bit $b = 0$ (resp., $b = 1$). The intermediate hybrids are defined in such a way that the distributions in any pair of subsequent hybrids from the first one to the last are statistically indistinguishable.

Since the proof is in the GGM, w.l.o.g. the challenger simulates all the generic bilinear group oracle queries for A. In particular, the challenger stores the actual computed elements in the list $\mathcal{L}_{\mathsf{t}}$ based on its group type $\mathsf{t} \in \{1, 2, \mathsf{T}\}$. The handle to an actual element stored in any of these lists are just a tuple $(\mathsf{t}, \mathsf{pos})$ specifying the group type t and its position in the table $\mathcal{L}_{\mathsf{t}}$. Since our scheme contains several variables, we will refrain from explicitly specifying the handles to the actual elements for convenience. Further, when we move to the SGM, we will denote any literal variable v as v and composite terms like $v_1 v_2$ (resp., $\frac{v_1}{v_2}$) as $\mathsf{v}_1\mathsf{v}_2$ (resp., $\frac{\mathsf{v}_1}{\mathsf{v}_2}$) to represent an individual monomial in a (possibly multivariate) polynomial. For variables denoted with Greek alphabets, say α, β, γ, we represent their corresponding formal variables as $\upalpha, \upbeta, \upgamma$. We also define $\mathbb{Z}_q\text{-}\mathsf{span}(\mathcal{S})$ as the set of $\mathbb{Z}_q$-linear combinations of all elements in any set $\mathcal{S}$. Assume A issues an arbitrary polynomial number $Q_{\mathsf{zt}}(\lambda)$ of queries to its Zt_{T} oracle in each hybrid.

$\mathbf{H}_1(\lambda)$: This is the real scheme corresponding to bit $b = 0$ in the GGM. In more detail, the hybrid goes as follows.

- **Setup phase:** A sends an attribute length $n = n(\lambda)$ and slot count $L = L(\lambda)$ to the challenger, upon which it first initializes $\mathsf{ctr} = 0$, a dictionary D, and the set $\mathcal{C}_L = \emptyset$ to account for corrupted slots. Next, it computes $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, g_1, g_2, e) \leftarrow_\$ \mathsf{GroupGen}(1^\lambda)$ and initializes three tables as $\mathcal{L}_{\mathsf{t}}[1] = g_{\mathsf{t}}, \forall \mathsf{t} \in \{1, 2, \mathsf{T}\}$. The challenger prepares a tuple $\left(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, \{(\mathsf{t}, 1)\}_{\mathsf{t}\in\{1,2,\mathsf{T}\}}\right)$, where $(\mathsf{t}, 1)$ represents the handle to $g_{\mathsf{t}}, \forall \mathsf{t} \in \{1, 2, \mathsf{T}\}$. To allow A to compute the group operations including the bilinear map e, the challenger also simulates all the oracles $\mathsf{Add}_{\mathsf{t}}, \mathsf{Neg}_{\mathsf{t}}, \mathsf{Map}_e, \mathsf{Zt}_\mathsf{T}$ with implicit access to the lists $\{\mathcal{L}_{\mathsf{t}}\}_{\mathsf{t}\in\{1,2,\mathsf{T}\}}$. It then computes the crs components as follows:
  1. Set $n' = n + 1$. Compute $h = g_1^\beta, \Gamma = g_1^\gamma \in \mathbb{G}_1$ and $Z = e(g_1, g_2)^\alpha \in \mathbb{G}_\mathsf{T}$ as in the real Setup algorithm. Update $\mathcal{L}_1$ with the elements β, γ and $\mathcal{L}_\mathsf{T}$ with α respectively.
  2. For each slot index $i \in [0, L]$, do the following:
     (a) $\forall w \in [n']$, compute $U_{w,i} = g_1^{u_{w,i}} \in \mathbb{G}_1$ as in the real scheme and update $\mathcal{L}_1$ with $u_{w,i}$.
     (b) $\forall i > 0$, compute $A_i = g_2^{t_i}, B_i = g_2^{\alpha+\beta\cdot t_i} \in \mathbb{G}_2$ as in the real scheme and update $\mathcal{L}_2$ with $t_i, (\alpha + \beta \cdot t_i)$ in order.
     (c) $\forall i > 0, w \in [n']$ and for each $j \in [0, L] \setminus \{i\}$, compute $W_{i,j,w} = g_2^{\frac{t_i \cdot u_{j,w}}{\gamma}} \in \mathbb{G}_2$ as in the real scheme and update $\mathcal{L}_2$ with $\frac{t_i \cdot u_{j,w}}{\gamma}$.
  3. For $\widetilde{\mathbf{x}}_0 = (\widetilde{x}_{1,0}, \ldots, \widetilde{x}_{n',0}) \leftarrow_\$ \mathbb{Z}_q^{n'+}$, set $\mathsf{pk}_0 = \left(T_0, \left\{\widetilde{W}_{i,0}\right\}_{i\in[L]}\right)$ as in the real scheme. Define $u'_0 = \sum_{w=1}^{n'} u_{w,0} \cdot \widetilde{x}_{w,0} = -\mathsf{DL}(T_0)$ so that
  $$T_0 = g_1^{u'_0} \in \mathbb{G}_1 \quad , \quad \widetilde{W}_{i,0} = g_2^{\frac{t_i \cdot u'_0}{\gamma}} \in \mathbb{G}_2, \forall i \in [L].$$
  Update $\mathcal{L}_1$ with u'_0 and $\mathcal{L}_2$ with $\left\{\frac{t_i \cdot u'_0}{\gamma}\right\}_{i\in[L]}$ in order.
  4. Set
  $$\mathsf{crs} = \Big(\mathcal{G}, Z, h, \Gamma, \{A_i, B_i\}_{i\in[L]}, \left\{\{U_{w,i}\}_{i\in[0,L]}, \{W_{i,j,w}\}_{i\in[L], j\in[0,L]\setminus\{i\}}\right\}_{w\in[n']}, \mathsf{pk}_0\Big).$$
  5. Return to A a tuple crs' that includes $\left(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, \{(\mathsf{t}, 1)\}_{\mathsf{t}\in\{1,2,\mathsf{T}\}}\right)$ along with the handles to all elements in the same order as they are arranged in the crs above.

- **Pre-challenge query phase:** A can issue *key generation* queries or *corruption* queries in this phase.
  1. Consider the key-generation queries first. Upon getting a slot index $i \in [L]$, the challenger updates $\mathsf{ctr} = \mathsf{ctr} + 1$, sets $\mathbf{x}_i^{\mathsf{ctr}} = \mathbf{x}_i$ and does the following:
     (a) Sample $\widetilde{r}_i^{\mathsf{ctr}} \leftarrow_\$ \mathbb{Z}_q^+$ and compute $\mathsf{pk}_i^{\mathsf{ctr}} = \left(T_i^{\mathsf{ctr}}, \left\{\widetilde{W}_{j,i}^{\mathsf{ctr}}\right\}_{j\in[L]\setminus\{i\}}\right)$ as in KGen.
     (b) Note that the element $T_i^{\mathsf{ctr}} \in \mathbb{G}_1$ from $\mathsf{pk}_i^{\mathsf{ctr}}$ has the following structure:

     $$T_i^{\mathsf{ctr}} = g_1^{-\widetilde{r}_i^{\mathsf{ctr}} u_{n',i}}, \text{ where } \mathsf{sk}_i^{\mathsf{ctr}} = \widetilde{r}_i^{\mathsf{ctr}} \text{ is the secret key.}$$

     Even given the handle to $u_{n',i}$, A cannot compute a handle for $\mathsf{DL}(T_i^{\mathsf{ctr}}) = -\widetilde{r}_i^{\mathsf{ctr}} u_{n',i}$ on its own. Hence, the challenger updates $\mathcal{L}_1$ with $\mathsf{DL}(T_i^{\mathsf{ctr}})$.
     (c) Further, each term in $\left\{\widetilde{W}_{j,i}^{\mathsf{ctr}} \in \mathbb{G}_2\right\}_{j\in[L]\setminus\{i\}}$ has the following structure:

     $$\widetilde{W}_{j,i}^{\mathsf{ctr}} = W_{j,i,n'}^{\widetilde{r}_i^{\mathsf{ctr}}} = g_2^{\frac{t_j u_{n',i}}{\gamma} \cdot \widetilde{r}_i^{\mathsf{ctr}}}$$

     For reasons similar to Item (b) above, the challenger updates $\mathcal{L}_2$ with each element individually from the set $\{\widetilde{r}_i^{\mathsf{ctr}} \cdot \frac{t_j u_{n',i}}{\gamma}\}_{j\in[L]\setminus\{i\}}$.
     (d) Define $\mathsf{pk}_{\mathsf{ctr}} = \mathsf{pk}_i^{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}} = \mathsf{sk}_i^{\mathsf{ctr}}$ and $\mathsf{pk}'_{\mathsf{ctr}}$ as a sequence of handles to all elements in the same order as they are arranged in $\mathsf{pk}_{\mathsf{ctr}}$.
     (e) Return the tuple $(\mathsf{ctr}, \mathsf{pk}'_{\mathsf{ctr}})$ to A and update $\mathsf{D}[\mathsf{ctr}] = (i, \mathsf{pk}_{\mathsf{ctr}}, \mathsf{sk}_{\mathsf{ctr}})$.
  2. When A sends $c \in [\mathsf{ctr}]$ issuing a corruption query, the challenger returns sk' to A where $\mathsf{D}[c] = (i', \mathsf{pk}', \mathsf{sk}')$.
- **Challenge phase:** In this phase, A specifies the following challenge information:

  $$\{(c_i, \mathbf{x}_i, \mathsf{pk}_i^*)\}_{i\in[L]} \quad \text{and} \quad ((\mathbf{y}_0, m_0), (\mathbf{y}_1, m_1)) \in (\mathbb{Z}_q^{n+} \times \mathbb{G}_\mathsf{T})^2.$$

  Preprocessing the challenge information. For each $i \in [L]$, the challenger checks that $\mathbf{x}_i \neq \mathbf{0}^n$ and does the following:
  1. If $c_i \in [\mathsf{ctr}]$, it checks $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$. If $i \neq i'$, it halts. Else, it sets $\mathsf{pk}_i^* = \mathsf{pk}'$. Further, if A issued a corruption query for c_i before, it updates $\mathcal{C}_L = \mathcal{C}_L \cup \{i\}$.
  2. If $c_i = \perp$, pk_i^* represents a corrupt secret key generated by A itself. Hence, it parses pk_i^* and halts if $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*) = 0$.[13] Else, it updates $\mathcal{C}_L = \mathcal{C}_L \cup \{i\}$.

  Computing key aggregation. The challenger then computes $(\mathsf{mpk}, (\mathsf{hsk}_i)_{i\in[L]}) = \mathsf{Aggr}\left(\mathsf{crs}, ((\mathsf{pk}_i^*, \mathbf{x}_i))_{i\in[L]}\right)$, where

  $$\mathsf{mpk} = \left(\mathcal{G}, g, h, Z, \Gamma, \left\{\widehat{U}_w\right\}_{w'\in[n'+1]}\right) \quad \text{and} \quad \mathsf{hsk}_i = \left(\mathcal{G}, i, A_i, B_i, \left\{\widehat{W}_{w,i}\right\}_{w\in[n'+1]}\right), \forall i \in [L].$$

  Computing the challenge ciphertext. The challenger now uses mpk and the pair $(\mathbf{y}_0, m_0)$, and generates $c^* \leftarrow_\$ \mathsf{Enc}(\mathsf{mpk}, \mathbf{y}_0, m_0)$ where $c^* = \left(C_1, C_2, \{C_{3,w}\}_{w\in[n'+1]}, C_4\right)$.
  1. Note that $C_1 = m_0 \cdot e(g_1, g_2)^{\alpha s} \in \mathbb{G}_\mathsf{T}$ and $C_2 = g_1^s \in \mathbb{G}_1$. Accordingly, the challenger updates $\mathcal{L}_\mathsf{T}$ with αs and $\mathcal{L}_1$ with s respectively.

[13] By Definition 17, A is supposed to send well-formed keys that passes the $\mathsf{IsValid}(\mathsf{crs}, \cdot, \cdot)$ test. Hence, from now on we do not mention it any more, but assume the challenger checks it implicitly.

2. With $\widetilde{\mathbf{y}}_0 = (\mathbf{y}_0, 0, 0) = (y_1^0, \ldots, y_n^0, 0, 0)$, note that the elements $\{C_{3,w} \in \mathbb{G}_1\}_{w\in[n'+1]}$ have the following structure:

$$\forall w \in [n],\ C_{3,w} = h^{y_w^0 \cdot r + s} \cdot \widehat{U}_w^{-z} = g_1^{r\beta y_w^0 + s\beta - z\cdot u_w} \text{ where } u_w = \sum_{i=0}^{L} u_{w,i},$$

$$\text{for } w = n',\ C_{3,n'} = g_1^{r\beta\cdot 0 + s\beta - z\cdot u'_n} = g_1^{s\beta - z\cdot u_{n'}} \text{ where } u_{n'} = \sum_{i=0}^{L} u_{n',i},$$

$$\begin{aligned}\text{for } w = n'+1, C_{3,n'+1} &= g_1^{r\beta\cdot 0 + s\beta} \cdot \widehat{U}_{n'+1}^{-z} = g_1^{s\beta} \cdot \prod_{i=0}^{L} T_i^{-z} \\ &= g_1^{s\beta} \cdot \prod_{i=0}^{L} g_1^{z \sum_{w=1}^{n'} \widetilde{x}_{w,i}\cdot u_{w,i}} \\ &= g_1^{s\beta} \cdot \prod_{i=0}^{L} g_1^{z\cdot u'_i} = g_1^{s\beta + z\cdot u'_0 - z\sum_{i=1}^{L} \mathsf{DL}(T_i)}\end{aligned}$$

Accordingly, the challenger updates $\mathcal{L}_1$ with the elements $\{r\beta y_w^0 + s\beta - z\cdot u_w\}_{w\in[n]}$, $(s\beta - z\cdot u_{n'})$, and $\left[s\beta + z\cdot u'_0 - z\cdot \sum_{i=1}^{L} \mathsf{DL}(T_i)\right]$ in order.
3. Since $C_4 = g_1^{\gamma z} \in \mathbb{G}_1$, it updates $\mathcal{L}_1$ with $z\gamma$.

Group oracle queries. Since Aggr is deterministic, A is able to compute $(\mathsf{mpk}, (\mathsf{hsk}_i)_{i\in[L]})$ on its own. In the GGM, A is able to compute handles for the elements in mpk and $\{\mathsf{hsk}_i\}_{i\in[L]}$. To this end, it queries the appropriate group oracles to generate such handles as follows:

1. A only needs to compute the handles for $\{\widehat{U}_w\}_{w\in[n'+1]}$ to complete its information about mpk. Note that $\forall w \in [n']$, $\widehat{U}_w = \prod_{i=0}^{L} U_{w,i} = g_1^{u_w}$, where $u_w = \sum_{i=0}^{L} u_{w,i}$. Hence, $\forall w \in [n']$, A invokes the Add_1 oracle L times *iteratively* on the handles in $\mathcal{L}_1$ for $\{u_{w,i}\}_{i\in[0,L]}$ and gets a handle for u_w. Further, to get a handle for $\widehat{U}_{n'+1} = \prod_{i=0}^{L} T_i$, it has to first a get a handle for each T_i that is fused with the predicate $\mathbf{x}_i$. Note the structure of each T_i after Step (2) in Aggr:

$$T_i = g_1^{\sum_{w=1}^{n'} -\widetilde{x}_{w,i}\cdot u_{w,i}} = g_1^{\sum_{w=1}^{n}(-x_{w,i}\cdot u_{w,i})} \times g_1^{\left(-\widetilde{r}_i\cdot u_{n',i}\right)} \in \mathbb{G}_1.$$

Given a handle for the second multiplicand, it is easy to note that the first one is publicly computable using Neg_1 and Add_1 oracles. Once A obtains the handles for $\{T_i\}_{i\in[L]}$, it can call Add_1 oracle on these handles to get the same for $\widehat{U}_{n'+1}$.
2. A only needs to compute the handles for $\{\widehat{W}_{w,i}\}_{w\in[n'+1]}$ to get complete information about hsk_i for each $i \in [L]$. Note that $\forall w \in [n']$, $\widehat{W}_{w,i} = \prod_{j\in[0,L]\setminus\{i\}} W_{i,j,w} = g_2^{t_i\cdot(u_w - u_{w,i})/\gamma}$, since $(u_w - u_{w,i}) = \sum_{j\in[0,L]\setminus\{i\}} u_{w,j}$. It is again easy to see that a handle for such an element can be computed by calling the Add_2 oracle $L-1$ times *iteratively* on the handles in $\mathcal{L}_2$ for $\left\{\frac{t_i\cdot u_{w,j}}{\gamma}\right\}_{j\in[0,L]\setminus\{i\}}$. Further, note that to get a handle for $\widehat{W}_{n'+1,i} = \prod_{j\in[0,L]\setminus\{i\}} \widetilde{W}_{i,j}^{-1}$, it has to first a get a handle for each $\widetilde{W}_{j,i}$ that is fused with the predicate $\mathbf{x}_i$. Note the structure of each $\widetilde{W}_{j,i}$ after Step (2) in Aggr:

$$\widetilde{W}_{j,i} = \left(\prod_{w=1}^{n} W_{i,j,w}^{\widetilde{x}_{w,i}}\right) \cdot W_{i,j,n'}^{\widetilde{r}_i} = g_2^{\sum_{w=1}^{n} \frac{t_j u_{w,i}}{\gamma}\cdot x_{w,i}} \times g_2^{\left(\frac{t_j u_{n',i}}{\gamma}\cdot \widetilde{r}_i\right)} \in \mathbb{G}_2.$$

Again, given a handle for the second multiplicand, the same can be computed publicly for the entire product using handles for $\{W_{i,j,w}\}$. Once A obtains the handles to each element in $\{\widetilde{W}_{j,i}\}_{j\in[L]\setminus\{i\}}$, it can call Add_2 oracle on these handles to get the same for $\widehat{W}_{n+1,i}$.
3. Finally, it defines mpk' and each hsk'_i as sequences of handles to all elements (except $i, \mathbf{x}_i$) in the same order as arranged in mpk and each $\mathsf{hsk}_i, \forall i \in [L]$.

• **Output phase:** A outputs a bit $b' \in \{0,1\}$.

For ease of presentation, in Table 2 we show all unit and composite terms generated in the scheme itself, and stored in the respective lists.

	$\mathcal{L}_1$	$\mathcal{L}_2$	$\mathcal{L}_\mathsf{T}$
crs	$\boxed{g_1}$, $\boxed{\beta}$, $\boxed{\gamma}$ $\boxed{u'_0} = \sum_{w=1}^{n'} u_{w,0}\widetilde{x}_{w,0}$, $\left\{\boxed{u_{w,i}}\right\}_{i\in[0,L],w\in[n']}$	$\boxed{g_2}$, $\left\{\boxed{t_i}, \boxed{\alpha+\beta t_i}\right\}_{i\in[L]}$ $\boxed{\frac{t_i u'_0}{\gamma}}$, $\left\{\boxed{\frac{t_i u_{w,j}}{\gamma}}\right\}_{\substack{i\in[L]\\ j\in[0,L]\setminus\{i\}\\ w\in[n']}}$	$\boxed{g_\mathsf{T}}$ $\boxed{\alpha}$
$\{\mathsf{pk}_c\}_{c\in[Q_\mathsf{k}]}$	$\left\{\boxed{-\widetilde{r}_i^c u_{n',i}}\right\}_{c\in[Q_\mathsf{k}]}$ $\left(\text{for } \{T_i^c\}_{c\in[Q_\mathsf{k}]}\right)$	$\left\{\boxed{\widetilde{r}_i^c \cdot \frac{t_j u_{n',i}}{\gamma}}\right\}_{\substack{j\in[L]\setminus\{i\}\\ c\in[Q_\mathsf{k}]}}$ $\left(\text{for } \left\{\widetilde{W}_{j,i}^c\right\}_{j\in[L]\setminus\{i\},c\in[Q_\mathsf{k}]}\right)$	–
c	$\boxed{s}$ (for C_2) , $\boxed{z\gamma}$ (for C_4) $\boxed{r\beta y_w^0 + s\beta - zu_w}$ (for $C_{3,w}, \forall w \in [n]$), $\boxed{s\beta - zu_{n'}}$ (for $C_{3,n'}$) , where $u_{n'} = \sum_{i=0}^{L} u_{n',i}$ $\boxed{s\beta - z\mathsf{DL}(T)}$ (for $C_{3,n'+1}$) , where $\mathsf{DL}(T) \coloneqq \sum_{i=0}^{L} \mathsf{DL}(T_i)$	–	$\boxed{\mathsf{DL}(m) + \alpha s}$

Table 2. The above table shows all terms from the scheme for which handles are stored in the respective lists $\mathcal{L}_\mathsf{t}, \forall \mathsf{t} \in \{1,2,\mathsf{T}\}$. Assume $\mathcal{A}$ issues some arbitrary polynomial number, Q_k, of key queries in the pre-challenge query phase (some of which may be corrupted). The table lists all the terms for each of these keys $\{\mathsf{pk}_c\}_{c\in[Q_\mathsf{k}]}$ received by $\mathcal{A}$ in the second row. Hence, these terms are also indexed with superscripts for the key query count $c \in [Q_\mathsf{k}]$ (along with the slot index, say $i \in [L]$, for which $\mathcal{A}$ asked the key). The terms corresponding to mpk and hsk_i are not shown in the table, since the handles for these are publicly computable by $\mathcal{A}$ using the group oracles. Note that such terms correspond to keys for all the registered L slots (possibly all of which may be corrupted or even adversarially generated). Hence, the individual variables in each of those terms in mpk and hsk_i are independent of the counter variable $c \in [Q_\mathsf{k}]$ respectively. In c, observe that we also have $(\mathsf{DL}(m) + \alpha s)$ in $\mathcal{L}_\mathsf{T}$, where $\mathsf{DL}(m) \in \mathbb{Z}_q$ is w.r.t. g_T.

$\mathbf{H}_2(\lambda)$: In this hybrid, we switch to the SGM *partially*. Namely, the interaction between challenger and A remains exactly as it was in $\mathbf{H}_1(\lambda)$, but now the challenger stores formal variables for all the terms from Table 2 in the respective lists $\mathcal{L}_\mathsf{t}, \forall \mathsf{t} \in \{1,2,\mathsf{T}\}$. Thus, all the handles A receives refer to multivariate polynomials from the following ring:

$$\zeta = \mathbb{Z}_q\left[\alpha, \beta, \gamma, \mathtt{u}'_0, \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_\mathtt{i}^\mathtt{c}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \frac{1}{\gamma}, \mathtt{s}, \mathtt{r}, \mathtt{z}, \{\mathtt{y}_\mathtt{w}\}_{\mathtt{w}\in[\mathtt{n}'+1]}\right].$$

Concretely, A gets handles to formal polynomials from $\mathcal{L}_\mathsf{t}$ for each $\mathsf{t} \in \{1,2,\mathsf{T}\}$, where:

1. $\mathcal{L}_\mathsf{T} = \{1, \alpha, \mathsf{DL}(m) + \alpha\mathtt{s}\}$.
2. $\mathcal{L}_1 = \mathcal{L}_1^{\mathsf{crs}} \cup \mathcal{L}_1^{\mathsf{key}} \cup \mathcal{L}_1^{c}$, where
   (a) $\mathcal{L}_1^{\mathsf{crs}} = \left(1, \beta, \gamma, \mathtt{u}'_0, \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[0,\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}\right)$,
   (b) $\mathcal{L}_1^{\mathsf{key}} = \left(\{-\widetilde{\mathtt{r}}_\mathtt{i}^\mathtt{c}\mathtt{u}_{\mathtt{n}',\mathtt{i}}\}_{c\in[Q_\mathsf{k}]}\right)$ for some $i \in [L]$, and

(c) $\mathcal{L}_1^c = \left(\mathtt{s}, \mathtt{z}\gamma, \left\{ \mathtt{r}\beta\mathtt{y}_\mathtt{w} + \mathtt{s}\beta - \mathtt{z}\sum_{\mathtt{i}=0}^{\mathtt{L}} \mathtt{u}_{\mathtt{w},\mathtt{i}} \right\}_{w\in[n]}, \mathtt{s}\beta - \mathtt{z}\mathtt{u}_{\mathtt{n}'}, \mathtt{s}\beta - \mathtt{z}\mathsf{DL}(\mathtt{T}) \right)$.

3. $\mathcal{L}_2 = \mathcal{L}_2^{\mathsf{crs}} \cup \mathcal{L}_2^{\mathsf{key}}$, where

(a) $\mathcal{L}_2^{\mathsf{crs}} = \left(1, \{\mathtt{t}_\mathtt{i}, \alpha + \beta\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \frac{\mathtt{t}_\mathtt{i}\mathtt{u}_0'}{\gamma}, \left\{ \frac{\mathtt{t}_\mathtt{i}\mathtt{u}_{\mathtt{w},\mathtt{j}}}{\gamma} \right\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{j}\in[0,\mathtt{L}]\setminus\{\mathtt{i}\},\mathtt{w}\in[\mathtt{n}']} \right)$, and

(b) $\mathcal{L}_2^{\mathsf{key}} = \left(\left\{ \frac{\widetilde{\mathtt{r}}_\mathtt{i}^c \mathtt{t}_\mathtt{j}\mathtt{u}_{\mathtt{n}',\mathtt{i}}}{\gamma} \right\}_{j\in[L]\setminus\{i\},c\in[Q_\mathsf{k}]} \right)$ for some $i \in [L]$.

However, when $\mathcal{A}$ issues any zero-test query via Zt_T oracle, the challenger replaces the formal variables with their corresponding elements from $\mathbb{Z}_q$. In this case, if the variable is not assigned a value in $\mathbb{Z}_q$, it samples the corresponding value from the same distribution as it did in $\mathbf{H}_1(\lambda)$. However, once a value is assigned to a variable, it is fixed throughout the rest of $\mathbf{H}_2(\lambda)$. We show in Lemma 1 that $\mathbf{H}_1(\lambda) \equiv \mathbf{H}_2(\lambda)$. Given the tuple $\mathsf{P} = (\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_\mathsf{T})$, we define $\mathcal{C}(\mathcal{L}_\mathsf{T}) = \mathcal{L}_\mathsf{T} \cup \{V_1 \cdot V_2 \mid \forall V_1 \in \mathcal{L}_1, V_2 \in \mathcal{L}_2\}$. Basically, it is the set of all monomials from ζ with variables in $\mathbb{G}_\mathsf{T}$ that $\mathcal{A}$ can compute querying Map_e on the handles it received for elements in $\mathcal{L}_1, \mathcal{L}_2$. We estimate the size of $\mathcal{C}(\mathcal{L}_\mathsf{T})$ as follows. Note that we have $|\mathcal{C}(\mathcal{L}_\mathsf{T})| = |\mathcal{L}_\mathsf{T}| + |\mathcal{L}_1| \cdot |\mathcal{L}_2|$ where $|\mathcal{L}_\mathsf{T}| = 3$,

$$\begin{aligned} |\mathcal{L}_1| &= |\mathcal{L}_1^{\mathsf{crs}}| + \left|\mathcal{L}_1^{\mathsf{key}}\right| + |\mathcal{L}_1^c| \\ &\le \{(L+1)n' + 4\} + LQ_\mathsf{k} + (n+4) = L(n + Q_\mathsf{k} + 1) + 2n + 9, \text{ and} \\ |\mathcal{L}_2| &= |\mathcal{L}_2^{\mathsf{crs}}| + \left|\mathcal{L}_2^{\mathsf{key}}\right| \\ &\le \{2 + 2L + n'L^2\} + \{L(L-1)Q_\mathsf{k}\} = L^2(n + Q_\mathsf{k} + 1) - L(Q_\mathsf{k} - 2) + 2. \end{aligned}$$

There are several variables in ζ and several terms in $\mathcal{L}_1, \mathcal{L}_2$. Hence, for brevity, we do not state all the elements of $\mathcal{C}(\mathcal{L}_\mathsf{T})$ explicitly with all possible cross combinations of the monomials from $\mathcal{L}_1, \mathcal{L}_2$. However, it is easy to see by inspection that the maximal total degree of each term in $\mathcal{C}(\mathcal{L}_\mathsf{T})$ is $d = 7$ corresponding to the term $\left[\mathtt{r}\beta\mathtt{y}_\mathtt{w} \cdot \frac{\widetilde{\mathtt{r}}_\mathtt{i}^c \mathtt{t}_\mathtt{j}\mathtt{u}_{\mathtt{n}',\mathtt{i}}}{\gamma}\right]$ for any $w \in [n'], i \in [L], j \in [0, L] \setminus \{i\}, c \in [Q_\mathsf{k}]$. We also note that any handle submitted by $\mathcal{A}$ to the Zt_T oracle during its interaction refers to a polynomial $f \in \zeta$ as

$$f\left(\alpha, \beta, \gamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_\mathtt{i}^c\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \frac{1}{\gamma}, \mathtt{s}, \mathtt{r}, \mathtt{z}, \{\mathtt{y}_\mathtt{w}\}_{\mathtt{w}\in[\mathtt{n}'+1]}\right) = \sum_{\theta\in\mathcal{C}(\mathcal{L}_\mathsf{T})} \eta_\Theta \Theta,$$

where the coefficients $\{\eta_\Theta \in \mathbb{Z}_q\}_{\Theta\in\mathcal{C}(\mathcal{L}_\mathsf{T})}$ can be computed efficiently. Further, since all the monomials in $\mathcal{C}(\mathcal{L}_\mathsf{T})$ are distinct, the coefficients η_Θ are unique.

$\mathbf{H}_3(\lambda)$: In this hybrid, *all* queries to Zt_T oracle are answered using formal variables. Namely, for any Zt_T query on a handle to a polynomial $f \in \zeta$, the challenger returns 1 if:

$$f\left(\alpha, \beta, \gamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_\mathtt{i}^c\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \frac{1}{\gamma}, \mathtt{s}, \mathtt{r}, \mathtt{z}, \{\mathtt{y}_\mathtt{w}\}_{\mathtt{w}\in[\mathtt{n}'+1]}\right) = 0.$$

We show in Lemma 2 that $\mathbf{H}_2(\lambda) \approx \mathbf{H}_3(\lambda)$.

$\mathbf{H}_4(\lambda)$: In this hybrid, the challenge ciphertext computes an encryption of m_0 with respect to $\mathbf{y}_1$. That is, everything remains as it is in $\mathbf{H}_3(\lambda)$ except that the challenger generates

$$c^* = (C_1, C_2, \{C_{3,w}\}_{w\in[n'+1]}, C_4) \leftarrow\$ \mathsf{Enc}(\mathsf{mpk}, \mathbf{y}_1, m_0).$$

We show in Lemma 3 that $\mathbf{H}_3 \approx \mathbf{H}_4$.

From here on, the challenger moves to $\mathbf{H}_6(\lambda)$ directly if $m_0 = m_1$. Else if $m_0 \neq m_1$, it still moves to $\mathbf{H}_6(\lambda)$, but via the next hybrids.

$\mathbf{H}_{5,1}(\lambda)$: In this hybrid, $Z^s \in \mathbb{G}_\mathsf{T}$ is replaced with $\mathfrak{U} \leftarrow\$ \mathbb{G}_\mathsf{T}$. We show in Lemma 4 that $\mathbf{H}_4(\lambda) \approx \mathbf{H}_{5,1}(\lambda)$.

$\mathbf{H}_{5,2}(\lambda)$: In this hybrid, the challenge ciphertext encrypts m_1 instead of m_0.

$\mathbf{H}_{5,3}(\lambda)$: In this hybrid, $\mathfrak{U}$ is changed to the honestly computed Z^s again. Since these hybrids are standard, we show that directly in Lemma 5 that $\mathbf{H}_{5,1}(\lambda) \approx \mathbf{H}_{5,2}(\lambda) \approx \mathbf{H}_{5,3}(\lambda)$.

$\mathbf{H}_6(\lambda)$: In this hybrid, the challenger moves to GGM from the symbolic setting of SGM. This is the real scheme corresponding to bit $b = 1$ in the GGM and Lemma 6 shows that $\mathbf{H}_{5,3}(\lambda) \approx \mathbf{H}_6(\lambda)$.

Hybrid Indistinguishability. Here, we show the indistinguishability of consecutive hybrids.

Lemma 1 **($\mathbf{H}_1(\lambda) \equiv \mathbf{H}_2(\lambda)$).** *$\mathbf{H}_1(\lambda)$ and $\mathbf{H}_2(\lambda)$ are perfectly indistinguishable.*

Proof. Note that A sees the same handles in both $\mathbf{H}_1(\lambda)$ and $\mathbf{H}_2(\lambda)$. So it can notice a difference between the hybrids only if some zero-test query via the $\mathsf{Zt_T}$ oracle is answered differently. However, these zero-test queries are answered using values sampled from the same distribution in both the hybrids. Thus A's view remains the same in both the hybrids.

Lemma 2 **($\mathbf{H}_2(\lambda) \approx \mathbf{H}_3(\lambda)$).** *$\mathbf{H}_2(\lambda)$ and $\mathbf{H}_3(\lambda)$ are statistically indistinguishable.*

Proof. Note that $\mathbf{H}_2(\lambda)$ and $\mathbf{H}_3(\lambda)$ differs only when A submits a handle for some $f \in \zeta$ satisfying

$$f\left(\alpha, \beta, \gamma, u_0', \{u_{w,i}\}_{i\in[L],w\in[n']}, \{\widetilde{r}_i^c\}_{i\in[L],c\in[Q_k]}, \{t_i\}_{i\in[L]}, \frac{1}{\gamma}, s, r, z, \{y_w^0\}_{w\in[n'+1]}\right) = 0, \text{ and}$$
$$f\left(\upalpha, \upbeta, \upgamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_{\mathtt{i}}^{\mathtt{c}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \frac{1}{\upgamma}, \mathtt{s}, \mathtt{r}, \mathtt{z}, \{\mathtt{y}_\mathtt{w}\}_{\mathtt{w}\in[\mathtt{n}'+1]}\right) \neq 0$$

to the $\mathsf{Zt_T}$ oracle. Denote this event as $\mathbf{E}_{2,3}$. It suffices to bound the probability of $\mathbf{E}_{2,3}$ occurring in $\mathbf{H}_2(\lambda)$. To this end, recall that the maximal total degree of any such polynomial $f \in \zeta$ that could be formed by linear combinations of the monomials in $\mathcal{C}(\mathcal{L}_\mathsf{T})$ is $d = 7$. Further, for any such $f \in \zeta$ in $\mathbf{H}_2(\lambda)$, observe that all variables input to f are randomly sampled except $\{y_w^0\}_{w\in[n'+1]}$. In particular, we have $\mathbf{y}_0 \in \mathbb{Z}_q^{n^+}$ supplied by A itself and $y_{n'}^0, y_{n'+1}^0 = 0$. Thus, fixing $\widetilde{\mathbf{y}}_0$, the maximal degree of each monomial in f becomes $d-1 = 6$. We then define a new polynomial $g \in \mathbb{Z}_q\left[\upalpha, \upbeta, \upgamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_{\mathtt{i}}^{\mathtt{c}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \{\mathtt{y}_\mathtt{w}\}_{\mathtt{w}\in[\mathtt{n}'+1]}\right]$ as

$$g\left(\upalpha, \upbeta, \upgamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_{\mathtt{i}}^{\mathtt{c}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \{y_w^0\}_{w\in[n'+1]}\right) =$$
$$\upgamma \cdot f\left(\upalpha, \upbeta, \upgamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_{\mathtt{i}}^{\mathtt{c}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \frac{1}{\upgamma}, \{y_w^0\}_{w\in[n'+1]}\right).$$

The polynomial g is introduced to clear any $\upgamma$ from the denominator that may appear in f and to make sure that g is in the ring

$$\mathbb{Z}_q\left[\upalpha, \upbeta, \upgamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_{\mathtt{i}}^{\mathtt{c}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \{\mathbf{y}_\mathtt{w}\}_{\mathtt{w}\in[\mathtt{n}'+1]}\right]$$

and not ζ. Note that since $\gamma \neq 0$, $\mathbf{E}_{2,3}$ occurs if and only if

$$g\left(\alpha, \beta, \gamma, \mathtt{u}_0', \{u_{w,i}\}_{i\in[L],w\in[n']}, \{\widetilde{r}_i^c\}_{i\in[L],c\in[Q_k]}, \{t_i\}_{i\in[L]}, \{y_w^0\}_{w\in[n'+1]}\right) = 0$$

and

$$g\left(\upalpha, \upbeta, \upgamma, \mathtt{u}_0', \{\mathtt{u}_{\mathtt{w},\mathtt{i}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{w}\in[\mathtt{n}']}, \{\widetilde{\mathtt{r}}_{\mathtt{i}}^{\mathtt{c}}\}_{\mathtt{i}\in[\mathtt{L}],\mathtt{c}\in[\mathtt{Q}_\mathtt{k}]}, \{\mathtt{t}_\mathtt{i}\}_{\mathtt{i}\in[\mathtt{L}]}, \{y_w^0\}_{w\in[n'+1]}\right) \neq 0.$$

However, this implies that $\deg(g) \leq 7$ ($\upgamma$ may increase the maximal degree of each monomial by 1, even if $\widetilde{\mathbf{y}}_0$ is fixed). We now note that all the remaining inputs in g are sampled independently and uniformly at random. Hence, by Schwarz-Zippel lemma we have that $\Pr[\mathbf{E}_{2,3}] \leq \frac{7}{q}$. As A issues $Q_{\mathtt{zt}}(\lambda)$ queries to the $\mathsf{Zt_T}$ oracle, a union bound implies that A can distinguish the two hybrids with probability at most $\frac{7\cdot Q_{\mathtt{zt}}(\lambda)}{q}$. Thus, $\mathbf{H}_2(\lambda) \approx \mathbf{H}_3(\lambda)$.

Lemma 3 **($\mathbf{H}_3(\lambda) \approx \mathbf{H}_4(\lambda)$).** *$\mathbf{H}_3(\lambda)$ and $\mathbf{H}_4(\lambda)$ are statistically indistinguishable.*

Proof. In $\mathbf{H}_3(\lambda)$ and $\mathbf{H}_4(\lambda)$, A interacts with the challenger in the SGM. In particular, all elements from $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_\mathsf{T}$ are treated "symbolically" and indexed by their discrete logarithms. The only information that A can learn in the SGM is by querying $\mathsf{Zt_T}$ oracle. Without loss of generality, we need to care only about the *successful* queries to the $\mathsf{Zt_T}$ oracle. Recall the challenge ciphertext $c^* = \big(C_1, C_2, \{C_{3,w}\}_{w\in[n'+1]}, C_4\big)$. The heart of this analysis is to prove properties about the coefficients that A assigns to the discrete logarithms of the elements in $(C_{3,1},\ldots,C_{3,n'+1})$. These are the most important coefficients because the above group elements are the only ones depending on the challenge attribute vector $\mathbf{y}_b$. Recall that $\forall w \in [n'+1]$, we have

$$C_{3,w} = h^{\widetilde{y}^b_w \cdot r+s} \cdot \widehat{U}_w^{-z} = g_1^{\widetilde{y}^b_w \cdot r\beta+s\beta} \cdot \widehat{U}_w^{-z} = g_1^{r\beta\widetilde{y}^b_w+s\beta-z\chi},$$

where $\chi \in \{u_1,\ldots,u_{n'},\mathsf{DL}(T)\}$. We now proceed with the following claims.

Claim 1 *The coefficients of all the terms* $(C_{3,1},\ldots,C_{3,n'+1})$ *that are* **not** *paired with some* A_i *must be equal to zero.*

Proof. Note that the only *symbolic* elements that A can access in $\mathbb{G}_2$, apart from $\mathtt{t_i}$ (representing A_i) for any $i\in[L]$, are the following:

1. 1 (representing g_2).
2. $\alpha+\beta\mathtt{t_i}$ (representing B_i), $\forall i\in[L]$.
3. $\frac{\mathtt{t_i u_0'}}{\gamma}$ (representing $\widetilde{W}_{i,0}$), $\forall i\in[L]$.
4. $\left\{\frac{\mathtt{t_i u_{w,j}}}{\gamma}\right\}_{\substack{j\in[0,L]\setminus\{i\}\\ w\in[n']}}$ (representing $W_{i,j,w}$), $\forall i\in[L]$.
5. $\left\{\frac{\widetilde{\mathtt{r}}^{\mathtt{c}}_{\mathtt{i}}\mathtt{t_j u_{n',i}}}{\gamma}\right\}_{\substack{j\in[L]\setminus\{i\}\\ c\in[Q_\mathsf{k}]}}$ (representing $W^{\widetilde{r}^c_i}_{j,i,n'}$ from the product $\widetilde{W}^c_{j,i}$ in pk_c), $\forall i\in[L]$.
6. Any arbitrary linear combination amongst the above 5 items (and possibly with $\mathtt{t_i}$).

Observe that items $2,3,4$ and 5 are all linearly independent. In particular, they do not cancel out internally as well as with each other. E.g., for $i,j\in[0,L], i\neq j$, $(\alpha+\beta\mathtt{t_i})$ cannot cancel out $(\alpha+\beta\mathtt{t_j})$, or they also do not cancel out with $\frac{\mathtt{t_i u_0'}}{\gamma},\frac{\mathtt{t_i u_{w,j}}}{\gamma}$ or $\frac{\widetilde{\mathtt{r}}^{\mathtt{c}}_{\mathtt{i}}\mathtt{t_j u_{n',i}}}{\gamma}$. We will now establish that they cannot cancel even when A uses the Map_e oracle to form products with the terms available to A from $\mathbb{G}_1$. We focus particularly on the terms representing $C_{3,w}$s' and C_4. The other terms in $\mathbb{G}_1$ from $\mathsf{crs}, \{\mathsf{pk}_c\}_{c\in[Q_\mathsf{k}]}$ and $C_2\in c^*$ follow from a simple inspection. For any $i\in[L]$, let us look at the coefficients of the symbolic representation for any $C_{3,w}$ (from above) multiplied with anything available from the same for elements in $\mathbb{G}_2$. For convenience, we argue both in the language of pairings and the equivalent symbolic setting in the following, switching between them as and when required. In particular, we focus on the variable $\mathtt{z}$, which is present only in the terms representing $C_{3,w}$ and C_4. Define $\Delta = \{\mathtt{u_1},\ldots,\mathtt{u_{n'}},\mathsf{DL}(\mathtt{T})\}$. Below we look at all possible pairings and show that they have linearly independent symbolic terms that cannot be cancelled out, so long as we forbid A_i in the pairing.

1. $e(C_{3,w},g_2)$: Here we have the *unique* terms $\mathtt{z}\chi,\forall\chi\in\Delta$ (no $\alpha,\beta,\gamma,\mathtt{t_i}$).
2. $e(C_{3,w},A_i)$: Here we have the *unique* terms $\mathtt{zt_i}\chi,\forall\chi\in\Delta$ (no α,β,γ). Note that this is the one that we are excluding, but we still have to consider it to make sure that it does not cancel out with the *other* pairings of $C_{3,w}$.
3. $e(C_{3,w},B_i)$: Here we have the *unique* terms $\mathtt{z}\alpha\chi$ and $\mathtt{zt_i}\beta\chi,\forall\chi\in\Delta$ (no γ).
4. $e(C_{3,w},\widetilde{W}_{i,0})$: Here we have the *unique* terms $\mathtt{zt_i u_0'}\chi/\gamma,\forall\chi\in\Delta$ (no α,β).
5. $e(C_{3,w},W_{i,j,w})$: Here we have the *unique* terms $\mathtt{zt_i u_{w,j}}\chi/\gamma$, $\forall\chi\in\Delta$ (no α,β).
6. $e(C_{3,w},W^{\widetilde{r}^c_i}_{j,i,n'})$: Here we have the *unique* terms $\mathtt{z}\widetilde{\mathtt{r}}^{\mathtt{c}}_{\mathtt{i}}\mathtt{t_j u_{n',i}}\chi/\gamma,\forall\chi\in\Delta$ (no α,β).
7. $e(C_4,g_2)$: Here we have the *unique* term $\mathtt{z}\gamma$ (no $\alpha,\beta,\mathtt{t_i}$).
8. $e(C_4,A_i)$: Here we have the *unique* term $\mathtt{z}\gamma\mathtt{t_i}$ (no α,β).
9. $e(C_4,B_i)$: Here we have the *unique* terms $\mathtt{z}\gamma\alpha$ (no β) and $\mathtt{z}\gamma\beta\mathtt{t_i}$ (no α).
10. $e(C_4,\widetilde{W}_{i,0})$: Here we have the *unique* term $\mathtt{zt_i u_0'}$ (no α,β).
11. $e(C_4,W_{i,j,w})$: Here we have the terms $\mathtt{zt_i u_{w,j}}$ (no α,β,γ). Note that this term includes terms from Item-(2) above setting $\chi=\mathtt{u_w}=\sum_{k=0}^{L}u_{w,k}$. Hence, it is *not* a unique term. But we are anyway excluding A_i from this inspection.
12. $e(C_4,W^{\widetilde{r}^c_i}_{j,i,n'})$: Here we have the *unique* terms $\mathtt{z}\widetilde{\mathtt{r}}^{\mathtt{c}}_{\mathtt{i}}\mathtt{t_j u_{n',i}}$ (no α,β,γ).

By inspection, we see that these monomials are indeed unique across all possible pairings (except A_i) and thus, are also linearly independent among each other. It is easy to see that the same holds true for the initial terms in $C_{3,w}$ as well. In particular, observe that for any $w \in [n'+1]$, $C_{3,w}$ is represented symbolically by the general term $(\mathtt{r}\beta\mathtt{y_w} + \mathtt{s}\beta - \mathtt{z}\chi)$ for $\chi \in \Delta$. The presence of $\mathtt{r}$ in $\mathtt{r}\beta y_w$ and β in $\mathtt{s}\beta$ makes all their possible combinations unique and linearly independent. Hence, no cross cancellations can occur if we forbid pairing $C_{3,w}$ with A_i. Formally, define

$$\Psi = \left\{1, \left\{\mathtt{t_i}, \alpha + \beta\mathtt{t_i}, \frac{\mathtt{t_i u'_0}}{\gamma}\right\}_{i\in[L]}, \left\{\frac{\mathtt{t_i u_{w,j}}}{\gamma}\right\}_{\substack{i\in[L],j\in[0,L]\setminus\{i\}\\ w\in[n']}}, \left\{\frac{\widetilde{\mathtt{r}}_\mathtt{i}^\mathtt{c}\mathtt{t_j u_{n',i}}}{\gamma}\right\}_{\substack{i\in[L],j\in[L]\setminus\{i\}\\ c\in[Q_\mathtt{k}]}}\right\}.$$

The above then implies that for any term $\psi \in \mathbb{Z}_q\text{-span}(\Psi) \setminus \mathbb{Z}_q\text{-span}(\{\mathtt{t_1}, \ldots, \mathtt{t_L}\})$:

$$\sum_{w\in[n'+1]} \eta_w \cdot (\mathtt{r}\beta\mathtt{y_w} + \mathtt{s}\beta - \mathtt{z}\chi) \cdot \psi = \psi \sum_{w\in[n'+1]} \eta_w \cdot (\mathtt{r}\beta\mathtt{y_w} + \mathtt{s}\beta - \mathtt{z}\chi) = 0.$$

To reiterate informally, we can consider the coefficients $C_{3,w}$ paired with *anything* (that is not A_i) individually since (i) all the "anything" are linearly independent and all "anything" paired with $C_{3,w}$ do not cancel out. We now show it must be the case that $\eta_w = 0, \forall w \in [n'+1]$. Substituting the discrete logarithm of $C_{3,w}$ and only looking at the terms involving $\mathtt{z}$, we have:

$$\sum_{w\in[n'+1]} \eta_w \cdot \mathsf{DL}\big(\widehat{\mathtt{U}}_\mathtt{w}^{-\mathtt{z}}\big) = -\mathtt{z} \sum_{w\in[n'+1]} \eta_w \cdot \mathsf{DL}\big(\widehat{\mathtt{U}}_\mathtt{w}\big) = 0.$$

We consider two cases:

– $\eta_{n'+1} = 0$: Recall that for all $w \in [n']$ we have

$$\mathsf{DL}\big(\widehat{\mathtt{U}}_\mathtt{w}\big) = \mathsf{DL}\left(\prod_{i\in[0,L]} g_1^{\mathtt{u_{w,i}}}\right) = \mathtt{u_w}.$$

The variable $\mathtt{u_w}$ is formed from the terms $\{\mathtt{u_{w,i}}\}_{\mathtt{i}\in[0,\mathtt{L}]}$ which are unit variables. Hence, the expression above never evaluates to 0 symbolically, unless all of the coefficients $(\eta_1, \ldots, \eta_{n'})$ are 0. Thus in this case, the coefficients must be all 0.

– $\eta_{n'+1} \neq 0$: Recall that

$$\mathsf{DL}\big(\widehat{\mathtt{U}}_{\mathtt{n'+1}}\big) = \mathsf{DL}\left(\prod_{\mathtt{i}\in[0,\mathtt{L}]} \mathtt{T_i}\right) = \mathsf{DL}(\mathtt{T}) = \mathsf{DL}(\mathtt{T_0}) + \sum_{\mathtt{i}\in[\mathtt{L}]} \mathsf{DL}(\mathtt{T_i}).$$

Further, recall that there is at least one honest public-key in the above sum (namely T_0). Hence, it follows that $\mathsf{DL}(\mathtt{T_0}) = -\mathtt{u'_0}$ is a unit variable. Note that the coefficient $\eta_{n'+1}$ is multiplied by $[\ldots] \cdot \mathtt{z} \cdot \mathtt{u'_0}$. The only other way to obtain the term $\mathtt{z} \cdot \mathsf{DL}(\mathtt{T_0})$ is to pair C_4 with $\widetilde{W}_{i,0}$ (this contains $\mathsf{DL}(\mathtt{T_0})$ as part of the user's key). However, depending on what $\psi \in \Psi$ is, we additionally have (or do not have) one of the following coefficients:

- $\psi = 1$: Here we do not have t_i.
- $\psi = \alpha + \beta\mathtt{t_i}$: Here we have an extra β.
- $\psi = \frac{\mathtt{t_i u_{w,j}}}{\gamma}$: Here we have an extra $1/\gamma$.
- $\psi = \frac{\widetilde{\mathtt{r}}_\mathtt{i}^\mathtt{c}\mathtt{t_j u_{n',i}}}{\gamma}$: Here we have the extra terms $\widetilde{\mathtt{r}}_\mathtt{i}^\mathtt{c}, 1/\gamma$.

Thus, $\eta_{n'+1}$ is multiplied with a variable that is not obtainable anywhere else. Hence, it must be the case that $\eta_{n'+1} = 0$. This establishes Claim 1.

Claim 2 *The coefficients of all the terms $(C_{3,1}, \ldots, C_{3,n'+1})$, when paired with A_i (for some $i \in [L]$), must form a vector orthogonal to $\mathbf{y}_b, \forall b \in \{0,1\}$.*

Proof. Recall that $\forall w \in [n'+1]$, the symbolic discrete logarithm of $C_{3,w}$ is $(\mathtt{r}\beta\mathtt{y_w} + \mathtt{s}\beta - \mathtt{z}\chi)$ for $\chi \in \Delta$. After mapping it to $\mathbb{G}_\mathsf{T}$ (for example, via pairing with $A_i = g_2^{\mathtt{t_i}}$ for some $i \in [L]$), the variable $\mathtt{r}$ cancels out only when

$$\sum_{w \in [n'+1]} \mathtt{r}\beta\mathtt{t_i} \cdot \mathtt{y_w} \cdot \eta_w = \mathtt{r}\beta\mathtt{t_i} \sum_{w \in [n'+1]} \eta_w \cdot \mathtt{y_w} = 0$$

where $(\eta_1 \ldots, \eta_{n'+1})$ denote the coefficients of the above terms. This must be the case as $\mathtt{r}$ is present in the first summand of $C_{3,w}$s' and nowhere else. This establishes Claim 2.

Claim 3 *The coefficients of all the terms $(C_{3,1}, \ldots, C_{3,n'+1})$ when paired with A_i (for some $i \in [L]$) must be of the form $c \cdot \widetilde{\mathbf{x}}_i$, for some non-zero constant $c \in \mathbb{Z}_q^+$, or all 0.*

Proof. Observe that Claim 2 also shows that the only way for A to obtain some information is to use the coefficients corresponding to a "valid" predicate $\mathbf{x}_i$ for some $i \in [L]$, (i.e., one which allows to decrypt the ciphertext). We establish in this claim that such an $\mathbf{x}_i$ is also registered.

On pairing $C_{3,w}$ with A_i for some $i \in [L]$, symbolically, a successful zero-test looks like:

$$\sum_{w \in [n'+1]} \eta_w \cdot (\mathtt{r}\beta\mathtt{y_w} + \mathtt{s}\beta - \mathtt{z}\chi) \cdot \mathtt{t_i} = 0 \tag{6}$$

In the above equation, we have $\chi = \mathtt{u_w}, \forall w \in [n']$ and for $w = n'+1$, we have $\chi = \mathsf{DL}(\mathtt{T})$. The case where $\eta_w = 0, \forall w \in [n'+1]$ is trivial. So we consider the case where there exists some $w \in [n'+1]$ such that $\eta_w \neq 0$. Note that it suffices to consider each $A_i (= g_2^{t_i})$ separately, since each $\mathtt{t_i}$ are unit variables, and in particular they are linearly independent. We look at the last term in above equation again, namely:

$$\sum_{w \in [n'+1]} -\eta_w \cdot \chi \cdot t_i \cdot z.$$

For $w \in [n']$, within each $\chi = \mathtt{u_w} = \sum_{k \in [0,L]} \mathtt{u_{w,k}}$, we isolate $\mathtt{u_{w,i}}$, since the terms $\{\mathtt{z}\mathtt{t_i}\mathtt{u_{w,i}}\}_{w \in [n']}$ are not present anywhere else, nor obtainable via any pairing. However, this it not the case for the terms $\mathtt{u_{w,j}}$ for $j \neq i$. So we can ignore them for clarity (though they are part of the sum). Similarly, for $w = n'+1$, we isolate the term $\mathsf{DL}(\mathtt{T_i})$ from $\chi = \mathsf{DL}(\mathtt{T}) = \sum_{k \in [L]} \mathsf{DL}(\mathtt{T_k})$. Equation (6) with Claim 2 then implies the following:

$$\begin{aligned}
&\sum_{w \in [n'+1]} \mathtt{s}\beta\mathtt{t_i} \cdot \eta_w - \sum_{w \in [n'+1]} \mathtt{z}\mathtt{t_i}\chi \cdot \eta_w = 0 \text{ (from Claim 2)} \\
\Rightarrow &\sum_{w \in [n']} \eta_w \cdot \mathtt{z}\mathtt{t_i} \cdot \mathtt{u_{w,i}} = \\
&\qquad -\eta_{n'+1} \cdot \mathtt{z}\mathtt{t_i} \cdot \mathsf{DL}(\mathtt{T_i}) \ (\because \mathtt{s}\beta\mathtt{t_i} \text{ and } \mathtt{z}\mathtt{t_i}\chi \text{ are linearly independent}) \\
\Rightarrow &\sum_{w \in [n']} \eta_w \cdot \mathtt{u_{w,i}} = \eta_{n'+1} \cdot \sum_{w \in [n']} \widetilde{x}_{w,i}\mathtt{u_{w,i}} \ (\because \mathsf{DL}(\mathtt{T_i}) = -\sum_{w \in [n']} \widetilde{x}_{w,i}\mathtt{u_{w,i}}) \\
\Rightarrow &(\eta_1, \ldots, \eta_{n'}) = \underbrace{\eta_{n'+1}}_{c} \cdot \widetilde{\mathbf{x}}_i \ (\because \{\mathtt{u_{w,i}}\}_{w \in [n']} \text{ are linearly independent})
\end{aligned}$$

If $c \neq 0$, this is already consistent with the claim. If $c = 0$, it must be that $\eta_w = 0, \forall w \in [n']$, since $\mathtt{u_{w,i}}$ are linearly independent variables that do not cancel out. This establishes Claim 3.
Claims 1 to 3 together implies that the only non-trivial queries that A can issue are by using vectors in the span of both "registered" *and* "valid" $\mathbf{x}_i \in \mathbb{Z}_q^{n^+}$. Thus, switching $\mathbf{y}_0$ to $\mathbf{y}_1$ does not create any difference in A's view. Hence, $\mathbf{H}_3 \approx \mathbf{H}_4$.

Lemma 4 **($\mathbf{H}_4(\lambda) \approx \mathbf{H}_{5,1}(\lambda)$).** $\mathbf{H}_4(\lambda)$ *and* $\mathbf{H}_{5,1}(\lambda)$ *are statistically indistinguishable.*

Proof. Both $\mathbf{H}_4(\lambda)$ and $\mathbf{H}_{5,1}(\lambda)$ are in SGM, and so Claims 1 to 3 still hold. However, note that A is admissible and we are in the setting where $m_0 \neq m_1$. Hence, A can only ask keys for predicates $\mathbf{x}_i \in \mathbb{Z}_q^{n^+}$ which are invalid (i.e. they do not allow decrypting c^*). In this case, by Claims 1 to 3, we have that $\eta_w = 0, \forall w \in [n'+1]$. Hence, we can safely ignore the ciphertext components $\{C_{3,w}\}_{w\in[n'+1]}$. The rest of the proof follows simply from the fact that A cannot get enough information about the component $C_1 = m_0 \cdot Z^{\mathtt{s}} = m_0 \cdot e(g_1, g_2)^{\alpha\mathtt{s}}$. In particular, $\mathbf{H}_4(\lambda)$ and $\mathbf{H}_{5,1}(\lambda)$ could be distinguished if A could gather information on $e(g_1, g_2)^{\alpha\mathtt{s}}$ (possibly using its oracles in SGM). From Table 2, note that no information about α (resp., $\mathtt{s}$) is ever released in $\mathbb{G}_1$ (resp., $\mathbb{G}_2$). So the only avenue left to A is to infer information about g_2^{α} (which can then be paired with $C_1 = g_1^{\mathtt{s}} \in c^*$). This is impossible as A only has access to $\mathsf{DL}(B_i) = \alpha + \beta\mathtt{t_i} \in \mathbb{G}_2$, where no other components allows it to learn anything about $\beta\mathtt{t_i} \in \mathbb{G}_2$. Hence, $\mathbf{H}_4(\lambda) \approx \mathbf{H}_{5,1}(\lambda)$.

Lemma 5 ($\mathbf{H}_{5,1}(\lambda) \approx \mathbf{H}_{5,2}(\lambda) \approx \mathbf{H}_{5,3}(\lambda)$). *$\mathbf{H}_{5,1}(\lambda)$, $\mathbf{H}_{5,2}(\lambda)$ and $\mathbf{H}_{5,3}(\lambda)$ are statistically indistinguishable.*

Proof. In $\mathbf{H}_{5,1}(\lambda)$, Z^s has been replaced with a uniformly sampled $\mathfrak{U} \in \mathbb{G}_\mathsf{T}$. Switching from m_0 to m_1 is thus information-theoretically secure, i.e., $\mathbf{H}_{5,1}(\lambda) \approx \mathbf{H}_{5,2}(\lambda)$. Further, an analysis similar to Lemma 4 shows that $\mathbf{H}_{5,2}(\lambda) \approx \mathbf{H}_{5,3}(\lambda)$.

Lemma 6 ($\mathbf{H}_4(\lambda) \approx \mathbf{H}_6(\lambda)$ or $\mathbf{H}_{5,3}(\lambda) \approx \mathbf{H}_6(\lambda)$). *$\mathbf{H}_4(\lambda) \approx \mathbf{H}_6(\lambda)$ are statistically indistinguishable (when $m_0 = m_1$). Else, $\mathbf{H}_{5,3}(\lambda)$ and $\mathbf{H}_6(\lambda)$ are statistically indistinguishable.*

Proof. The proof follows similar to Lemmas 1 and 2 (in the reverse order).

Final pairing-based RIPE scheme. By combining the slotted RIPE scheme of Construction 1 and the ("power-of-two") transformation of Construction 3 (Appendix B), we obtain the following corollary.

Corollary 1. *In the* GGM*, there exists a secure and perfectly correct RIPE scheme with message space $\mathcal{M} = \mathbb{G}_\mathsf{T}$ and attribute space $\mathcal{U} = \mathbb{Z}_q^{n^+}$, satisfying the following properties:*

- $(n \cdot L^2 \cdot \mathsf{poly}(\lambda, \log L), n \cdot \mathsf{poly}(\lambda, \log L), n \cdot \mathsf{poly}(\lambda, \log L))$*-compactness, and*
- $(L \cdot \mathsf{poly}(\lambda, \log L), n \cdot L^2 \cdot \mathsf{poly}(\lambda, \log L), O(\log L), n \cdot \mathsf{poly}(\lambda, \log L))$*-efficiency.*

Recall that L stands for the maximum bound on the number of supported users (bounded case).

Proof. The corollary follows by combining Theorems 3 to 6 and 11 to 13. □

7 Slotted RFE from Indistinguishability Obfuscation

Here, we build slotted RFE for arbitrary (exponentially large) function spaces. The construction leverages iO, SSB hash functions, and PRGs.

Construction 2 *Let $\mathcal{F} = \{f_i : \mathcal{M} \to \mathcal{Y}\}$ be a function space of exponential size. Without loss of generality, we assume that any function $f_i \in \mathcal{F}$ can be described (in binary) using $O(\log(|\mathcal{F}|))$ bits. Also, consider the following ingredients:*

- *A length-doubling PRG* $\mathsf{G} : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$.
- *A SSB scheme* $\Pi_{\mathsf{SSB}} = (\mathsf{SSB.Setup}, \mathsf{SSB.Hash}, \mathsf{SSB.Open}, \mathsf{SSB.Verify})$.
- *An iO obfuscator* Obf.

We build a slotted RFE scheme $\Pi_{\mathsf{sRFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M} = \{0,1\}^*$ *and function space* $\mathcal{F} = \{f_i : \mathcal{M} \to \mathcal{Y}\}$ *as follows:*

$\mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|)$**:** *On input the security parameter 1^λ, the slot parameter 1^L, and the size $|\mathcal{F}|$ (in binary) of the function space $\mathcal{F}$, the randomized setup algorithm sets $\ell_{\mathsf{blk}} = 2\lambda + O(\log(|\mathcal{F}|))$ and samples an hash key* $\mathsf{hk} \leftarrow_\$ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$*. It outputs* $\mathsf{crs} = \mathsf{hk}$.

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

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

$\mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i, f_i))_{i\in[L]})$: *On input the common reference string* $\mathsf{crs} = \mathsf{hk}$, *and* L *pairs* $(\mathsf{pk}_1, f_1), \ldots, (\mathsf{pk}_L, f_L)$ *each composed of a public key* $\mathsf{pk}_i \in \{0,1\}^{2\lambda}$ *and a function* $f_i \in \mathcal{F}$ *(recall that* f_i *can represented using a binary string of size* $O(\log(|\mathcal{F}|))$*), the deterministic aggregation algorithm sets* $\mathsf{mpk} = (\mathsf{hk}, h)$ *where* $h = \mathsf{SSB.Hash}(\mathsf{hk}, ((\mathsf{pk}_i, f_i))_{i\in[L]})$. *Then, for each user* $i \in [L]$, *it computes the helper decryption key* $\mathsf{hsk}_i = (i, \mathsf{pk}_i, f_i, \pi_i)$ *where* $\pi_i = \mathsf{SSB.Open}(\mathsf{hk}, ((\mathsf{pk}_j, f_j))_{j\in[L]}, i)$. *Finally, it outputs* mpk *and* $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$.

$\mathsf{Enc}(\mathsf{mpk}, m)$: *On input the master public key* $\mathsf{mpk} = (\mathsf{hk}, h)$ *and a message* $m \in \mathcal{M}$, *the randomized encryption algorithm outputs* $c = C' \leftarrow\$ \mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m})$ *where the circuit* $C_{\mathsf{hk},h,m}$ *is defined in the following way:*

$C_{\mathsf{hk},h,m}(i, \mathsf{pk}_i, f_i, \pi_i, \mathsf{sk}_i)$
If $\mathsf{SSB.Verify}(\mathsf{hk}, h, i, (\mathsf{pk}_i, f_i), \pi_i) = 1 \wedge \mathsf{pk}_i = \mathsf{G}(\mathsf{sk}_i)$ **then: return** $f_i(m)$
Else: return $\perp$

Circuit $C_{\mathsf{hk},h,m}$ *is padded to match the size* $\gamma = \max\{C_{\mathsf{hk},h,m}, C_{\mathsf{hk},h,m,m',0}, \ldots, C_{\mathsf{hk},h,m,m',L}\}$ *where* $C_{\mathsf{hk},h,m,m',j}$ *is defined in Theorem 10.*

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, c)$: *On input the secret key* sk, *the helper decryption key* $\mathsf{hsk} = (i, \mathsf{pk}_i, f_i, \pi_i)$, *and a ciphertext* $c = C'$, *the deterministic decryption algorithm outputs* $C'(i, \mathsf{pk}_i, f_i, \pi_i, s)$.

We start with proving that Construction 2 is complete, correct, and efficient.

Theorem 7 (Completeness of Construction 2). *Let* G, $\mathit{\Pi}_{\mathsf{SSB}}$, Obf *as above. The slotted RFE scheme* $\mathit{\Pi}_{\mathsf{sRFE}}$ *from Construction 2 is complete (Definition 10).*

Proof. Completeness follows by observing that $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1$ whenever $|\mathsf{pk}_i| = 2\lambda$ (the output size of the PRG G). □

Theorem 8 (Perfect correctness of Construction 2). *Let* G, $\mathit{\Pi}_{\mathsf{SSB}}$, Obf *as above. If* $\mathit{\Pi}_{\mathsf{SSB}}$ *is perfectly correct (Definition 2), and* Obf *is correct (Definition 1), then the slotted RFE scheme* $\mathit{\Pi}_{\mathsf{sRFE}}$ *from Construction 2 is correct (Definition 11).*

Proof. Correctness follows by by the correctness of the underlying $\mathit{\Pi}_{\mathsf{SSB}}$ and Obf schemes. □

Theorem 9 (Compactness and efficiency of Construction 2). *Let* G, $\mathit{\Pi}_{\mathsf{SSB}}$, Obf *as above. If* $\mathit{\Pi}_{\mathsf{SSB}}$ *is succinct and efficient (Definition 5), then the slotted RFE scheme* $\mathit{\Pi}_{\mathsf{sRFE}}$ *from Construction 2 is*

- $(\mathsf{poly}(\lambda, \log L, \log|\mathcal{F}|), \mathsf{poly}(\lambda, \log L, \log|\mathcal{F}|), \mathsf{poly}(\lambda, \log L, \log|\mathcal{F}|))$*-compact;*
- $(\mathsf{poly}(\lambda), \mathsf{poly}(\lambda), L \cdot \mathsf{poly}(\lambda, \log L, \log|\mathcal{F}|))$*-efficient (Definition 12).*

Proof. We demonstrate each property individually.

- $\mathsf{poly}(\lambda, \log L, \log|\mathcal{F}|)$**-compact** crs**:** The common reference string crs is composed of $\mathsf{hk} \leftarrow\$ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$. Hence, the property holds by leveraging the succinctness of $\mathit{\Pi}_{\mathsf{SSB}}$, i.e., the size of crs is bounded by $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log L)$ where $\ell_{\mathsf{blk}} = 2\lambda + O(\log|\mathcal{F}|)$.
- $\mathsf{poly}(\lambda, \log L, \log|\mathcal{F}|)$**-compact** mpk**:** The master public keys mpk are composed of $\mathsf{mpk} = (\mathsf{hk}, h)$ where $\mathsf{k} \leftarrow\$ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$ and $h = \mathsf{SSB.Hash}(\mathsf{hk}, ((\mathsf{pk}_i, f_i))_{i\in[L]})$. Hence, the property holds by leveraging the succinctness of $\mathit{\Pi}_{\mathsf{SSB}}$, i.e., the size of mpk is bounded by $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log L)$ where $\ell_{\mathsf{blk}} = 2\lambda + O(\log(|\mathcal{F}|))$.
- $\mathsf{poly}(\lambda, \log L, \log|\mathcal{F}|)$**-compact** hsk**:** The helper decryption keys hsk_i are composed of $\mathsf{hsk}_i = (i, \mathsf{pk}_i, f_i, \pi_i)$ where $\pi_i = \mathsf{SSB.Open}(\mathsf{hk}, ((\mathsf{pk}_j, f_j))_{j\in[L]}, i)$ and $\mathsf{pk}_i \in \{0,1\}^{2\lambda}$. By leveraging the compactness of SSB we have $|\pi_i| \leq \mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log L)$ where $\ell_{\mathsf{blk}} = 2\lambda + O(\log|\mathcal{F}|)$. Moreover, by definition we have $|\mathsf{pk}_i| = 2\lambda$, $|f_i| \leq O(\log|\mathcal{F}|)$, and $|i| \leq O(\log L)$. Hence, the size of hsk is bounded by $\mathsf{poly}(\lambda, \log L, \log(|\mathcal{F}|))$

- $\mathsf{poly}(\lambda)$**-efficient** KGen**:** The key generation algorithm KGen performs a single PRG evaluation. Hence, the running time of KGen is bounded by $\mathsf{poly}(\lambda)$.
- $\mathsf{poly}(\lambda)$**-efficient** $\mathsf{IsValid}$**:** The validation algorithm simply checks if $|\mathsf{pk}_i| \leq 2\lambda$. Hence, $\mathsf{IsValid}$ running time is polynomial in the security parameter.
- $(L \cdot \mathsf{poly}(\lambda, \log L, \log |\mathcal{F}|))$**-efficient** Aggr**:** The aggregation algorithm Aggr executes $\mathsf{SSB.Hash}$ once. Moreover, it executes L times (one for each aggregated public key) the SBB opening algorithm $\mathsf{SSB.Open}$. Hence, by leveraging the efficiency of SSB (i.e., running times of $\mathsf{SSB.Hash}$ and $\mathsf{SSB.Open}$ are bounded by $L \cdot \mathsf{poly}(\lambda, \ell_{\mathsf{blk}})$ and $\mathsf{poly}(\lambda, \ell_{\mathsf{blk}}, \log L)$), we have that the running time of Aggr is bounded by $L \cdot \mathsf{poly}(\lambda, \log L, \log |\mathcal{F}|)$ where $\ell_{\mathsf{blk}} = 2\lambda + O(\log |\mathcal{F}|)$.

□

We now prove the formal security for Construction 2 below.

Theorem 10 (Security). *Let* G, Π_{SSB}, Obf *as above. If* G *is secure (Definition 6),* Π_{SSB} *is index hiding (Definition 3) and somewhere statistically binding (Definition 4), and* Obf *is secure (Definition 1), then the slotted RFE scheme* Π *from Construction 2 is secure (Definition 13).*

Proof. Consider the following hybrid experiments:

$\mathbf{H}^b_{-1}(\lambda)$**:** This is exactly the experiment $\mathbf{Game}^{\mathsf{slot\text{-}rfe}}_{\Pi_{\mathsf{sRFE}},\mathsf{A}}(\lambda, b)$ where the challenge bit is b.

$\mathbf{H}^b_0(\lambda)$**:** Same as $\mathbf{H}^b_{-1}$ except that the challenge ciphertext is computed as $c = C' \leftarrow\$ \mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},j})$ where $j = 0$, m^*_0 and m^*_1 are the challenge messages output by the adversary, and the circuit $C_{\mathsf{hk},h,m,m',j}$ is defined as follows:

$C_{\mathsf{hk},h,m,m',j}(i, \mathsf{pk}_i, f_i, \pi_i, \mathsf{sk}_i)$

If $\mathsf{SSB.Verify}(\mathsf{hk}, h, i, (\mathsf{pk}_i, f_i), \pi_i) = 1 \wedge \mathsf{pk}_i = \mathsf{G}(\mathsf{sk}_i)$ **then:**
  If $i > j$ **then: return** $f_i(m)$
  Else: return $f_i(m')$
Else: return $\perp$

The circuit $C_{\mathsf{hk},h,m,m',j}$ is padded to match the size γ defined as $\gamma = \max\{C_{\mathsf{hk},h,m}, C_{\mathsf{hk},h,m,m',0}, \dots, C_{\mathsf{hk},h,m,m',L}\}$ where $C_{\mathsf{hk},h,m}$ is defined in Construction 2.

$\mathbf{H}^b_i(\lambda)$**:** Same as $\mathbf{H}^b_{i-1}$ except that the challenge ciphertext is computed as $c = C' \leftarrow\$ \mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},j})$ where $j = i$ (instead of $j = i - 1$).

$\mathbf{H}^b_{L+1}(\lambda)$**:** Same as $\mathbf{H}^b_L$ except that the challenge ciphertext is computed as $c = C' \leftarrow\$ \mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_{1-b}})$ where the circuit $C_{\mathsf{hk},h,m}$ is defined as in Construction 2. Observe that this is exactly the experiment $\mathbf{H}^{1-b}_{-1}(\lambda)$.

Also, consider the following intermediate hybrid experiment that will help us demonstrating the computational indistinguishability of $\mathbf{H}^b_i$ and $\mathbf{H}^b_{i+1}$:

$\widetilde{\mathbf{H}}^b_i(\lambda)$**:** Same as $\mathbf{H}^b_i$ except that the challenger computes $\mathsf{hk} \leftarrow\$ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i + 1)$ (instead of $\mathsf{hk} \leftarrow\$ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$).

We now prove the following lemmas.

Lemma 7. $\mathbf{H}^b_{-1}(\lambda) \approx_c \mathbf{H}^b_0(\lambda)$*, for* $b \in \{0, 1\}$*.*

Proof. The lemma follows by simply observing that $C_{\mathsf{hk},h,m^*_b,j}$ and $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},j}$ are functionally-equivalent when $j = 0$. Hence, the lemma follows by the security of the iO obfuscator Obf.

Lemma 8. $\mathbf{H}^b_i(\lambda) \approx_c \widetilde{\mathbf{H}}^b_i(\lambda)$*, for* $b \in \{0, 1\}$ *and* $i \in \{0\} \cup [L]$*.*

Proof. Suppose that exists a PPT adversary A with a non-negligible advantage in distinguishing between $\mathbf{H}^b_i(\lambda)$ and $\widetilde{\mathbf{H}}^b_i(\lambda)$. We construct an adversary A' that breaks the index hiding property of Π_{SSB}. A' is defined as follows:

1. Receive the number of slots 1^L by A.
2. Send the parameters $\ell_{\mathsf{blk}} = 2\lambda + \log(|\mathcal{F}|)$, $N = L$, and the challenge indexes $(i_0 = 1, i_1 = i+1)$ to the challenger. The challenger will play the index hiding experiment with respect to ℓ_{blk}, N and indexes (i_0, i_1).
3. Receive hk^* from the challenger and send $\mathsf{crs} = \mathsf{hk}^*$ to A.
4. Play the rest of the experiment as defined in $\mathbf{H}_i^b(\lambda)$.
5. Return the output of A.

Let d be the challenge bit sampled by the challenger. If $d = 0$, A' correctly simulates $\mathbf{H}_i^b(\lambda)$ since hk^* is generated as $\mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, 1)$. On the other hand, if $d = 1$, A' simulates $\widetilde{\mathbf{H}}_i^b(\lambda)$ since hk^* is generated as $\mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i+1)$. Thus, A' has the same non-negligible advantage of A. This concludes the proof.

Lemma 9. *$\widetilde{\mathbf{H}}_i^b(\lambda) \approx_c \widetilde{\mathbf{H}}_{i+1}^b(\lambda)$, for $b \in \{0,1\}$ and $i \in \{0\} \cup [L-1]$.*

Proof. Fix $i \in \{0\} \cup [L-1]$. Let $\mathbf{NonCorrupt}_i$ be an event that occurs when the following two conditions hold:

1. $c_{i+1}^* \in [\mathsf{ctr}]$ (see Definition 13). This implies that the $(i+1)$-th public key pk_{i+1}^* (chosen by the adversary during the challenge phase) has been generated by the challenger on the (c_{i+1}^*)-th key-generation query submitted by the adversary.
2. $c_{i+1}^* \notin \mathcal{Q}_{\mathsf{Corr}}$ where $\mathcal{Q}_{\mathsf{Corr}}$ is the set of corruption queries submitted by the adversary to the corruption oracle during the query phase.

Observe that

$$\mathbb{P}\Big[\widetilde{\mathbf{H}}_i^b(\lambda) = 1\Big] = \mathbb{P}\Big[\widetilde{\mathbf{H}}_i^b(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big] + \mathbb{P}\Big[\widetilde{\mathbf{H}}_i^b(\lambda) = 1 \wedge \overline{\mathbf{NonCorrupt}_i}\Big],$$
$$\mathbb{P}\Big[\widetilde{\mathbf{H}}_{i+1}^b(\lambda) = 1\Big] = \mathbb{P}\Big[\widetilde{\mathbf{H}}_{i+1}^b(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big] + \mathbb{P}\Big[\widetilde{\mathbf{H}}_{i+1}^b(\lambda) = 1 \wedge \overline{\mathbf{NonCorrupt}_i}\Big].$$

Hence, it suffices to prove that the following two equations hold:

$$\Big|\mathbb{P}\Big[\widetilde{\mathbf{H}}_i^b(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big] - \mathbb{P}\Big[\widetilde{\mathbf{H}}_{i+1}^b(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big]\Big| \le \mathsf{negl}(\lambda), \text{ and} \tag{7}$$
$$\Big|\mathbb{P}\Big[\widetilde{\mathbf{H}}_i^b(\lambda) = 1 \wedge \overline{\mathbf{NonCorrupt}_i}\Big] - \mathbb{P}\Big[\widetilde{\mathbf{H}}_{i+1}^b(\lambda) = 1 \wedge \overline{\mathbf{NonCorrupt}_i}\Big]\Big| \le \mathsf{negl}(\lambda). \tag{8}$$

Indeed, $\widetilde{\mathbf{H}}_i^b(\lambda) \approx_c \widetilde{\mathbf{H}}_{i+1}^b(\lambda)$ would then follow by the triangular inequality and the combination of Equations (7) and (8).

Claim 4 (Equation (7)) *If $\mathbf{NonCorrupt}_i$ occurs then $\widetilde{\mathbf{H}}_i^b(\lambda) \approx_c \widetilde{\mathbf{H}}_{i+1}^b(\lambda)$, for $b \in \{0,1\}$ and $i \in [0, L-1]$.*

Proof. This claim implies that Equation (7) holds. The proof relies on the fact that $\mathbf{NonCorrupt}_i$ occurs, i.e., the $(i+1)$-th slot is not corrupted.

Consider the following intermediate hybrid experiments:

$\widetilde{\mathbf{H}}_{1,i}^b(\lambda)$: Same as $\widetilde{\mathbf{H}}_i^b(\lambda)$ except that the challenger samples $k \leftarrow\!\!\$\, [K]$ where $K = K(\lambda)$ is a bound on the number of key generation queries submitted by the adversary during the query phase. Let pk_k be the public key returned by the challenger as the answer of the k-th key generation query (if there is one). The challenger aborts if either of the following condition hold:
1. The challenge $(i+1)$-th tuple $(c_{i+1}^*, f_{i+1}^*, \mathsf{pk}_{i+1}^*)$ (chosen by the adversary during the challenge phase) satisfies $c_{i+1}^* \neq k$.
2. $k \in \mathcal{Q}_{\mathsf{Corr}}$ where $\mathcal{Q}_{\mathsf{Corr}}$ is the set of corruption queries submitted by the adversary during the query phase.

Otherwise, the challenger proceeds as in $\widetilde{\mathbf{H}}_i^b(\lambda)$.

$\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$**:** Same as $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$ except that pk_k is sampled at random from $\{0,1\}^{2\lambda}$.

$\widetilde{\mathbf{H}}^b_{3,i}(\lambda)$**:** Same as $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$ except that the challenge ciphertext is computed as $c = C' \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},j})$ where $j = i+1$.

$\widetilde{\mathbf{H}}^b_{4,i}(\lambda)$**:** Same as $\widetilde{\mathbf{H}}^b_{3,i}(\lambda)$ except that pk_k is computed as $\mathsf{pk}_k = \mathsf{G}(s)$ where $s \leftarrow\!\!\$\ \{0,1\}^\lambda$.

$\widetilde{\mathbf{H}}^b_{5,i}(\lambda)$**:** Same as $\widetilde{\mathbf{H}}^b_{4,i}(\lambda)$ except that for the following differences:

If $i+2 \leq L$**:** The challenger computes $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i{+}2)$ (instead of $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i+1)$).

If $i+2 > L$**:** $\widetilde{\mathbf{H}}^b_{5,i}(\lambda)$ is identical to $\widetilde{\mathbf{H}}^b_{4,i}(\lambda)$.

Subclaim 1 *For $b \in \{0,1\}$ and $i \in \{0\} \cup [L-1]$, we have that*

$$\mathbb{P}\Big[\widetilde{\mathbf{H}}^b_i(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big] = K \cdot \mathbb{P}\Big[\widetilde{\mathbf{H}}^b_{1,i}(\lambda) = 1\Big].$$

Proof. $\widetilde{\mathbf{H}}^b_{i,1}(\lambda)$ and $\widetilde{\mathbf{H}}^b_i(\lambda)$ are identical except for the aborting condition of $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$. If $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$ outputs 1 (i.e., the challenger does not abort during the execution of $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$), then it must be that $\widetilde{\mathbf{H}}^b_i(\lambda) = 1$, $c_{i+1} = k$, and $k \notin \mathcal{Q}_{\mathsf{Corr}}$. In other words, the event $\mathbf{NonCorrupt}_i$ must occur. As a conseguence,

$$\begin{aligned}
\mathbb{P}\Big[\widetilde{\mathbf{H}}^b_{1,i}(\lambda) = 1\Big] &= \mathbb{P}\Big[\widetilde{\mathbf{H}}^b_i(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i \wedge k = c^*_{i+1}\Big] \\
&= \mathbb{P}\Big[k = c_{i+1} \;\Big|\; \widetilde{\mathbf{H}}^b_i(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big] \cdot \mathbb{P}\Big[\widetilde{\mathbf{H}}^b_i(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big] \\
&= 1/K \cdot \mathbb{P}\Big[\widetilde{\mathbf{H}}^b_i(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\Big],
\end{aligned}$$

In the latter equality we used the fact that $c^*_{i+1} \in [\mathsf{ctr}] \subseteq [K]$ and the challenger (in $\widetilde{\mathbf{H}}^b_i(\lambda)$) samples k randomly in $[K]$.

Subclaim 2 $\widetilde{\mathbf{H}}^b_{1,i}(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{2,i}(\lambda)$*, for $b \in \{0,1\}$ and $i \in \{0\} \cup [L-1]$.*

Proof. Suppose there exists a PPT adversary A with a non-negligible advantage in distinguishing between $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$. We construct an adversary A' that breaks the security of the PRG G. A' is defined as follows:

1. Receive y from the challenger.
2. Sample $k \leftarrow\!\!\$\ [K]$ and play the rest of the experiment as defined in $\widetilde{\mathbf{H}}^b_{i,1}(\lambda)$ except that, on k-th key-generation query, set $\mathsf{pk}_k = y$. Moreover, if A submits k to the corruption oracle, A' aborts as in $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$.
3. Return the output of A.

Note that both $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$ output 0 (i.e., abort condition) if the adversary A submits a corruption query on index k. This means that A' does not need to know the seed $s \leftarrow\!\!\$\ \{0,1\}^\lambda$ (i.e., the secret key associated to the k-th public key) sampled by the challenger during our reduction. Having said that, if $y = \mathsf{G}(s)$, A' perfectly simulates $\widetilde{\mathbf{H}}^b_{1,i}(\lambda)$. On the other hand, if $y \leftarrow\!\!\$\ \{0,1\}^{2\lambda}$, A perfectly simulates $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$. This concludes the proof.

Subclaim 3 $\widetilde{\mathbf{H}}^b_{2,i}(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{3,i}(\lambda)$*, for $b \in \{0,1\}$ and $i \in \{0\} \cup [L-1]$.*

Proof. The only difference between $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{3,i}(\lambda)$ is that the challenge ciphertext is computed in one as $\mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i})$ and $\mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i+1})$. We show that, with overwhelming probability over the choice of hk and pk_k, these two circuits are functionally-equivalent. Let $x = (i_x, \mathsf{pk}_x, f_x, \pi_x, \mathsf{sk}_x)$ be an input for the above circuits. Then, the following conditions hold:

Case $i_x \neq i+1$**:** Both circuits $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i}$ and $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i+1}$ have identical input/output behavior.

Case $i_x = i+1 \wedge (\mathsf{pk}_x, f_x) \neq (\mathsf{pk}^*_{i+1}, f^*_{i+1})$: In both $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{3,i}(\lambda)$, hk and h are generated as $\mathsf{hk} \leftarrow\!\!\$\ \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i+1)$ and $h = \mathsf{SSB.Hash}(\mathsf{hk}, ((\mathsf{pk}^*_1, f^*_1), \ldots, (\mathsf{pk}^*_L, f^*_L)))$. This means that, with overwhelming probability over the choice of hk, the SSB's instantiation (with respect to hk) is statistically binding (Definition 4) on position $i+1$. As a consequence, with overwhelming probability, there does not exist a $(\mathsf{pk}_x, f_x, \pi_x)$ such that $(\mathsf{pk}_x, f_x) \neq (\mathsf{pk}^*_{i+1}, f^*_{i+1})$ and $\mathsf{SSB.Ver}(\mathsf{hk}, h, i+1, (\mathsf{pk}_x, f_x), \pi_x) = 1$. This implies that, with overwhelming probability, both circuits output $\bot$.

Case $i_x = i+1 \wedge (\mathsf{pk}_x, f_x) = (\mathsf{pk}^*_{i+1}, f^*_{i+1})$: Assume that $\widetilde{\mathbf{H}}^b_{2,i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{3,i}(\lambda)$ do not abort. This implies that $\mathsf{pk}_x = \mathsf{pk}^*_{i+1} = \mathsf{pk}_k$ is sampled at random from $\{0,1\}^{2\lambda}$ where pk_k is the public key sampled in the k-th key-generation query. Since G is a length-doubling PRG, the following probability hold:

$$\mathbb{P}\left[\exists \mathsf{sk} \in \{0,1\}^\lambda, \mathsf{G}(\mathsf{sk}) = \mathsf{pk}_k \;\middle|\; \mathsf{pk}_k \leftarrow\!\!\$\ \{0,1\}^{2\lambda}\right] \leq 2^\lambda/2^{2\lambda} = 2^{-\lambda}.$$

This implies that, with overwhelming probability over the choice of pk_k, both circuits output $\bot$.

By combining the above cases, we conclude that the circuits $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i}$ and $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i+1}$ are functionally-equivalent with overwhelming probability over the choice of hk and pk_k. Hence, the claim follows by the security of the iO obfuscator Obf.

Subclaim 4 $\widetilde{\mathbf{H}}^b_{3,i}(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{4,i}(\lambda)$*, for* $b \in \{0,1\}$ *and* $i \in \{0\} \cup [L-1]$.

Proof. The claim follows by using an identical argument to that of Subclaim 2.

Subclaim 5 $\widetilde{\mathbf{H}}^b_{4,i}(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{5,i}(\lambda)$*, for* $b \in \{0,1\}$ *and* $i \in \{0\} \cup [L-1]$.

Proof. By definition, $\widetilde{\mathbf{H}}^b_{4,L-1}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{5,L-1}(\lambda)$ are identical, for $i = L-1$. On the other hand, for $i < L-1$, the claim follows by the index hiding property of SSB and the proof is identical to that of Lemma 8.

Subclaim 6 *For* $b \in \{0,1\}$ *and* $i \in \{0\} \cup [L-1]$*, we have that*

$$\mathbb{P}\left[\widetilde{\mathbf{H}}^b_{i+1}(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\right] = K \cdot \mathbb{P}\left[\widetilde{\mathbf{H}}^b_{5,i}(\lambda) = 1\right].$$

Proof. The claim follows by using an identical argument to that of Subclaim 1.

By combining Subclaims 2 to 4 we have that $\widetilde{\mathbf{H}}^b_{1,i}(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{5,i}(\lambda)$. Moreover, by leveraging Subclaims 1 and 6 we conclude that:

$$\mathbb{P}\left[\widetilde{\mathbf{H}}^b_{i}(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\right] = K \cdot \mathbb{P}\left[\widetilde{\mathbf{H}}^b_{1,i}(\lambda) = 1\right] \leq K \cdot \left(\mathbb{P}\left[\widetilde{\mathbf{H}}^b_{5,i}(\lambda) = 1\right] + \mathsf{negl}(\lambda)\right), \text{ and}$$
$$\mathbb{P}\left[\widetilde{\mathbf{H}}^b_{i+1}(\lambda) = 1 \wedge \mathbf{NonCorrupt}_i\right] = K \cdot \mathbb{P}\left[\widetilde{\mathbf{H}}^b_{5,i}(\lambda) = 1\right].$$

By taking into account that $K \in \mathsf{poly}$, the above equations imply that $\widetilde{\mathbf{H}}^b_i(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{i+1}(\lambda)$ whenever $\mathbf{NonCorrupt}_i$ occurs (Equation (7)). This concludes the proof of Claim 4.

Claim 5 (Equation (8)) *If* $\overline{\mathbf{NonCorrupt}_i}$ *occurs then* $\widetilde{\mathbf{H}}^b_i(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{i+1}(\lambda)$*, for* $b \in \{0,1\}$ *and* $i \in \{0\} \cup [L-1]$.

Proof.

This claim implies that Equation (8) holds. The proof relies on the fact that $\overline{\mathbf{NonCorrupt}_i}$ occurs, i.e., the $(i+1)$-th slot is corrupted. Hence, the adversary must be valid, i.e., $f^*_{i+1}(m^*_0) = f^*_{i+1}(m^*_1)$.

Consider the following intermediate hybrid experiments:

$\widetilde{\mathbf{H}}^b_{6,i}(\lambda)$: Same as $\widetilde{\mathbf{H}}^b_i(\lambda)$ except that the challenge ciphertext is computed as $c = C' \leftarrow\!\!\$\ \mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},j})$ where $j = i+1$.

$\widetilde{\mathbf{H}}^b_{7,i}(\lambda)$: Same as $\widetilde{\mathbf{H}}^b_{6,i}(\lambda)$ except for the following differences:

If $i+2 \leq L$**:** The challenger computes $\mathsf{hk} \leftarrow_{\$} \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i{+}2)$ (instead of $\mathsf{hk} \leftarrow_{\$} \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i+1)$).

If $i+2 > L$**:** $\widetilde{\mathbf{H}}^b_{7,i}(\lambda)$ is identical to $\widetilde{\mathbf{H}}^b_{6,i}(\lambda)$.

Note that, in both cases, $\widetilde{\mathbf{H}}^b_{7,i}(\lambda)$ is exactly the experiment $\widetilde{\mathbf{H}}^b_{i+1}(\lambda)$.

Subclaim 7 *If* $\overline{\mathbf{NonCorrupt}_i}$ *occurs then* $\widetilde{\mathbf{H}}^b_{i}(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{6,i}(\lambda)$*, for* $b \in \{0,1\}$ *and* $i \in \{0\} \cup [L-1]$.

Proof. The only difference between $\widetilde{\mathbf{H}}^b_{i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{6,i}(\lambda)$ is that the challenge ciphertext is computed as $\mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i})$ and $\mathsf{Obf}(1^\lambda, C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i+1})$, respectively. We show that, with overwhelming probability over the choice of hk, these two circuits are functionally-equivalent. Let $x = (i_x, \mathsf{pk}_x, f_x, \pi_x, \mathsf{sk}_x)$ be an input for the above circuits. Then, the following conditions hold:

Case $i_x \neq i+1$**:** Both circuits $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i}$ and $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i+1}$ have identical input/output behavior.

Case $i_x = i+1 \wedge (\mathsf{pk}_x, f_x) \neq (\mathsf{pk}^*_{i+1}, f^*_{i+1})$**:** In both $\widetilde{\mathbf{H}}^b_{i}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{i+1}$, hk is generated as $\mathsf{hk} \leftarrow_{\$} \mathsf{SSB.Setup}(1^\lambda, 1^{\ell_{\mathsf{blk}}}, L, i{+}1)$. Hence, with overwhelming probability over the choice of hk, the SSB's scheme is somewhere statistically binding (Definition 4) on position $i+1$, i.e., there does not exist a $(\mathsf{pk}_x, f_x, \pi_x)$ such that $(\mathsf{pk}_x, f_x) \neq (\mathsf{pk}^*_{i+1}, f^*_{i+1})$ and $\mathsf{SSB.Ver}(\mathsf{hk}, h, i{+}1, (\mathsf{pk}_x, f_x), \pi_x) = 1$. This implies that, with overwhelming probability, both circuits output $\perp$.

Case $x = i+1 \wedge (\mathsf{pk}_x, f_x) = (\mathsf{pk}^*_{i+1}, f^*_{i+1})$**:** We consider the following cases.

1. If $\mathsf{SSB.Ver}(\mathsf{hk}, h, i+1, (\mathsf{pk}_x, f_x), \pi_x) = 0 \vee \mathsf{G}(\mathsf{sk}_x) \neq \mathsf{pk}_x$, both circuits output $\perp$.
2. Otherwise (i.e., $\mathsf{SSB.Ver}(\mathsf{hk}, h, i{+}1, (\mathsf{pk}_x, f_x), \pi_x) = 1 \wedge \mathsf{G}(\mathsf{sk}_x) = \mathsf{pk}_x$), $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i}$ and $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i+1}$ output $f_x(m^*_0)$ and $f_x(m^*_1)$, respectively. Since $\overline{\mathbf{NonCorrupt}_i}$ occurs, the adversary must be valid (with respect to the $(i{+}1)$-th slot). Hence, the circuits return the same output $f^*_{i+1}(m^*_0) = f_x(m^*_0) = f_x(m^*_1) = f^*_{i+1}(m^*_1)$.

By combining the above cases, we conclude that, with overwhelming probability over the choice of hk, $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i}$ and $C_{\mathsf{hk},h,m^*_b,m^*_{1-b},i+1}$ are functionally-equivalent. Hence, Subclaim 7 follows by the security of the iO obfuscator Obf.

Subclaim 8 $\widetilde{\mathbf{H}}^b_{6,i}(\lambda) \approx_c \widetilde{\mathbf{H}}^b_{7,i}(\lambda)$*, for* $b \in \{0,1\}$ *and* $i \in \{0\} \cup [L-1]$.

Proof. By definition, $\widetilde{\mathbf{H}}^b_{6,L-1}(\lambda)$ and $\widetilde{\mathbf{H}}^b_{7,L-1}(\lambda)$ are identical, for $i = L-1$,. On the other hand, for $i < L-1$, the claim follows by the index hiding property of SSB and the proof is identical to that of Lemma 8.

Claim 5 follows by combining Subclaims 7 and 8.

Finally, by combining Claim 4 and Claim 5, Equations (7) and (8), and the triangular inequality, we conclude that Lemma 9 holds.

Lemma 10. $\mathbf{H}^b_L(\lambda) \approx_c \mathbf{H}^b_{L+1}(\lambda)$*, for* $b \in \{0,1\}$.

Proof. The lemma follows by using an identical argument to that of Lemma 7.
By combining Lemmas 7 to 10 and the fact that $\mathbf{H}^b_{L+1}(\lambda) \equiv \mathbf{H}^{1-b}_{-1}(\lambda)$, we conclude that Construction 2 is secure.

Final iO-based RFE scheme. By combining the slotted RFE scheme of Construction 2 and the ("power-of-two") transformation of Construction 3 (Appendix B), we obtain the following corollary.

Corollary 2. *Under (succinct and efficient) SSB hash functions and iO, there exists a secure and perfectly correct RFE scheme supporting any class of functions* $\mathcal{F} = \{f_i : \mathcal{M} \to \mathcal{Y}\}$ *of size* $|\mathcal{F}| = 2^{\mathsf{poly}(\lambda)}$ *and satisfying the following properties:*

- $(\mathsf{poly}(\lambda), \mathsf{poly}(\lambda), \mathsf{poly}(\lambda))$*-compactness;*
- $(\mathsf{poly}(\lambda), L \cdot \mathsf{poly}(\lambda), O(\log L), \mathsf{poly}(\lambda))$*-efficiency.*

Recall that L *stands for* current *number of registered users (unbounded case).*

Proof. The corollary follows by leveraging Definition 5, Theorems 7 to 10 and 11 to 13 and by setting the maximum number of users/slots 2^ℓ of Construction 3 and Theorem 12 to 2^λ. ◻

Acknowledgments. The first author was supported by the Carlsberg Foundation under the Semper Ardens Research Project CF18-112 (BCM). The second and the sixth author were partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU and by Sapienza University under the project SPECTRA. The third author was partially supported by the European Union (ERC AdG REWORC - 101054911), and by True Data 8 (Distributed Ledger & Multiparty Computation) under the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE. The fourth author was partially funded by the German Federal Ministry of Education and Research (BMBF) in the course of the 6GEM research hub under grant number 16KISK038 and by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972.

References

ABSV15. Prabhanjan Ananth, Zvika Brakerski, Gil Segev, and Vinod Vaikuntanathan. From selective to adaptive security in functional encryption. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 657–677. Springer, Heidelberg, August 2015.

Agr19. Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: New methods for bootstrapping and instantiation. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 191–225. Springer, Heidelberg, May 2019.

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, Heidelberg, August 2015.

AJL+19. Prabhanjan Ananth, Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. Indistinguishability obfuscation without multilinear maps: New paradigms via low degree weak pseudorandomness and security amplification. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 284–332. Springer, Heidelberg, August 2019.

AS17. Prabhanjan Ananth and Amit Sahai. Projective arithmetic functional encryption and indistinguishability obfuscation from degree-5 multilinear maps. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 152–181. Springer, Heidelberg, April / May 2017.

AY20. Shweta Agrawal and Shota Yamada. Optimal broadcast encryption from pairings and LWE. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 13–43. Springer, Heidelberg, May 2020.

BCFG17. Carmen Elisabetta Zaira Baltico, Dario Catalano, Dario Fiore, and Romain Gay. Practical functional encryption for quadratic functions with applications to predicate encryption. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 67–98. Springer, Heidelberg, August 2017.

BDGM20. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate iO from homomorphic encryption schemes. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part I*, volume 12105 of *LNCS*, pages 79–109. Springer, Heidelberg, May 2020.

BDGM22. Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Factoring and pairings are not necessary for io: Circular-secure lwe suffices. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

BFF+19. Gilles Barthe, Edvard Fagerholm, Dario Fiore, John C. Mitchell, Andre Scedrov, and Benedikt Schmidt. Automated analysis of cryptographic assumptions in generic group models. *Journal of Cryptology*, 32(2):324–360, April 2019.

BGG+14. Dan Boneh, Craig Gentry, Sergey Gorbunov, Shai Halevi, Valeria Nikolaenko, Gil Segev, Vinod Vaikuntanathan, and Dhinakaran Vinayagamurthy. Fully key-homomorphic encryption, arithmetic circuit ABE and compact garbled circuits. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 533–556. Springer, Heidelberg, May 2014.

BGI+01. Boaz Barak, Oded Goldreich, Russell Impagliazzo, Steven Rudich, Amit Sahai, Salil P. Vadhan, and Ke Yang. On the (im)possibility of obfuscating programs. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 1–18. Springer, Heidelberg, August 2001.

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

Bit20. Nir Bitansky. Verifiable random functions from non-interactive witness-indistinguishable proofs. *Journal of Cryptology*, 33(2):459–493, April 2020.

BS15. Zvika Brakerski and Gil Segev. Function-private functional encryption in the private-key setting. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 306–324. Springer, Heidelberg, March 2015.

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, Heidelberg, March 2011.

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.

CDSG+20. Jérémy Chotard, Edouard Dufour-Sans, Romain Gay, Duong Hieu Phan, and David Pointcheval. Dynamic decentralized functional encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 747–775. Springer, Heidelberg, August 2020.

CES21. Kelong Cong, Karim Eldefrawy, and Nigel P. Smart. Optimizing registration based encryption. In Maura B. Paterson, editor, *18th IMA International Conference on Cryptography and Coding*, volume 13129 of *LNCS*, pages 129–157. Springer, Heidelberg, December 2021.

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.

GGHZ16. Sanjam Garg, Craig Gentry, Shai Halevi, and Mark Zhandry. Functional encryption without obfuscation. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part II*, volume 9563 of *LNCS*, pages 480–511. Springer, Heidelberg, January 2016.

GHKW17. Rishab Goyal, Susan Hohenberger, Venkata Koppula, and Brent Waters. A generic approach to constructing and proving verifiable random functions. In Yael Kalai and Leonid Reyzin, editors, *TCC 2017, Part II*, volume 10678 of *LNCS*, pages 537–566. Springer, Heidelberg, November 2017.

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, Heidelberg, April 2019.

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, Heidelberg, November 2018.

GJLS21. Romain Gay, Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from simple-to-state hard problems: New assumptions, new techniques, and simplification. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 97–126. Springer, Heidelberg, October 2021.

GKMR22. Noemi Glaeser, Dimitris Kolonelos, Giulio Malavolta, and Ahmadreza Rahimi. Efficient registration-based encryption. Cryptology ePrint Archive, Paper 2022/1505, 2022.

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.

GLSW15. Craig Gentry, Allison Bishop Lewko, Amit Sahai, and Brent Waters. Indistinguishability obfuscation from the multilinear subgroup elimination assumption. In Venkatesan Guruswami, editor, *56th FOCS*, pages 151–170. IEEE Computer Society Press, October 2015.

GP21. Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 736–749, 2021.

GPSW06. Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-based encryption for fine-grained access control of encrypted data. In Ari Juels, Rebecca N. Wright, and Sabrina De Capitani di Vimercati, editors, *ACM CCS 2006*, pages 89–98. ACM Press, October / November 2006. Available as Cryptology ePrint Archive Report 2006/309.

GPSZ17. Sanjam Garg, Omkant Pandey, Akshayaram Srinivasan, and Mark Zhandry. Breaking the sub-exponential barrier in obfustopia. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 156–181. Springer, Heidelberg, April / May 2017.

GV20. Rishab Goyal and Satyanarayana Vusirikala. Verifiable registration-based encryption. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 621–651. Springer, Heidelberg, August 2020.

HJO+16. Brett Hemenway, Zahra Jafargholi, Rafail Ostrovsky, Alessandra Scafuro, and Daniel Wichs. Adaptively secure garbled circuits from one-way functions. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 149–178. Springer, Heidelberg, August 2016.

HLWW22. Susan Hohenberger, George Lu, Brent Waters, and David J Wu. Registered attribute-based encryption. *Cryptology ePrint Archive*, 2022.

HW15. Pavel Hubacek and Daniel Wichs. On the communication complexity of secure function evaluation with long output. In Tim Roughgarden, editor, *ITCS 2015*, pages 163–172. ACM, January 2015.

JLMS19. Aayush Jain, Huijia Lin, Christian Matt, and Amit Sahai. How to leverage hardness of constant-degree expanding polynomials overa $\mathbb{R}$ to build $i\mathcal{O}$. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part I*, volume 11476 of *LNCS*, pages 251–281. Springer, Heidelberg, May 2019.

JLS21. Aayush Jain, Huijia Lin, and Amit Sahai. Indistinguishability obfuscation from well-founded assumptions. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 60–73, 2021.

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

KSW08. Jonathan Katz, Amit Sahai, and Brent Waters. Predicate encryption supporting disjunctions, polynomial equations, and inner products. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, pages 146–162. Springer, Heidelberg, April 2008.

Lin16. Huijia Lin. Indistinguishability obfuscation from constant-degree graded encoding schemes. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part I*, volume 9665 of *LNCS*, pages 28–57. Springer, Heidelberg, May 2016.

Lin17. Huijia Lin. Indistinguishability obfuscation from SXDH on 5-linear maps and locality-5 PRGs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 599–629. Springer, Heidelberg, August 2017.

LV16. Huijia Lin and Vinod Vaikuntanathan. Indistinguishability obfuscation from DDH-like assumptions on constant-degree graded encodings. In Irit Dinur, editor, *57th FOCS*, pages 11–20. IEEE Computer Society Press, October 2016.

MQR22. Mohammad Mahmoody, Wei Qi, and Ahmadreza Rahimi. Lower bounds for the number of decryption updates in registration-based encryption. Cryptology ePrint Archive, Report 2022/1285, 2022. https://eprint.iacr.org/2022/1285.

O'N10. Adam O'Neill. Definitional issues in functional encryption. Cryptology ePrint Archive, Report 2010/556, 2010. https://eprint.iacr.org/2010/556.

OPWW15. Tatsuaki Okamoto, Krzysztof Pietrzak, Brent Waters, and Daniel Wichs. New realizations of somewhere statistically binding hashing and positional accumulators. In Tetsu Iwata and Jung Hee Cheon, editors, *ASIACRYPT 2015, Part I*, volume 9452 of *LNCS*, pages 121–145. Springer, Heidelberg, November / December 2015.

Sha84. Adi Shamir. Identity-based cryptosystems and signature schemes. In G. R. Blakley and David Chaum, editors, *CRYPTO'84*, volume 196 of *LNCS*, pages 47–53. Springer, Heidelberg, August 1984.

SSW09. Emily Shen, Elaine Shi, and Brent Waters. Predicate privacy in encryption systems. In Omer Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 457–473. Springer, Heidelberg, March 2009.

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, Heidelberg, May 2005.

SW14. Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: deniable encryption, and more. In David B. Shmoys, editor, *46th ACM STOC*, pages 475–484. ACM Press, May / June 2014.

WW21. Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious LWE sampling. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part III*, volume 12698 of *LNCS*, pages 127–156. Springer, Heidelberg, October 2021.

A Slotted Registered Inner-Product Encryption

Bilinear groups. Our slotted RIPE is based on asymmetric bilinear groups for which we first provide some notations here. We use cyclic groups of prime order q with an asymmetric bilinear map endowed on them. In particular, we assume a PPT algorithm $\mathsf{GroupGen}$ that takes a security parameter λ as input and outputs $\mathcal{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}, q, g_1, g_2, e)$, where $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_\mathsf{T}$ are cyclic groups of prime order q, g_1 (resp. g_2) is random generator in $\mathbb{G}_1$ (resp. $\mathbb{G}_2$) and $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_\mathsf{T}$ is a non-degenerate bilinear map.

We now present the slotted RIPE definitions below. Let $n = n(\lambda)$ be a polynomial in λ and q be a prime. A slotted RIPE with message space $\mathcal{M}$ and attribute space $\mathcal{U}$ is composed of the following polynomial-time algorithms:

$\mathsf{Setup}(1^\lambda, 1^n, 1^L)$: On input the security parameter 1^λ, the vector length 1^n, and the number of slots 1^L, the randomized setup algorithm outputs a common reference string crs.

$\mathsf{KGen}(\mathsf{crs}, i)$: On input the common reference string crs and a slot index $i \in [L]$, the randomized key-generation algorithm outputs a public key pk_i and a secret key sk_i.

$\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i)$**:** On input the common reference string crs, a slot index $i \in [L]$, and a public key pk_i, the deterministic key validation algorithm outputs a decision bit $b \in \{0, 1\}$.

$\mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_i, \mathbf{x}_i))_{i\in[L]})$: On input the common reference string crs and a L pairs $(\mathsf{pk}_1, , \mathbf{x}_1), \ldots, (\mathsf{pk}_L, , \mathbf{x}_L)$ each composed of a public key pk_i and its corresponding (non-zero) vector $\mathbf{x}_i \in \mathcal{U}$, the deterministic aggregation algorithm outputs the master public key mpk and a L helper decryption keys $\mathsf{hsk}_1, \ldots, \mathsf{hsk}_L$.

$\mathsf{Enc}(\mathsf{mpk}, \mathbf{y}, m)$**:** On input the master public key mpk, a (non-zero) attribute vector $\mathbf{y} \in \mathcal{U}$, and a message $m \in \mathcal{M}$, the randomized encryption algorithm outputs a ciphertext c.

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, c)$**:** On input a secret key sk, an helper decryption key hsk, and a ciphertext c, the deterministic decryption algorithm outputs a message $m \in \mathcal{M} \cup \{\perp\}$.

Definition 14 (Completeness of slotted RIPE). *A slotted RIPE scheme* $\Pi_\mathsf{sRIPE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and attribute space* $\mathcal{U}$ *is complete if* $\forall\lambda \in \mathbb{N}, n \in \mathbb{N}, L \in \mathbb{N}$ *and* $\forall i \in [L]$,

$$\mathbb{P}\big[\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i) = 1 \big| \mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^n, 1^L), (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{crs}, i)\big] = 1.$$

Definition 15 (Perfect correctness of slotted RIPE). *A slotted RIPE scheme* $\Pi_\mathsf{sRIPE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and attribute space* $\mathcal{U}$ *is correct if* $\forall\lambda \in \mathbb{N}, n \in \mathbb{N}, L \in \mathbb{N}, i \in [L]$, $\forall\mathsf{crs}$ *output by* $\mathsf{Setup}(1^\lambda, 1^n, 1^L)$, $\forall(\mathsf{pk}_i, \mathsf{sk}_i)$ *output by* $\mathsf{KGen}(\mathsf{crs}, i)$, $\forall$ *collection of public key* $\{\mathsf{pk}_j\}_{j\in[L]\setminus\{i\}}$ *such that* $\mathsf{IsValid}(\mathsf{crs}, j, \mathsf{pk}_j) = 1$, $\forall m \in \mathcal{M}$, $\forall \mathbf{x}_1, \ldots, \mathbf{x}_L \in \mathcal{U}$, *and* $\forall \mathbf{y} \in \mathcal{U}$ *such that* $\langle \mathbf{x}_i, \mathbf{y}\rangle = 0$ *for every* $i \in [L]$, *we have:*

$$\mathbb{P}\left[\mathsf{Dec}(\mathsf{sk}_i, \mathsf{hsk}_i, c) = m \,\middle|\, \begin{array}{c}(\mathsf{mpk}, (\mathsf{hsk}_j)_{j\in[L]}) = \mathsf{Aggr}(\mathsf{crs}, ((\mathsf{pk}_j, \mathbf{x}_j))_{j\in[L]}),\\ c \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{mpk}, \mathbf{y}, m)\end{array}\right] = 1.$$

Definition 16 (Compactness and efficiency for slotted RIPE). *The definition is identical to that of RFE (Definition 12).*

Definition 17 (Security of slotted RIPE). *Let* $\Pi_\mathsf{sRIPE} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{IsValid}, \mathsf{Aggr}, \mathsf{Enc}, \mathsf{Dec})$ *be a slotted RIPE scheme with message space* $\mathcal{M}$ *and attribute space* $\mathcal{U}$. *For any adversary* A, *we define the following security experiment* $\mathbf{Game}^{\mathsf{sRIPE}}_{\Pi_\mathsf{sRIPE},\mathsf{A}}(\lambda, b)$ *with respect to bit* $b \in \{0, 1\}$ *between* A *and a challenger.*

- **Setup phase:** *Upon getting an attribute length* n *and a slot count* L *from the adversary* A, *the challenger samples* $\mathsf{crs} \leftarrow\!\!\$\ \mathsf{Setup}(1^\lambda, 1^n, 1^L)$ *and gives* crs *to* A. *The challenger also initializes a counter* $\mathsf{ctr} = 0$, *a dictionary* D, *and a set of slot indices* $\mathcal{C}_L = \emptyset$ *to account for corrupted slots.*
- **Pre-challenge query phase:** A *can issue the following queries.*
  - **Key-generation query:** A *specifies a slot index* $i \in [L]$. *As a response, the challenger increments* $\mathsf{ctr} = \mathsf{ctr} + 1$, *samples* $(\mathsf{pk}_\mathsf{ctr}, \mathsf{sk}_\mathsf{ctr}) \leftarrow\!\!\$\ \mathsf{KGen}(\mathsf{crs}, i)$, *updates the dictionary as* $\mathsf{D}[\mathsf{ctr}] = (i, \mathsf{pk}_\mathsf{ctr}, \mathsf{sk}_\mathsf{ctr})$ *and replies with* $(\mathsf{ctr}, \mathsf{pk}_\mathsf{ctr})$ *to* A.

– **Corruption query:** A *specifies an index* $c \in [\mathsf{ctr}]$. *In response, the challenger looks up the tuple* $\mathsf{D}[c] = (i', \mathsf{pk}', \mathsf{sk}')$ *and replies with* sk' *to* A.

- **Challenge phase:** *For each* $i \in [L]$, A *specifies a tuple* $(c_i, \mathbf{x}_i, \mathsf{pk}_i^*)$ *where:*
  - *either* $c_i \in [\mathsf{ctr}]$ *that refers to a challenger-generated key from before which it associates with a non-zero predicate* $\mathbf{x}_i \in \mathcal{U}$*: in this case, the challenger looks up* $\mathsf{D}[c_i] = (i', \mathsf{pk}', \mathsf{sk}')$ *and halts if* $i \neq i'$. *Else, the challenger sets* $\mathsf{pk}_i^* = \mathsf{pk}'$. *Further, if* A *issued a corrupt query before on* c_i*, the challenger adds* i *to* $\mathcal{C}_L$.
  - *or* $c_i = \perp$ *that refers to a self-generated (and corrupt) secret key for an arbitrary non-zero predicate* $\mathbf{x}_i \in \mathcal{U}$*: in this case, the challenger aborts if* $\mathsf{IsValid}(\mathsf{crs}, i, \mathsf{pk}_i^*) = 0$. *Else if* pk_i^* *is valid, it adds the index* i *to* $\mathcal{C}_L$.

  Additionally, A *sends a challenge pair* $(\mathbf{y}_0, m_0), (\mathbf{y}_1, m_1) \in \mathcal{U} \times \mathcal{M}$. *In response, the challenger computes* $(\mathsf{mpk}, (\mathsf{hsk}_i)_{i\in[L]}) = \mathsf{Aggr}(\mathsf{crs}, (\mathsf{pk}_i^*, \mathbf{x}_i)_{i\in[L]})$ *and* $c^* \leftarrow\!\!\$\ \mathsf{Enc}(\mathsf{mpk}, \mathbf{y}_b, m_b)$*, and replies with* c^* *to* A.
- **Output phase:** A *returns a bit* $b' \in \{0,1\}$ *which is also the output of the experiment.*

A *is called admissible if the challenge pair* $(\mathbf{y}_0, m_0), (\mathbf{y}_1, m_1)$ *satisfy the following:*

– $\forall \mathbf{x}_i \in \mathcal{U}$ *with* $i \in \mathcal{C}_L$*, it holds that:*

$$\text{either } \langle \mathbf{x}_i, \mathbf{y}_0\rangle = \langle \mathbf{x}_i, \mathbf{y}_1\rangle = 0 \quad \text{or} \quad \text{both } \langle \mathbf{x}_i, \mathbf{y}_0\rangle, \langle \mathbf{x}_i, \mathbf{y}_1\rangle \neq 0, \text{ and}$$

– *if* $\exists \mathbf{x}_i \in \mathcal{U}$ *with* $i \in \mathcal{C}_L$ *such that* $\langle \mathbf{x}_i, \mathbf{y}_0\rangle = \langle \mathbf{x}_i, \mathbf{y}_1\rangle = 0$*, then* $m_0 = m_1$.

We say that Π_{sRIPE} *is secure if for all polynomials* $n = n(\lambda), L = L(\lambda)$ *and for all PPT and admissible* A *in the above security hybrid, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for all* $\lambda \in \mathbb{N}$,

$$\left|\Pr\left[\mathbf{Game}^{\mathsf{sRIPE}}_{\Pi_{\mathsf{sRIPE}},\mathsf{A}}(\lambda, 0) = 1\right] - \Pr\left[\mathbf{Game}^{\mathsf{sRIPE}}_{\Pi_{\mathsf{sRIPE}},\mathsf{A}}(\lambda, 1) = 1\right]\right| = \mathsf{negl}(\lambda).$$

B From slotted RFE to RFE

In this section, we show a construction that transform a slotted RFE to (standard) RFE. The construction is identical to that proposed by Hohenberger *et al.* [HLWW22] (for the RABE case). For self-containment, we recall the construction below.

Construction 3 *Let* $\Pi_{\mathsf{sRFE}} = (\mathsf{sRFE.Setup}, \mathsf{sRFE.KGen}, \mathsf{sRFE.IsValid}, \mathsf{sRFE.Aggr}, \mathsf{sRFE.Enc}, \mathsf{sRFE.Dec})$ *be a slotted RFE scheme with message space* $\mathcal{M}$ *and function space* $\mathcal{F} = \{f_i : \mathcal{M} \to \mathcal{Y}\}$. *We build a RFE scheme* $\Pi_{\mathsf{RFE}} = (\mathsf{Setup}, \mathsf{KGen}, \mathsf{RegPK}, \mathsf{Enc}, \mathsf{Update}, \mathsf{Dec})$ *with message space* $\mathcal{M}$ *and function space* $\mathcal{F}$ *as follows. The construction makes use of the following conventions:*

- *Without loss of generality, we assume that the number of users is a power of two* $L = 2^\ell$ *for* $\ell \in \mathsf{poly}(\lambda)$.
- *The construction uses* $\ell+1$ *independent instantiations of* Π_{sRFE}. *The* k*-th slotted RFE handles* 2^{k-1} *slots where* $k \in [\ell+1]$.
- *The state* α *(managed by the KC) is composed of the following elements:*
  - *A counter* ctr *that tracks the current number of registered users.*
  - *A dictionary* D_1 *that maps* $(k, i) \in [\ell+1] \times [2^{k-1}]$ *into pairs* (pk, f)*, i.e., it stores the public key and the corresponding function associated to the* i*-th slot of the* k*-th slotted RFE scheme (observe that the* k*-th slotted RFE scheme supports* 2^{k-1} *slots).*
  - *A dictionary* D_2 *that maps* $(k, i) \in [\ell+1] \times [L]$ *into an helper decryption key* $\mathsf{hsk}_{k,i}$*, i.e., it stores the helper decryption key* $\mathsf{hsk}_{k,i}$ *of the* i*-th slot of the* k*-th slotted RFE scheme.*
  - *The current master public key* $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_1, \dots, \mathsf{mpk}_{\ell+1})$.

  On initialization, the initial state is set to $\alpha = \perp$ *which is parsed as* $\alpha = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$ *where* $\mathsf{ctr} = 0$, $\mathsf{D}_1 = \emptyset$, $\mathsf{D}_2 = \emptyset$*, and* $\mathsf{mpk} = (0, \perp, \dots, \perp)$.

$\mathsf{Setup}(1^\lambda, 1^L, |\mathcal{F}|)$**:** *On input the security parameter* 1^λ*, a bound (represented in unary) on the number of users* 1^L*, and the size* $|\mathcal{F}|$ *(in binary) of the function space* $\mathcal{F}$*, the randomized setup algorithm computes* $\mathsf{crs}_i \leftarrow\!\!\$\ \mathsf{sRFE.Setup}(1^\lambda, 1^L, |\mathcal{F}|)$ *for* $i \in [\ell+1]$*, and outputs* $\mathsf{crs} = (\mathsf{crs}_1, \dots, \mathsf{crs}_{\ell+1})$.

$\mathsf{KGen}(\mathsf{crs}, \alpha)$**:** *On input the common reference string* $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_{\ell+1})$ *and a state* $\alpha = (\mathsf{ctr}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$*, the randomized key-generation algorithm runs* $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow\!\!\$\, \mathsf{sRFE.KGen}(\mathsf{crs}_k, i_k)$ *for each* $k \in [\ell+1]$ *where* $i_k = (\mathsf{ctr} \mod 2^{k-1}) + 1 \in [2^{k-1}]$ *is the slot index corresponding to the* k*-th slotted RFE scheme. Finally, it outputs the public key* $\mathsf{pk} = (\mathsf{ctr}, \mathsf{pk}_1, \ldots, \mathsf{pk}_{\ell+1})$ *and the secret key* $\mathsf{sk} = (\mathsf{ctr}, \mathsf{sk}_1, \ldots, \mathsf{sk}_{\ell+1})$*.*

$\mathsf{RegPK}(\mathsf{crs}, \alpha, \mathsf{pk}, f)$**:** *On input the common reference string* $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_{\ell+1})$*, a state* $\alpha = (\mathsf{ctr}_\alpha, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk} = (\mathsf{ctr}_\alpha, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_{\ell+1}))$*, a public key* $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_1, \ldots, \mathsf{pk}_{\ell+1})$*, and a function* $f \in \mathcal{F}$*, the deterministic registration algorithm proceeds as follows:*

1. *For each* $k \in [\ell + 1]$*, let* $i_k = (\mathsf{ctr}_\alpha \mod 2^{k-1}) + 1 \in [2^{k-1}]$ *be the slot index corresponding to the* k*-th slotted RFE.*
2. *For each* $k \in [\ell + 1]$*, if* $(\mathsf{sRFE.IsValid}(\mathsf{crs}_k, i_k, \mathsf{pk}_k) = 0 \lor \mathsf{ctr}_{\mathsf{pk}} \neq \mathsf{ctr}_\alpha)$ *then the registration algorithm halts and returns the state* α *and master public key* mpk *(i.e., registration failed).*
3. *Otherwise, for each* $k \in [\ell+1]$*, the registration algorithm sets* $\mathsf{D}_1[k, i_k] = (\mathsf{pk}, f)$*. Moreover, proceeds as follows:*
   - *If* $i_k = 2^{k-1}$ *(i.e., the* k*-th slotted RFE scheme is full) executes the following steps:*
     (a) *Compute* $(\mathsf{mpk}'_k, \mathsf{hsk}'_{k,1}, \ldots, \mathsf{hsk}'_{k,2^{k-1}}) = \mathsf{sRFE.Aggr}(\mathsf{crs}_k, \mathsf{D}_1[k, 1], \ldots, \mathsf{D}_1[k, 2^{k-1}])$*, i.e., it aggregates all the public keys associated to the* k*-th slotted RFE scheme.*
     (b) *Set* $\mathsf{D}_2[\mathsf{ctr}_\alpha + 1 - 2^{k-1} + i, k] = \mathsf{hsk}'_{k,i}$ *for every* $i \in [2^{k-1}]$*.*
   - *If* $i_k \neq 2^{k-1}$*, it sets* $\mathsf{mpk}'_k = \mathsf{mpk}_k$ *(the* k*-th master public key of the* k*-th slotted RFE is unchanged).*

Finally, it returns the new master public key $\mathsf{mpk}' = (\mathsf{ctr}_\alpha + 1, \mathsf{mpk}'_1, \ldots, \mathsf{mpk}'_{\ell+1})$ *and the new state* $\alpha' = (\mathsf{ctr}_\alpha + 1, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk}')$*.*

$\mathsf{Enc}(\mathsf{mpk}, m)$**:** *On input the master public key* $\mathsf{mpk} = (\mathsf{ctr}, \mathsf{mpk}_1, \ldots, \mathsf{mpk}_{\ell+1})$ *and a message* $m \in \mathcal{M}$*, the randomized encryption algorithm proceeds as follows:*

1. *For every* $k \in [\ell + 1]$ *such that* $\mathsf{mpk}_k \neq \bot$*, it computes* $c_k \leftarrow\!\!\$\, \mathsf{sRFE.Enc}(\mathsf{mpk}_k, m)$*.*
2. *For every* $k \in [\ell + 1]$ *such that* $\mathsf{mpk}_k = \bot$*, it sets* $c_k = \bot$*.*

Finally, it outputs $c = (\mathsf{ctr}, c_1, \ldots, c_{\ell+1})$*.*

$\mathsf{Update}(\mathsf{crs}, \alpha, \mathsf{pk})$**:** *On input the common reference string* $\mathsf{crs} = (\mathsf{crs}_1, \ldots, \mathsf{crs}_{\ell+1})$*, the state* $\alpha = (\mathsf{ctr}_\alpha, \mathsf{D}_1, \mathsf{D}_2, \mathsf{mpk})$*, and a public key* $\mathsf{pk} = (\mathsf{ctr}_{\mathsf{pk}}, \mathsf{pk}_1, \ldots, \mathsf{pk}_{\ell+1})$*, the deterministic update algorithm returns* $\bot$ *if* $\mathsf{ctr}_{\mathsf{pk}} \geq \mathsf{ctr}_\alpha$*. Otherwise, it returns* $\mathsf{hsk} = (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_{\ell+1})$ *where* $\mathsf{hsk}_k = \mathsf{D}_2[\mathsf{ctr}_{\mathsf{pk}} + 1, k]$ *for every* $k \in [\ell + 1]$*.*

$\mathsf{Dec}(\mathsf{sk}, \mathsf{hsk}, m)$**:** *On input a secret key* $\mathsf{sk} = (\mathsf{ctr}_{\mathsf{sk}}, \mathsf{sk}_1, \ldots, \mathsf{sk}_{\ell+1})$*, a helper description key* $\mathsf{hsk} = (\mathsf{hsk}_1, \ldots, \mathsf{hsk}_{\ell+1})$*, and a ciphertext* $c = (\mathsf{ctr}_c, c_1, \ldots, c_{\ell+1})$*, the deterministic decryption algorithm returns* $\bot$ *if* $\mathsf{ctr}_c \leq \mathsf{ctr}_{\mathsf{sk}}$*. Otherwise, it computes the largest* $k \in [\ell + 1]$ *such that the* k*-th bit of* ctr_c *and* $\mathsf{ctr}_{\mathsf{sk}}$ *differ (here, we assume that bits are* 1*-indexed starting from the least significant bit). If* $\mathsf{hsk}_k = \bot$*, then the decryption algorithm returns* $\mathsf{getUpdate}$*. Otherwise, it returns* $y = \mathsf{sRFE.Dec}(\mathsf{sk}_k, \mathsf{hsk}_k, c_k)$*.*

Correctness, efficiency, and security of Construction 3 follow by using an identical argument to that of [HLWW22].

Theorem 11 (Perfect correctness of Construction 3). *Let* Π_{sRFE} *as above. If* Π_{sRFE} *is complete (Definition 10) and perfectly correct (Definition 11), then* Π_{RFE} *from Construction 3 is perfectly correct (Definition 7).*

Proof. The theorem follows by using an identical argument to that of [HLWW22, Theorem 6.2]. □

Theorem 12 (Compactness and efficiency of Construction 3). *Let* Π_{sRFE} *as above where* $\mathcal{F}$ *is the class of functions supported by* Π_{sRFE}*. and* $t_{\mathsf{crs}} = t_{\mathsf{crs}}(\lambda, L, |\mathcal{F}|), t_{\mathsf{mpk}} = t_{\mathsf{mpk}}(\lambda, L, |\mathcal{F}|), t_{\mathsf{hsk}} = t_{\mathsf{hsk}}(\lambda, L, |\mathcal{F}|), t_{\mathsf{KGen}} = t_{\mathsf{KGen}}(\lambda, L, |\mathcal{F}|), t_{\mathsf{IsValid}} = t_{\mathsf{IsValid}}(\lambda, L, |\mathcal{F}|), t_{\mathsf{Aggr}} = t_{\mathsf{Aggr}}(\lambda, L, |\mathcal{F}|)$ *be polynomials in the security parameter,* L*, and* $|\mathcal{F}|$*. If* Π_{sRFE} *is* $(t_{\mathsf{crs}}, t_{\mathsf{mpk}}, t_{\mathsf{hsk}})$*-compact and* $(t_{\mathsf{KGen}}, t_{\mathsf{IsValid}}, t_{\mathsf{Aggr}})$*-efficient (Definition 12), then* Π_{RFE} *from Construction 3 is*

- $(O(t_{\mathsf{crs}} \cdot \log L), O(t_{\mathsf{mpk}} \cdot \log L), O(t_{\mathsf{hsk}} \cdot \log L))$*-compact, and*
- $(O(t_{\mathsf{KGen}} \cdot \log L), O(t_{\mathsf{IsValid}} \cdot \log L + t_{\mathsf{Aggr}} + t_{\mathsf{hsk}} \cdot \widetilde{L}), O(\log \widetilde{L}), O(t_{\mathsf{hsk}} \cdot \log L))$*-efficient (Definition 8),*

where $L = 2^\ell$ *is the maximum number of supported users (see Construction 3) and* $\widetilde{L} \leq L$ *is the* current *number of registered users at the time of execution.*

Proof. Let $L = 2^\ell$ (as in Construction 3). We demonstrate each property individually:

- $O(t_{\mathsf{crs}} \cdot \log L)$**-compact** crs**:** The common reference string (of Construction 3) is composed of $\ell+1$ common reference strings (each of size t_{crs}) of the underlying slotted RFE scheme. See also [HLWW22, Theorem 6.5].
- $O(t_{\mathsf{mpk}} \cdot \log L)$**-compact** mpk**:** Each master public key (generated by Construction 3) is composed of a counter ctr (of size $|\mathsf{ctr}| \leq \ell$) and $\ell + 1$ master public keys (each of size t_{mpk}) of the underlying slotted RFE scheme. See also [HLWW22, Theorem 6.5].
- $O(t_{\mathsf{hsk}} \cdot \log L)$**-compact** hsk**:** Similarly, each helper decryption key (generated by Construction 3) is composed of $\ell + 1$ helper decryption keys (each of size t_{hsk}) of the underlying slotted RFE scheme. See also [HLWW22, Theorem 6.5 and 6.6].
- $O(t_{\mathsf{KGen}} \cdot \log L)$**-efficient** KGen**:** The key-generation (of Construction 3) executes $\ell + 1$ times the key-generation algorithm of the underlying slotted RFE. Hence, its running time is bounded by $O(t_{\mathsf{KGen}} \cdot \log L)$ where t_{KGen} is the running time of the slotted RFE key-generation algorithm.
- $O(t_{\mathsf{IsValid}} \cdot \log L + t_{\mathsf{Aggr}} + \widetilde{L} \cdot t_{\mathsf{hsk}})$**-efficient** RegPK**:** The (worst-case) running time of RegPK can be derived by estimating the running time for registering the a generic $\widetilde{L} = 2^k$-th user (for $k \in [\ell + 1]$). In such a case, the running time of RegPK is composed of $\ell + 1$ executions of IsValid and a single execution of Aggr whose (individual) running times are bounded by t_{IsValid} and t_{Aggr}, respectively. In addition, the newly generated $\widetilde{L} = 2^k$ helper decryption keys (output by Aggr) are stored into the dictionary D_2 (this takes time linear in $2^k = \widetilde{L}$ in the RAM model of computation) and each helper decryption key (of the underlying slotted RFE) is of size t_{hsk}. Hence, the final (worst-case) running time of the registration algorithm is $O(t_{\mathsf{IsValid}} \cdot \log L + t_{\mathsf{Aggr}} + t_{\mathsf{hsk}} \cdot \widetilde{L})$ in the RAM model of computation.
- $(O(\log \widetilde{L}), O(t_{\mathsf{hsk}} \cdot \log L))$**-efficient** Update**:** The challenger executes Update at most $O(\log \widetilde{L})$ (for a generic number $\widetilde{L} = 2^k$ of current registered users) because of the following reasons: (i) each helper decryption key hsk is composes of $\ell + 1$ helper decryption keys $(\mathsf{hsk}_1, \ldots, \mathsf{hsk}_{\ell+1})$ of the underlying slotted RFE scheme, (ii) Update is invoked (by an user) only when one of the the $\ell + 1$ helper decryption keys hsk_i is $\perp$, and (iii) after the execution of Update, hsk_i is no longer $\perp$.
  Regarding the running time, Update simply compares two $(\ell+1)$-bits counters and looks up for the $\ell+1$ helper decryption keys stored in the dictionary D_2. Moreover, by definition, each helper decryption key of slotted RFE is of size t_{hsk}. We conclude that Update runs in time $O(t_{\mathsf{hsk}} \cdot \log L)$ in the RAM model of computation.
  See also [HLWW22, Theorem 6.6].

□

Theorem 13 (Security of Construction 3). *Let Π_{sRFE} as above. If Π_{sRFE} is secure (Definition 13), then Π_{RFE} from Construction 3 is secure (Definition 9).*

Proof. The theorem follows by using an identical argument to that of [HLWW22, Theorem 6.7] except that we make use of the validity of the adversary with respect to the RFE experiment (instead of the validity for the RABE case). □

Remark 5. As noted by [HLWW22], if the running time of Setup and the sizes of crs, mpk, and hsk (of the underlying slotted RFE) are all poly-logarithmic in the number of users, the resulting RFE scheme (output by Construction 3) supports an arbitrary number of users. This is because the resulting RFE will have the same poly-logarithmic efficiency/compactness and, in turn, this allows us to set $L = 2^\lambda$ to support an arbitrary polynomial number of users. Note that our iO-based slotted RFE scheme satisfy this requirements (see Theorem 9).